M113
Systèmes d'Exploitation et Programmation Système

Prof. Fadoua YAKINE

M113
Systèmes d'Exploitation et Programmation Système

Chapitre 3 : Communication interprocessuss

Inter-Process Communication (IPC)

23 Octobre 2024

Prof. Fadoua YAKINE

Synchronisation et communication entre processus

- Les processus peuvent avoir besoin de communiquer entre eux pour échanger des données.
- Dans tous ces cas, les processus ne sont plus indépendants et ils effectuent des accès concurrents aux ressources.

Synchronisation et communication entre processus

- Les processus doivent fréquemment se synchroniser et communiquer avec d'autres processus.
  - Par exemple, dans un pipeline shell, la sortie du premier processus doit être transmise au deuxième processus, et ainsi de suite.
- Il y a donc un besoin de communication entre les processus, de préférence d'une manière bien structurée et en évitant les interruptions.
- Nous aborderons quelques-uns des problèmes liés à cette communication interprocessus (IPC, InterProcess Communication).

Problemes de synchronisation et communication entre processus

On rencontre essentiellement trois problèmes :

1. Le premier coule de source : Comment un processus fait-il pour passer des informations à un autre processus ?
2. Le deuxième repose sur la nécessité de s'assurer que deux processus, ou plus, ne produisent pas de conflits lorsqu'ils s'engagent dans des activités critiques.
3. Le troisième concerne le séquençage en présence de dépendances : si le processus A produit des données et que le processus B les imprime, B doit attendre que A ait terminé pour pouvoir remplir sa tâche.

Conditions de concurrence (race conditions)

- Une condition de concurrence est une situation qui se produit lorsque processus ou plus accèdent à une ressource partagée, telle qu'un fichier ou une variable, en même temps.

- Le résultat de leur exécution dépend de l'ordre dans lequel ils sont exécutés.
  - Cela peut entraîner un comportement inattendu ou incorrect du système.

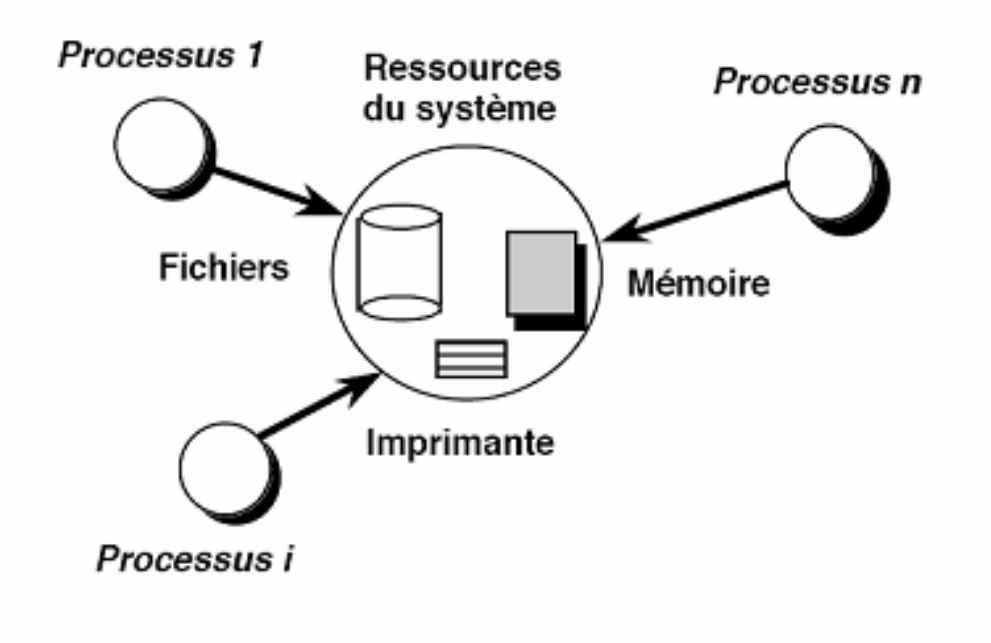
Conditions de concurrence (race conditions)

- Pour illustrer le problème de synchronisation :

```
Réservation :
si nbplace > 0
alors
réserver une place;
nbplace = nbplace – 1;
fsi
```

- On considère l'exécution de deux processus Client_1 et Client_2, alors que nbplace est égale à 1.

