CHAPITRE I : INTRODUCTION

Dans ce chapitre nous présenterons la problématique des accès concurrentes : le fait que plusieurs processus entrent en compétition pour occuper une ressource. Nous définirons également les concepts fondamentaux concernée par la programmation concurrente.

1.1 PROBLEMATIQUE :

Lorsque plusieurs processus s'exécutent sur un ordinateur , monoprocesseur ou multiprocesseur avec mémoire commune , ils sont amenés à partager des variables communes soit volontairement s'ils coopèrent pour traiter un même problème, soit involontairement parce qu'ils sont obligés de se partager les ressources de l'ordinateur.

Malheureusement, le partage des variables sans précaution particulière peut conduire à des résultats imprévisibles.

Exemple : Le compte bancaire

Considérons un compte bancaire , dont le montant est mémorisé dans un emplacement mémoire A. Le programme qui consiste à ajouter 100 à ce compte pourrait être le suivant , où N est une variable locale du programme :

```
Algorithme Compte
Début
        Lire(N, A) ;

        N :=N+100 ;

        Ecrire(N, A)
Fin.
```

Si maintenant deux processus différents consistent en l'exécution de ce même programme, le processeur sera alloué à chacun d'eux dans un ordre quelconque. En particulier, on peut imaginer l'ordre suivant :

Ordre d'exécution	*Processus P1*	*Processus P2*
1	Lire(N, A)	
2		Lire(N, A) ;
3		N :=N+100 ;
4		Ecrire(N, A) ;
5	N :=N+100 ;	
6	Ecrire(N, A)	

N étant une variable locale du programme, cela implique qu'il en existe un exemplaire pour chacun des deux processus. Il s'ensuit que si la valeur initiale du compte est 1000, on constate qu'après ces exécutions , il est égal à 1100 au lieu de 1200. Les deux processus ne sont pas en fait totalement indépendants. Ils partagent la ressource commune qu'est la valeur du compte bancaire à l'adresse A sur disque. Cette ressource doit avoir un seul point d'accès ; c'est donc une ressource critique , et les processus sont en exclusion mutuelle sur cette ressource critique.

Si la variable N est commune aux deux processus, le problème n'est pas résolu pour autant. En effet, l'instruction N :=N+100 n'est pas une instruction « indivisible ». En fait, elle peut être décomposée en trois micro-instructions au niveau machine , comme le montre le code suivant :

Load N Add 100 Store N	⇔	N :=N+100

Il est important de savoir que l'allocateur du processeur prend en compte les micro-instructions et non les instruction du langage évolué pour déterminer les moments où il peut remplacer le processus actif par un autre. On peut donc imaginer le scénario suivant d'allocation du processeur aux deux processus :

Ordre d'exécution	*Processus P1*	*Processus P2*
1	Load N	
2		Load N
3		Add 100
4		Store N
5	Add 100	
6	Store N	

Ainsi , on voit là aussi que le résultat obtenu à l'issue des deux processus est erroné.

On voit à travers cet exemple que l'exécution simultanées de programmes (processus) peut conduire à des résultats erronés si des précautions ne sont pas prises.

1.2 CONCEPTS FONDAMENTAUX :

Dans cette section nous donnons quelques définitions des concepts fondamentaux essentielles pour suivre la suite de l'ouvrage.

1.2.1 SECTION CRITIQUE

Une section critique (SC) est un ensemble d'instruction d'un programme qui peuvent engendrer des résultats imprévisibles lorsqu'elles sont exécutées simultanément par des processus différents.

D'une manière générale on peut dire qu'un ensemble d'instructions peut constituer une section critique (SC) s'il y a des variables partagées ; et non dans l'absolu. Autrement dit, l'existence de section critique implique l'utilisation de variable partagées , mais l'inverse n'est pas vrai.

De ce qui précède on peut facilement déduire que l'exécution simultanées de deux sections critiques appartenant à deux ensembles différents et ne partageant pas de variables ne peut poser aucun problème d'accès concurrent.

1.2.2 RESSOURCE CRITIQUE

On appelle ressource critique tout objet : variable, table, fichier, périphérique, … qui peut faire l'objet d'un accès concurrent par plusieurs processus. Par exemple deux processus qui tentent d'envoyer chacun, simultanément, un fichier sur l'imprimante ; le périphérique imprimante devient ressource critique pour eux.

1.2.3 EXCLUSION MUTUELLE

Les problèmes posés par les accès concurrents décrits au §1.1 montrent que la solution consiste à exécuter les sections critiques en **exclusion mutuelle** (mutex). C'est à dire qu'une SC ne peut être entamée que si aucune autre SC du même ensemble n'est en exécution.

Ainsi, voici donc le principe général d'une solution garantissant que l'exécution simultanée de plusieurs processus ne conduirait pas à des résultats imprévisibles : 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. Dans le cas contraire, il ne devra pas progresser tant que l'autre processus n'aura pas terminé sa SC.

Avant d'entrer en SC, le processus doit exécuter un protocole d'entrée. Le but de ce protocole est de vérifier justement si la SC n'est occupée par aucun autre processus.

A la sortie de la SC, le processus doit exécuter un protocole de sortie de la SC. Le but de ce protocole est d'avertir les autres processus en attente que la SC est devenue libre.

Le schéma suivant résume ce principe de fonctionnement :

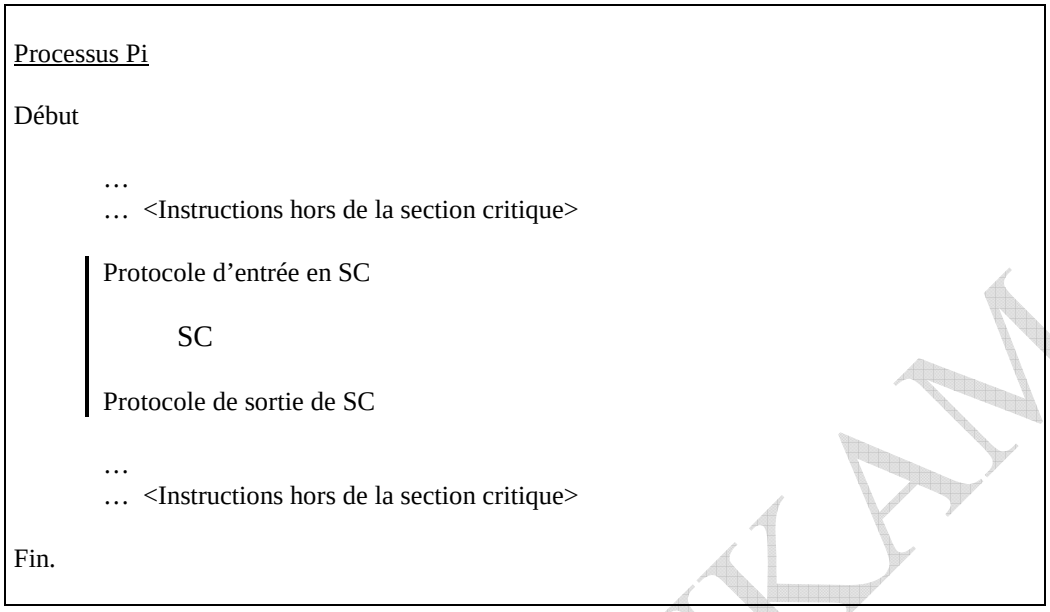

Figure 1.1 Structure d'un processus qui exécute une section critique.

1.2.4 CONTRAINTES A RESPECTER POUR LA REALISATION D'UNE EXCLUSION MUTUELLE :

Pour réaliser une exclusion mutuelle , nécessaire pour résoudre le problème des accès concurrents, on admet que certaines contraintes doivent être respectées :

1. **Le déroulement** : Le fait qu'un processus ne demande pas à entrer en section critique ne doit pas empêcher un autre processus d'y entrer. Cette contrainte exclut les méthodes fondées sur un tour de rôle strict.
2. **L'attente finie** : Si plusieurs processus sont en compétition pour entrer en SC, le choix de l'un d'eux ne doit pas être repoussé indéfiniment. Autrement dit, la solution proposée doit garantir que tout processus n'attend pas indéfiniment.
3. Tous les processus doivent être égaux vis à vis de l'entrée en SC.