Algorithmes du cours

Algorithmique distribuée - Polytech Grenoble INFO4

Algorithme 1 Exclusion mutuelle de Le Lann

Variables

$Initiateur \in \{Vrai, Faux\}$ (vrai pour un unique processus)
$Etat \in \{Req, IN, OUT\}$

Algorithmepour un processus quelconque

Si $Initiateur$ **alors**
    `Envoyer` $\langle Jeton \rangle$ `à` D
Fin Si
Pour toujours
    `Réceptionner` $\langle Jeton \rangle$ `de` G
    Si $Etat = Req$ **alors**
        $Etat \leftarrow IN$
        `CS` ▷ Section critique
        $Etat \leftarrow OUT$
    Fin Si
    `Envoyer` $\langle Jeton \rangle$ `à` D
Fin Pour toujours

Algorithme 2 Élection de Le Lann pour un processus p

Variables

$Id_p \in \mathbb{Z}$ (identifiant unique du processus)
$Leader \in \{Vrai, Faux\}$ (représente le résultat de l'élection : le processus est-il le leader ?)
$List \in \mathbb{Z}^*$ (liste des identifiants collectés)

Algorithme pour un processus

$List \leftarrow \{Id_p\}$
`Envoyer` $\langle Election, Id_p \rangle$ `à` D
Pour toujours
    `Réceptionner` $\langle Election, Id \rangle$ `de` G
    Si $Id_p \neq Id$ **alors**
        $List \leftarrow List \cup \{Id\}$
        `Envoyer` $\langle Election, Id \rangle$ `à` D
    Sinon
        $Leader \leftarrow Id_p = \min(List)$
    Fin Si
Fin Pour toujours

Algorithme 3 Élection de Chang et Roberts pour un processus p

Variables

$Id_p \in \mathbb{Z}$ (identifiant unique du processus)
$Leader \in \{Vrai, Faux\}$ (représente le résultat de l'élection : le processus est-il le leader ?)

Algorithme pour un processus initiateur

`Envoyer` $\langle Election, Id_p \rangle$ `à` D
Pour toujours
  `Réceptionner` $\langle Election, id \rangle$ `de` G
  Si $Id_p = id$ **alors**
    $Leader \leftarrow Vrai$
  Sinon Si $id < Id_p$
    $Leader \leftarrow Faux$
    `Envoyer` $\langle Election, id \rangle$ `à` D
  Fin Si
Fin Pour toujours

Algorithme pour un processus non-initiateur

$Leader \leftarrow Faux$
Pour toujours
  `Réceptionner` $\langle Election, Id \rangle$ `de` G
  `Envoyer` $\langle Election, Id \rangle$ `à` D
Fin Pour toujours

Algorithme 4 Élection de Itai et Rodeh pour un processus p (version simplifiée de Fokkink et Pang, qui suppose les canaux FIFO)

Variables

$n \in \mathbb{N}$ (nombre exact de noeuds dans le réseau)
$Id_p \in \mathbb{Z}$ (identifiant tiré aléatoirement du processus)
$Etat_p \in \{Actif, Passif, Leader\}$ (représente le résultat de l'élection : le processus est il (encore) candidat à l'élection ? Le processus est-il le leader ?)
`Les messages sont de la forme` $\langle Election, id, saut, unique \rangle$

Algorithme pour un processus initiateur

$Id_p \leftarrow$ tirage aléatoire de 1 à n
$Etat_p \leftarrow Actif$
`Envoyer` $\langle Election, Id_p, 1, Vrai \rangle$ `à` D
Pour toujours
    `Réceptionner` $\langle Election, id, saut, unique \rangle$ `de` G
    Si $Etat_p = Passif$ **alors**
        `Envoyer` $\langle Election, id, saut + 1, unique \rangle$ `à` D
    Sinon Si $saut = n \wedge unique$ **alors**
        $Etat_p \leftarrow Leader$
    Sinon Si $saut = n \wedge \neg unique$ **alors**
        $Id_p \leftarrow$ tirage aléatoire de 1 à n
        `Envoyer` $\langle Election, Id_p, 1, Vrai \rangle$ `à` D
    Sinon Si $Id_p = id$ **alors**
        `Envoyer` $\langle Election, id, saut + 1, Faux \rangle$ `à` D
    Sinon Si $id < Id_p$ **alors**
        $Etat_p \leftarrow Passif$
        `Envoyer` $\langle Election, id, saut + 1, unique \rangle$ `à` D
    Sinon($id > Id_p$)
        // ne rien faire (message non propagé)
    Fin Si
Fin Pour toujours

Algorithme pour un processus non-initiateur

$Etat_p \leftarrow Passif$
Pour toujours
    `Réceptionner` $\langle Election, id, saut, unique \rangle$ `de` G
    `Envoyer` $\langle Election, id, saut + 1, unique \rangle$ `à` D
Fin Pour toujours

Algorithme 5 Exclusion mutuelle de Lamport

Variables

H tableau des horloges locales connues
Req tableau des dates de demandes non servies ($\perp$ si pas de demande)
Les tableaux sont indicés sur l'ensemble des processus.

