Protocoles simples pour l'implémentation répartie des sémaphores

Michel Raynal

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

1 9 9 2

25*ème anniversaire*

N° 1604

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes distribués

PROTOCOLES SIMPLES POUR L'IMPLEMENTATION REPARTIE DES SEMAPHORES

Michel RAYNAL

Février 1992

RR-1604

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX FRANCE
Tél. : 99 84 71 00 - Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Protocoles simples pour l'implémentation répartie des sémaphores

Michel RAYNAL
IRISA
Campus de Beaulieu
35042 Rennes-Cédex
FRANCE
raynal@irisa.fr

Programme 1, Projet ADP
Publication Interne n° 631 - Janvier 1992 - 14 pages

Résumé

L'utilisation croissante des machines parallèles à mémoire répartie constitue un terrain de prédilection relativement nouveau pour les concepteurs de systèmes répartis, le paradigme classique "réseau (local) + messages" y étant souvent remplacé par celui de la mémoire virtuelle répartie. Dans cette nouvelle problématique (qui consiste en fait à implémenter sur ces nouvelles architectures des concepts fondamentaux des systèmes centralisés) on examine ici la mise en oeuvre répartie du mécanisme de base pour la synchronisation qu'est le sémaphore. Plusieurs protocoles sont proposés ; ils sont tous construits à partir de l'invariant fondamental associé au sémaphore. Ces protocoles, joints à un mécanisme de mémoire virtuelle répartie, peuvent ainsi permettre une programmation classique de ce type de machines parallèles.

Simple protocols to implement distributed semaphores

Abstract

Thanks to their structure and regularity, distributed memory parallel machines allow to implement well-known paradigms such as the shared memory one. We examine in this paper the distributed implementation of a basic synchronization tool, namely the semaphore concept. When they are joined to a distributed shared memory implementation these protocols achieve portability of classical parallel centralized programs.

Protocoles simples pour l'implémentation répartie des sémaphores

Michel RAYNAL
IRISA
Campus de Beaulieu
35042 Rennes-Cédex
FRANCE
raynal@irisa.fr

Résumé

L'utilisation croissante des machines parallèles à mémoire répartie constitue un terrain de prédilection relativement nouveau pour les concepteurs de systèmes répartis, le paradigme classique "réseau (local) + messages" y étant souvent remplacé par celui de la mémoire virtuelle répartie. Dans cette nouvelle problématique (qui consiste en fait à implémenter sur ces nouvelles architectures des concepts fondamentaux des systèmes centralisés) on examine ici la mise en oeuvre répartie du mécanisme de base pour la synchronisation qu'est le sémaphore. Plusieurs protocoles sont proposés ; ils sont tous construits à partir de l'invariant fondamental associé au sémaphore. Ces protocoles, joints à un mécanisme de mémoire virtuelle répartie, peuvent ainsi permettre une programmation classique de ce type de machines parallèles.

1 Introduction

Le contexte réparti est généralement caractérisé par l'impossibilité de capter un état global de façon instantanée [LEL 77,LAM 78]. Les processus, placés sur les différents sites du système, ne peuvent en effet s'échanger des informations qu'à l'aide de messages qui transitent via des canaux de communication, et ces échanges ne peuvent être instantanés. Si pour certaines applications réparties cela ne pose pas de problèmes particuliers [BER 89], il n'en est pas de même dans le cas général. L'impossibilité de fournir une vision cohérente et instantanée de l'état d'une application oblige à se satisfaire d'une perception approchée de cet état. Le rôle du système consiste alors à offrir des mécanismes permettant d'obtenir un état approché le plus récent possible, cohérent du point de vue du problème à résoudre [LEL 77,LAM 78,CHA 85,RAY 92]. Cette difficulté à capter un état global a ainsi donné naissance au niveau des systèmes d'une part à un certain nombre de mécanismes de synchronisation répartie (jetons [LEL 77], séquenceurs [REE 79], marqueurs [MIS 83,CHA 85], temps logique [LAM 78,FID 91], etc) et d'autre part à la définition de primitives de communication qui, en un certain sens, éliminent le "bruit" apporté par l'asynchronisme des communications (diffusion ordonnée [CRI 85,GAR 91,SCH 90], ordre causal [RAY 91a,BIR 91], etc). Il convient également de noter qu'un certain nombre de langages ont été proposés pour aider à maitriser ces problèmes [BAL 89].