Problèmes de synchronisation entre processus



- Notion de ressource critique qui correspond à une ressource ne pouvant être utilisée que par un seul processus à la fois.

Ressource critique, section critique

- Pour illustrer le problème de synchronisation :

```
Réservation :
si nbplace > 0
alors
réserver une place;
nbplace = nbplace – 1;
fsi
```

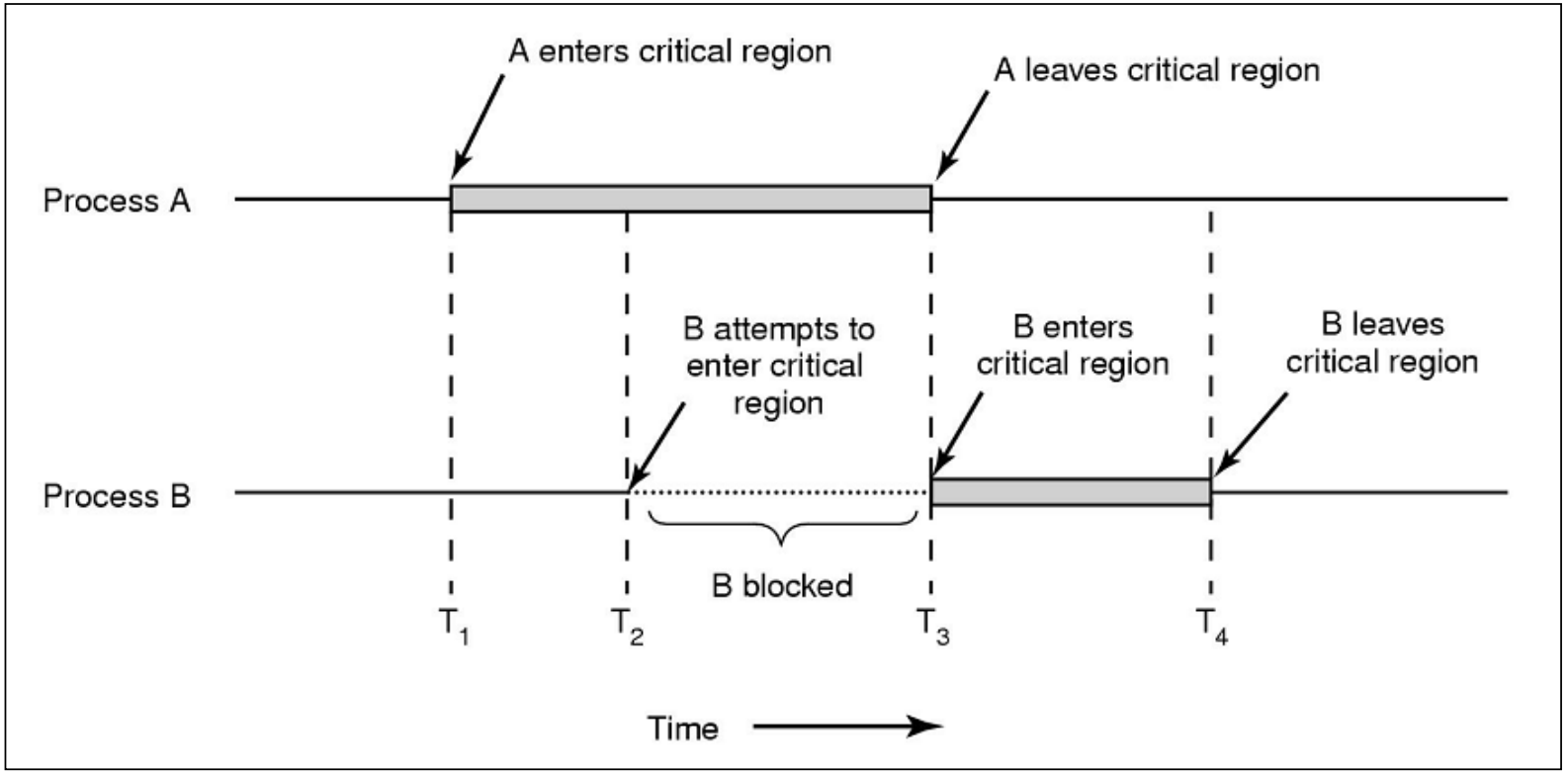
- On considère l'exécution de deux processus Client_1 et Client_2, alors que nbplace est égale à 1.
- Nbplace devra être considéré comme une ressource critique.
- Le code d'utilisation d'une ressource critique est appelé section critique.

