Contrôle de concurrence par sémaphores(suite)

Sémaphores privés

Principe

-un sémaphore ne contient qu'un compteur, c'est parfois insuffisant pour exprimer le contrôle d'un problème concurrent

-Dans le schéma des sémaphores privés, l'action de synchronisation est associée non plus à une ressource mais à un processus :

c'est le processus qui détecte s'il doit se bloquer ou non

Sémaphores privés

Définition

un sémaphore privé **spriv** d'un processus P_i est tel que

-seul P_i peut exécuter **P(spriv)** et **V(spriv)**

les autres processus ne peuvent exécuter que **V(spriv)**

-la valeur initiale du sémaphore doit être 0 : **E0(spriv,0)**

-la file d'attente d'un sémaphore privé contient au plus un processus

-la valeur d'un sémaphore privé vérifie **spriv.E≥-1**

Sémaphores privés

Utilisation de données partagées et de sémaphores privés pour le contrôle de la concurrence

-un processus se bloque lui-même sur son sémaphore privé, si nécessaire, en utilisant des données partagées reflétant l'état du système :les données de contrôle

-les données de contrôle doivent être lues et écrites en section critique, celle-ci doit être libérée avant le blocage éventuel du processus

-les données de contrôle servent à un processus à

Noter son blocage

Noter qu'un processus doit être réveillé

Sémaphores privés

Exemple : schéma clients/serveur

Un processus serveur dont le rôle est d'offrir un service à des processus clients ne peut s'exécuter que sur demande des clients, il traite une seule demande à la fois.

```
Contexte commun sémaphore spriv; E0(spriv,0);
```

Processus serveur	Processus client$_1$	Processus client$_2$
Tant que vrai faire	...	...
P(spriv);	V(spriv);	V(spriv);
// service	...	...
…		
Fait;		

Sémaphores privés

Schéma de synchronisation

```
Contexte commun N processus; //identifiés par 0,…,N-1
Sémaphore sempriv[N];tout i,E0(sempriv[i],0);
Sémaphore mutex; E0(mutex,1);
D; //ensemble de variables partagées
Processus Pi
Booleen OK;//contrôle du blocage hors SC
  //demande de ressources et blocage éventuel
 Utilisation des ressources
//libération des ressources et réveil éventuel de
processus
```

Sémaphores privés

Schéma de synchronisation

```
//demande de ressources et blocage éventuel
P(mutex);
 si cond1(D)// condition de non blocage pour Pi
 alors noter le non blocage dans D; OK=true;
 sinon noter dans D le blocage;OK=false;
 finsi;
V(mutex);
//blocage éventuel hors section critique
si non OK alors P(sempriv[i]);finsi;
 Utilisation des ressources
//libération des ressources et réveil éventuel
de processus
```

Sémaphores privés

Schéma de synchronisation

```
//libération des ressources et réveil éventuel
de processus
P(mutex);
 mise à jour des variables de D;
 pour tout processus bloqué Y faire
  si cond2(Y)//condition de réveil de Y
  alors mise à jour de D;
   V(sempriv[Y]); // réveil de Y
  finsi;
 fait;
V(mutex);
```

Sémaphores privés
Le repas des philosophes

Principe de la solution

- allocation globale des deux baguettes
- les variables de contrôle reflète l'état d'un philosophe

Un philosophe pense, demande les baguettes ou mange
il mange en un temps fini

Un philosophe peut acquérir ses baguettes si ses voisins ne mangent pas, sinon il se bloque

Un philosophe libère ses baguettes en notant qu'il pense

Un philosophe réveille éventuellement un philosophe voisin demandeur et au préalable note que ce dernier mange

Sémaphores privés
Le repas des philosophes

```
Contexte commun
Type statut=(pense, demande, mange);
statut état[5];tout i de 0 à 4, etat[i]=pense;
//initialement tous les philosophes pensent
Sémaphore mutex; E0(mutex,1);//accès aux variables
Sémaphore sempriv[5]; tout i de 0 à 4 E0(sempriv[i],0);
Processus philosophe i
Booleen OK;
  tant que vrai faire
   penser;
   prélude; // demande à manger
   manger;
   postlude; // finit de manger
  fait;
```