L'arrivée des machines parallèles à mémoire répartie (telles que, par exemple, l'iPSC d'Intel) apporte une problématique nouvelle au domaine du réparti. Alors qu'avec la problématique classique (qui, d'une façon un peu réductrice, était celle des protocoles et des réseaux locaux) on pensait un programme en termes de processus coopérant par messages (et en conséquence l'accent était mis sur la répartition proprement dite), cette nouvelle donne de machines cherche, grâce entre autres à la régularité de leurs structures, à faciliter

la définition d'un système qui permette rapidement (i.e. avec peu de couches de logiciel) aux programmeurs d'application d'être indépendants du support et d'obtenir des programmes efficaces. Des concepts classiques des systèmes centralisés sont ainsi "réhabilités" et adaptés à ce type d'architectures ; le concept de mémoire virtuelle en est l'exemple le plus typique [LI 85,LAII 91]. Cet article se situe dans cette ligne de pensée et s'intéresse à la mise en oeuvre des sémaphores sur ce type de machines parallèles. Ainsi la donnée conjointe d'un des protocoles présentés et d'une implémentation de mémoire virtuelle permet le transport de programmes parallèles "classiques" (compris ici comme étant les programmes où la communication se fait via une mémoire partagée et la synchronisation par des sémaphores) sur des machines parallèles à mémoire répartie, l'adressage et le contrôle utilisés par ces programmes étant alors indépendants du support effectivement utilisé (centralisé ou réparti).

L'article est composé de 2 parties principales. La première (§2) propose une implémentation *abstraite* des sémaphores ; le terme *abstrait* fait ici référence à la notion de niveau d'abstraction : le protocole proposé à ce niveau définit un schéma auquel peuvent être associées plusieurs mises en oeuvre possibles. La seconde partie (§3) présente 3 implémentations concrètes du schéma précédent ; celles-ci peuvent être considérées comme canoniques au sens où elles reposent sur les 3 mécanismes fondamentaux à partir desquels sont construits les algorithmes répartis d'exclusion mutuelle.

2 Une implémentation abstraite

2.1 Une définition des sémaphores

Un sémaphore S est un compteur d'événements [DIJ 71]. Représenté par une variable entière non-négative intinitialisée à une valeur $s_0 \geq 0$, il ne peut être manipulé que par les 2 opérations suivantes appelées P et V ; V permet de signaler l'occurence d'un événement, P permet de consommer un événement et d'assujettir la progression d'un processus à la présence d'un événement non encore consommé. Au plan conceptuel ces opérations sont éxécutées atomiquement.

$$P(S) : \textbf{ attendre } (S > 0);\ S := S - 1$$
$$V(S) :\ S := S + 1$$

L'invariant suivant est associé à tout sémaphore S [DIJ 71,MAR 81] ($\#x$ représente le nombre d'exécutions de x terminées depuis le début) :

$$S \geq 0 \text{ et } S = s_0 + \#V(S) - \#P(S)$$

Les sémaphores constituent un mécanisme primitif de synchronisation qui peut servir de cible pour implémenter la plupart des structures de synchronisation de haut niveau proposées dans les langages parallèles. Le lecteur intéressé trouvera dans [AND 91] un très bon développement sur les sémaphores et sur leur utilisation.

2.2 Une implémentation abstraite

Au concept de sémaphore est associée la notion de variable partagée. En donner une mise en oeuvre peut donc reposer sur une méthode générale de distribution d'un objet ; c'est la démarche suivie par Schneider [SCH 82]. Comme nous ne sommes intéressés que par l'objet particulier qu'est le sémaphore nous allons nous appuyer sur ses propriétés pour en obtenir des mises en oeuvre efficaces, moins coûteuses en temps et en nombre de messages.

On considère dans ce qui suit un seul sémaphore S. On supposera de plus, sans perte de généralité, qu'il y a un processus du programme par processeur de la machine et on les identifiera par les entiers : $1, 2, \ldots, i, \ldots, n$.

Les opérations P sur un sémaphore sont potentiellement conflictuelles : en effet si plusieurs P se présentent simultanément sur le même sémaphore S lorsque celui-ci a la valeur 1, alors une seule pourra être exécutée, les autres étant bloquantes pour les processus qui les ont invoquées. Une solution pour régler ces conflits consiste à n'exécuter qu'une opération P à la fois sur un sémaphore, c'est à dire en section critique. Les opérations V ne rentrent jamais en conflit au sens où, quel que soit l'état des sémaphores, l'exécution des unes n'empêche jamais l'exécution des autres. Du point de vue des opérations V, la cohérence de S requiert simplement de n'oublier aucune incrémentation. A ce niveau d'abstraction, ces remarques conduisent à l'implémentation abstraite suivante :

procédure *P(S)*= **début** *acquérir l'exclusion mutuelle associée à S;*
attendre $(S > 0)$;
$S := S - 1$;
libérer l'exclusion associée à S
fin
procédure *V(S)*= **début** *S:=S+1* **fin**

Les problèmes à résoudre sont alors :

i) trouver une implémentation répartie pour la variable S,

ii) rechercher un algorithme d'exclusion efficace et adapté pour les opérations P.

