SYSTÈME D'EXPLOITATION CENTRALISES : RAPPORT PROJET MINICHAT

XAMBILI Robin

Table des matières

1 INTRODUCTION

Le but de ce projet est de développer une application de messagerie interactive afin de mettre en oeuvre les notions de base vues en TD et TP, autour de l'interaction par tubes et par segments de mémoire partagée.

2 ARCHITECTURE ET CHOIX DE CONCEPTION

2.1 RENDU

L'archive rendue contient :

Le code final console.c, serveur.c, chatmmap.c

Le rapport du projet.

2.2 VERSION SERVEUR

"Pour cette version, les messages échangés entre processus participants transitent par des tubes, selon le principe suivant :

- un processus particulier, le serveur, centralise les messages émis par les participants, et les retransmet à chacun des participants. Le serveur est donc relié à chacun des participants par deux tubes : ? un tube du participant vers le serveur, par lequel le participant envoie les messages saisis ? un tube du serveur vers le participant, par lequel le participant reçoit les messages émis par le serveur
- les processus participants attendent des messages soit de l'entrée standard (auquel cas ils les retransmettent au serveur), soit du serveur (auquel cas ils actualisent la liste des derniers messages de la conversation, et en rafraîchissent l'affichage)
- un tube particulier, lu par le serveur, écrit par les participants, permet à un nouveau participant de s'enregistrer auprès du serveur, en fournissant son identité et celle de ses tubes de communication."

2.2.1 Connexions

Lorsqu'un client veut se connecter,il crée ses tubes d'entrée et de sortie puis il écrit son pseudo dans le tube "écoute", le serveur ouvre ensuite les tubes en "read only" et "write only" puis initialise le participants dans le tableau. Le client et le serveur doivent ouvrir les deux tubes dans le même ordre pour éviter un interblocage. Le serveur ajoute un client seulement si le nombre de participants actifs est inférieur au nombre maximal de participants.

2.2.2 Déconnexions

Il y a deux cas de déconnexion pour un participants :

- Il entre "au revoir" : dans ce cas le serveur le détecte, désactive le participant et un message service aux autres participants. (Le client attend avant de fermer ses tubes (sleep(3)) pour éviter qu'il y ai un descripteur erroné dans le select du serveur)
- Un client de pseudo "fin" se connecte : dans ce cas le serveur, envoie un message service à tout les participants. Les participants détectent ce message et se déconnectent proprement. De la même façon le serveur attend avant de fermer ses tubes.

2.2.3 Select + blocage

Comme les clients produisent peu, on utilise select() pour éviter de faire un read() bloquant. Ainsi, le select() bloque tant que aucun tube d'entrée n'est prêt à la lecture.

2.2.4 Réinitialisation des buffers

Pour éviter de transmettre des résidus de message, on vide les buffers avant chaque read() ou write().

2.2.5 Retour à la ligne après read() sur l'entrée standard

La lecture de la saisie retourne un retour à la ligne '\n' en fin de chaîne, pour l'enlever on détecte la position de '\n' à l'aide de strchr() puis on le remplace par '\0'.

2.3 VERSION TABLEAU BLANC

"Dans cette version, les messages sont directement écrits par les participants dans un segment de mémoire partagée, sans transiter par un serveur intermédiaire. Tous les processus sont donc similaires."

2.3.1 Connexions

Lorsqu'un client se connecte, on envoie le message service dans la mémoire partagée et on initialise dernier0 (le numéro du dernier message) comme le numéro du dernier message de la discussion.

2.3.2 Déconnexions

Lorsqu'un client entre "au revoir", on quitte la boucle principale et on envoie le message service dans la mémoire partagée.

2.3.3 Mise à jour de l'affichage

On définit un traitant pour le signal SIGUSR1 qui va affiche la discussion et mets a jour dernier0 si le dernier message de discussion a un numéro plus grand que dernier0. Afin de pouvoir mettre à jour l'affichage à chaque nouveau message et pas seulement à l'envoi d'un message, on passe la lecture sur l'entrée standard en mode non-bloquant à l'aide de fcntl() et on envoi le signal SIGUSR1 à chaque passage de boucle.

2.3.4 Envoi de message

On ajoute un procédure envoyer(struct message) qui permet d'envoyer et message dans la mémoire partagée. Pour chaque message ajoutée, on réorganise la discussion en décalant chaque message d'un indice. Ensuite on provoque l'affichage en envoyant le signal SIGUSR1.

2.3.5 Retour à la ligne après read() sur l'entrée standard

On rencontre le même problème que dans la version serveur et on le résout de la même façon.

2.4 COMPARAISON DES VERSIONS

2.4.1 Avantages et désavantages de la version serveur

Avantages :

+ Gestion des informations de la discussion simplifiée (nombre de participants, ...)
+ Possibilité de déconnecter un participant et de mettre fin à la discussion depuis le serveur

Désavantages :

- Besoin d'un processus supplémentaire, le serveur doit être actif
- Nécessite d'ouvrir deux tubes nommés par participants : beaucoup de ressources réservées
- Nombre de participants limité

2.4.2 Avantages et désavantages de la version tableau blanc

Avantages :
+ Moins de ressources réservées
+ Possibilité de continuer une discussion arrêtée
+ Les processus sont similaires
+ Pas de limite de participants

Désavantages :
- Nécessite de faire passer plus d'informations dans les messages (Ex : numéro d'ordre)
- Le nombre de participants ne peut pas être limité, seul les participants gèrent les connexions/déconnexions

3 MÉTHODOLOGIE DE TEST

3.1 Connexions

On connecte un client et on regarde si les autres participants reçoivent bien le message de service et s'ils ne sont pas déconnectés. Dans le cas de la version serveur on vérifie que le serveur ne se bloque pas. De plus, dans le cas du serveur, on vérifie qu'on ne puisse pas connecter plus de participants que le maximum autorisé et que la demande soit bien prise en compte lorsqu'un client se déconnecte.

3.2 Déconnexions

On déconnecte un client et on regarde si les autres participants reçoivent bien le message de service et s'ils ne sont pas déconnectés. Dans le cas de la version serveur on vérifie que le serveur ne se bloque pas. On vérifie si le processus se termine bien. Dans le cas de la version serveur, on vérifie que la déconnexion de tous les participants ne provoque pas de blocage et si la reconnexion est toujours possible. De plus, on regarde si la fin du serveur provoque bien la fin de tous les clients.

3.3 Envoi de message

On envoi plusieurs messages avec différents clients et on regarde si ils s'affichent bien pour tout les participants à la discussion. On vérifie aussi si les messages sont bien préfixé du bon pseudo.

3.4 Taille des messages

On essaie d'envoyer un message dépassant la taille limite et on regarde si le message est bien couper et si le message suivant possède la fin du message précédent.

3.5 Nombre de participants

On vérifie si le serveur affiche le bon nombre de participants actifs.

4 CONCLUSION

Ce projet m'a permis de mettre en pratique des notions vus en TP/TD et donc de mieux les comprendre. La version serveur m'a posé plus de difficultés que la version tableau blanc. Une estimation du temps de travail serait : 8h sur la version serveur, 3h sur la version tableau blanc et 2h30 sur le rapport.