Sémaphores privés
Le repas des philosophes

Prélude

```
P(mutex);//accès aux variables de contrôle
 état[i]=demande;
 si (état[(i+1)%5]≠mange et état[(i-1)%5]≠mange)
 alors état[i]=mange;OK=vrai;
 sinon OK=faux; finsi;
V(mutex);
 si non OK alors P(sempriv[i]); finsi;
```

Sémaphores privés
Le repas des philosophes

```
Postlude

P(mutex);//accès aux variables de contrôle
 état[i]=pense;
//le philosophe i essaie de réveiller ses voisins
 si (état[(i+1)%5]=demande et état[(i+2)%5]≠mange)
 alors état[(i +1)%5]=mange;V(sempriv[(i+1)%5]);finsi;
 si (état[(i-1)%5]=demande et état[(i-2)%5]≠mange)
 alors état[(i-1)%5]=mange;V(sempriv[(i-1)%5]);finsi;
V(mutex);
```

La solution est sans interblocage, mais n'est pas équitable : risque de famine

Sémaphores privés
Lot de ressources banalisées

Un lot de N ressources banalisées est partagé par un ensemble de M processus; chaque ressource est en accès exclusif

Un processus peut demander X ressources $X \leq N$; les ressources sont allouées globalement

Si un processus détecte qu'il doit se bloquer par manque d'un nombre suffisant de ressources, il faut mémoriser sa requête dans une file d'attente qui est une donnée partagée

Lorsqu'un processus libère ses ressources, il doit réveiller éventuellement un ou plusieurs processus en consultant les requêtes de la file d'attente et en allouant le cas échéant le nombre de ressources demandées

Sémaphores privés
Lot de ressources banalisées

```
Contexte commun
 entier stock=N;//nombre de ressources disponibles
 file F de requete;
//une requête est un couple (i,X) i est le numero du
// processus et X le nombre de ressources demandées
 procedure ajouter(file F;entier i;entier X)
 //ajoute le processus i et sa demande X dans la file F
 procedure ôter(file F;entier i;entier X)
 //retire de la file la requête (i,X)
 procedure premier(file F;entier i;entier X)
 //lit la première requête de la file F;
 sémaphore mutexStock; E0(mutexStock,1);
 sémaphore sempriv[M]; tout i de 0 à M-1,E0(sempriv[i],0);
```

Sémaphores privés
Lot de ressources banalisées

```
Processus proc_i //i, 0 à N-1 identifiant du processus
 booleen OK;entier X;
 tant que vrai faire
  calculer X;
 //demande de X ressources
 P(mutexStock);
  si X≤stock alors stock=stock-X;OK=vrai;
  sinon
   ajouter(F,i,X);OK=faux;
  finsi;
 V(mutexStock);
 si non OK alors P(sempriv[i]);finsi;
 choix de X ressources //en exclusion mutuelle
 utilisation de X ressources
```

Sémaphores privés
Lot de ressources banalisées

```
//Restitution de X ressources par proc_i
Rendre les ressources;//en E.M.avant réveil de processus
Entier j,Y;
P(mutexStock);
stock=stock+X;
si non file-vide(F) alors
// on tente de satisfaire la première requête
 premier(F,j,Y);
 si Y≤stock alors //on peut satisfaire la requête
  stock=stock-Y;
  ôter(F,j,Y);//retrait de la requête de F
  V(sempriv[j]);//réveil du processus j
 finsi;
V(mutexStock);
```

Sémaphores privés
Lot de ressources banalisées

Variante réveil de plusieurs processus

```
//Restitution de X ressources par proc_i
Rendre les ressources;//en EM, avant réveil de processus
entier j,Y;
P(mutexStock);
stock=stock+X;
// on satisfait les requêtes dans l'ordre de la file
tant que non file-vide(F) faire
 premier(F,j,Y);
 si Y≤stock alors //on peut satisfaire la requête
  stock=stock-Y;
  ôter(F,j,Y);//retrait de la requête de F
  V(sempriv[j]);//réveil du processus j
 sinon break;finsi;//la requête ne peut pas être satisfaite
fait;
V(mutexStock);
```