Une solution à ce poblème consiste à utiliser l'invariant associé au sémaphore :

$$S = s_0 + \#V - \#P$$

Cette relation montre en effet que la valeur de S peut être décomposée en 2 parties $s_0 + \#V$ et $\#P$ qui peuvent être gérées différemment.

Tout d'abord chacun des sites i qui peut invoquer *P(S)* est doté d'une variable locale nv_i qui constitue une image de $\#V$. Lorsqu'un site j exécute *V(S)*, il diffuse un message *incr* à tous les sites qui ont une image nv_i. A la réception d'un tel message, le site i incrémente nv_i de 1. On a ainsi $\forall i : nv_i \leq \#V$ et l'égalité est obtenue lorsqu'il n'y a pas de messages *incr* en transit.

L'opération P exécutée par un site s'écrit maintenant, en matérialisant le compteur $\#P$ par une variable globale np :

procédure *P(S)* = **début** *acquérir l'exclusion;*
attendre $(s_0 + nv_i - np > 0)$;
$np := np + 1$;
libérer l'exclusion
fin

La variable globale np n'est accédée qu'en section critique, sa gestion va donc se confondre avec celle de la section critique. Selon le protocole d'exclusion choisi on va obtenir une mise en oeuvre concrète particulière ; c'est ce que nous allons examiner dans ce qui suit. Les 3 solutions proposées se distinguent ainsi par la façon dont est représentée la variable np et le contrôle des accès qui lui est associé.

3 Trois implémentations concrètes

3.1 Un regard sur les algorithmes d'exclusion mutuelle

Comme les algorithmes d'exclusion constituent l'élément de base choisi pour réaliser les opérations P, nous en rappelons brièvement la classification proposée dans [RAY 91b]. Celle-ci distingue 2 classes : les algorithmes

d'exclusion fondés sur les permissions et ceux fondés sur un jeton.

Dans la première classe un processus doit demander et obtenir la permission d'autres processus afin de pouvoir pénétrer en section critique. Cette classe se subdivise en 2 sous-classes selon la nature des permissions. Dans la première les permissions sont individuelles ; le conflit d'un site i avec les autres est éclaté en autant de conflits bilatéraux $(i,j), \forall j \neq i$ qui sont réglés individuellement, et donc lorsqu'un site j donne sa permission à i il n'engage que lui. L'algorithme de Ricart et Agrawala est le représentant de cette classe [RIC 81]. Dans l'autre sous-classe il s'agit de permissions "issues d'arbitres" au sens où lorsque j donne sa permission à i il s'engage pour tous les sites qui lui demandent la permission. L'algorithme de Maekawa est le représentant de cette classe [MAE 85]. On parle également dans ce cas d'algorithmes à quorums [NEI 91] : le quorum Q_i associé au site i désignant l'ensemble des sites auxquels i doit demander (puis restituer) la permission ; ces algorithmes sont caractérisés par la relation suivante [MAE 85,SAN 87,BAR 86,AGR 90] : $\forall i,j : Q_i \cap Q_j \neq \phi$. Quel que soit le type des permissions utilisées, ce sont elles qui garantissent la propriété de sûreté (i.e. la cohérence de la section critique : au plus un utilisateur). La façon la plus simple, et la plus utilisée, de garantir la propriété de vivacité (toute demande sera honorée) consiste à estampiller [LAM 78] les demandes.

Dans la classe des algorithmes fondés sur un jeton, l'unicité de celui-ci garantit la propriété de sûreté. Ces algorithmes se distinguent les uns des autres par la façon dont ils assurent la propriété de vivacité. Deux schémas de base pour cela sont le mouvement perpétuel du jeton sur un anneau [LEL 77] et l'utilisation d'une arborescence, qui se reconfigure, pour router les requêtes de demande du jeton [NAI 87,RAY 89,VdS 87].

Le lecteur intéressé par de plus amples développements sur les algorithmes répartis d'exclusion mutuelle pourra consulter [RAY 92] qui, en accord avec cette classification, étudie dans le détail les plus représentatifs d'entre-eux.

