Contrôle de concurrence par sémaphores

Rappel du concept de sémaphore

Définition (Dijkstra-1965)

Un sémaphore S est un objet partagé constitué de

- un entier **E** initialisé à une valeur ≥0
- une file d'attente **F** des processus bloqués

Un sémaphore est accessible uniquement par 3 primitives atomiques :

P(S) : Puis-je (Proberen)
contrôle d'autorisation + blocage éventuel du demandeur

V(S) : Vas-y (Verhogen)
ajout d'une autorisation + déblocage éventuel d'un demandeur

E0(S,I) : initialisation du sémaphore à I≥0 autorisations et file d'attente vide

Rappel du concept de sémaphore

Définition

```
Primitive P(sémaphore S) :
début
  S.E=S.E-1;//retrait d'une autorisation
  si S.E<0 alors bloquer le processus;
  placer son id dans la file;
fin
Primitive V(sémaphore S) :
début
  S.E=S.E+1;//ajout d'une autorisation
  si S.E≤0 alors //il y a au moins un processus bloqué
  choix et retrait d'un processus de F;
  réveil du processus;finsi;
fin
Primitive E0(sémaphore S,entier I) :
début
  S.E=I; S.F=vide;//I≥0
fin
```

Rappel du concept de sémaphore

Propriétés

- Un processus qui se bloque en exécutant P(S) termine son exécution de P(S)
- Un processus ne peut accéder à un sémaphore que par les primitives P(S) et V(S), l'initialisation E0(S,I) doit précéder tout accès au sémaphore
- Un seul processus peut exécuter P(S) ou V(S) à un instant donné
- S.E initialisé à I≥0 signifie qu'on peut exécuter I fois P(S) sans blocage
-S.E≥0 représente le nombre d'autorisations de franchissements de P(S) sans blocage, à un instant donné
-S.E<0 |S.E| représente le nombre de processus bloqués à un instant donné

Rappel du concept de sémaphore

Propriétés

Soient

NP(S) le nombre d'appels à P(S), NV(S) le nombre d'appels à V(S)

NF(S) le nombre de franchissements de P(S)

NBLOC(S) le nombre de processus bloqués

-S.E=I-NP(S)+NV(S)

-NBLOC(S)= max(0, -S.E)

-NF(S)=Min(NP(S), I+NV(S))

Exclusion mutuelle

Accès exclusif à une ressource critique ou à un ensemble de variables partagées par N processus s'exécutant en concurrence

```
Contexte commun ressource ou variables partagées
Sémaphore mutex; E0(mutex,1);
Processus Pi
Début
 tant que vrai faire
   P(mutex); //entrée en section critique
    Section_critique;
   V(mutex);//sortie de section critique
    Hors section critique;
Fin;
```

Exclusion mutuelle

Respect des propriétés

1- Accès exclusif

L'accès est autorisé dans P(mutex) si mutex.E ≥ 0,

soit $NP(mutex) \leq 1+NV(mutex)$

c'est-à-dire $NP(mutex)-NV(mutex) \leq 1$.

Il y a donc au plus un processus au plus en section critique

2-Pas d'interblocage actif : si aucun processus en SC

mutex.E=1

3-pas d'interblocage passif

La cohorte

N processus au plus coopèrent pour
-se répartir une tâche
-partager N ressources banalisées
Schéma d'exécution

```
Processus Pi
 tant que vrai faire
 entrée-groupe//contrôle l'accès au groupe
 groupe //participation au groupe
 sortie-groupe  // libère un accès
 hors groupe;
fait
```

La cohorte

```
Contexte commun sémaphore S_C; E0(S_C,N);
Processus P_i
 tant que vrai faire
 P(S_C);//entrée-groupe
 groupe //participation au groupe
 V(S_C) //sortie-groupe
 hors groupe;
fait
```

Cohorte : lot de ressources banalisées

Problématique

Un ensemble de processus se partagent un stock de N ressources banalisées, par exemple des pages mémoires, allouables dynamiquement une à une.