Fonction $min_{<<}(Req)$ (minimum des valeurs de Req, selon $<<$)

return (x, T) ssi $\forall q \neq p, Req(q) \neq \perp \implies (x, T) << (q, Req(q))$

Fonction Accès() (tentative d'accès à la section critique)

Si $Req(p) \neq \perp$ **alors**
    Si $(p, T) = min_{<<}(Req)$ et $\forall q \neq p, H(q) > T$ **alors**
        `CS`
        $Req(p) \leftarrow \perp$
        `Envoyer` $\langle Sortie, p, H(p) \rangle$ `à` tous les autres processus
    Fin Si
Fin Si

Fonction Demande() (exécutée par l'application)

Si $Req(p) = \perp$ **alors** $Req(p) \leftarrow H(p)$ **Fin Si**
`Envoyer` $\langle Demande, p, H(p) \rangle$ `à` tous les autres processus
Accès()
$H(p) \leftarrow H(p) + 1$

Algorithme pour un processus p

Pour tout processus q **faire** $H(q) \leftarrow 0; Req(q) \leftarrow \perp$ **Fin Pour**
Pour toujours
    `Réceptionner` $\langle M, q, T \rangle$ `de` C_q
    $H(p) \leftarrow \max(H(p), T + 1)$
    $H(q) \leftarrow T$
    Si $M = Demande$ **alors**
        Si $Req(q) = \perp$ **alors** $Req(q) \leftarrow H(q)$ **Fin Si**
        `Envoyer` $\langle Ack, p, H(p) \rangle$ `à` C_q
    Sinon Si $M = Sortie$ **alors**
        $Req(q) \leftarrow \perp$
    Fin Si
    Accès()
    $H(p) \leftarrow H(p) + 1$
Fin Pour toujours

Algorithme 6 Circulation de jeton dans un arbre pour tout processus p

Algorithme pour le processus initiateur $init$

Pour $i \leftarrow 1$ à δ_{init} **faire**
  Envoyer $\langle Jeton \rangle$ à i
  Réceptionner $\langle Jeton \rangle$ de i
Fin Pour
décide

Algorithme pour tout autre processus p

Pour $i \leftarrow 1$ à δ_p **faire**
  Réceptionner $\langle Jeton \rangle$ de q
  Envoyer $\langle Jeton \rangle$ à $(q \bmod \delta_p) + 1$
Fin Pour

Algorithme 7 Propagation de la donnée d avec retour, pour tout processus p

Variables

Cpt : compteur du nombre de messages reçus
$Parent$: pointeur vers le canal du noeud parent

Algorithme pour le processus initiateur $init$

$Parent \leftarrow \perp$
Envoyer $\langle Brd{,}d \rangle$ à tous les voisins
Pour $Cpt \leftarrow 1$ à δ_{init} **faire**
  Réceptionner $\langle msg \rangle$ de q
Fin Pour
décide

Algorithme pour tout autre processus p

Réceptionner $\langle Brd, d \rangle$ de q
$Parent \leftarrow q$
Envoyer $\langle Brd{,}d \rangle$ à tous les voisins sauf q
Pour $Cpt \leftarrow 2$ à δ_p **faire**
  Réceptionner $\langle msg \rangle$ de q
Fin Pour
Envoyer $\langle Ack \rangle$ à $Parent$

Algorithme 8 Algorithme de Shandy Lamport - Snapshot

Modification d'un algorithme $\mathcal{A}$ pour qu'il effectue une sauvegarde de son état global.

On suppose que l'algorithme $\mathcal{A}$ pour le processus p est de la forme :

<init>
Pour toujours
    `Réceptionner` $\langle M \rangle$ `de` q
    ...
Fin Pour toujours

Variables supplémentaires pour $\mathcal{A}$:

save : tableau de booléens indicés sur p et ses voisins.
Cqp : tableau indicé sur les voisins de p, $Cqp[q]$ contient le contenu du canal de q vers p, une fois la sauvegarde faite.

En plus de l'initialisation <init> de $\mathcal{A}$:

$save[p] \leftarrow faux$
Pour tous les voisins q de p, $save[q] \leftarrow faux$; $Cqp[q] \leftarrow \emptyset$

Sur demande de sauvegarde :

`enregistrer` S_p
$save[p] \leftarrow vrai$
`Envoyer` $\langle SAVE \rangle$ `à` tous les voisins de p

Dans la boucle "Pour Toujours" de l'algorithme $\mathcal{A}$:

Si $M = SAVE$ **alors**
    enregistrer $Cqp[q]$; $save[q] \leftarrow vrai$
    Si non $save[p]$ **alors**
        `enregistrer` S_p
        $save[p] \leftarrow vrai$
        `Envoyer` $\langle SAVE \rangle$ `à` tous les voisins de p
    Fin Si
    Si $save[p]$ et pour tout q, $save[q]$ **alors** `enregistrement fini` **FinSi**
Fin Si

Dans la boucle "Pour Toujours" de l'algorithme $\mathcal{A}$, ajout pour tous les autres types de messages :

Si $M \neq SAVE$ et $save[p]$ et non $save[q]$ **alors**
    $Cqp[q] \leftarrow M + +Cqp[q]$
Fin Si