3.2 Utilisation d'un jeton

Une des façons les plus simples pour implémenter une variable globale dans un système réparti est d'utiliser un jeton qui va la transporter. Cette méthode, bien connue [LEL 78], a été formalisée dans [CHA 88]. Une solution pour implémenter l'opération *P(S)* consiste donc à choisir un algorithme d'exclusion fondé sur un jeton et à valuer celui-ci avec la variable d'intérêt *np*. Avec les primitives *acquérir(jeton)* et *libérer(jeton)* offertes par l'algorithme choisi on obtient la mise en oeuvre suivante :

procédure *P(S)*=
    début *acquérir(jeton(np));*
        attendre $(s_0 + nv_i - np > 0)$;
        $np := np + 1$;
        libérer(jeton(np))
    fin

La propriété de vivacité (tout site qui veut exécuter *P* pourra le faire) se confond donc avec l'obtention du jeton. Les algorithmes d'exclusion à jeton assurent tous cette propriété : un site qui désire le jeton l'obtiendra (sous l'hypothèse que tout site qui l'a obtenu, le libère). Nous ne détaillerons pas ici les algorithmes à jeton ; le lecteur intéressé consultera [RAY 92]. Il convient de remarquer que les mécanismes utilisés dans cette solution sont très proches des concepts de compteurs d'événements et de séquenceurs introduits par Reed et Kanodia [REE 79].

3.3 Une solution fondée sur les permissions individuelles

Lorsqu'un site i désire entrer en section critique il envoie une requête estampillée à chacun des autres sites j. Lorsqu'un site j reçoit une telle requête il renvoie à i un message *permission* par retour du courrier s'il n'est

pas intéressé par la section critique ou si, bien qu'intéressé, sa requête n'est pas prioritaire par rapport à celle de i ; dans le cas où il est prioritaire il diffère l'envoi de sa permission à sa sortie de la section critique. La priorité entre les requêtes est définie par l'ordre total créé par les estampilles qui leur sont associées [LAM 78] ; la propriété de vivacité est garantie par cet ordre total.

Rappelons que le problème consiste à donner une représentation cohérente de la variable np. Dans l'implémentation abstraite, np est incrémentée de 1 à chaque sortie de section critique ; on a donc, lorsqu'un site i, détenteur de l'exclusion, exécute le test $s_0 + nv_i - np > 0$:

$$np = \#(\textit{entrées en section critique}) - 1$$

Comme les requêtes sont servies dans l'ordre des estampilles cette valeur peut être calculée localement par un site i détenteur de l'exclusion. Chaque fois que i envoie un message *permission* c'est, de son point de vue, un autre site qui rentre en section critique ; on a donc lorsque i vient d'obtenir la section critique :

$$np = \#(\textit{messages permission envoyés par } i) + \#(\textit{entrées en section critique par } i) - 1$$

Ce nombre peut facilement être calculé par tout site i à l'aide d'une variable locale np_i, et l'on a $np_i = np$ lorsque i est en section critique. On donne en annexe le texte complet d'un tel protocole pour la mise en oeuvre d'une opération P généralisée [PRE 75].

3.4 Une solution fondée sur les quorums

Dans ce cas, à tout site i est associé un ensemble de sites Q_i, ou quorum, auquel celui-ci doit envoyer des requêtes, puis recevoir des permissions afin de rentrer en section critique. Nous avons vu que l'on doit avoir $Q_i \cap Q_j \neq \phi, \forall i, j$ afin d'assurer la sûreté (deux sites doivent avoir au moins un arbitre en commun) [MAE 85]. Après utilisation, les permissions sont restituées (ce qui n'était pas le cas avec les permissions individuelles). La vivacité est obtenue par l'utilisation d'estampilles [MAE 85,SAN 87,RAY 92]. Un tel algorithme fournit donc à tout site i un quorum Q_i et les 2 primitives :

- *acquérir_quorum* : envoie des requêtes aux sites de Q_i et en attend les permissions correspondantes
- *libérer_quorum* : restitue les permissions aux sites de Q_i.

Entre ces 2 appels le site i a obtenu les permissions des sites de Q_i et chacun d'eux ne peut alors satisfaire d'autres requêtes tant que sa permission ne lui a pas été retournée.