-un sémaphore S_C initialisé à N modélise le nombre d'autorisations d'accès au stock

-une structure de données partagée reflète l'état d'allocation de chaque ressource, on doit y accéder en exclusion mutuelle un sémaphore mutex initialisé à 1 contrôle la section critique

Cohorte : lot de ressources banalisées

Cas de demande d'une seule ressource

```
Contexte commun
sémaphore S_C, mutex; E0(S_C,N); E0(mutex,1);
booleen stock[N]; initialisé à faux;
Processus P_i
 entier j;
 tant que vrai faire
  //acquisition d'une autorisation;
  P(S_C);
  // choix d'une ressource
  P(mutex);
  j=1;
  tant que stock[j] faire j=j+1;fait
  stock[j]=vrai;
  V(mutex);
```

Cohorte : lot de ressources banalisées

Cas de demande d'une seule ressource(suite)

```
  Utilisation de la ressource j;
  //restitution de la ressource
  P(mutex); stock[j]=faux;V(mutex);
  V(S_C) //sortie-groupe
  hors groupe;
 fait
```

Cohorte : lot de ressources banalisées

Problème : Cas de demande de plusieurs ressources

```
Contexte commun
sémaphore S_C, mutex; E0(S_C,N); E0(mutex,1);
booleen stock[N] initialisé à faux;
Processus P_i
 tant que vrai faire
  //acquisition des autorisations;
   P(S_C); P(S_C);…
  // choix des ressources
  P(mutex);… V[mutex];
   //utilisation des ressources
  //restitution des ressources
  P(mutex);…V(mutex);
   V(S_C);V(S_C);… //sortie-groupe
  hors groupe;
 fait;
```

Risque d'interblocage!

Producteurs-Consommateurs

Spécification

Contexte commun **`tampon de N cases`**

```
producteur                    consommateur
 tant que vrai faire           tant que vrai faire
 produire un message;          retirer un message;
 déposer un message;           consommer le message;
 fait                          fait
```

Objectif

Asservir la vitesse moyenne de production à la vitesse moyenne de consommation en ralentissant le moins possible le producteur

Producteurs-Consommateurs

Hypothèses

1- vitesses relatives quelconques
2- tampon de taille fixe, 1 case=1message, vide initialement
3- tout message est déposé et retiré une fois et une seule
4- le dépôt et le retrait se font en un temps fini

Producteurs-Consommateurs

Propriétés de la solution

1- exclusion mutuelle d'accès aux messages
2- le producteur attend si le tampon est plein, il est réveillé dès que le tampon n'est plus plein
3- le consommateur attend si le tampon est vide, il est réveillé dès que le tampon n'est plus vide
4- pas d'interblocage

Producteur-Consommateur

Schéma d'exécution

```
Contexte commun tampon de N cases vide;
Processus producteur
  tant que vrai faire
    produire un message;
    si tampon plein alors se bloquer;
    déposer un message;
    si consommateur bloqué alors réveil du consommateur;
   fait;
Processus consommateur
tantque vrai faire
   si tampon vide alors se bloquer;
   retirer un message;
   si producteur bloqué alors réveil du producteur;
   consommer le message;
fait;
```

Producteur-Consommateur

Schéma de synchronisation

- un sémaphore Nvide initialisé à N pour le contrôle du nombre de cases vides
  le producteur exécute P(Nvide) pour réserver une case
  le consommateur exécute V(Nvide) pour libérer une case
- un sémaphore Nplein initialisé à 0 pour le contrôle du nombre de messages (cases pleines)
  le producteur exécute V(Nplein) pour signaler au consommateur 1 nouveau message dans le tampon
  le consommateur exécute P(Nplein) pour attendre un message

Producteur-Consommateur

Schéma de synchronisation

Contexte commun **`tampon de N cases vide;`**

`Sémaphore Nplein,Nvide;`

`E0(Nplein,0);E0(Nvide,N)`

```
Processus producteur
message m;
  tant que vrai faire
    produire un message(m);
    P(Nvide);
    déposer(m);
    V(Nplein);
  fait;
```

```
Processus consommateur
message m;
  tant que vrai faire
    P(Nplein);
    retirer(m);
    V(Nvide);
    consommer(m);
  fait;
```

Producteur-Consommateur

Respect des propriétés

2- le producteur attend si le tampon est plein : P(Nvide)
il est réveillé dès que le tampon n'est plus plein : V(Nvide)
3- le consommateur attend si le tampon est vide : P(Nplein)
il est réveillé dès que le tampon n'est plus vide : V(Nplein)
4- pas d'interblocage : le producteur et le consommateur ne peuvent être bloqués simultanément, respectivement sur P(Nvide) et P(Nplein)

Producteur-Consommateur

Gestion du tampon

-Le tampon est géré circulairement, implanté par un tableau

-On définit un indice de production **queue** et un indice de consommation **tête**

-les messages doivent être consommés dans l'ordre de leur production, **tête** et **queue** sont initialisées à la même valeur

Contexte commun

```
Sémaphore Nplein,Nvide;
E0(Nplein,0);E0(Nvide,N)
Message tampon[N];
```

Producteur-Consommateur

Schéma avec gestion du tampon

```
Processus producteur
message m;
entier queue=0;
  tant que vrai faire
    produire un message(m);
    P(Nvide);
    tampon[queue]=m;//déposer(m);
    V(Nplein);
    queue=(queue+1)%N;
   fait;
```

