Synchronisation des Processus

PLAN

- Rappel de notion de processus
- Spécification du problème
- Section Critique (SC)
- Exclusion Mutuelle
  - Principe
  - Propriétés
- Réalisation d'exclusion Mutuelle
  - Matérielle
  - Logicielle;

Rappel de notion de processus

- Processus?
- Etats d'un processus
- Contexte d'un processus
- Classes de processus
  - Processus indépendants
  - Processus coopérants
  - Processus concurrents
- Création et destruction de processus

Spécification du Problème

- Machines monoprocesseurs ou Multi-processeurs;
- Processus s'exécutent sur une machine mono/ multi Processeurs avec mémoire partagée;
- Partager des variables:
  - volontairement: coopérer pour traiter un Problème
  - involontairement: se partager des ressources;

Problème de synchronisation: Exemple

- Le partage de variables sans précaution particulière peut conduire à des résultats erronés:

Processus Crédit	Processus Débit
a: Compte:=Compte + C	b: Compte:= compte - D

Hypothèses d'exécution

- L'évolution dans l'exécution de chaque processus est à priori indépendante;
- Le délai entre deux instructions d'un processus est non nul, mais fini ;
- Deux accès à une même case mémoire ne peuvent être simultanés ;
- Les registres sont sauvegardés et restaurés à chaque commutation

Hypothèses d ’exécution: exemple

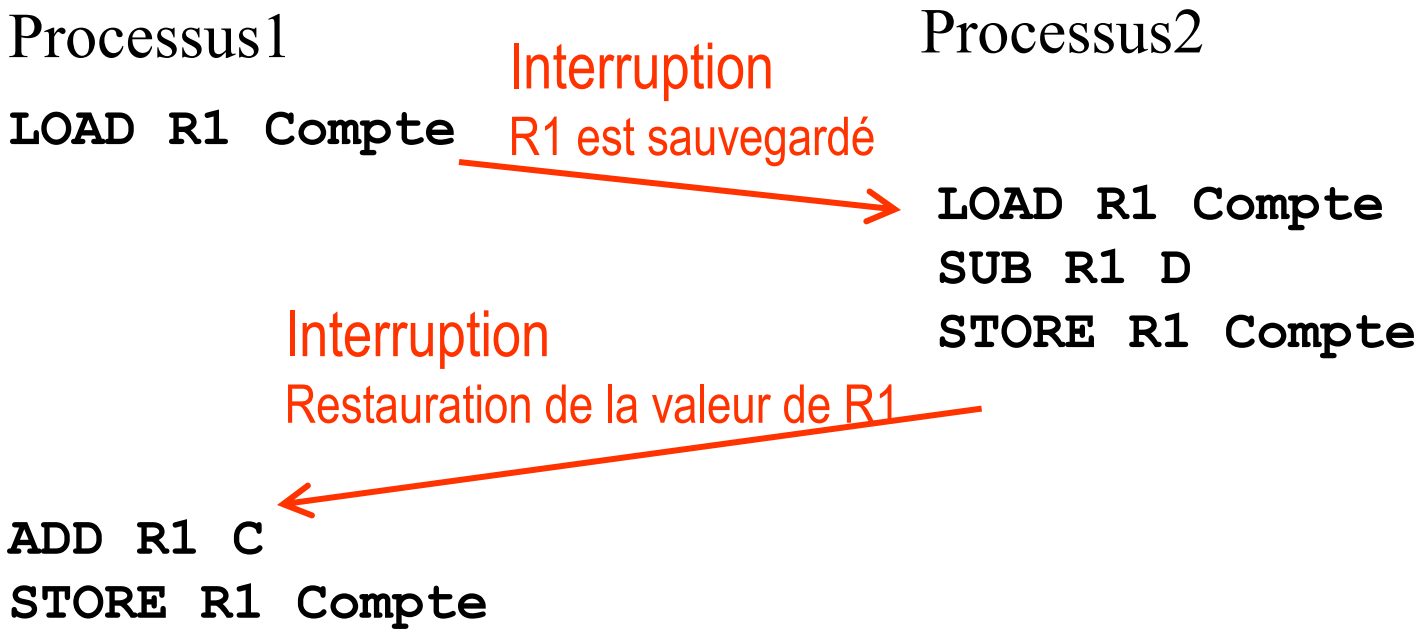
Processus1	Processus2
a1: LOAD R1 Compte a2: ADD R1 C a3: STORE R1 Compte	b1: LOAD R1 Compte b2: SUB R1 D b3: STORE R1 Compte

