MINI RAPPORT
ET
HOW TO

Projet Réseaux 2016 : Création d'un RING
- De peer en peer -

LUTTRINGER
Jean-Romain
L3S5

Etat global du programme :

Le programme est terminé et possède toutes les commandes demandées dans l'énoncé. De plus, il n'y a plus de père initiateur, i.e un pair peut JOIN depuis n'importe quel pair déjà présent dans l'anneau, et le pair ayant crée le ring peut partir de l'anneau.

NOTE : Petit erreur lors de la création de la commande RING : elle s'appelle ici PING et se lance donc avec «:P » et non «:R »

Lancement :

Le programme peut se lancer de deux manières différentes.

- Si le pair veut créer un nouvel anneau, il faut taper, depuis le dossier principal :

./bin/peer [ring_port] [external_listening_port] INIT

ring port étant le port d'écoute au sein de l'anneau (pour écouter le pair précédent), et external listening port le port d'écoute vers l'extérieur de l'anneau (si un nouveau pair veut joindre .

-Si le pair veut joindre un pair d'un anneau déjà existant :

./bin/peer [ring_port] [external_port] [peer_add] [peer_port]

avec peer_add l'adresse du pair à joindre, et peer_port le port d'écoute du pair à joindre.

Exemple :

./bin/peer 5001 5100 INIT
./bin/peer 5002 5102 localhost 5100
./bin/peer 5003 5103 localhost 5102

Ici, un peer va créer un anneau, et écouter si quelqu'un veut rejoindre sur le port 5100.
Le deuxième peer va rejoindre l'anneau via le peer 1, et écouter d'éventuels nouveaux peers sur le port 5102
Le troisième peer va rejoindre le même anneau via le peer 2, en le joignant sur son port d'écoute.

TAPEZ MAKE POUR COMPILER LE FICHIER DEPUIS LE DOSSIER PRINCIPAL
PROGRAMME TESTE AVEC 0 ERREUR SUR VALGRIND
/ ! \ il n'y a pas de fonction de hash pour l'id, c'est un rand % 255. Il reste donc une chance minime que deux pairs récupère le même id, dans ce cas le programme ne fonctionnera pas correctement/!

Explications supplémentaires

Toute communication sur le Ring, à l'exception du transfert de fichiers, se fait via une structure « segment » , commentée et détaillée dans structure.h.

Un peer va donc lire un segment, identifier son type , et réagir en fonction.

Chaque peer possède plusieurs sockets (certaines initialisées que dans certains cas)

- une socket d'écoute vers l'extérieur, pour capter de nouveaux peers. Lors d'activité sur cette dernière, l'activité est *accept* sur une socket *service.*

- une socket next, permettant d'entrer en contact avec le pair suivant dans l'anneau.

-une socket eprev, captant une activité sur le port *précédent,* et étant accepté sur la socket *prev*

Toutes les commandes fonctionnent sur le même principe : si un pair reçoit une commandes qu'il a émise cela signifie qu'elle a bien été transférée à l'intégralité de l'anneau.

La commande méritant une brève explication est la commande GET. Comme ce programme a été conçu et testé en local, le moyen de simuler différents répertoires a été fait comme suit :

Quand un client a JOIN, un fichier est créée suivant le format suivant : ID_file. Ce fichier fait office de dossier visible par tous les pairs de l'anneau. Par conséquent, quand le pair 4 va recevoir une demande pour un fichier, il va vérifier dans son dossier (4_file) si il possède le fichier. Si il le possède, il va envoyer un segment NOTIFY au pair demandant le fichier, en passant par son socket d'écoute (afin de le contacter directement et sans passer par l'anneau pour le transfert, limitant ainsi le trafic). Le pair demandant 'X' va alors accepter la connexion et réceptionner le fichier demandé dans son propre fichier X_file.

NOTE : Le dossier crée est supprimé dès que le pair quitte l'anneau.

Le *select* appliqué uniquement à un file descriptor lors de la connexion au serveur, ou lors qu'un pair capte une activité sur sa socket d'écoute (ligne 75 et 330 dans peer.c) est un moyen trouvé pour résoudre le problème suivant :

Lorsqu'une connexion est captée sur la socket d'écoute, elle est acceptée sur service. Le pair attend ensuite la demande du nouveau pair. Or, si ce dernier ne rentre rien, le pair ayant accepté la connexion était bloqué sur ce read. Le select permet donc de faire un read avec un timeout de, ici, 8 secondes. Le client a donc 8 secondes pour entrer sa demande (qui ne pourra être que «:J ») , avant d'avoir été déclaré comme inactif et d'être déconnecté.

PRESENTATION RAPIDE DES SEGMENTS :

Segment PING :

L'id de l'émetteur est stocké dans la structure, et chaque pair rajoute son ID dans le champ « buffer » de la structure.

LANCE VIA LA COMMANDE < :P >

Segment BCAST :

Le champ « buffer » sert ici à stocker le message à broadcast. Lorsque le segment retourne à l'envoyeur, ce dernier affiche « I said [message] » , confirmant le fait que le message a bien fait le tour de l'anneau. **<:B>**

Segment JOIN :

Segment dont le type est réglé sur JOIN. Il permet de savoir si un pair ayant été contacté sur son socket d'écoute a reçu un NOTIFY ou un JOIN. **< :J>**

Segment EXIT :

Segment envoyé au pair suivant pour prévenir de son départ. Il permet de prévenir ce pair qu'il doit accepter une nouvelle connexion. **< :E>**

NOTE : Dans cet anneau, un pair n'est contacté par son successeur que si il doit se connecter à un nouveau pair.

Segment GET :

Le segment le plus complexe. Il contient dans le buffer le nom du fichier demandé. Les ID des pairs auxquels le fichier a été demandé sont stockés dans « Passed_by ». Lorsque le message est envoyé, le successeur détecte si il est le premier à le recevoir. Si il l'est, il rajoute les informations du pair ayant envoyé ce GET dans « emitter_info », afin les autres pairs sachent comment le contacter. Le port d'écoute a lui été déjà renseigné par le pair créateur de la demande. **<:G>**

Segment NOTIFY :

Ce segment est en réalité le segment GET ayant été reçu, avec simplement son type modifié. Ceci permet de conserver les informations importantes du segment get (en particulier le nom du fichier demandé, pour que le demandeur sache de quel fichiers il vient d'être « NOTIFY »). En somme un get fonctionne de cette manière.

→ J'envoie un segment GET contenant toutes les informations.

→ J'ai le fichier, je NOTIFY ce segment GET en l'envoyant au demandeur, en modifiant uniquement le type du segment.

EXEMPLE D'EXECUTION PAS A PAS :

Peer1> ./bin/peer 5000 5100 INIT // création du ring
Peer2> ./bin/peer 5001 5101 localhost 5100
Peer2>:J // join via le peer 1
Peer3>./bin/peer 5002 5102 localhost 5101
Peer3>:J //join via peer2
Peer4>./bin/peer 5003 5103 localhost 5100
Peer4>:J // join via peer1

Peer2> :P → 2,4,3,1
Peer1>:E
Peer3>:P ->3,2,4
// En admettant que le fichier Lorem soit présent chez Peer2

Peer4>:G Lorem
Beginning to receive file Lorem
File Transfered
File found via 3,1
//le fichier est maintenant chez Peer4

Peer4 >:E
Peer3 >:E
Peer2 >:P → {2}
…
etc