Université Assane Seck de Ziguinchor

Système d'Exploitation

PARALLELISME ET INTERACTION DES PROCESSUS

Sommaire

INTRODUCTION

On peut distinguer deux types de parallélisme : le parallélisme réel qui est matériel qui consiste à faire fonctionner plusieurs unités de traitement en même temps. Le pseudo-parallélisme qui est logiciel, qui consiste à partager le temps d'utilisation du processeur entre plusieurs processus.

I.1. Terminologies

Un système est dit :

- **multi-tâches** lorsque plusieurs entités concurrentes s'exécutent sur une machine mono processeur.
- **Parallèle** lorsque plusieurs entités concurrentes s'exécutent sur plusieurs processeurs avec une **mémoire commune.**
- **Réparti** lorsque plusieurs entités concurrentes s'exécutent sur des machines distinctes reliées en réseau et **ne partageant pas de mémoire commune.**

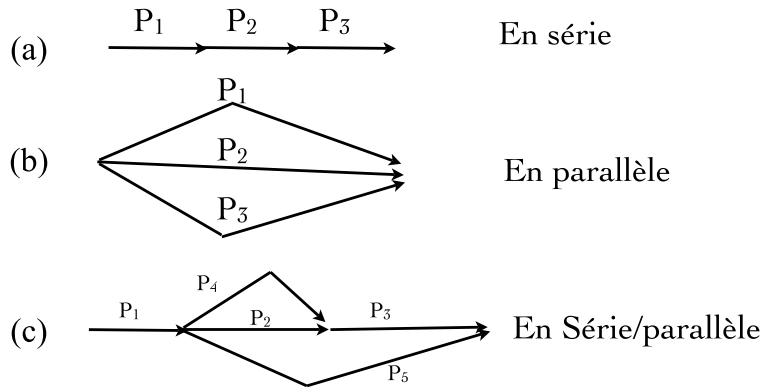
Deux processus sont :

– indépendants si l'évolution de l'un n'influe pas et ne dépend pas de l'autre

- en coopération si chacun mène un travail distinct de l'autre mais avec un objectif commun et la mise en place de synchronisation et de communication et qu'il n'y a pas compétition sur l'accès à certaines ressources.
- en compétition lorsqu'ils sont en coopération mais lorsque, de plus, ils sont en compétition sur l'accès de certaines ressources.

Selon le niveau d'observation, deux processus peuvent être vus en coopération ou en compétition.

Les coroutines sont des processus qui s'exécutent de manière concurrentes et pouvant manipuler des données communes sans hiérarchie d'appel. Chaque coroutine peut être vue comme un processus.



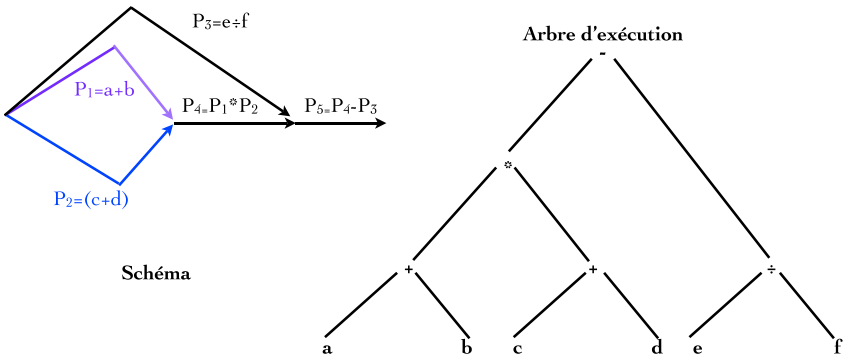
Représentation de la séquence des processus par un graphe

(a) : P1 et P2 ne sont pas des coroutines.

(b) : P1, P1 et P2 s'exécutent de manière concurrente.

(c) :P3 s'exécute après P4 et/ou P2

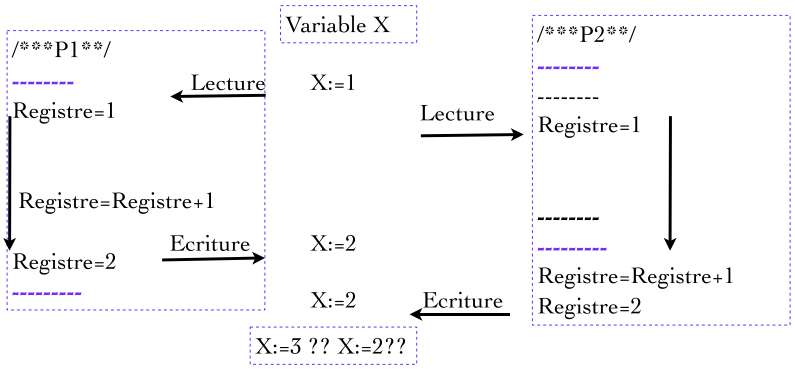
Soient l'opréation (a+b)*(c+d)-(e÷f), si P1 effectue t1= (a+b), P2 effectue t2=(c+d), P3 effectue t3= (e÷f), P4 effectue t4 = t1*t2 et P5 effectue t5= t4-t3, ce qu'on peut représenter par le schéma suivant.