Afin de représenter np on associe à chaque site i une variable locale np_i, initialisée à 0, et gérée de la façon suivante. Lorsque j envoie sa permission à i il lui transmet la valeur de np_j, et lorsque i la lui retourne il lui transmet la valeur de np_i (et j exécute alors $np_j := np_i$) ; tout message *permission* transporte donc la valeur de la variable locale np_k de son émetteur k. Contrairement à l'algorithme fondé sur les permissions individuelles où tout site i envoyait (recevait) des requêtes vers (de) chacun des autres, ici le site n'envoie des requêtes qu'aux sites $j \in Q_i$ et n'en reçoit que des sites k tels que $i \in Q_k$; une simple comptabilité comme la précédente est donc inopérante pour calculer np ; mais comme :

- $\forall i, j : Q_i \cap Q_j \neq \phi$
- et que, lorsque i exécute *libérer_quorum* il renvoie aux sites $j \in Q_i$ leurs permissions dotées de la valeur de np_i et que ceux-ci exécutent alors la mise à jour $np_j := np_i$

on a, lorsqu'un site i, après avoir exécuté *acquérir_quorum*, a obtenu l'exclusion et les valeurs np_j de tous les sites $j \in Q_i$:

$$np = \max_{j \in Q_i}(\{np_j\})$$

L'opération *P(S)* exécutée par un site i peut alors être implémentée par :

procédure *P(S)=*
début *acquérir_quorum;*
%i a alors obtenu les valeurs $np_j, \forall j \in Q_i$%
$np_i := \max_{j \in Q_i}(\{np_j\})$;
attendre $(s_0 + nv_i - np_i > 0)$;
$np_i := np_i + 1$;
libérer_quorum;
%provoque les mises à jour $np_j := np_i, \forall j \in Q_i$%
fin

Remarque. Les variables np_i sont ici gérées comme des numéros de version ; chaque site i possède la copie d'un objet (vide) et son numéro de version np_i. La section critique utilise le dernier numéro de version (i.e. le plus grand) et définit le numéro de la prochaine version (le numéro suivant).

3.5 Remarques

3.5.1 Remise à zéro

Un inconvénient potentiel des solutions précédentes, qui a pour origine l'utilisation directe de l'invariant, réside dans la croissance arbitraire des variables nv_i et np_i (ou np si l'on considère le jeton). Ceci est le propre des algorithmes fondés sur l'utilisation de compteurs à évolution monotone. Si cela est rédhibitoire il est possible d'introduire un protocole simple de "remise à zéro" pour borner le domaine d'évolution de ces compteurs.

Lorsque le site i, possesseur de l'exclusion mutuelle, constate que la variable np_i (ou le champ np du jeton) a atteint un seuil prédéfini x, il exécute le protocole suivant :

i) le site i envoie à tous les sites un message *raz(x)* et en attend les acquittements.

ii) lorsqu'un site j reçoit *raz(x)* il exécute $nv_j := nv_j - x$ et, s'il ne s'agit pas de l'algorithme à jeton, $np_j := np_j - x$.

iii) ce n'est qu'après avoir reçu tous les acquittements, et exécuté $np := np - x$ s'il s'agit de l'algorithme à jeton, que le site i peut libérer la section critique.

Dans le cas des 2 protocoles fondés sur les permissions individuelles et les quorums il est également possible de remplacer nv_i et np_i par une seule variable vs_i définie par l'invariant qui donne la valeur du sémaphore :

$$vs_i = s_0 + nv_i - np_i$$

Les incrémentations de nv_i et de np_i étant alors implémentées respectivement par une incrémentation et une décrémentation de vs_i. On retrouve alors en réparti la mise en oeuvre traditionnelle de la valeur du sémaphore : les valeurs locales vs_i bien qu'à instant donné non égales entre elles sont des images cohérentes de S au sens où les décisions prises avec leur aide sont correctes.

3.6 Défaillances

Bien qu'une étude de la résistance aux défaillances soit au delà des objectifs de cet article, nous pouvons faire les remarques suivantes. La résistance aux défaillances de divers types des algorithmes proposés est en fait celle des algorithmes d'exclusion mutuelle dont ils sont issus. En ce qui concerne les algorithmes à jeton, nous renvoyons le lecteur à [LEL 77,MIS 83]. Les algorithmes fondés sur les permissions individuelles sont peu résistants : en effet la défaillance d'un site a besoin d'être connue de (et prise en compte par) tous les autres sites [RIC 81]. Par contre les algorithmes à quorums (dont les algorithmes à vote majoritaire sont un cas particulier [THO 79,NEI 91]), sont résistants à un grand nombre de défaillances ; le lecteur pourra consulter à ce sujet les références [BAR 86,AGR 90].

4 Conclusion

Les protocoles proposés pour implémenter les sémaphores dans les systèmes à mémoire répartie sont relativement simples : ils s'appuient tous sur l'invariant associé au comportement des sémaphores. Ceci a permis de les définir de façon progressive : une implémentation abstraite a fixé un schéma général fondé sur l'utilisation de compteurs et de l'exclusion mutuelle ; les implémentations, appelées concrètes, ont ensuite utilisé des mises en oeuvre particulières pour les compteurs et l'exclusion mutuelle. (Remarquons qu'une démarche analogue, procédant par niveaux d'abstraction avec une implémentation abstraite et une famille d'implémentations concrètes associées, a été suivie pour implémenter le rendez-vous généralisé [BAG 89]).

