La communication et la synchronisation inter-tâches

Samia Bouzefrane

Maître de Conférences

CEDRIC –CNAM

samia.bouzefrane@cnam.fr
http://cedric.cnam.fr/~bouzefra

Sommaire

- **Introduction aux problèmes de synchronisation**
- **Sémaphores : principe et utilisation**
- **Moniteurs : principe et utilisation**

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Introduction

- **Programme multi-tâches : coopération inter-tâches**

– Échange ou partage d'informations

– Synchronisation pour la protection des données

- **Deux modèles**

– Système centralisé : privilégie la communication et la synchronisation par **mémoire commune**

– Système distribué : privilégie la communication et la synchronisation par **messages**

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Caractéristiques des systèmes distribués

- **Ensemble de machines reliées en réseau (pouvant être temps réel)**
- **Pas de mémoire commune**
- **Coopération et synchronisation basées sur *l'échange de messages***

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Mécanismes de communication dans les systèmes distribués

Appel de procédures distantes
– envoi d'une demande à un serveur
– plusieurs sémantiques (appel bloquant (synchrone) ou non bloquant (asynchrone))
– mécanisme système
– exemple : RPC

Invocation de méthodes distantes
– appel d'une méthode d'un objet situé sur un site distant
– mécanisme langage
– exemple : Java RMI (Remote Method Invocation)

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Caractéristiques des systèmes centralisés

- **Une seule machine pour plusieurs tâches :** ***partage obligatoire***
- **Mémoire commune**
- **Communication et synchronisation par** ***partage de données*** **en mémoire**

– simple à mettre en œuvre

– pose le problème de la protection des données par rapport à des accès multiples (problème de cohérence) lorsque les opérations ne sont pas atomiques

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Mécanismes présents dans les systèmes centralisés

- **Masquage des interruptions**

– dangereux, à réserver à des zones très ciblées (noyau système)

- **Test and Set**

– bas niveau, plutôt matériel (au niveau du processeur)

- **Sémaphore (Dijkstra - 1965)**

– assez bas niveau mais très commun et simple dans le cas général

- **Moniteur**

– haut-niveau, le plus commode et le plus sûr (au niveau langage)

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Synchronisation dans les systèmes centralisés

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Mécanismes de synchronisation

Deux mécanismes classiques peuvent être utilisés pour protéger des données accédées par plusieurs tâches :

– **Les sémaphores** qui autorisent l'accès à une ressource critique pour un nombre déterminé de fois

– **Les moniteurs** qui permettent « d'encapsuler » des données en définissant des règles d'accès exclusif à ces données

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Les Sémaphores

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Les sémaphores

• **Un sémaphore (Dijkstra 1965) S est un élément de synchronisation caractérisé par trois informations :**
– un compteur interne entier (≥0)
– une file d'attente
– deux opérations atomiques (exécution indivisible) nommées P et V

La création et l'initialisation sont faites à l'aide d'une opération atomique.

• **Si S est un sémaphore, on notera P(S) ou V(S) l'appel d'une opération sur le sémaphore S**

• **On dit qu'un sémaphore est binaire si son compteur ne peut prendre que les valeurs 0 ou 1**

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Les sémaphores : sémantique/1

Primitive P(S) : prend le sémaphore; appel bloquant

```
Val(S) = Val(S)-1;
Si (Val(S) < 0) Alors
Bloquer l'appelant et le mettre dans la file d'attente associée à S;
Finsi
```

Primitive V(S) : rend le sémaphore; jamais bloquant

```
Val(S) = Val(S)+1;
Si (Val(S) <= 0) Alors
Choisir un processus dans la file d'attente, le retirer de celle-ci
et le réveiller (le débloquer);
Finsi
```

Phase Initiale :

```
Val(S):= V;
```


samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Les sémaphores : sémantique/2

- **On a l'invariant suivant :**

Si *Val(S) < 0* **Alors** *|(Val(S)|= Longueur (File d'attente de S)*

- **Dans le cas d'un sémaphore binaire, l'opération V(S) n'aura aucun effet si le compteur du sémaphore est déjà à un 1**

- **Le concept de sémaphore ne précise pas quelle tâche est réveillée dans le cas où il y en a plusieurs en attente**

– gestion FIFO (on réveille la plus ancienne dans la file)
– gestion par priorité (on réveille la plus prioritaire)

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Les sémaphores : Exemple d'utilisation

- **Les primitives P et V encadrent les opérations à exécuter de manière atomique :**
  P(S)
  suite d'opérations (section critique)
  V(S)

- Si **Val(S)** est initialisé à **1** alors la section critique est exécutée en **exclusion mutuelle**.