Quelle sera la valeur de compte si l’ordre d’exécution est le suivant ?
a1< a2 < a3 < b1 < b2 < b3

Quelle sera la valeur de compte si l’ordre d’exécution est le suivant ?
a1< b1 < a2 < a3 < b2 < b3

Quelle sera la valeur de compte si l’ordre d’exécution est le suivant ?
a1< b1 < a2 < b2 < b3 < a3

Exemple d'exécution: a1< b1< b2< b3< a2< a3
Valeurs des variables: Compte = 100; C= 20; D= 30
Valeur de compte=?



Sections critiques(SC): Définition

- Section Critique = ensemble de suites d'instructions qui peuvent produire des résultats erronés lorsqu'elles sont exécutées simultanément par des processus différents.
- L 'exécution de deux SC appartenant à des ensembles différents et ne partagent pas de variables ne pose aucun problème.

Détermination des SC

- L ’existence implique l’utilisation de variables partagées, mais l ’inverse n ’est pas vrai;
- Pratiquement les SC doivent être détectées par les concepteurs de programmes;
- Dès qu’il y a des variables partagées, il y a forte chance de se retrouver en présence de SC.

Exclusion Mutuelle: Principe(1)

- Les SC doivent être exécutés en Exclusion Mutuelle:
  - une SC ne peut être commencée que si aucune autre SC du même ensemble n'est en cours d 'exécution;
- Avant d 'exécuter une SC, un processus doit s'assurer qu'aucun autre processus n 'est en train d'exécuter une SC du même ensemble.

Exclusion Mutuelle: Principe(2)

- Dans le cas contraire, il devra pas progresser, tant que l'autre processus n'aura pas terminé sa SC;

- Nécessité de définir un protocole d 'entrée en SC et un protocole de sortie de SC

Protocole d ’entrée/sortie en SC

- *Protocole d ’entrée en SC* (prologue): ensemble d ’instructions qui permet cette vérification et la non progression éventuelle;
- *Protocole de sortie de SC* (épilogue): ensemble d’instructions qui permet à un processus ayant terminé sa SC d ’avertir d ’autres processus en attente que la ressource soit libre.

Structure des processus

Début

Section non Critique

prologue

SC

épilogue

Section non critique

Fin.

Propriétés de l'exclusion Mutuelle

1. Un seul processus en SC;
2. Un processus qui veut entrer en SC ne doit pas attendre qu'un autre processus passe avant lui pour avoir le droit.
3. Un processus désirant entrer en SC y entre au bout d 'un temps fini; pas de privation d'y entrer vis à vis d 'un processus

Exclusion Mutuelle

- L ’exclusion Mutuelle n’est pas garantie si:
  - a) un processus peut entrer en SC alors qu’un autre s’y trouve déjà;
  - b) un processus désirant entrer en SC ne peut pas y entrer alors qu’il n’y a aucun processus en SC;
  - c) un processus désirant entrer en SC n ’y entrera jamais car il sera jamais sélectionné lorsqu’il est en concurrence avec d ’autres processus

Réalisation d'exclusion Mutuelle

- Solutions logicielles : attente active (Dekker, Peterson,), attente passive (Dijkstra);

- Solutions Matérielles:
  - Monoprocesseurs: masquage d 'interruptions;
  - Multiprocesseurs: instruction indivisible.

