Modélisation des applications réparties

Algorithme d'exclusion mutuelle de L. LAMPORT

J. Christian Attiogbé

Janvier 2017, maj 2022

IUT Nantes
Pôle Sciences et technologie

Plan du cours

1. Algorithme d'exclusion mutuelle

2. Exercice : programmation de l'algorithme d'EM

Algorithme d'exclusion mutuelle

Mécanisme fondamental : exclusion mutuelle
Imaginons une section critique, convoitée par n processus
donc *race condition*

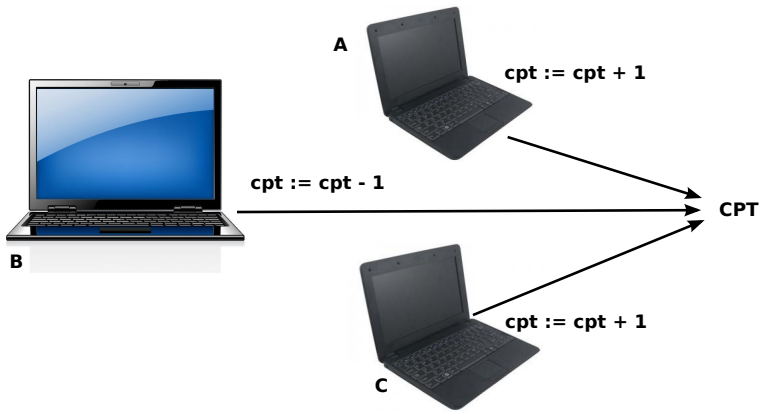

Figure: Concurrence d'accès

La modification de la variable doit se faire en section critique

Algorithme d'exclusion mutuelle

Chaque processus peut :

- demander à entrer en SC
- attendre la/les réponses à sa demande de SC
- recevoir une demande d'entrée en SC
- répondre à une demande
- entrer en SC
- sortir de la SC

Mais les processus sont distants, séparés par le réseaux, simultanés, ...

L'algo assure la synchronisation dans les systèmes répartis

Exclusion mutuelle : un algorithme réparti (distribué)

Lamport 1978, Ricart et Agrawal 1981
L'algorithme demande l'existence d'un ordre total sur tous les événements du système.
Quand un processus veut entrer en section critique :

- il construit un message contenant :
  - le nom de la section critique dans laquelle il veut entrer,
  - son numéro de processus et
  - l'heure courante (estampille).
- Il envoie ce message à tous les autres processus + lui même.

En supposant que l'émission des messages est fiable, chaque message est acquitté.

Exclusion mutuelle : un algorithme réparti

Quand un processus reçoit un message *demande d'entrée en SC* :

- il prend une décision qui dépend de son état et de la section critique sollicitée par le message.
  1. si le récepteur n'est pas dans la section critique et ne veut pas entrer dans celle-ci, il retourne simplement un message OK à l'emetteur ;
  2. si le récepteur est déjà dans la section critique, il ne répond pas, il mémorise la demande dans une file d'attente ;
  3. si le récepteur veut entrer en section critique, mais ne l'a pas encore fait, il compare l'estampille du message venant d'arriver à celle contenue dans le message qu'il a envoyé aux autres. La plus ancienne gagne. Si le message est plus vieux, le récepteur répond par un message OK. Si c'est l'estampille de son message qui est la plus ancienne, le récepteur mémorise la demande et n'envoie rien.

Exclusion mutuelle : un algorithme réparti (distribué)

Après avoir envoyé ses demandes d'entrée en section critique, un processus (demandeur) se met en état d'attente jusqu'à ce que tous les autres lui donnent leur autorisation.

Dès que toutes les autorisations sont accordées, il peut entrer en section critique.

Quand il sort, il envoie un message OK à tous les processus présents dans la file d'attente et les supprime tous de la file.

Dans les conditions normales, cet algorithme fonctionne correctement.

Que se passe-t-il quand une machine est en panne ?

Exclusion mutuelle : un algorithme réparti (distribué)

L'algorithme est réparti ; toutes les machines y sont impliquées.
Quand une machine est bloquée, tout le reste est bloqué.

Le problème de l'algorithme

S'il y a n ordinateurs, il y a n points faibles (contrairement à un algorithme centralisé ou il y aurait 1 point faible).

Solution

- Quand un message arrive, il doit être acquitté : accord ou refus.
- L'émetteur utilise un temporisateur et arme un délai quand il envoie une demande vers les autres machines.
- Si au bout du délai de temporisation il n'a pas obtenu d'acquittement, le message est déclaré perdu et l'algorithme peut fonctionner sans la machine concernée.

Conclusion sur l'exclusion mutuelle

- On sait écrire des algorithmes répartis
- Ils sont plus difficiles à élaborer et à prouver
- Ils sont mis en œuvre dans les systèmes

Assistance au développement

- Décharger le programmeur des tâches de bas niveau, pour écrire les applications réparties
- Proposer des mécanismes et outils de plus haut niveau pour interagir avec l'environnement : les *middleware*.

Exercice : programmation en GO

Programmons l'algorithme décrit avec les hypothèses suivantes :

- nous allons exécuter en parallèle sur un processeur, n processus,
- les n processus ont accès à une variable en mémoire, qui est donc critique,
- nous voulons observer les traces du déroulement des processus et observer quels processus accèdent à/libèrent la variable.

En vous inspirant des TD/TP déjà effectués en GO, proposer une programmation simple

Programmation en Go : exemple de scnénario

Dans une application `myApp` on déclare **une variable globale `globVar`** et on déclare **4 fonctions représentant 4 processus distincts**.

- Chaque processus a une section critique dans laquelle elle accède à la variable `globVar`.
- Lorsqu'on exécute l'application `myApp`, elle lance l'exécution en parallèle des 4 processus, **chacun étant identifié par un numéro**.
- Chaque processus tente d'**entrer dans sa section critique en faisant la partie appropriée** de l'algorithme étudié.
- S'il obtient l'accord des autres processus, alors il imprime son numéro puis effectue le reste de l'algorithme : **libération de la section critique**.

Programmation en GO : quelques indications

- chaque processus connait tous les autres, leurs horloges logiques, et les messages reçus de chacun,
- lorsqu'un processus n'utilise pas la section critique, il scrute de temps en temps sa boite au messages et répond aux messages reçus ;
- chaque processus devra envoyer un message aux autres processus, quelle est la structure du message ?
- chaque processus doit analyser les messages reçus (venant des autres processus) ; quelles structures de communications sont adaptées (canaux bufferisés/non bufferisés) ?
- quelles structures de données sont nécessaires pour chaque processus ?
- effectuer une programmation incrémentale.

Références

- Synchronisation et état global dans les systèmes répartis, Michel Raynal, Eyrolles, 1992