Exclusion mutuelle

Quatre conditions pour obtenir l'exclusion mutuelle:

1. Un seul processus à la fois dans la section critique
2. Aucune supposition sur le nombre et la vitesse des CPUs
3. Aucun processus hors de la section critique ne peut bloquer un autre processus.
4. Aucun processus ne doit attendre indéfiniment pour entrer en section critique.

Exclusion mutuelle



Ressource critique, section critique et exclusion mutuelle

- La section critique doit au moins offrir la propriété essentielle de l'exclusion mutuelle qui garantit que la ressource critique manipulée dans la section critique ne peut effectivement être utilisée que par un seul processus à la fois.
- Il faut Protéger la zone critique
  - **entrée à la section critique**

```
Si nbplace > 0
alors
réserver une place;
nbplace = nbplace – 1;
fsi
```

  - **sortie de la section critique**

Solution 1: Le verrou

- Variable de verrou
  - Un verrou est une variable binaire partagée qui indique la présence d'un processus en Section Critique
  - Un processus peut entrer seulement si c=0 Il met alors c=1 pour empêcher les autres processus d'entrer
  - Protection par verrou :

```
Int  c ; c = 0
Réservation : si c=0 alors aller à Réservation
c = 1 ;
Si nbplace > 0
Alors
Réserver une place ;
nbplace = nbplace – 1 ;
c=0 ;
fsi
```

Solution 2: Alternance stricte

- On utilise une variable partagée (tour) qui mémorise le numéro du processus autorisé à entrer en section critique

```
while (TRUE) {
    while (turn != 0)        /* loop */ ;
    critical_region( );
    turn = 1;
    noncritical_region( );
}
```

(a)

```
while (TRUE) {
    while (turn != 1)        /* loop */ ;
    critical_region( );
    turn = 0;
    noncritical_region( );
}
```

(b)

Proposition de solution au problème de la section critique

(a) Processus 0. (b) Processus 1.

- Inconvénient : **un processus possédant *turn, peut ne*** pas être intéressé immédiatement par la section critique.

Solution 3 : Sémaphores

- Un sémaphore est une variable qui :
  - Contrôle l'accès à une ressource partagée
  - Indique le nombre d'éléments de la ressource qui sont disponibles
  - Maintient une liste des processus bloqués en attente de cette ressource (s'il y en a)
- Le sémaphore a été inventé par Edsger Dijkstra
- Un sémaphore sem est une structure système composée d'une file d'attente L de processus et d'un compteur S, appelé niveau du sémaphore et contenant initialement une valeur val.
- Cette structure ne peut être manipulée que par trois opérations init(sem, val) et P(sem), V(sem)

Solution 3 : Sémaphores

- L'opération init(sem,val)
  - a pour but d'initialiser le sémaphore, c'est-à-dire qu'elle met à vide la file d'attente L et initialise avec la valeur val le compteur S
- L'opération P (sem) (du néerlandais Proberen signifiant tester et en français "Puis-je ?«)
  - attribue un jeton au processus appelant s'il en reste au moins un et sinon bloque le processus dans la file d'attente L.

```
SI (S > 0) ALORS S--;
        SINON (attendre sur L)
FSI
```

Solution 3 : Sémaphores

- L'opération V(sem)) (du néerlandais Verhogen signifiant tester incrémenter et en français « Vas y ?"
  - a pour but de rendre un jeton au sémaphore. De plus, si il y a au moins un processus bloqué dans la file d'attente L du sémaphore, un processus est réveillé.

```
S++;
SI (des processus sont en attente sur S)
 ALORS laisser l'un deux continuer (reveil)
FSI
```

