Synchronisation

Xavier Merrheim

Problématique

- Lorsque deux tâches (processus ou threads) accèdent à une même zone mémoire, elles peuvent interagir de manière indésirable, même si chaque tâche, prise indiduellement, se comporte correctement.

Exemple

- On imagine qu'il y a dans un variable en RAM la valeur 1000
- Un processus P1 veut augmenter cette valeur de 300
- P2 veut la diminuer de 500
- On aimerait que lorsque les 2 programmes sont terminés, il y ait 800 dans compte

Traduction de P1 en pseudo-assembleur

- Mettre dans R1 la valeur de compte
- Mettre dans R2 la constante 300
- Ajouter R2 à R1
- Mettre R1 dans compte

Traduction de P2 en pseudo-assembleur

- Mettre dans R1 la valeur de compte
- Mettre dans R2 la constante 500
- Soustraire R2 à R1
- Mettre R1 dans compte

Définition

- Une ressource est dite ressource critique lorsque des accès concurrents à cette ressources la mettre dans un état incohérent.
- On parle aussi de situation de compétition (race condition) pour décrire une situation dont l'issue dépend de l'ordre dans lequel les opérations sont effectuées.
- Une section critique est une section de programme manipulant une ressource critique.

Ordonnancement

- L'ordonnanceur peut à tout moment interrompre un processus et lancer un autre processus.
- Lors d'un changement de comptexte on sauvegarde la valeur des registres
- On aimerait que le réésultat final soit indépendant des choix de l'ordonnanceur.

Exécution 1

- Mettre dans R1 la valeur de compte
- Mettre dans R2 la contante 300
- Ajoouter R2 à R1
- Mettre R1 dans compte
  - Mettre dans R1 la valeur de compte
  - Mettre dans R2 la contante 500
  - Soustraire R2 à R1
  - Mettre R1 dans compte

Compte	R1	R2
1000		
	1000	
	1000	300
	1300	300
1300	1300	300
	1300	300
	1300	500
	800	500
800	800	500

Exécution 2

		Compte	R1	R2
		1000		
• Mettre dans R1 la valeur de compte		1000	1000	
• Mettre dans R2 la contante 300		1000	1000	300
• Ajouter R2 à R1		1000	**1300**	**300**
sauvegarde du contexte	• Mettre dans R1 la valeur de compte	1000	1000	300
	• Mettre dans R2 la contante 500	1000	1000	500
	• Soustraire R2 à R1	1000	500	500
	• Mettre R1 dans compte	500	500	500
Restauration du contexte		500	**1300**	**300**
• Mettre R1 dans compte		1300	1300	300

Problème

- La valeur finale de compte dépepend donc des choix de l'ordonnanceur !
- Nos 2 programmes sont de plus parfaitement corrects !
- Il faut encadrer l'accès aux ressources partagées entre 2 processus.

Exclusion mutuelle

- Une section de programme est dite atomique lorsqu'elle ne peut pas être interrompue par un autre processus manipulant les mêmes ressources critiques.

  → c'est donc une atomicité relative à la ressource
- Un mécanisme d'exclusion mutuelle sert à assurer que 2 processus ne peuvent être simultanement en section critique pour la même ressource

  → en anglais : mutual exclusion, ou mutex

Ecriture des programmes de P1 et P2

Programme de P1

```
Entrer_sc()
compte=compte+300
Sortir_sc()
```

Programme de P2

```
Entrer_sc()
compte=compte-300
Sortir_sc()
```

Exclusion mutuelle

- Exclusion : deux tâches ne doivent pas se trouver en même temps en SC
- Progression : une tâche doit pouvoir entrer en SC si aucune autre ne s'y trouve
- Équité : une tâche ne devrait pas attendre indéfiniment pour entrer SC
- L'exclusion mutuelle doit fonctionner dans un contexte multi-cœurs

Une mauvaise solution

- int verrou = 0;
- void entrer_SC() { while (verrou) {}
  verrou = 1; }
- void sortir_SC() { verrou = 0; }

- Problème : attente active et en plus ça ne marche pas. Deux processus peuvent se retrouver en section critique.