Réalisation d'exclusion Mutuelle: solutions logicielles

Solution naïve: résoudre le problème de partage de variables par d'autre variables:

1ère tentative

Var barrière : (fermée, ouverte) % variable partagée %
Barrière := ouverte % initialement la section critique est libre

Processus P1
Test: ***Tantque*** barrière = fermée ***faire***
allera test
Fintantque
barrière := fermée;
< section critique >;
barrière := ouverte;

Processus P2

Test: ***Tantque*** barrière = fermée ***faire***
allera test
Fintantque
barrière := fermée;
< section critique >;
barrière := ouverte;

2ème tentative

Var Tour = 1..2 ; % variable partagée %
Tour := 1 ; % initialisation %

" Processus1"

Test: ***Tantque*** tour=2 ***faire***
allera test
Fintantque
< section critique >;
Tour := 2;

" Processus2 "

Test: ***Tantque*** tour=1 ***faire***
allera test
Fintantque
< section critique >;
Tour := 1;

Exclusion Mutuelle: Algorithmes de Dekker

- Solutions pour deux processus;
- Chaque processus boucle indéfiniment sur l'exécution de la section critique;

Solution correcte de Dekker

```
Var  process1, process2 :(interieur, exterieur);
        tour  : 1..2;
  process1:= exterieur;
  process2:= exterieur;
```

Processus P1

```
    ----------
process1:= interieur;
tour:= 2;
test: tantque process2 = interieur et tour = 2 faire
        aller à test
       Fintantque
 <section critique>
process1:= exterieur;
  ----------
```

Processus P2

```
    ----------
process2 := interieur;
tour := 1;
test: tantque process1 = interieur et tour = 1 faire
            allerà test
      Fintantque
<section critique>;
process2:=exterieur;
  ----------
```

Exclusion Mutuelle: Algorithmes de Peterson

- Solution symétrique pour N processus (généralisation de la solution de Dekker;

- L'interblocage est évité grâce à l'utilisation d'une variable partagée *Tour;*
  - la variable tour est utilisée de manière absolue et non relative;

Exclusion Mutuelle: solutions matérielles sur monoprocesseur

- Solution brutale: masquage d ’interruptions
  - On empêche les commutations de processus qui pourraient violer l’exclusion Mutuelle des SC;
  - donc Seule l’interruption générée par la fin du quantum de temps nous intéresse

Exclusion mutuelle: solution brutale

- Les IT restent masquées pendant toute la SC, d'où risque de perte d'IT ou de retard de traitement.
- Une SC avec while(1) bloque tout le système
- Les sytèmes ne permettent pas à tout le monde de masquer n'importe comment les IT.

Exclusion Mutuelle:solution monoprocesseur(1)

```
Processus 1 (P1)

Masquage IT
LOAD R1 Compte
ADD R1 C
STORE R1 Compte
Démasquer IT
```

Processeur
Exécute P1

```
Processus 2

Masquage IT
LOAD R1 Compte
SUB R1 D
STORE R1 Compte
Démasquer IT
```

Processus bloqué

Exclusion Mutuelle:solution monoprocesseur (2)

- Avantage: masquage des interruptions pendant la modification de *occupé*;
- Inconvénient:
  - Le masquage des IT n'est accessible qu'aux programmeurs privilégies pour des raisons de fiabilité : exemple super_ utilisateur.
  - Cette solution ne fonctionne pas sur des Multiprocesseurs.

Exclusion Mutuelle:solution Multiprocesseur (1)

- Instruction indivisible: réalisée une seule fois par le matériel:
  - Test_and_Set(TAS) : instruction indivisible de consultation et de modification d'un mot mémoire.

Soit C une variable globale indiquant l'état (ou l'occupation) d'une section critique, c'est-à-dire: C = 1 ==> section critique occupée, et C = 0 ==> section critique libre. L'algorithme de fonctionnement de TAS est le suivant:

Exclusion Mutuelle:solution Multiprocesseur (2)

- TAS(C)
- ***begin***
- < verrouiller l'accès à C >;
- lire C
- ***if*** C = 0 ***then*** ***begin***
- C := 1
- co := co + 2 % co = compteur ordinal % (1)
- ***end*** % ou compteur d'instructions %
- ***else*** co := co + 1 % (2) %
- ***endif***
- < libérer l'accès à C >
- ***end***

Exclusion Mutuelle:solution Multiprocesseur (3)

- Par exemple, la programmation de l'exemple précédent peut être:

