Problèmes et stratégies de cohérence et de synchronisation

Motivation : Émergence du calcul distribués, machines parallèles → nouveaux problèmes

Préliminaires

- Un **système** est un ensemble de processeurs $p_0, \dots, p_{n-1}$ et de registres $R_0, \dots, R_{m-1}$.
- Un **processeur** est une machine à états
- Un **registre** (ou objet) est la donnée d'un domaine et de primitives.

  Une primitive est définie par sa spécification ~~séquentielle~~.
  (NB : on ne considère que des primitives atomiques)

  → Exemples : primitives **read** ou write

Définitions : Une **configuration** est un $n+m$-uplet ~~ou~~

$(q_0, \dots, q_{n-1}, r_0, \dots, r_{m-1})$ où q_i est l'état de p_i

Un **événement** est une étape de calcul d'un processeur p_i, c.-à-d. la suite suite d'actions suivantes (atomique) :

1. p_i choisit un registre et appelle une primitive
2. L'opération est exécutée
3. L'état de p_i change.

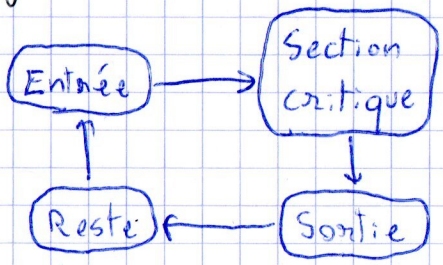
Le choix en (1) est uniquement déterminé par l'état de p_i

Une **exécution** est une suite $C_0, \phi_1, C_1, \phi_2, C_2, \dots$ où C_i est une configuration, $\phi_i \in \Pi 0; n-1 \Pi$ (processeur), et C_{i+1} correspond au nouvel état après l'événement du processeur p_{ϕ_i} dans l'état $C_i(\phi_i)$.

I - Section critique, exclusion mutuelle

Motivation : besoin de réserver une variable pour opérations du genre `i = i+1;` ou `if (x!=0) return 1/x;`

Définition : Étant donné un système dont chaque processus peut voir ses états partionnés comme ci-contre, on définit :



(1) **Exclusion mutuelle** : pour toute exécution issue d'une configuration initiale, pour toute configuration de cette exécution, au plus 1 processeur en SC.

(2) **Pas de blocage** : pour toute configuration d'une exécution admissible issue d'une configuration initiale dans laquelle un processeur est dans l'entrée, il y a une configuration future avec un processeur en SC.

(3) **Pas de ~~famine~~ recalage** : idem à (2) en imposant que ce soit le même processeur.

A. Exclusion mutuelle avec primitive forte

Définition : test&set est une primitive de registre binaire R définie comme suit :

- Si R contient 1, R.test&set() renvoie 1
- Si R contient 0, R.test&set() renvoie 0 et stocke 1 dans R.

Algorithme :

n processeurs $p_0, \dots, p_{n-1}$

1 registre R avec test&set, read et write

Entrée de p_i :

TantQue (R.test&set() != 0)

Attendre

Sortie de p_i :

R.write(0)

Proposition : Cet algorithme implémente une procédure d'exclusion mutuelle sans blocage et sans ~~famine~~ inégalité

B. Avec primitives read et write

Algorithme : Verrou de Peterson

2 processeurs p_0, p_1

3 registres : flag[2] (binaire), victime (binaire)

Entrée de p_i :

flag[i].write(1)

victime.write(i)

TantQue (flag[1-i].read() = 1 et victime.read() = i)

Attendre

Sortie de p_i :

flag[i].write(0)

Théorème : le verrou de Peterson satisfait à (1) et (3) (donc à (2))

Note : On peut généraliser l'algorithme de Peterson à n > 2 processus

Algorithme de la boulangerie (Lamport)

Théorème : l'algorithme de la boulangerie satisfait ~~à (1) et (2). De plus pour toute exécution admissible, il satisfait à (3)~~ à (1), (2) et (3)

DEV 1

Définition : Une exécution $C_0 \phi_1 C_1 \phi_2 \dots$ est admissible ssi pour tout $k \in \mathbb{N}$, $0 \leq i < n$ tel que p_i n'est pas dans son état final à C_k, il existe $\ell > k$ tel que $\phi_\ell = i$

Note : pour les algorithmes ci-dessus (exclusion mutuelle en général) il faut l'admissibilité pour que (3) soit vraie : pas de tolérance aux pannes.

II - Hiérarchie des primitives de synchronisation

Définition : Une fonction propose : $\alpha \to \alpha$ est un protocole de consensus si, pour chaque exécution dans laquelle propose est appelée au plus une fois par chaque processeur

- Validité : propose renvoie une des valeurs proposée par un processus
- Cohérence : propose renvoie la même valeur à tous les processus
- Sans attente : propose renvoie une valeur en temps fini à chaque processus

Définition : Le numéro de consensus d'une classe C de registres est le plus grand entier $n \in \mathbb{N} \cup \{+\infty\}$ tel que on peut implanter un protocole de consensus pour n processeurs avec uniquement des objets de C, et des registres atomiques

Proposition

- La classe des registres atomiques (entiers, read, write) a un numéro de consensus de 1
- La classe des files (entiers, pop, push) a un numéro de consensus de 2

Définition ~~Un registre RMW est est RMW s'il admet une primitive call et un ensemble de fonctions $\mathcal{F}$ tel~~

Une primitive p est RMW pour un ensemble de fonctions $\mathcal{F}$ si $R.p(a_0, \dots, a_{k-1})$ renvoie la valeur v de R et stocke dans R la valeur $f(v)$ où $f \in \mathcal{F}$ ne dépend que des arguments

Exemples :

- read est RMW ($\mathcal{F} = \{Id\}$)
- test&set est RMW ($\mathcal{F} = \{x \mapsto 1\}$)

Définition Une primitive RMW est dans Common2 si, pour tous $f, g \in \mathcal{F}$, pour toute valeur v,

$f(g(v)) = g(f(v))$ ou $f(g(v)) = f(v)$ ou $g(f(v)) = g(v)$

Théorème Tout registre dont les primitives sont RMW dans Common2 et dont l'ensemble $\mathcal{F}$ n'est pas réduit à Id a un numéro de consensus de 2

Preuve DEV 2

Ouverture

- Tolérance aux pannes plus fine (plantage)
- Universalité du consensus
- Modèle réseau