Les processus P1 et P2 sont indépendants et sont en coopération. Dans (a+b)*(b+d)-(e÷f), si P1 effectue t1= (a+b), alors P1 et P2 sont en competition sur l'accès à la ressource partagée qui est la variable b.

I.2. Corruption des données

Soient deux processus A et B et une variable X initialisée à 1. A et B executent chacun l'instruction d'incrémentation X=X+1. Si l'accès à la variable partagée X n'est pas protégé, et si le processus B lit la variable X après l'instruction de lecture de X par A et avant l'instruction d'écriture en mémoire de X=2 par ce dernier, le résultat de X après l'écrémentation sera erroné (X=2 au lieu de 3)



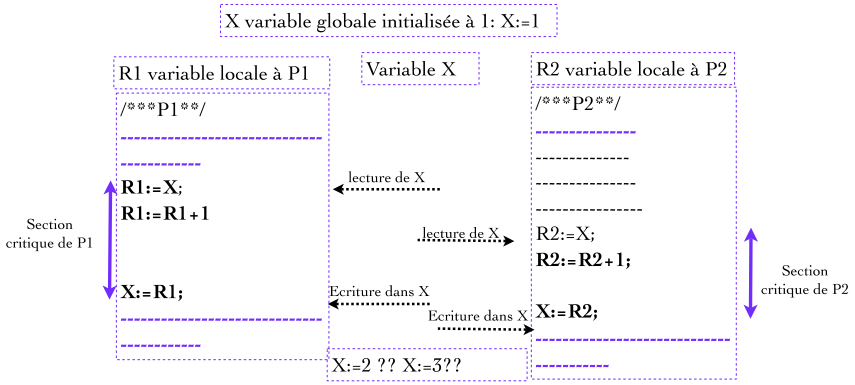
Problème de corruption des données

II. SECTION CRITIQUE

Le code exécuté par un processus peut être regroupé en sections, certaines d'entre elles ont besoin d'accéder aux ressources partagées, d'autres non. Les premières sont appelées sections critiques. Pour éviter les compétions d'accès, un mécanisme est indispensable afin de bien synchroniser l'exécution au sein des sections critiques.

Le problème de l'accès concurrent peut être illustré par l'exemple suivant :

Soient deux processus : P1 et P2.



Les deux processus P1 et P2 sont des coroutines.

Si les vitesses d'exécution et la synchronisation sont telles qu'une coroutine s'exécute entre les instructions d'affectation et d'incrémentation de l'autre coroutine, les résultats seront faussés. Il existe donc des parties du programme, au cours de leur exécution, le développement d'une autre coroutine ne doit pas être utilisé. De telles parties des programme sont appelées section critique. Chaque coroutine contient des sections critiques qui exigent une exclusion mutuelle, c'est-à-dire si un processus entre dans la section critique et manipule des variables x et y par exemple, aucun autre processus ne doit pas entrer dans la section critique où l'on manipule ces variables x et y, mais dans d'autres sections critiques où l'on manipule d'autres variables différentes de x et y.

III. EXCLUSION MUTUELLE

Pour réaliser l'exclusion mutuelle sur une section critique liée à une variable partagée, il faut tenir en compte de certains paramètres :

- A un moment donné, un seul processus peut être dans la section critique.
- Aucun processus ne peut rester indéfiniment dans la section critique.
- Aucun processus ne doit attendre indéfiniment sans entrer dans la section critique.

En d'autres termes ce qu'il faut éviter c'est l'exécution simultanée de la section critique et l'interblocage. Par exemple, le processus A attend le processus B, qui lui même attend le processus C qui lui même attend le processus A : il y a un interblocage (ou deadlock).

Un caractère important pour la réalisation de l'exclusion mutuelle est l'**atomicité**. Dans le problème posé, le point critique se trouve entre la lecture d'une variable partagée et son écriture. Si un changement de contexte intervient entre ces deux actions, et qu'un autre processus écrive dans la variable: l'état de la mémoire du premier processus est incohérantUne instruction atomique, est une commande fournie par le SE. Elle ne peut pas être interrompue par un changement de contexte.