L'efficacité comparée des protocoles obtenus dépend de celle des algorithmes d'exclusion utilisés. En effet si l'efficacité des opérations V est toujours la même, celle des opérations P varie en fonction du protocole d'exclusion. Quelle que soit la mise en oeuvre choisie (en fonction de l'application, du support, de la possibilité des défaillances, de leurs types, etc) il est important de remarquer que la mise en oeuvre répartie d'un sémaphore est toujours plus coûteuse que celle d'un algorithme d'exclusion mutuelle : il y a en effet en plus la mise en oeuvre de l'opération V. En conséquence les sémaphores répartis ne doivent être utilisés que pour résoudre les problèmes de synchronisation (allocation de ressources, coordination, etc) qui ne peuvent l'être avec la seule aide de l'exclusion mutuelle. Il s'agit là d'une différence essentielle par rapport aux systèmes centralisés. Utilisés conjointement avec la mémoire virtuelle répartie, les sémaphores et l'exclusion mutuelle constituent ainsi deux mécanismes de contrôle complémentaires pour exploiter des programmes "classiques" sur les machines parallèles à mémoire répartie.

Annexe

Cette annexe détaille le second des trois protocoles présentés (celui fondé sur les permissions individuelles). Afin de montrer la généralité de ces protocoles on donne l'implémentation de l'opération P généralisée suivante [PRE 75] :

$$P(S,k) : \ attendre\ (S-k \geq 0);\ S := S-k$$

Cette opération permet de "croquer" (chunk operation) plusieurs valeurs de façon atomique.

L'algorithme de base est celui de Ricart et Agrawala dont les grandes lignes ont été décrites dans le texte (une description explicative est donnée dans [RAY 92]) ; on se limite ici à donner les déclarations et le comportement associé à tout site i.

var $état_i$: {*dehors, dedans*} **init à** *dehors;*
$h_i, dernier_i$: **entier croissant init à** 0;
$priorité_i$: **boolén;**
$attendus_i$: **ensemble de sites init à** ϕ;
$différé_i$: **ensemble de couples***(site,valeur)* **init à** ϕ;
nv_i, np_i : **entier init à** 0

La variable $état_i$ a la valeur *dehors* lorsque le site i n'est pas intéressé par la section critique (c'est à dire n'est pas engagé dans une opération P) ; h_i est l'horloge logique du site i et $dernier_i$ représente la dernière valeur d'horloge qu'il a utilisée pour estampiller une requête. L'ensemble $attendus_i$ sert à mémoriser de qui le site i attend des permissions et $différé_i$ vers qui il en a retardé. Le paramètre k des opérations P est utilisé pour incrémenter la variable locale np_i lorsque le site i renvoie sa permission ou lorsqu'il termine l'exécution de l'opération P. Le comportement du site i est décrit par 3 énoncés : une procédure qui décrit la mise en oeuvre proprement dite de *P(S,k)*, et 2 traitements associés aux réceptions de messages nécessaires à la mise en oeuvre de P.

procédure *P(S,k)*=
début $état_i := dedans$;
$dernier_i := h_i + 1$;
envoyer $requête(k,(h_i,i))$ **à** j **pour** $1 \leq j \neq i \leq n$;
attendre $(attendus_i = \phi)$;
attendre $(s_0 + nv_i - (np_i + k) \geq 0)$;
$np_i := np_i + k$;
$\forall (j,k_j) \in différé_i$: **faire envoyer** $permission(i)$ **à** j;
$np_i := np_i + k_j$
fait;
$différé_i := \phi$;
$état_i := dehors$
fin

réception de $permission(j)$
$attendus_i := attendus_i - \{j\}$

réception de $requête(k_j,(h,j))$
début
$h_i := max(h_i,h)$;
$priorité_i := (état_i = dedans)$ **et** $((dernier_i,i) < (h,j))$;
si $priorité_i$ **alors** $différé_i := différé_i \cup \{(j,k_j)\}$
sinon envoyer $permission(i)$ **à** j;
$np_i := np_i + k_j$
fsi

La mise en oeuvre de l'opération *V(S,m)*, qui incrémente S de m, utilise la diffusion d'un message *incr(m)* qui provoque l'incrémentation des variables nv_i. Les textes associés aux procédures *P, V* et aux réceptions de messages sont ininterruptibles à l'exception de la primitive **attendre**. Les textes des programmes utilisateurs sont interruptibles.

