le cnam

LA COMMUNICATION ET LA SYNCHRONISATION INTER-TÂCHES

Samia Bouzefrane

CEDRIC –CNAM
samia.bouzefrane@cnam.fr
https://samia.roc.cnam.fr

lecnam

Sommaire

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

Les moniteurs

Principe des 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*

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

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

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 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)*

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.

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.

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 »*
  - *Méthodes « **await/signal/signalAll** » en Java et méthodes « **lock/unlock** »*
  - *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, ...)*

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: sera  bloquant si au moins
    un processus n'est pas arrivé au point de rendez-vous }
```

Exemple RDV/1

- Supposons que nous avons 3 processus qui doivent s'attendre. Ils vont tous exécuter **Rendez_vous.Arriver().**
- Le moniteur s'exécute en exclusion mutuelle, ce qui garantit la cohérence de la variable critique **Nb_arrivés**. Une file d'attente implicite est associée au moniteur.
- A la variable condition **Tous_Arrivés**, sont associées les fonctions **Wait** et **Signal** ainsi qu'une file d'attente initialement vide.

Variable *Nb_Arrivés*=0

Variable *N*=3

File d'attente Moniteur 

File d'attente cond. *Tous-Arrivés* 

Exemple RDV/2

Processus P1 exécute Rendez_vous.Arriver()

Rendez_vous = moniteur

Var Nb_arrivés : entier ;
Tous_Arrivés : condition ;

Procedure Entry Arriver ;

Début

Nb_arrivés := Nb_arrivés + 1 ;

Si Nb_arrivés < N Alors {

Tous_Arrivés.Wait ;}

Tous_Arrivés.Signal;

Fin

File d'attente Moniteur P2

File d'attente cond. *Tous-Arrivés* P1

Variable *Nb_Arrivés*=0=1

Variable *N*=3

Exemple RDV/2

Processus P2 exécute Rendez_vous.Arriver()

Rendez_vous = moniteur

```
Var Nb_arrivés : entier  ;
Tous_Arrivés : condition   ;

Procedure Entry Arriver ;

Début

        Nb_arrivés := Nb_arrivés + 1 ;

        Si Nb_arrivés < N Alors {

                Tous_Arrivés.Wait ;}

        Tous_Arrivés.Signal;

Fin
```

File d'attente Moniteur P3

File d'attente cond. *Tous-Arrivés* P1 P2

Variable *Nb_Arrivés*=0=1=2

Variable *N*=3

Exemple RDV/2

Processus P3 exécute Rendez_vous.Arriver()

Rendez_vous = moniteur

```
Var Nb_arrivés : entier  ;
Tous_Arrivés : condition   ;

Procedure Entry Arriver ;

Début

    Nb_arrivés := Nb_arrivés + 1 ;

    Si Nb_arrivés < N Alors {

        Tous_Arrivés.Wait ;}

    Tous_Arrivés.Signal;

Fin
```

File d'attente Moniteur 

File d'attente cond. *Tous-Arrivés* 

Variable *Nb_Arrivés*=0=1=2=3

Variable *N*=3

P3 exécute Tous_Arrivés.Signal et réveille P1.
Une fois P1 réveillé, P1 poursuit son exécution.

Exemple RDV/3

Processus P3 exécute Rendez_vous.Arriver()

Rendez_vous = moniteur

Var Nb_arrivés : entier ;
Tous_Arrivés : condition ;

Procedure Entry Arriver ;

Début

Nb_arrivés := Nb_arrivés + 1 ;

Si Nb_arrivés < N Alors {

Tous_Arrivés.Wait ;}

Tous_Arrivés.Signal;

Fin

File d'attente Moniteur 

File d'attente cond. *Tous-Arrivés* 

Variable *Nb_Arrivés*=0=1=2=3

Variable *N*=3

P1 exécute Tous_Arrivés.Signal et réveille P2.
P2 poursuit son exécution et exécute Tous_Arrivés.Signal.
Ce signal est perdu car pas de processus à réveiller.

En deux mots le concept de moniteur

- **Concept** proposé par **Hoare** en 1974 pour résoudre le problème de synchronisation.

Type m = moniteur

Début

Déclaration des variables locales;

Déclaration et corps des procédures du moniteur;

// **accessibles en exclusion mutuelle**

Initialisation;

Fin

- **Les procédures du moniteur se synchronisent à l'aide de deux primitives :** **Wait()**

Signal()

qui permettent de bloquer ou de réveiller un processus sur une condition.

Une condition est une variable qui n'a pas de valeur mais qui est implémentée à l'aide d'une file d'attente.

Syntaxe des primitives :

Cond.Wait() : bloque toujours le processus appelant

Cond.Signal() : réveille un processus bloqué dans la file d'attente associée à **Cond**.

lecnam

Les moniteurs Posix

Moniteurs Posix

- Un moniteur Posix est l'association
  - *d'un mutex (type pthread_mutex_t) qui sert à protéger la partie de code où l'on teste les conditions de progression*
  - *et d'une variable condition (type pthread_cond_t) qui sert de point de signalisation :*
    - on se met en attente sur cette variable par la primitive :

      pthread_cond_wait (&laVariableCondition,&leMutex);

    - on est réveillé sur cette variable avec la primitive :

      pthread_cond_signal (&laVariableCondition);

Schéma d'utilisation

- Soit la condition de progression C,
- Le schéma d'utilisation des moniteurs Posix est le suivant :