Toute réalisation de l'exclusion mutuelle est basée sur l'utilisation de moyens matériels et logiciel offerts aux systèmes informatiques. Parmi ces moyens, le plus utilisé est le mécanisme offert par **les sémaphores**

IV. LES SEMAPHORES

Le sémaphore est une variable utilisée en commun par plusieurs processus. Sa forme générale est un entier non négatif. Dans un cas particulier, il peut prendre 2 valeurs 0 et 1 correspondantes à son ouverture et sa fermeture : on parle de **sémaphore binaire.** Chaque sémaphore est associé à une variable partagé et une liste de processus.

On définit 2 opération sur le sémaphore : Proberen=tester, Verhogen=augmenter.

Ces opérations sont atomiques c'est-à-dire quand on commence l'une, il est impossible de l'interrompre sans terminaison de l'opération, Proberen est noté P(), Verhogen est noté V().

Soit un sémaphore S (qui est une variable partagés) :

V(s) est défini comme :V(S) : S :=S+1 en une opération atomique (l'accès, l'incrémentation, l'écriture en mémoire ne peuvent être interrompues et S n'est pas accessible aux autres processus pendant cette opération V(S)) .

P(S) est définie de la manière suivante : Si $S \neq 0$ alors P(S) : S :=S-1 en une opération atomique.

Ainsi P(S) peut empêcher l'entrée dans la section critique et V(S) peut autoriser l'entrée dans la section critique. Ceci permet de réaliser l'exclusion mutuelle de manière simple.

Exemple : si la variable Exmut est un sémaphore utilisé pour organiser l'entrée dans la section critique, alors pour n coroutines, la réalisation de l'exclusion mutuelle peut être de la forme suivante :

Le **cobegin** est une construction réalisant l'exécution concurrente d'un ensemble de séquences d'instructions.

cobegin

Suite_D_Instructions_1

Suite_D_Instructions_2

...

Suite_D_Instructions_N

coend

Le bloc termine quand toutes les séquences ont terminées

Exemple : Si la variable Exmut est un sémaphore utilisé pour organiser l'entrée dans la section critique (l'exclusion mutuelle), alors pour n coroutines, la réalisation de l'exclusion mutuelle peut être de la forme suivante.

```
Variable globale                Sémaphore Exmut
Variable de type Sémaphore      Exmut :=1   //initialisation de Exmut
                  ------------------
                  ------------------
              Cobegin  //début d'exécution des coroutines
                   ----------------
                   ----------
                    Pi begin
                   ------------
                  P(Exmut)
                 Section critique
                   V(Exmut)
                 --------------------
                   end
                    Coend
```

Réalisation de l'exclusion mutuelle avec les sémaphores

Le blocage mutuel n'est pas possible car les tentatives simultanées d'exécuter P(Exmut) seront ordonnées séquentiellement de manière aléatoire par le système. L'exclusion mutuelle est garantie puisqu'un seul processus peut diminuer Exmut jusqu'à 0 avec P(Exmut).

Les sémaphores peuvent être utilisés pour organiser la comptabilisation des ressources et pour synchroniser l'accès à ces ressources.

Exemple : le problème du **producteur** et du **consommateur**

Nous disposons d'une zone tampons contenant n tampons de même taille. Un processus produit un enregistrement et le place dans un tampon libre. Un autre processus retire l'enregistrement du tampon et l'imprime. Les opérations d'écriture et de lecture se font de manière asynchrone.

1 sémaphore pour indiquer l'exclusion mutuelle : bufsem : sémaphore binaire

2 sémaphores pour la comptabilisation des ressources :

bufvide : voir combien de tampons sont vides

bufplein : voir combien de tampons sont plein

Sémaphore bufsem, bufvide, bufplein

bufsem :=1 ; bufvide :=N ; bufplein :=0 ;

```
/*Producteur*/
 Cobegin
   --
   ----
  Begin
Adr1 : Produire_enregistrementSuivant() ;
   P (bufvide) ;
   P (bufsem) ;
  EcrireDansBuffer () ;
  V (bufsem) ;
  V (bufplein) ;
  ---
  ---
 Aller à Adr1
  End
  coend
```

```
/*Consommateur*/
  Cobegin
     --
     ----
     Begin
Adr2 : P (bufplein) ;
     P (bufsem) ;
    RetirerDuBuffer () ;
    V (bufsem) ;
   V (bufvide) ;
       ---
    Consommer () ;  / *imprimer*/
       ---
     Aller à Adr2
      End
      coend
```

En général les sémaphores sont réalisés de manière logicielle dans la majorité des machines actuelles. Cette réalisation est incluse dans le système d'Exploitation. Si tel n'est pas le cas, il ressort du programmeur d'écrire un logiciel de modélisation des sémaphores.