References

[AGR 90] AGRAWALA D., EL ABBADI A. *Exploiting logical structures in replicated data bases.* Inf. Proc. Letters, vol.33, (1990), pp. 255-230.

[AND 91] ANDREWS G.R. *Concurrent programming : principles and practice.* The Benjamin/Cummings Pub. Co., (1991), 537 p.

[BAG 89] BAGRODIA R.L. *Process synchronization : design and performance evaluation of distributed algorithms.* IEEE Trans. on SE, vol.15,9, (september 1989), pp. 3-23.

[BAL 89] BAL H.E., STEINER J.G., TANENBAUM A. *Programming languages for distributed computing.* Computing Surveys, vol.21,3, (september 1989), pp. 261-322.

[BAR 86] BARBARA D., GARCIA-MOLINA H. *Mutual exclusion in partitionned distributed systems.* Distributed Computing, vol.1, (1986), pp. 119-132.

[BER 89] BERTSEKAS D.P., TSITSIKLIS J.N. *Parallel and distributed computation.* Prentice Hall, (1989), 715 p.

[BIR 91] BIRMAN K., SCHIPER A., STEPHENSON P. *Lightweight causal and atomic group multicast.* ACM TOCS, vol.9,3, (1991) pp. 272-314.

[CHA 84] CHANDY K.M., MISRA J. *The drinking philosopher problem.* ACM Toplas, vol.6,4, (1984), pp. 632-646.

[CHA 85] CHANDY K.M., LAMPORT L. *Distributed snapshots : determining global states of distributed programs.* ACM TOCS, vol.3,1, (1985), pp. 63-75.

[CHA 88] CHANDY K.M., MISRA J. *Parallel program design : a foundation.* Addison Wesley Pub. Co., (1988), 516 p.

[CRI 85] CRISTIAN F., AGHILI H., STRONG R., DOLEV D. *Atomic broadcast : from simple message diffusion to byzantine agreement.* Proc. FTCS 15, Ann Arbor, (1985).

[DIJ 71] DIJKSTRA E.W.D. *Hierarchical ordering of sequential processes.* Acta Informatica, vol.1,2, (1971), pp. 115-138.

[FID 91] FIDGE C. *Logical time in distributed computing systems.* Computer, vol.24,8, (august 1991), pp. 28-33.

[GAR 91] GARCIA-MOLINA H., SPAUSTER A. *Ordered and reliable multicast communication.* ACM TOCS, vol.9,3, (1991) pp. 242-271.

[LAH 91] LAHJOMRI Z., PRIOL Th. *KOAN : a shared virtual memory for the iPSC/2 hypercube.* Rapport de Recherche Inria, n°1504, (september 1991), 29 p. (Soumis à publication).

[LAM 78] LAMPORT L. *Time, clocks and the ordering of events in a distributed system.* Comm. ACM, vol.21,7, (july 1978), pp. 558-565.

[LEL 77] LE LANN G. *Distributed systems : towards a formal approach.* IFIP Congress, (1977), pp. 155-160.

[LEL 78] LE LANN G. *Algorithms for distributed data sharing systems which use tickets.* Proc. 3rd Berkeley Workshop on Data Management, (1978), pp. 259-272.

[LI 85] LI K., HUDAK P. *Memory coherence in shared virtual memory.* Proc. 5th ACM Symposium on Principles of Dist. Computing, (august 1985), pp. 229-239.

[MAE 85] MAEKAWA M. *A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems.* ACM TOCS, vol.3,2, (1985), pp. 145-159.

[MAR 81] MARTIN A.J. *An axiomatic definition of synchronization primitive.* Acta Informatica, vol.16, (1981), pp. 219-235.

[MIS 83] MISRA J. *Detecting termination detection of distributed computations using markers.* Proc. 2nd ACM Conf. on PODC, (1983), pp. 290-294.

[NAI 87] NAIMI M., TREHEL M. *A distributed algorithm for mutual exclusion based on data structure and fault-tolerance.* Proc. IEEE Phoenix Conf. on Comp. and Comm., (1987), pp. 256-276.

[NEI 91] NEILSEN M.L., MIZUNO M., RAYNAL M. *A general method to define quorums.* Rapport de Recherche Inria, n°1529, (september 1991), 18 p.

[PRE 75] PRESSER L. *Multiprogramming coordination.* Computing Surveys, vol.7,1, (1975), pp. 21-44.