- Si **Val(S)** est initialisé à **n>1** alors **n tâches** peuvent exécuter simultanément la section critique **(exemple du modèle Lecteurs/Rédacteurs)**

- Si **Val(S)** est initialisé à **0** alors la première tâche qui veut accéder à la section critique sera bloquée jusqu'à ce qu'une autre tâche l'autorise en exécutant un V(S), (**c'est le modèle du Client/Serveur**)

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Les sémaphores : Guide d'utilisation

- **Assurer une bonne protection**

– N'utiliser les sémaphores que sur de petites sections critiques
– Faire en sorte que celui qui rend le sémaphore (V) soit celui qui l'a pris (P)
– Isoler les actions en section critique du reste du code

- **Éviter l'interblocage en ordonnant la prise des sémaphores ou utiliser des tableaux de sémaphores à la Unix**

- **Exemple d'interblocage :**

– la tâche A prend S1, la tâche B prend S2 (*S1 et S2 sont deux sémaphores binaires*)
– la tâche A tente de prendre S2 : elle se bloque
– la tâche B tente de prendre S1 : elle se bloque
⇒ ***on est dans une situation d'interblocage***

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

L’exclusion mutuelle à l’aide des sémaphores

Soient N processus se partageant une ressource critique.

Un processus Pi s'écrira :
Contexte commun :
mutex : sémaphore initialisé à **1** ;

Processus Pi
Début
    Première partie du programme ;
    P(mutex)
    Appel de la procédure A {section critique }
    V(mutex)
    Reste du programme ;
Fin

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Modèle du Prod/Cons à l'aide des sémaphores/1

Solution pour un producteur :

Contexte commun :

SNbVides : **sémaphore** initialisé à **N** ;
SNbPleins : **sémaphore** initialisé à **0**;
Tampon: Tableau de N messages ;

Processus **Producteur**Var
Queue : entier initialisé à 0;
MessProd : message;
Debut Répéter
Fabriquer(MessProd);
P(SNbVides);
Déposer(MessProd,Tampon(Queue));
V(SNbPleins);
Jusqua Faux;
FinProcessus

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Modèle du Prod/Cons à l'aide des sémaphores/2

Solution pour un consommateur :

Consommateur

Var Tete : entier initialisé à 0;
MessCons : message;
Debut Répéter
    P(SNbPleins);
    Prélever(MessCons,Tampon(Tete));
    V(SNbVides);
    Traiter(MessCons);
Jusqua Faux;

Fin

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Modèle Client/Serveur à l'aide des sémaphores

Contexte commun: *spriv*: sémaphore initialisé à **0**;

Processus *Serveur*
Tantque vrai faire
P(*spriv*)
{***Réalisation du service*** }
.....
Fintantque

Processus *Client1*
.....
V(*spriv*)
.....

Processus *Client2*
.....
V(*spriv*)
.....

