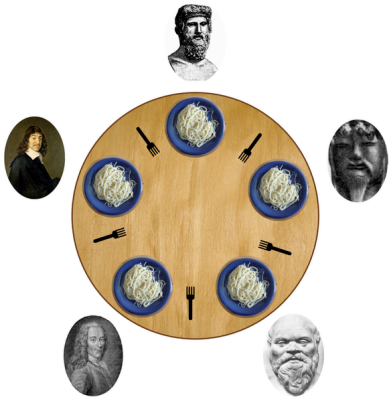
TME 2 - Implantation sous MPI d'horloges de Lamport

Exercice(s)

Exercice 1 – Dîner de philosophes

Le problème du dîner de philosophes est un problème classique dans le domaine des systèmes répartis, énoncé par E. Dijkstra. Ce problème est posé de la manière suivante :

- N philosophes (a priori asiatiques car ils consomment des nouilles avec des baguettes) se trouvent autour d'une table ronde ;
- chacun des philosophes a devant lui un bol de nouilles ;
- entre chaque bol se trouve exactement une baguette.



Un philosophe n'a que trois états possibles :

- `THINKING` : penser pendant un temps indéterminé ;
- `HUNGRY` : être affamé (pendant un temps déterminé et fini sinon il y a famine) ;
- `EATING` : manger pendant un temps déterminé et fini.

Des contraintes extérieures s'imposent à cette situation :

- pour manger, un philosophe a besoin de deux baguettes : celle qui se trouve à gauche de sa propre assiette, et celle qui se trouve à gauche de celle de son voisin de droite (c'est-à-dire les deux baguettes qui entourent sa propre assiette) ;
- quand un philosophe a faim, il passe dans l'état `HUNGRY` et il y reste jusqu'à ce qu'il ait obtenu les deux baguettes ;
- quand il termine de manger, il repose les deux baguettes.

Nous nous intéressons dans un premier temps au mécanisme permettant à un philosophe de s'approprier une baguette. La baguette est un élément passif, elle ne peut donc pas émettre ou recevoir des messages.

Question 1
Quel échange de message permet à un philosophe de prendre une baguette ? Quelles sont les variables locales à gérer ? Que se passe-t-il si deux voisins font leur demande en même temps ?

Question 2
Que se passe-t-il si tous les philosophes font leur demande en même temps ?

Une solution est d'instaurer des priorités croissantes entre philosophes. Pour ce faire, on fournit à chaque philosophe un identifiant entier unique. Tout philosophe est prioritaire par rapport à son voisin si ce dernier possède un identifiant de valeur supérieure. Lorsqu'il n'est pas en train de manger, un processus doit céder la baguette qu'il possède si elle est demandée par un processus plus prioritaire que lui.

Comme les identifiants sont uniques, on est certain qu'un philosophe (celui dont l'identifiant est le plus petit) pourra toujours obtenir ses baguettes et qu'il n'y aura donc pas d'interblocage

Question 3
Pour l'implémentation, nous décidons que chaque philosophe souhaite manger `NB_MEALS` repas.

À quel moment le processus représentant un philosophe doit-il se terminer ?

Question 4
Proposez un algorithme décrivant le comportement d'un philosophe dans la solution proposée ci-dessus. Prenez soin de détailler les opérations effectuées (1) à l'initialisation (ie. avant chaque repas), et (2) à la réception de chaque type de message.

Question 5
Ecrivez un programme MPI qui implémente la solution proposée pour un nombre `NB` (fixé à l'exécution) de processus philosophe. Chaque processus consommera `NB_MEALS` repas.

Question 6
Ajoutez un mécanisme d'horloges logiques à votre système et affichez, pour chaque philosophe, la date de consommation de chacun de ses repas. Le but est de pouvoir recomposer une vision globale de l'exécution selon laquelle les dates des repas sont cohérentes avec les échanges de messages.