Une bonne solution avec sémaphore

- On associe à la ressource un jeton, que les processus peuvent prendre et reposer
- Seul le processus possédant le jeton devrait manipuler la ressource
- Donc, si un processus souhaite manipuler la ressource et que le jeton est pris, il doit d'abord attendre que le jeton redevienne disponible
- Utilisation d'un sémaphore initialisé à 1

Solution avec sémaphore

- **Programme P1**

```
int compte;                          /* commun */
sem_t mutex_compte;                  /* commun */
status = sem_init(&mutex_compte, 1, 1);
status = sem_wait(&mutex_compte);
compte += 1000;
status = sem_post(&mutex_compte);
```

- **Programme P2**

```
status = sem_wait(&mutex_compte);
compte -= 500;
status = sem_post(&mutex_compte);
```

Remarque

- Les sections critiques sont un mal nécessaire:
  - un mal, parce qu'elles empêches le parallélisme qu'on a eu tant de mal à mettre en place… elles réduisent donc les performances.
  - nécessaire, parce qu'elles garantissent l'intégrité des ressources critiques
- Conséquence : les éviter lorsqu'on le peut mais les mettre quand elles sont indispensables.

Interblocage

- Un inter-blocage (deadlock) est une situation où plusieurs tâches sont bloquées car chacune attend un événement que doit produite une autre.
- Un exemple classique avec 2 ressources critiques et qui se produit lorsque deux tâches attendent chacune un « jeton » détenu par l'autre :

Programme P1
```
sem_wait(&mutex_A);
  utilise(A);
sem_wait(&mutex_B);
    utilise(A, B);
sem_post(&mutex_B);
sem_post(&mutex_A);
```

Programme P2
```
sem_wait(&mutex_B);
  utilise(B);
sem_wait(&mutex_A);
    utilise(A, B);
sem_post(&mutex_A);
sem_post(&mutex_B);
```


Une solution possible

Programme P1
```
sem_wait(&mutex_A);
utilise(A);
sem_wait(&mutex_B);
utilise(A, B);
sem_post(&mutex_B);
sem_post(&mutex_A);
```

Programme P2
```
sem_wait(&mutex_B);
utilise(B);
sem_post(&mutex_B);
sem_wait(&mutex_A);
sem_wait(&mutex_B);
utilise(A, B);
sem_post(&mutex_B);
sem_post(&mutex_A);
```


Le problème des producteurs/consommateurs

- Un ensemble de processus, divisé en deux catégories, partage une zone mémoire.
- Les premiers (producteurs) remplissent la mémoire partagée, avec des éléments.
  - la mémoire ne peut contenir qu'un nombre d'éléments limité et connu à l'avance
- Les seconds (consommateurs) utilisent ces éléments et les retirent de la mémoire.
- Exemple : file d'impression

Problème

- Un producteur doit se bloquer lorsque la mémoire partagée est pleine
- Un consommateur doit se bloquer lorsque la mémoire partagée est vide

Principe

- Utilisation de deux sémaphores, implémentant les deux critères de blocage
  - l'un représente le nombre de cases libres
  - l'autre représente le nombre de cases occupées
  - Analogie : deux piles de jetons, avec une quantité de jetons constante
- les jetons passent d'une pile dans une autre lorsqu'une case change d'état
- Si les opérations sur la zone mémoire partagée ne sont pas atomiques, il faut les protéger par une section critique → troisième sémaphore

Algorithme

```
elt_t tab[N]
sem_t libres, occupes, mutex;
sem_init(&libres,  1, N);
sem_init(&occupes, 1, 0);
sem_init(&mutex,   1, 1);
```


Algorithme : producteur

```
void producteur() {
elt_t e = produire();
sem_wait(&libres);
sem_wait(&mutex);
ajouter(e, tab);
sem_post(&mutex);
sem_post(&occupes);
}
```

Algorithme consommateur

```
void consommateur() {
sem_wait(&occupes);
sem_wait(&mutex);
elt_t e = retirer(tab);
sem_post(&mutex);
sem_post(&libres);
consommer(e);
}
```


Problème lecteurs/rédacteurs

- Un ensemble de processus, divisé en deux catégories, partage une zone mémoire.
- Certains processus (les lecteurs) font des accès en lecture seule à cette zone.
- D'autres processus (les rédacteurs) modifient le contenu de cette zone.

Principe de solution

- On protège la mémoire partagée par une exclusion mutuelle, mais…
- les lecteurs n'ont besoin de cette exclusion mutuelle que si aucun autre lecteur n'utilise la mémoire ;
- pour le vérifier, ils utilisent un compteur (nombre de lecteurs en train d'utiliser la zone), lui aussi protégé par une exclusion mutuelle.

