Détection de terminaison

Question 1

On doit insérer le code de détection de terminaison dans le code du serveur.

Question 2

Les cas d'utilisation de notre algorithme de terminaison est le suivant :

On souhaite empêcher nos clients d'envoyer des messages supplémentaires. Nous souhaitons ensuite que l'esemble des messages clients soient délivrés. Une fois qu'ils auront été tous délivrés, nous éteindront nos serveurs.

Donc pour permettre ceci, il nous faut :

1. Empêcher que les clients génèrent de nouveaux messages entre nos serveurs
2. Attendre la terminaison de notre application (tous les messages des clients sont arrivés à destination). Pour se faire, nous utiliserons une variante de l'algorithme de Safra.
3. Une fois le point précédent terminé, nous pourrons en toute sécurité éteindre nos serveurs.

La condition **« forall process p, statep = passif »** signifie donc, avec ce qui précède

aucun serveur n'accepte plus de messages de client; tous nos serveurs sont en attente de messages de client

Le client détecte juste que le serveur n'est plus disponible et s'arrête. Il n'est pas nécéssaire d'ajouter un protocole supplémentaire étant donné que couper la connection coté serveur entraine l'arrêt du client.

La ligne 17 de l'algorithme permet de savoir si un message a été reçu entre deux réceptions du jeton. On réinitialise à la fin de tout traitement du jeton la couleur à **white** et elle sera basculé à **black** en cas de réception d'un message *utilisateur*. Il suffit ensuite, quand on reçoit le jeton, de lire la couleur du serveur pour savoir si un message a été reçu depuis le dernier passage. Pour l'implémenter, il suffit d'avoir une variable correspondant à la couleur dans notre classe chargée de la détection de terminaison. Nous passons sa valeur à **white** après chaque passage du jeton, et la passons à **black** lors de toute réception de message utilisateur.

La variable **statep** permet de savoir si notre serveur est actif (si on travaille sur des requêtes utilisateur). Il faut le basculer en active quand on commence à traiter un message utilisateur, et le rebasculer à passive une fois que l'on a finit cette tache.

Question 3

La variable **mcp** correspond à la variable **messageCount** de ntre classe **EndManager**. Nous comptons dedans les messages reçus par nos clients, donc les join et leave notifications, et les messages privés et publiques transitant entre nos serveurs.

Mon implémentation est grandement simplifié, étant donné que ces messages sont les seuls à utiliser le broadcast manager causal. Ce sont donc les seuls du type InterServerMessage à être estampillé avec le type 5.

Question 4

Comme nous nous sommes fixé comme contrainte une topologie sous-jacente quelconque, nous ne pouvons faire d'hypothèses sur notre réseau. Nous n'avons donc aucune certitude de réussir à extraire un anneau de notre topologie. J'ai choisis de mettre en place un anneau virtuel en utilisant les propriétés suivantes :

- Chaque serveur a connaissance de la même liste ordonnée de l'ensemble des autres serveurs.
- Chaque serveur peut communiquer avec n'importe quel autre serveur par l'intermédiaire d'autre serveurs, par exemple en utilisant de la diffusion fiable, et en rajoutant un identifiant qui adresse ce message à l'autre serveur.

Nous pouvons ainsi recréer un anneau virtuel si à chaque serveur on adresse le message suivant au serveur suivant dans notre tableau d'identifiants serveurs ordonné. Il sera le seul à le recevoir via R Broadcast puis reconnaissance de son identifiant.

Cette technique permet de s'affranchir facilement de toute contrainte topologique mais génère une augmentation de la complexité en nombre de messages.

Question 5

Le processus **p0** est le processus responsable de la détection de terminaison. Nous choisirons **p0** comme étant le

processus élu (ce qui nous assure et son existance et son unicité).

Question 6

Compte tenu de notre scénario d'extinction, nous appelons la méthode **startEndingDetection** (qui correspond aux ligne 8 et 9 de l'algorithme). Elle est appelée quand l'administrateur système souhaite éteindre proprement son infrastructure. Elle a pour responsabilité de rendre les clients inactifs (le serveur n'accepte plus leurs messages), et d'envoyer le premier jeton.

Le jeton circule à une vitesse raisonnable (une attente de 10 ms est observée à chaque envoie de jeton afin de soulager les CPUs).

Question 7

Comme indiqué précédemment, on réinitialise à la fin de tout traitement du jeton la couleur à **white** et elle sera basculé à **black** en cas de réception d'un message *utilisateur*. Il suffit ensuite, quand on reçoit le jeton, de lire la couleur du serveur pour savoir si un message a été reçu depuis le dernier passage. Pour l'implémenter, il suffit d'avoir une variable correspondant à la couleur dans notre classe chargée de la détection de terminaison. Nous passons sa valeur à **white** après chaque passage du jeton, et la passons à **black** lors de toute réception de message utilisateur.

Question 8

Avec ce qui précède, l'action **Sp** est à appeler une fois par nombre de C broadcast propagé (que ce soit au sein du RBroadcast manager ou au sein de l'envoi).

Question 9

L'action **Rp** est à déclencher à chaqe réception de message relié à du C Broadcast.

Question 10

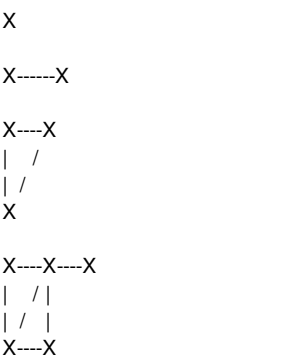
L'action **Ip** doit se déclencher dès que le traitement d'un message C Broadcasté est finit.

Question 11

L'action **Tp** doit être réalisé dès que le jeton est reconnu et accepté par le serveur.

Question 12

J'ai testé mon implémentation sur les structures suivantes :



Question 13

J'ai mis en place un message RBroadcasté qui déclenche l'arrêt de tous nos serveurs.