Soient 6 coroutines chargées en mémoires et les processus initiés correspondants P1-P6. P1 commence à se développer immédiatement, les autres étant en attente. P2, P3 et P4 se développent lorsqu'ils sont libérés par P1. P5 est libéré par P2 ou P3 alors que P6 l'est par P5 et P4. Un processus libère un autre juste avant de se terminer.

Donnons les fragments de programmes montrant la synchronisation de ces processus en utilisant les sémaphores.

Semaphore : S_1, S_2, S_3, S_4, S_5, S_6;
S_1=0, S_2=0, S_3=0, S_4=0, S_5=0, S_6=0;
Cobegin

P_1	P_2	P_3	P_4	P_5	P6
Begin	Begin	Begin	Begin	Begin	Begin
.	P (S_1)	P (S_2)	P (S_3)	P (S_4)	P (S_5)
.	.	.	.	.	P
V (S_1)	.	.	.	.	(S_6)
V	.	.	.	.	.
(S_2)	V	V (S_4)	V (S_5)	V	.
V (S_3)	(S_4)	End	End	(S_6)	.
End	End			End	End

Coend

Synchronisation avec les sémaphores

Les processus ont besoin de s'exclure mutuellement des sections critiques, mais aussi d'être informé de la fin d'une opération entamée par un autre processus. Il y a nécessité de

synchroniser les actions des processus. Cette synchronisation peut être réalisée avec des sémaphores. L'exemple du consommateur et du producteur. Le processus utilisant la ressource exécute P(S) sur le sémaphore lié et celui qui produit la ressource exécute V(S) sur le même sémaphore. Ainsi un processus peut être bloqué par P(S) et débloqué par V(S) venant d'un autre processus. De ce fait par un échange de signaux, la synchronisation est assurée entre les processus. Un processus devant attendre un événement complet avant de poursuivre son exécution. Ceci peut être illustré par le fragment de programme suivant :

```
Cobegin
Semaphore S;
S :=0;
P1
begin
------
---
Attend de P2
Signal d'un événement A
P(S) ;
---
End
Coend
```

```
P2
begin
------
---
Envoyer le signal de l'événement A à P1 ;
V(S) ;
---
End
```

On peut considérer que P1 utilise la ressource et P2 produit une unité de la ressource S.

V. LES MONITEURS

Les mécanismes d'exclusion mutuelle des sémaphores permettent l'exécution parallèle des processus. Cependant leur programmation se fait essentiellement en assembleur ou sous une forme analogue. Il est souvent plus commode d'avoir des opérations de haut niveau permettant de réaliser du parallélisme. Les moniteurs constituent l'une de ces opérations (moyens). Le moniteur est donc un ensemble de variable communes et de procédures permettant d'accéder à ces variables. A un moment donné, le moniteur peut être utilisé par un seul processus, le processus qui veut accéder aux variables en fait la demande au moniteur. S'il n'est pas occupé, il exécute la procédure correspondantes, autrement dit il faut que le processus appelant attend dans une file d'attente ou alors qu'il continue à se développer tout en se privant de cette variable momentanément. Le processus peut être bloqué selon une condition ; cette condition sera testée par le moniteur qui débloquera le processus.

Soit une ressource qui est un moniteur
Deux variables : libre, occupe

```
Libre : condition ; occupe : booleen ;
Begin
Occupe :=false                    /*ressource non occupée*/
Procedure DemandeRessource ;
begin
if occupe then wait (libre)       /*wait teste si la condition libre est vraie ou non,
autorisation d'accés                au cas ou occupe est vrai*/
occupe:=true
end
Procedure LibererRessource
Begin
Occupe:=false
Perform (libre)                   /*PERFORM met la condition libre à vraie*/
End
End
```

Nous avons une ressource, deux procédures permettant d'accéder à la ressource : **DemandeRessource** et **LibererRessource** que tentent d'exécuter les processus, nous avons aussi deux opérations du moniteur **wait** (attendre), **perform** (exécuter). Si la ressource est occupée alors la procédure DemandRessource exécute l'opération du moniteur wait qui bloque les processus qui vont se mettre en file d'attente jusqu'à ce que la condition libre soit vérifiée. Lorsque la procédure utilisant la ressource demande d'exécuter la procédure LibererRessource alors l'opération du moniteur Perform débloque le processus au début de la file.

Le moniteur n'est pas un processus en tant que tel, ses procédures sont exécutées seulement à la demande des processus. Si les procédures utilisent de la même manière une variable commune au moyen des sémaphores, alors pour chaque sémaphore il faut sa propre copie de la section critique. Si on utilise un moniteur, on a besoin d'une seule copie.