Processus Debit	Processus Credit
.	.
.	.
.	.
Test: TAS(C);	Test: TAS(C);
Aller a Test;	aller a Test;
a1: Rla := Compte ;	b1: Rlb := Compte ;
a2: Rla := Rla + C ;	b1: Rlb := Rlb - D ;
a3: Compte := Rla ;	b3: Compte := Rlb ;
C := 0;	C := 0;

- **Inconvénient: attente active.**

Verrou

Initialement V est nul et f(V) est vide;

Verrouiller*(V)**: ***debut
si V = 0 ***alors*** V:=1;
sinon debut
mettre le processus appelant P dans f(V)
état(P) := bloqué; % voir figure 1. %
fin
finsi
fin

Deverrouiller*(V)**: ***debut
si f(V) ≠ vide ***alors debut***
sortir Q de f(V);
état(Q) := prêt;
fin
sinon V := 0;
finsi
fin

- **Avantage : Attente passive**

Les sémaphores

- Introduit par Dijkstra en 1965 pour résoudre le problème d'exclusion mutuelle.
- Permettent l'utilisation de m ressources identiques (exple imprimantes) par n processus.
- Un sémaphore est une structure contenant deux champs :
  - Struct {n : entier ;
    en_attente : file de processus
    }

Sémaphores: Définition(1)

- Un sémaphore est une variable globale protégée, c'est à dire on peut y accéder qu'au moyen des trois procédures :
  - initialiser le sémaphore S à une certaine valeur x;
  - P(S) ; Peut -on passer ?/peut-on continuer?
  - V(S) ; libérer?/vas y?

Sémaphores: définition (2)

- Un *sémaphore binaire* est un sémaphore dont la valeur peut prendre que deux valeurs positives possibles : en générale 1 et 0.
- Un *sémaphore de comptage* : la valeur peut prendre plus de deux valeurs positives possibles.
  - Il est utile pour allouer une ressource parmi plusieurs exemplaires identiques : la valeur est initialisée avec le nombre de ressources.

Sémaphores: Réalisations logicielles

- Initialisation du compteur du sémaphore S
- P(S) /*compteur est tjs modifié par P(S)*/

  { compteur := compteur -1 ;

  Si compteur < 0 alors bloquer le processus en fin de S.en_attente ;}
- V(S) /* compteur est tjs modifié par V(S)*/

  {compteur := compteur + 1 ;

  Si compteur <= 0 alors réveiller le processus en tête de S.en_attente ; }

Réalisations logicielles des primitives P et V

- Problème de l'exclusion mutuelle:
  - initialiser S à 1, et la procédure d'entrée est P(S), et la procédure de sortie est V(S)
- P et V sont des primitives plutôt que des procédures car elles sont non interruptibles
  - possible sur monoprocesseur par masquage d'Interruption.

Réalisations logicielles des primitives P et V (2)

- L'initialisation dépend du nombre de processus pouvant effectuer en même temps une même " section critique " :
  - Exemple: m, si on a m imprimantes identiques;
- Cette implémentation donne à chaque fois dans compteur le nombre de ressources libres :
- lorsque compteur est négative, sa valeur absolue donne le nombre de processus dans la file.

Sémaphores: une deuxième implantation logicielle

- Traduction directe de la spécification fonctionnelle:
  - P(S) {

    Si compteur > 0 alors compteur = compteur -1 ;

    Sinon bloquer le processus en fin de S.en_attente ;
    }
  - V(S) {

    Si S.en_attente non-vide alors réveiller le processus en tête de S.en_attente ;

    sinon compteur := compteur +1 ;}

Sémaphore d'exclusion Mutuelle

Var mutex : sémaphore init. à 1

Processus Pi

Début

..

P(mutex)

SC

V(mutex)

...

Fin.

Sémaphores d'exclusion mutuelle: interblocage

Processus1	Processus2
P(semA) P(semB) SC V(semA) V(semB)	P(semB) P(semA) SC V(semB) V(semA)

Sémaphore de synchronisation: principe

- Un processus doit attendre un autre pour continuer (ou commencer) son exécution.

Processus1	Processus2
1 er travail V(sem)//réveil process 2	P(sem)//attente process1 2eme travail