Producteur-Consommateur

Schéma avec gestion du tampon

```
Processus consommateur
message m;
entier tête=0;
  tant que vrai faire
    P(Nplein);
    m=tampon[tête];//retirer(m);
    V(Nvide);
    tête=(tête+1)%N;
   fait
```

Producteur-Consommateur

Propriété

1- exclusion mutuelle d'accès aux messages

Si le producteur et le consommateur "travaillent" sur la même case c'est que tête=queue

tête=queue tampon vide alors le producteur est bloqué

tête=queue tampon plein alors le consommateur est bloqué

Producteurs-Consommateurs

Producteurs consommateurs multiples

Le schéma de synchronisation assure une exclusion mutuelle entre producteurs et consommateurs

- plusieurs producteurs ne doivent pas déposer dans la même case, l'indice de production queue est partagé

⇒ Exclusion mutuelle entre producteurs

- plusieurs consommateurs ne doivent pas prélever le même message, l'indice de consommation tête est partagé

=>exclusion mutuelle entre consommateurs

Producteurs-Consommateurs

Producteurs consommateurs multiples

Contexte commun

```
Sémaphore Nplein,Nvide;
E0(Nplein,0);E0(Nvide,N);
Sémaphore mutexProd,mutexCons;
E0(mutexProd,1); E0(mutexCons,1);
Entier tête,queue=0;
Message tampon[N];
```

Producteurs-Consommateurs

Producteurs consommateurs multiples

```
Processus producteur
message m;
  tant que vrai faire
    produire un message(m);
    P(Nvide);
    P(mutexProd);
    tampon[queue]=m;//déposer(m);
    queue=(queue+1)%N;
    V(mutexProd)
    V(Nplein);
    fait;
```

Producteurs-Consommateurs

Producteurs consommateurs multiples

```
Processus consommateur
message m;
tant que vrai faire
    P(Nplein);
    P(mutexCons);
     m=tampon[tête];//retirer(m);
     tête=(tête+1)%N;
     V(mutexCons);
     V(Nvide);
   fait;
```

Lecteurs-Rédacteurs

Compétition d'accès à un ensemble de données par un ensemble de processus

- lecteurs accès seulement en lecture
- rédacteurs accès en lecture et écriture

Objectif

Garantir la cohérence des données

Spécification

- plusieurs lectures simultanément
- les écritures sont en exclusion mutuelle

Lecteurs-Rédacteurs

Hypothèse complémentaire

tout processus termine sa lecture ou son écriture au bout d'un temps fini

Spécification de comportement

- Les lectures sont concurrentes et cohérentes, les écritures sont en exclusion mutuelle
- Priorité aux lecteurs, équité entre lecteurs et rédacteurs, service à l'ancienneté

Lecteurs-Rédacteurs

Schéma d'exécution

```
Processus unLecteur
  tant que vrai faire
   début-lecture;//contrôle l'accès en lecture
    lecture;
   fin-lecture;//libère un accès en lecture ou
      écriture
  fait;
Processus unRédacteur
  tant que vrai faire
   début-écriture;//contrôle l'accès en écriture
     écriture;
   fin-écriture;//libère un accès en lecture ou
                écriture
   fait;
```

Lecteurs-Rédacteurs

Schéma de solution

- Les rédacteurs doivent écrire en exclusion mutuelle avec un groupe de lecteurs et avec les autres rédacteurs

soit **mutex_A** un sémaphore d'exclusion mutuelle

Un rédacteur exécute **P(mutex_A)** pour obtenir l'accès

Le premier lecteur d'un groupe exécute **P(mutex_A)** pour obtenir l'accès pour le groupe

Une variable partagée **NL** permet de compter le nombre de lecteurs : s'il y a déjà un lecteur alors autoriser un nouveau lecteur

Un sémaphore d'exclusion mutuelle **mutex_NL** permet de gérer l'accès à NL et de bloquer tout lecteur non autorisé

Lecteurs-Rédacteurs

Schéma de solution

- Un rédacteur bloqué est réveillé par un autre rédacteur ou par le dernier lecteur qui a fini de lire par **V(mutex_A)**
- Un lecteur bloqué sur **P(mutex_A)** est réveillé par un rédacteur, s'il est bloqué par **P(mutex_L)** il est réveillé par le premier lecteur en exécutant **V(mutex_NL)**