Solution

```
int c = 0;
sem_t mutex_m, mutex_c;
sem_init(&mutex_m,  1, 1);
sem_init(&mutex_c,  1, 1);
```

Solution : redacteur()

```
void redacteur(e) {
sem_wait(&mutex_m);
ecrire(e, tab);
sem_post(&mutex_m);
}
```


Solution lecteur

```
void lecteur() {            //
sem_wait(&mutex_c);
if (c == 0)sem_wait(&mutex_m);
c = c+1;
sem_post(&mutex_c);
lire();
sem_wait(&mutex_c);
if (c == 1)sem_post(&mutex_m);
c=c-1;
sem_post(&mutex_c);
}
```


- Le lecteur qui prend le jeton de mutex_m n'est pas forcément celui qui le rend
- On note que les lecteurs peuvent parfois se bloquer dans la section critique du compteur c→ Cela peut-il créer un inter-blocage ?
- L'algorithme proposé fonctionne, mais n'assure pas l'équité : un rédacteur peut attendre indéfiniment avant d'avoir accès à la mémoire (famine) → Comment le modifier pour assurer l'équité ?

Conclusion

- Les problèmes de synchronisation sont nombreux et complexes.
- Nous avons étudié des solutions avec des sémaphores.
- Vous devez connaitre :
  - La solution pour créer une section critique
  - Le problème d'interblocage
  - La solution du problème des lecteurs/rédacteurs
  - La solution du problèmes des producteurs consommateurs.

Problème

- Lorsqu'un rédacteur accède à la mémoire partagée, aucune autre processus (qu'il soit lecteur ou rédacteur) ne doit y avoir accès.

  → problème d'exclusion mutuelle classique

- En revanche, les lecteurs peuvent être plusieurs à utiliser la zone en même temps, cela ne pose pas de problème.

Question 1

- Qu'est ce qu'une ressource critique et une section critique ?

Question 2

- Qu'est-ce qu'une section de programme atomique ?

Question 3

- Qu'est-ce qu'une exclusion mutuelle ?
- Un mécanisme d'exclusion mutuelle sert à assurer que 2 processus ne peuvent être simultanement en section critique pour la même ressource

Question 4

- Quels sont les 3 principes que doit respecter une exclusion mutuelle.
-

Question 5

- Quels sont les 3 appels système manipulant les sémaphore ?

Question 6

- Ecrire un programme P1 qui incrémente de 500 une ressource critique toto partagée par plusieurs processus.

Question 7

- Qu'est-ce qu'un interblocage ?

Question 8

- Donnez un exemple d'interblocage

Question 9

- Qu'appelle-t-on le problème des lecteurs rédacteurs ?

Question 10

- Proposez une solution à ce problème

Question 1

- Qu'est ce qu'une ressource critique et une section critique ?
- Une ressource est dite ressource critique lorsque des accès concurrents à cette ressources peuvent résulter dans un état incohérent.
  Une section critique est une section de programme manipulant une ressource critique.

Question 2

- Qu'est-ce qu'une section de programme atomique ?
- Une section de programme est dite atomique lorsqu'elle ne peut pas être interrompue par un autre processus manipulant les mêmes ressources critiques.

Question 3

- Qu'est-ce qu'une exclusion mutuelle ?
-

Question 4

- Quels sont les 3 principes que doit respecter une exclusion mutuelle.
- Exclusion : deux tâches ne doivent pas se trouver en même temps en SC

  Progression : une tâche doit pouvoir entrer en SC si aucune autre ne s'y trouve

  Équité : une tâche ne devrait pas attendre indéfiniment pour entrer SC

Question 5

- Quels sont les 3 appels système manipulant les sémaphore ?
- sem_init
  sem_wait
  sem_post

Question 6

- Ecrire un programme P1 qui incrémente de 500 une ressource critique toto partagée par plusieurs processus.
- int compte;

  sem_t mutex_compte;

  status = sem_init(&mutex_compte, 1, 1);

  status = sem_wait(&mutex_compte);

  compte += 1000;

  status = sem_post(&mutex_compte);

Question 7

- Qu'est-ce qu'un interblocage ?
- Un inter-blocage (deadlock) est une situation où plusieurs tâches sont bloquées car chacune attend un événement que doit produite une autre.

Question 8

- Donnez un exemple d'interblocage
- Programme P1
  sem_wait(&mutex_A);
  utilise(A);
  sem_wait(&mutex_B);
  utilise(A, B);
  sem_post(&mutex_B);
  sem_post(&mutex_A);

  Programme P2
  sem_wait(&mutex_B);
  utilise(B);
  sem_wait(&mutex_A);
  utilise(A, B);
  sem_post(&mutex_A);
  sem_post(&mutex_B);

Question 9

- Qu'appelle-t-on le problème des lecteurs rédacteurs ?
- Un ensemble de processus, divisé en deux catégories, partage une zone mémoire. Certains processus (les lecteurs) font des accès en lecture seule à cette zone. D'autres processus (les rédacteurs) modifient le contenu de cette zone.

Question 10

- Proposez une solution à ce problème
- Voir cours