[RAY 89] RAYMOND K. *A tree-based algorithm for mutual exclusion.* ACM TOCS, vol.7,1, (1989), pp. 61-77.

[RAY 91a] RAYNAL M., SCHIPER A., TOUEG S. *The causal ordering abstraction and a simple way to implement it.* Inf. proc. Letters, vol.30, (1991), pp. 343-350.

[RAY 91b] RAYNAL M. *A simple taxonomy for distributed mutual exclusion algorithms.* ACM Operating Systems Review, vol.25,2, (april 1991), pp. 47-51.

[RAY 92] RAYNAL M. *Synchronisation et état global dans les systèmes répartis.* Eyrolles, Collection EDF, (janvier 1992), 230 p.

[REE 79] REED D., KANODIA R.K., *Synchronization with eventcounts and sequencers.* Comm. ACM, vol.21,3, (february 1979), pp. 115-123.

[RIA 81] RICART G., AGRAWALA A.K. *An optimal algorithm for mutual exclusion in computer networks.* Comm. ACM, vol.24,1, (1981), pp. 9-17.

[SAN 87] SANDERS B.A. *The information structure of distributed mutual exclusion algorithms.* ACM TOCS, vol.5,3, (1987), pp. 284-299.

[SCH 82] SCHNEIDER F.B. *Synchronization in distributed programs.* ACM Toplas, vol.4,2, (april 1982), pp. 125-148.

[SCH 90] SCHNEIDER F.B. *Implementing fault-tolerance services using the state machine approach : a tutorial.* ACM Computing Surveys, vol.22,4, (1990), pp. 299-320.

[THO 79] THOMAS R.H. *A majority consensus approach to concurrency control.* ACM TODS, vol.4, (1979), pp. 180-209.

[VdS 87] Van de SNEPSCHEUT J.L. *Fair mutual exclusion on a graph of processes.* Distributed Computing, vol.2, (1987), pp. 113-115.

LISTE DES PUBLICATIONS INTERNES IRISA 1992

PI 624 SIGNAL AS A MODEL FOR REAL-TIME AND HYBRID SYSTEMS
Albert BENVENISTE, Michel LE BORGNE, Paul LE GUERNIC
Janvier 1992, 22 pages.

PI 625 ON THE CENTRAL-LIMIT THEOREM FOR TRACKING ESTIMATORS WITH SMALL GAIN - INFINITE HORIZON CASE
Bernard DELYON, Anatoli JUDITSKY
Janvier 1992, 16 pages.

PI 626 A MONTE CARLO METHOD BASED ON ANTITHETIC VARIATES FOR NETWORK RELIABILITY COMPUTATIONS
Mohamed EL KHADIRI, Gerardo RUBINO
Janvier 1992, 28 pages.

PI 627 CONSTRAINED MULTISCALE MARKOV RANDOM FIELDS AND THE ANALYSIS OF VISUAL MOTION
Fabrice HEITZ, Patrick PEREZ, Patrick BOUTHEMY
Janvier 1992, 40 pages.

PI 628 ON ITERATIVE REFINEMENT FOR THE SPECTRAL DECOMPOSITION OF SYMMETRIC MATRICES
Alexander N. MALYSHEV
Janvier 1992, 26 pages.

PI 629 STRUCTURAL OPERATIONAL SPECIFICATIONS AND TRACE AUTOMATA
Eric BADOUEL, Philippe DARONDEAU
Janvier 1992, 36 pages.

PI 630 EREBUS, A DEBUGGER FOR ASYNCHRONOUS DISTRIBUTED COMPUTING SYSTEM
Michel HURFIN, Noël PLOUZEAU, Michel RAYNAL
Janvier 1992, 14 pages.

PI 631 PROTOCOLES SIMPLES POUR L'IMPLEMENTATION REPARTIE DES SEMAPHORES
Michel RAYNAL
Janvier 1992, 14 pages.

PI 632 L-STABLE PARALLEL ONE-BLOCK METHODS FOR ORDINARY DIFFERENTIAL EQUATIONS
Philippe CHARTIER, Bernard PHILIPPE
Janvier 1992, 28 pages.

PI 633 ON EFFICIENT CHARACTERIZING SOLUTIONS OF LINEAR DIOPHANTINE EQUATIONS AND ITS APPLICATION TO DATA DEPENDENCE ANALYSIS
Christine EISENBEIS, Olivier TEMAM, Harry WIJSHOFF
Janvier 1992, 22 pages.

PI 634 UN NOYAU DE SYSTEME REPARTI POUR LES APPLICATIONS GEREES PAR UN TEMPS VIRTUEL
Janvier 1992, 20 pages.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249-6399