Exclusion mutuelle de Ricard et Agrawala

Question 1

Ce sont nos serveurs qui accèdent à une ressource de manière concurente. Ce sont donc eux qui doivent implémenter un mécanisme les protégeant des accès concurents. Les clients n'ont pas ce genre de problématiques, nous n'avons donc pas besoin de mettre en place ce genre de solutions pour eux.

Question 2

L'algorithme est écrit en suivant la convention orientée évenements.

Question 3

Les serveurs échangent des messages de type InterServerMessage, RBroadcasté (donc le champ type est à 3), soit transportant une demande d'entrée en session critique (subtype = 8, avec nsp comme champ message, et l'identité du processus dans le champs identifier), soit un transfert de jeton (subtype=9, le jeton est dans le champ message, et le processus a qui est destiné le jeton est situé dans le champs NeededData).

Question 4

La demande de jeton se fait à l'aide de la méthode **askLock** du LockManager. Nous pouvons la lancer depuis le clavier, une commande est prévue à cet effet.

Question 5

La topologie du réseau sous tendue par l'algorithme de Ricart et Agrawala est un graphe fortement connexe (chaque serveur est relié à chaque autre serveur). Néanmoins,nous pouvons nous passer de cette hypothèse si chaque message est broadcasté (quitte à identifier le serveur de destination sur un message destiné à un seul serveur : le serveur en question saura se reconnaitre et les autres réaliserons qu'il ne leur est pas dédié). Avec cette modification, nous n'avons plus de contraintes topologiques. C'est cette solution là que je vais mettre en place.

Question 6

Liste des variables de l'algorithme :

- **nsp** : L'estamplille de nos demandes de jeton.
- **dem** : Le compte pour chaque serveur des demandes reçues par chaque serveur.
- **jet** : Le compte des demandes satisfaites par le jeton. Ce n'est pas à proprement parlé une variable propre à un serveur, mais une valeur qui transite avec le jeton.

Question 7

Le jeton est donné au processus élu, qui n'est pas en session critique. Le fait d'utiliser le serveur élu nous garanti l'unicité du jeton. Par ailleurs, nous nous assurons qu'il y a bien un et un seul serveur élu. Il semble donc logique de se placer dans cette optique.

Question 8

Le jeton est un tableau qui, à chaque processus associe le nombre de fois ou ce processus a accédé cette ressource. Au début de l'éxécution, les valeurs associées à chaque processus sont initialisées à 0.

Question 9

A la réception du jeton, nous sommes les seuls à pouvoir accéder la ressource distante. Nous pouvons donc commencer à l'utiliser.

Question 10

Nos serveurs, compte tenu de nos exigeances, connaissent déjà l'ensemble des serveurs situés sur leur réseau. Ils capable de les identifier, et donc d'échanger des messages via RBroadcast sans ajout d'un nouveau sous protocole de notre part.

Question 11

L'ensemble des demandes est une table de hashage qui associe à un identifiant serveur le nombre de demandes qu'il a fait pour ce verrou.

Question 12

Quand il reçoit une demande, le processus ajoute une demande au processus qui l'a fait. Ensuite, si il n'est pas en session critique mais possède le jeton, il le redistribue en utilisant la méthode de sortie de session critique.

Question 13

Pour pouvoir tester l'exclusion mutuelle, nous pouvons effectuer des demandes de verrou de la part de plusieurs processus (au moins 3) et des relachement de verrou. Il faut tester les cas de figures suivants :

- Un serveur demande le verrou et personne n'est en session critique
- Un serveur demande le verrou alors qu'un autre serveur est en session critique. On vérifie qu'il obtient bien le jeton à la sortie de la session critique du premier processus.
- Deux serveurs demandent le verrou alors qu'un autre utilise les ressources. On vérifie bien que chacun libère la ressource et transmet le jeton au suivant.
- Il nous faut nous assurer aussi de l'hypothèse de vivacité. Pour se faire, on peut réaliser une séquence de ce style :
    - Lock 1
    - Lock 2
    - Lock 3
    - Release 1
    - Lock 1
    - Ici disons que 2 obtienne le jeton
    - Release 2
    - Ici 3 doit obtenir le jeton. Si ce n'est pas le cas alors nous aurons un problème avec la propriété de vivacité.

Notes complémentaires

Architecture permettant de personnaliser le comportement du LockManager

J'ai implémenté le patron de conception du visiteur (voir les classes ResourceVisitor et LogResourceVisitor) afin de pouvoir personaliser le comportement de notre lock manager. Maintenant, l'utilisateur de cette classe peut spécifier le comportement de cette classe sans avoir à l'hériter.

Ce patron de conception aurait pu être utilisé pour personaliser l'EchoManager. Je n'y ai pensé qu'après, mais cela s'y preterai bien...

Système de log

J'ai ajouté un système de gestion des logs. Je me suis appuyé sur **log4j** (http://logging.apache.org/log4j/1.2/).

Changement de topologie

Tous les verrous sont levés en cas de changement de topologie. Il s'agit d'une contrainte assez forte. Une autre stratégie plus maline consisterais à vérifier que nous avons bien un (et un seul) jeton dans notre réseau. Si nous n'en avons aucun, il suffit au serveur élu d'en regénérer un. Si il y en a plusieur, le serveur élu peut demander la destruction de certain jetons afin de ramener le compte à un. Je n'ai pas eu le temps d'implémenter ce mécanisme.