Lecteurs-Rédacteurs

Schéma de solution

```
Contexte commun entier NL=0;Sémaphore mutex_A, mutex_NL;
E0(mutex_A,1); E0(mutex_NL,1);
```

```
Processus unLecteur;
tant que vrai faire
//début lecture
 P(mutex_NL);
 NL=NL+1;
 si NL=1 alors P(mutex_A);
 finsi;
 V(mutex_NL);
 lectures;
//fin lecture
 P(mutex_NL);
 NL=NL-1;
 si NL=0 alors V(mutex_A);
V(mutex_NL; fait;
```

```
Processus unRédacteur;
tant que vrai faire
//début écriture
 P(mutex_A);
 écritures;
//fin écriture
V(mutex_A);
fait;
```

Lecteurs-Rédacteurs

Schéma de solution

Remarque

- la solution donne une priorité aux lecteurs si un lecteur à partir du moment où une lecture est déjà en cours : **famine possible des rédacteurs**

Égalité entre les classes de lecteurs et de rédacteurs

- les requêtes d'accès de tous les processus sont mises dans une seule file d'attente sous le contrôle d'un sémaphore d'exclusion mutuelle FIFO

Lecteurs-Rédacteurs

Schéma de solution égalité de priorité

```
Contexte commun entier NL=0;sémaphore mutex_A,mutex_NL;
E0(mutex_A,1); E0(mutex_NL,1);
sémaphore FIFO; E0(FIFO,1);

Processus unLecteur;
tant que vrai faire
//début lecture
 P(FIFO);
 P(mutex_NL);
 NL=NL+1;
 si NL=1 alors P(mutex_A);
 finsi;
 V(mutex_NL);
 V(FIFO);
  lectures;
//fin lecture
 P(mutex_NL);
 NL=NL-1;
 si NL=0 alors V(mutex_A);
 V(mutex_NL; fait;
```

Lecteurs-Rédacteurs

Schéma de solution égalité de priorités

```
Processus unRédacteur;
tant que vrai faire
//début écriture
   P(FIFO);
   P(mutex_A);
   V(FIFO);
  écritures;
//fin écriture
   V(mutex_A);
fait;
```

Le repas des philosophes

Hypothèses

• chaque assiette (philosophe) a une place fixe

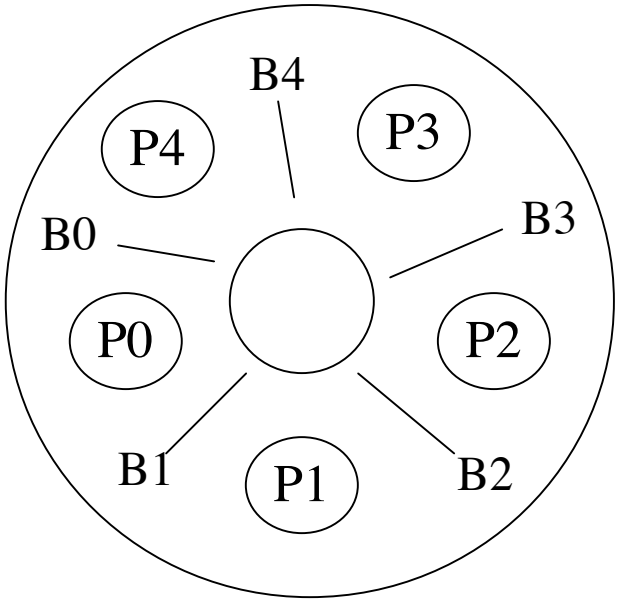
• chaque baguette a une place fixe

• accès à chaque baguette en exclusion mutuelle

Propriétés

• pas d'interblocage

• pas de famine



Le repas des philosophes

Une première solution prendre les baguettes une par une, la sienne; puis celle de droite

```
Contexte commun
Sémaphore baguette[n]; tout i, E0(baguette[i],1);

Processus Philosophe i // i identifiant 0..n-1
tant que vrai faire
  penser;
  P(baguette[i]);//prendre la baguette i;
  P(baguette[(i+1)%n])//prendre la baguette de droite;
  manger;
  V(baguette[i]);// rendre la baguette i;
  V(baguette[(i+1)%n])//rendre la baguette de droite;
fait;
```

Interblocage possible!

Le repas des philosophes

Une solution sans interblocage : autoriser seulement n-1 philosophes à demander à manger, il n'y a que n-1 chaises!

```
Contexte commun
sémaphore baguette[n]; tout i, E0(baguette[i],1);
sémaphore chaise; E0(chaise,n-1))
Processus Philosophe i // i identifiant 0..n-1
tant que vrai faire
  penser;
 P(chaise);
 P(baguette[i]);//prendre la baguette i;
 P(baguette[(i+1)%n])//prendre la baguette de droite;
  manger;
 V(baguette[i]);// rendre la baguette i;
 V(baguette[(i+1)%n]);//rendre la baguette de droite;
 V(chaise);
fait;
```