↓ Exercices

Exercice 1

Proposer une implémentation de l'exclusion mutuelle utilisant l'abstraction de consensus vue dans ce cours (sous forme de boite noire).

Garantissez notamment

- que toute demande sera un jour satisfaite, *même si un autre processus demande la SC sans cesse* ;
- qu'aucune entrée en SC ne sera autorisée avant la fin de celle en cours.

Exercice 2

Nous souhaitons avoir un algorithme de consensus permettant de s'accorder non plus sur une proposition, mais sur **un ordonnancement de propositions**, contenant toutes les propositions des processus valides, et autorisant les propositions de processus en panne.

Proposer une première version supposant un réseau synchrone.

Proposer ensuite une version ne faisant pas de supposition sur le réseau, utilisant (sous forme de boite noire) un algorithme de consensus tolérant un réseau partiellement synchrone.

Comment un tel algorithme pourrait-il être utilisé dans votre projet ChatsApp, afin d'offrir la garantie d'ordre global de réception des messages ?

Solution 1

Des consensus sont lancés en boucle.

À chaque instance de consensus, je partage ma demande si j'en ai une, ainsi que le TS associé.

La fonction de décision déterministe sélectionne la proposition dont le TS est le plus ancien.

Aussi, un processus en cours de SC doit attendre la fin de sa SC avant de faire sa proposition au prochain consensus. Sinon, il serait possible d'avoir plusieurs processus en SC en même temps. Une autre solution serait de continuer de proposer sa demande avec son TS tant qu'on n'est pas sorti de SC, pour que le consensus continue de nous l'accorder.

Solution 2

La solution simple est de modifier le consensus vu en cours pour qu'il retourne la liste des propositions au moment de la décision, triée de manière arbitraire mais déterministe (e.g. par processus croissant), sans en sélectionner une arbitrairement.

En utilisant un consensus comme boite noire, on doit

- broadcast notre proposition avant d'entrer dans le consensus,
- créer une liste des propositions reçues, avec un ordre arbitraire,
- proposer cette liste au consensus.

Si notre proposition n'apparait pas dans le résultat du consensus, c'est que notre message n'avait pas eu le temps d'arriver ; on relance donc un consensus avec une liste de toutes les propositions reçues jusqu'ici, la nôtre inclue. Quand on est d'accord, on en informe tout le monde. Quand tous les voisins corrects sont d'accord, on retourne la liste.

Avec un tel algorithme, on pourrait ne plus utiliser de mutex. À la place, on aurait des consensus en boucle. À chaque nouvelle instance de consensus, on lui passe les messages qu'on aimerait broadcast, puis on attend le résultat du consensus, qui sera les messages à afficher ainsi que leur ordre. Si nos messages n'y apparaissent pas tous, on les proposera à nouveau au prochain consensus.