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Les moniteurs

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Les moniteurs

- **Les moniteurs proposent une solution de "haut-niveau" pour la protection de données partagées (Hoare 1974)**

- **Ils simplifient la mise en place de sections critiques**

- **Ils sont définis par**

– des données internes (appelées aussi variables d'état)
– des primitives d'accès aux moniteurs (points d'entrée)
– des primitives internes (uniquement accessibles depuis l'intérieur du moniteur)
– une ou plusieurs files d'attentes

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Structure d'un moniteur

Type *m* = **moniteur**

Début

Déclaration des variables locales (ressources partagées);
Déclaration et corps des procédures du moniteur (points d'entrée);
Initialisation des variables locales;

Fin

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Les moniteurs: sémantique/1

- **Seul un processus (ou tâche ou thread) peut être actif à un moment donné à l'intérieur du moniteur**

- **La demande d'entrée dans un moniteur (ou d'exécution d'une primitive du moniteur) sera bloquante tant qu'il y aura un processus actif à l'intérieur du moniteur**

⇒ *L'accès à un moniteur construit donc implicitement une exclusion mutuelle*

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Les moniteurs: sémantique/2

• Lorsqu'un processus actif au sein d'un moniteur ne peut progresser dans son travail (une certaine condition est fausse), il libère l'accès au moniteur avant de se bloquer.

•Lorsque des variables internes du moniteur ont changé, le moniteur doit pouvoir « réveiller » un processus bloqué.

•Pour cela, il existe deux types de primitives :
– *wait* : qui libère l'accès au moniteur et bloque le processus appelant sur une condition
– *signal* : qui réveille sur une condition un des processus en attente à l'intérieur du moniteur (un processus qui a exécuté précédemment un *wait* sur la même condition)

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Les variables condition/1

- **Une variable condition** : est une variable
  - qui est définie à l'aide du type *condition*;
  - qui a un identificateur mais,
  - qui n'a **pas de valeur** (contrairement à un sémaphore).

- **Une condition :**
  - ne doit pas être initialisée
  - ne peut être manipulée que par les primitives Wait et Signal.
  - est représentée par une **file d'attente** de processus bloqués sur la même cause;
  - est donc assimilée à sa file d'attente.

- **La primitive *Wait*** bloque **systématiquement** le processus qui l'exécute

- **La primitive *Signal*** réveille un processus de la file d'attente de la condition spécifiée, si cette file d'attente n'est pas vide; sinon elle ne fait absolument rien.

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Les variables condition/2

- ***Syntaxe*** :

  cond.**Wait**;

  cond.**Signal**;

  /* *cond* est la variable de type condition déclarée comme variable locale */

- ***Autre synthaxe*** :

  Wait(*cond*) ;

  Signal(*cond*);

- Un processus réveillé par *Signal* continue son exécution à l'instruction qui suit le *Wait* qui l'a bloqué.

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Les moniteurs dans les langages de programmation

- **Selon les langages (ou les normes), ces mécanismes peuvent être implémentés de différentes façons**
  – méthodes « *wait* / *notify* / *notifyAll* » en Java et méthodes « *synchronized* »
  – primitives « *pthread_cond_wait* / *pthread_cond_signal* » en Posix et variables conditionnelles
  – objets protégés en Ada

- **La sémantique des réveils peut varier :**
  – Qui réveille t-on (le plus ancien, le plus prioritaire, un choisi au hasard, …)
  – Quand réveille t-on (dès la sortie du moniteur, au prochain ordonnancement, …)

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Un RDV entre N processus à l'aide des moniteurs

Type Rendez_vous = **moniteur**

Var Nb_arrivés : **entier** ; *Tous_Arrivés* : **condition** ; {*variables locales* }
Procedure Entry Arriver ; {*procédure accessible aux programmes utilisateurs* }
Début
Nb_arrivés := Nb_arrivés + 1 ;
Si Nb_arrivés < N Alors *Tous_Arrivés*.Wait ;
Tous_Arrivés.Signal;
Fin
Début {*Initialisations* }
Nb_arrivés := 0;
Fin.

Les programmes des processus s'écrivent alors :

Processus Pi

.....
***Rendez_vous*.Arriver** ; {*Point de rendez-vous : Pi sera bloqué si au moins*
.....

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Références

Samia Bouzefrane, Les Systèmes d'exploitation: Cours et Exercices corriges Unix, Linux et Windows XP avec C et JAVA (566 pages), Dunod Editeur, Octobre 2003, ISBN : 2 10 007 189 0.

Jean-François Peyre, supports de cours sur l'informatique industrielle-systèmes temps réel, CNAM(Paris).

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -