Chapitre 1 : Communication inter-processus par partage d'information

Imene LAHYANI

Introduction (1)

- Programme parallèle = programme qui contient plusieurs processus (ou threads) qui coopèrent
- Coopération → Communication, échange d'information
- Deux principales façons de communiquer:
  - **Par l'intermédiaire de variables partagées**
  - Par l'échange de messages

Introduction (2)

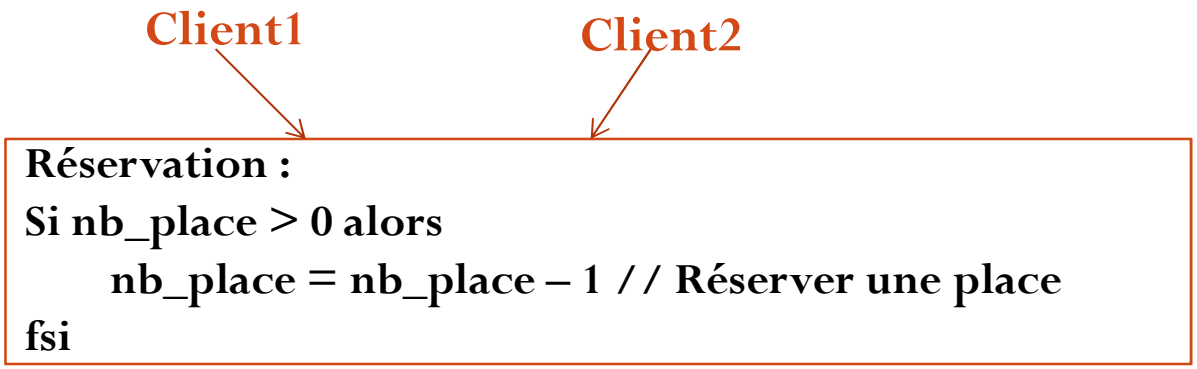
- La communication inter-processus via le partage de variable s'effectue lorsqu'un processus écrit une variable (partagée) qui va être lue par un autre processus

Problème :

- L'accès aux cellules partagées peut s'effectuer de façon **désordonnée**
  - deux processus peuvent accéder simultanément à des cellules partagées.
- Ceci peut introduire un degré d'incertitude dans l'exécution du programme.
- Un tel degré d'incertitude est inacceptable!!!.

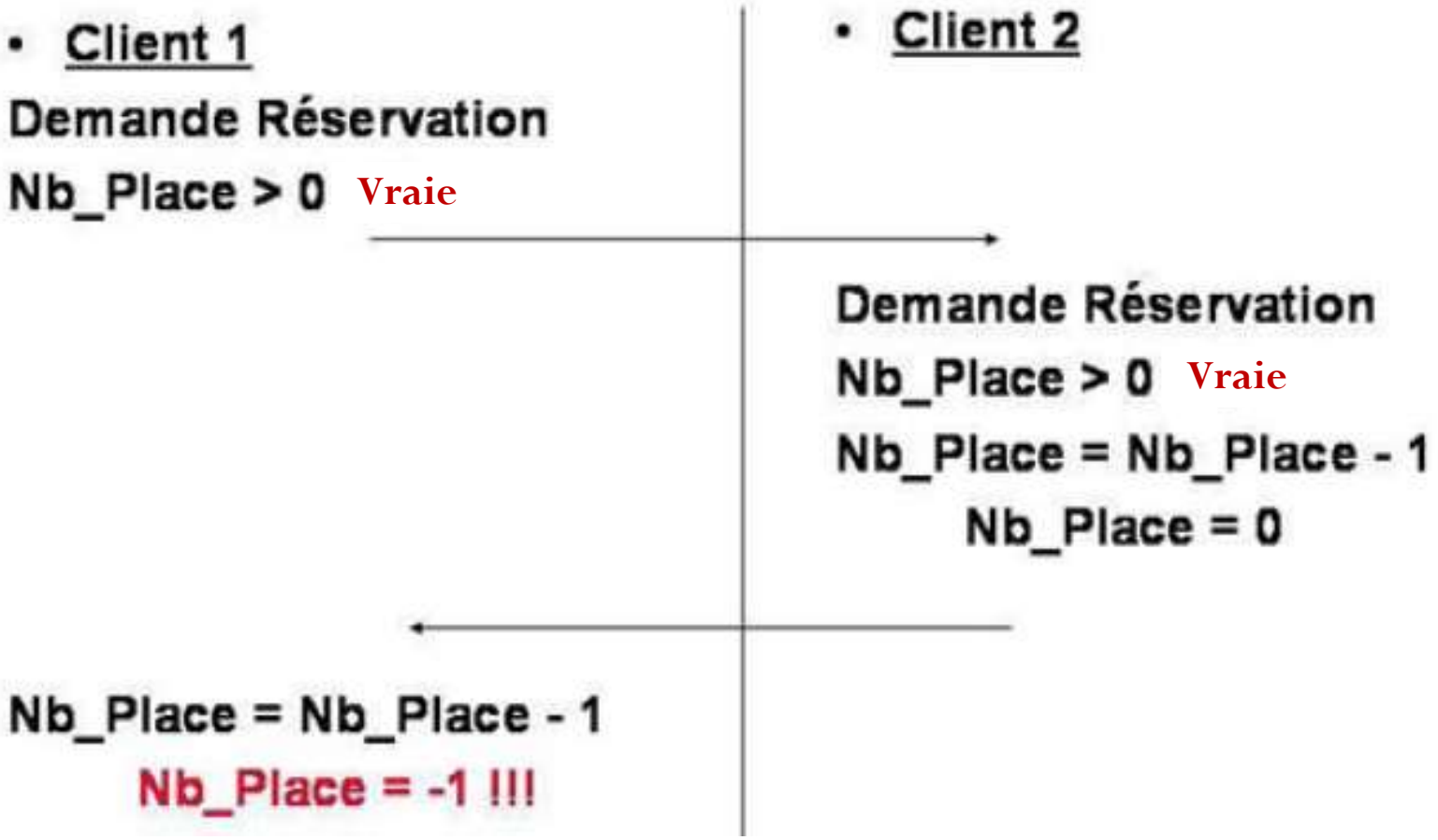
Introduction (3)

- **Exemple** : Problème de réservation de places dans un avion:
- Deux clients Client 1 et Client 2 veulent réserver chacun une place dans un avion (ils doivent accéder à la variable nb_places, initialement = 1)



- Comme résultat : Nb_places peut avoir la valeur 0 ou -1.
- Ce qui est faux!!!
  - Nb_places devra avoir la valeur 0 à la fin de l'algorithme

Introduction (4)



Introduction (5)

- **Solution:**
- Il est nécessaire de disposer d'un mécanisme permettant d'organiser l'accès aux informations partagées,

⇒ Il faut garantir **l'accès exclusif** aux informations partagées.

- Par conséquent, une instruction d'affectation doit être **atomique** (indivisible),

Sections Critiques

- **Définition** : Une section critique est une portion du code là ou on accède à la variable partagée.

- Toute variable partagée ne doit être accédée que de l'intérieur d'une section critique ;

Exclusion mutuelle

- **Définition** : **Deux processus ne peuvent jamais se trouver simultanément dans leurs sections critiques.**

Comment assurer l'exclusion mutuelle ?

- **Sol:** On ajoute un protocole avant et après chaque section critique
- Ces protocoles assureront l'exclusion mutuelle

process Pi(i=1..n)

section non-critique …
protocole d'entrée
section critique (SC)
protocole de sortie

Contraintes à respecter pour que les solutions soient acceptables !! :

- Les protocoles doivent garantir les propriétés suivantes lors de l'exécution du programme :
  - **Sûreté :** Les deux processus ne sont jamais simultanément dans leurs sections critiques.
  - **Vivacité :** Si un processus est prêt à entrer dans sa section critique, il finira par y arriver
  - **Robustesse :** Si un processus se trouve hors de sa section critique et hors du protocole, l'autre pourra toujours accéder à sa section critique.

Les sémaphores

- Un sémaphore est un objet constitué d'un entier sur lequel peuvent opérer deux primitives appelées P et V.
- Ces primitives sont atomiques, ce qui signifie que deux processus différents ne peuvent les exécuter simultanément sur le même sémaphore.
- Lors de sa création, un sémaphore s est initialisé à une certaine valeur entière s0.

wait(s)

P(s)

If s > 0 **then** s := s - 1

else "wait"

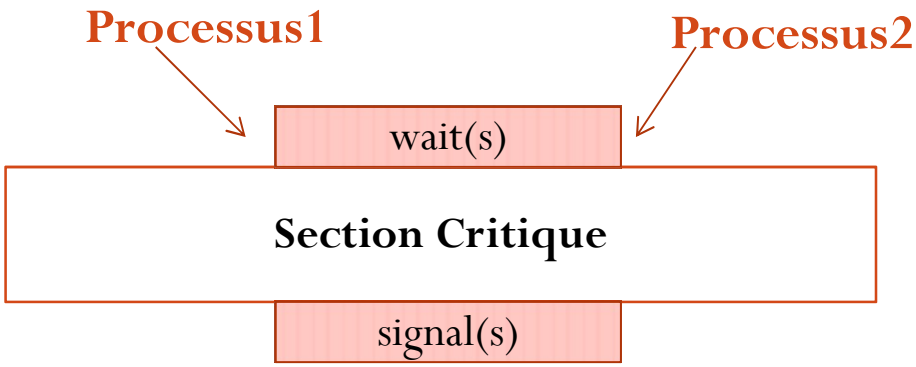
signal(s) :

V(s)

s := s + 1

RQ:
Les opérations permises sur les sémaphores doivent être exécutées de façon indivisible.

Application des sémaphores : l'exclusion mutuelle



S est un sémaphore initialisé à une valeur 1

Les sémaphores en java

- En java, le paquetage java.util.concurrent : fournit une classe appelée Semaphore qui permet la synchronisation à l'aide des sémaphores

public class Semaphore extends Object implements Serializable

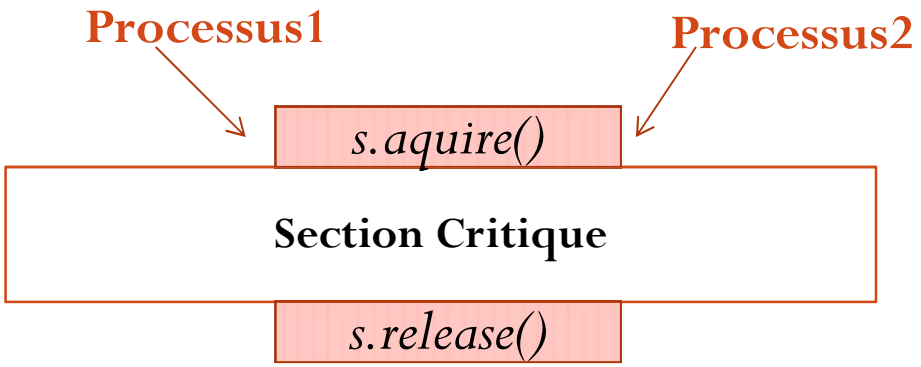
- Constructeur de la classe Semaphore:
  - Semaphore(int permits)
    - Crée un Sémaphore avec une valeur initiale (nb d'appels non bloquants)
  - Semaphore(int permits, boolean fair)
    - Crée un Sémaphore avec:
      - une valeur initiale (nb d'appels non bloquants).
      - fair = true pour garantir une gestion FIFO des processus en cas de conflit.

Les méthodes de la classe Sémaphore

La classe Semaphore expose différentes méthodes :

- acquire () et acquire (int permits) :
  - demande une ou plusieurs autorisations. Ces deux méthodes sont bloquantes : elles ne rendent la main que lorsque le nombre d'autorisations demandé est disponible.
  - Elles jettent une exception *InterruptedException* si le thread est interrompu.
- release() et release(int permits) :
  - rend une autorisation, ou le nombre d'autorisations passées en paramètre.
- availablePermits() :
  - retourne le nombre d'autorisations disponibles pour ce sémaphore.
- …

Application de la classe Sémaphore



S est un sémaphore déclaré comme suit :

Semaphore s = new Semaphore (1, true);