Solution 3 : Sémaphores

- Quand une sémaphore ne peut pas prendre de valeur plus grande que 1 (ressource unique), on parlera de sémaphore binaire, ou de mutex (exclusion mutuelle)
- Protection par mutex :

```
init (mutex, 1);
Réservation :
P(mutex); — entrée en section critique
si nbplace > 0
alors
reserver une place;
nbplace = nbplace – 1;
fsi
V(mutex); — sortie de la section critique
```

Solution 3 : Sémaphore binaire (Mutex)

- Un sémaphore binaire est un sémaphore qui est initialisé avec la valeur 1.
- Ceci a pour effet de contrôler l'accès une ressource unique.
- Le sémaphore binaire permet l'exclusion mutuelle(mutex) : une ressource est en exclusion mutuelle si seul un processus peut utiliser la ressource à un instant donné

exercice

- Supposons qu'il y a une **ressource partagée** et que **trois processus** peuvent y accéder simultanément. Nous utilisons un **sémaphore** avec une valeur initiale de 3 pour contrôler cet accès.

Les opérations du sémaphore sont :

- Processus A arrive ;
- Processus B arrive ;
- Processus C arrive ;
- Processus D arrive ;
- Processus A se termine.
- E arrive

1. Pour chaque étape, décrivez la valeur du sémaphore.
2. Quels processus attendent l'accès à la ressource à chaque moment ?
3. Comment le sémaphore gère-t-il la synchronisation des processus pour éviter les conflits d'accès ?

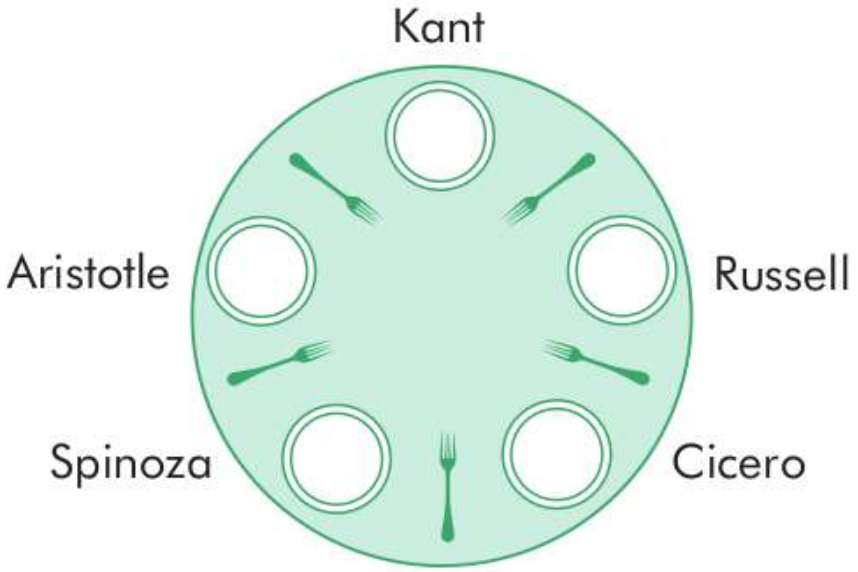
Un Classique : le diner des philosophes

- Le problème
  - cinq philosophes se trouvent autour d'une table ronde ;
  - chacun des philosophes a devant lui un plat de spaghetti ;
  - à gauche de chaque plat de spaghetti se trouve une fourchette.
- Etats des philosophes
  - penser ;
  - Avoir faim
  - Manger

Un Classique : le diner des philosophes

- Les philosophes mangent et pensent
- Mangent avec 2 fourchettes
- Prennent une seule fourchette à la fois

Comment prévenir les interbloquages ?



Un Classique : le diner des philosophes

```
#define N 5                                 /* number of philosophers */

void philosopher(int i)                     /* i: philosopher number, from 0 to 4 */
{
      while (TRUE) {
            think( );                       /* philosopher is thinking */
            take_fork(i);                   /* take left fork */
            take_fork((i+1) % N);           /* take right fork; % is modulo operator */
            eat( );                         /* yum-yum, spaghetti */
            put_fork(i);                    /* put left fork back on the table */
            put_fork((i+1) % N);            /* put right fork back on the table */
      }
}
```

Cette solution ne fonctione pas