```
pthread_mutex_lock (&leMutex);
évaluer C;
while  ( ! C ) {
        pthread_cond_wait (&laVariableCondition,&leMutex);
        ré-évaluer C si nécessaire
}

Faire le travail;
pthread_mutex_unlock(&leMutex);
```

Exemple du Prod/Cons avec les moniteurs Posix

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

/* définition du tampon */
#define N      10  /* Nb de cases du tampon */
#define NbMess 20  /* Nb de messages échangés */
int NbPleins=0;
int tete=0, queue=0;
int tampon[N];

/* définition des conditions et du mutex */
pthread_cond_t  vide;
pthread_cond_t  plein;
pthread_mutex_t mutex;
pthread_t tid[2];
```

Exemple (suite)

```
void Deposer(int m){
pthread_mutex_lock (&mutex);
        if(NbPleins == N)
pthread_cond_wait (&plein, &mutex);
        tampon[queue]=m;
        queue=(queue+1)%N;
        NbPleins++;
        pthread_cond_signal (&vide);
pthread_mutex_unlock (&mutex);
}
```

```
int Prelever(void){
int m;
pthread_mutex_lock (&mutex);
if(NbPleins ==0) pthread_cond_wait (&vide, &mutex);
        m=tampon[tete];
        tete=(tete+1)%N;
        NbPleins--;
        pthread_cond_signal (&plein);
pthread_mutex_unlock (&mutex);
return m;
}
```

Exemple/suite

```
void * Prod(void * k)/********** PRODUCTEUR */
{
int i;
int mess;
srand(getpid());
for(i=0;i<=NbMess; i++){
        usleep(rand()%10000);  /* fabrication du
message */
        mess=rand()%1000;
        Deposer(mess);
        printf("Mess depose: %d\n",mess);
        }
}
```

```
void * Cons(void * k)
        /********** CONSOMMATEUR */
{
int i;
int mess;
srand(getpid());
for(i=0;i<=NbMess; i++){
        mess=Prelever();
        printf("\tMess preleve: %d\n",mess);
        usleep(rand()%1000000);  /* traitement du
message */
        }
 }
```

Exemple (fin)

```
void main()
{
int i,  num;
pthread_mutex_init(&mutex,0);
pthread_cond_init(&vide,0);
pthread_cond_init(&plein,0);

/* creation des threads */
pthread_create(tid,  0, (void * (*)()) Prod, NULL);
pthread_create(tid+1, 0, (void * (*)()) Cons, NULL);
```

```
// attente de la fin des threads
pthread_join(tid[0],NULL);
pthread_join(tid[1],NULL);

// libération des ressources
pthread_mutex_destroy(&mutex);
pthread_cond_destroy(&vide);
pthread_cond_destroy(&plein);
exit(0);
}
```

lecnam

Bibliographie

- Samia Bouzefrane, Les Systèmes d'exploitation: Cours et Exercices corrigés 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).

- Claude Kaiser, supports de cours de SMB137.