



université
PARIS-SACLAY



CentraleSupélec

SYSTÈMES D'EXPLOITATION

SYNCHRONISATION DES PROCESSUS ET DES THREADS



3A - Cours Ingénieurs



CentraleSupélec



2023/2024



Idir AIT SADOUNE

idir.aitsadoune@centralesupelec.fr

PLAN

- Sections critiques
- Solution programmée
- Méthodes de synchronisation
- Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

INTER PROCESS COMMUNICATION - IPC

INTER PROCESS COMMUNICATION - IPC

- L'OS garantie l'**indépendance des processus**

INTER PROCESS COMMUNICATION - IPC

- L'OS garantie l'**indépendance des processus**
 - Par l'ordonnanceur CPU

INTER PROCESS COMMUNICATION - IPC

- L'OS garantie l'**indépendance des processus**
 - Par l'ordonnanceur CPU
 - Par la gestion mémoire que l'on verra plus tard

INTER PROCESS COMMUNICATION - IPC

- L'OS garantit l'**indépendance des processus**
 - Par l'ordonnanceur CPU
 - Par la gestion mémoire que l'on verra plus tard
- Un processus peut **communiquer** avec un autre processus ou avec des périphériques (fichiers, imprimantes, réseaux, ...).

INTER PROCESS COMMUNICATION - IPC

- L'OS garantit l'**indépendance des processus**
 - Par l'ordonnanceur CPU
 - Par la gestion mémoire que l'on verra plus tard
- Un processus peut **communiquer** avec un autre processus ou avec des périphériques (fichiers, imprimantes, réseaux, ...).
- Il est alors nécessaire de mettre en oeuvre un mécanisme de **communication inter-processus**.

PRINCIPALES MÉTHODES DE COMMUNICATIONS

Méthode	Description
---------	-------------

PRINCIPALES MÉTHODES DE COMMUNICATIONS

Méthode	Description
Signal	Un message système est envoyé d'un processus à un autre.

PRINCIPALES MÉTHODES DE COMMUNICATIONS

Méthode	Description
Signal	Un message système est envoyé d'un processus à un autre.
Pipe	Un canal unidirectionnel ; les données émises sont accumulées dans une mémoire tampon (FIFO).

PRINCIPALES MÉTHODES DE COMMUNICATIONS

Méthode	Description
Signal	Un message système est envoyé d'un processus à un autre.
Pipe	Un canal unidirectionnel ; les données émises sont accumulées dans une mémoire tampon (FIFO).
File	Lecture/Écriture dans un fichier.

PRINCIPALES MÉTHODES DE COMMUNICATIONS

Méthode	Description
Signal	Un message système est envoyé d'un processus à un autre.
Pipe	Un canal unidirectionnel ; les données émises sont accumulées dans une mémoire tampon (FIFO).
File	Lecture/Écriture dans un fichier.
Socket	Un flux de données envoyé à travers une interface réseau à un autre processus.

PRINCIPALES MÉTHODES DE COMMUNICATIONS

Méthode	Description
Signal	Un message système est envoyé d'un processus à un autre.
Pipe	Un canal unidirectionnel ; les données émises sont accumulées dans une mémoire tampon (FIFO).
File	Lecture/Écriture dans un fichier.
Socket	Un flux de données envoyé à travers une interface réseau à un autre processus.
Mémoire Partagée	Espace de mémoire alloué à plusieurs processus.

PRINCIPALES MÉTHODES DE COMMUNICATIONS

Méthode	Description
Signal	Un message système est envoyé d'un processus à un autre.
Pipe	Un canal unidirectionnel ; les données émises sont accumulées dans une mémoire tampon (FIFO).
File	Lecture/Écriture dans un fichier.
Socket	Un flux de données envoyé à travers une interface réseau à un autre processus.
Mémoire Partagée	Espace de mémoire alloué à plusieurs processus.
Moniteur/Sémaphore	Une structure de synchronisation pour les processus travaillant sur des ressources partagées.

PROBLÈME DE LA CONCURRENCE

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Soit la gestion d'un compte bancaire

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Soit la gestion d'un compte bancaire
 - Une variable partagée **balance**

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Soit la gestion d'un compte bancaire
 - Une variable partagée `balance`
 - Une fonction `add(1)` (`balance = balance + 1`)

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Soit la gestion d'un compte bancaire
 - Une variable partagée `balance`
 - Une fonction `add(1)` (`balance = balance + 1`)
 - Une fonction `sub(1)` (`balance = balance - 1`)

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Soit la gestion d'un compte bancaire
 - Une variable partagée `balance`
 - Une fonction `add(1)` (`balance = balance + 1`)
 - Une fonction `sub(1)` (`balance = balance - 1`)
 - Le montant initial du compte est de 9€

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Soient 2 threads en parallèle

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Soient 2 threads en parallèle
 - La première thread ajoute 10 000 000 fois 1€

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Soient 2 threads en parallèle
 - La première thread ajoute 10 000 000 fois 1€
 - La deuxième thread soustrait 10 000 000 fois 1€

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Soient 2 threads en parallèle
 - La première thread ajoute 10 000 000 fois 1€
 - La deuxième thread soustrait 10 000 000 fois 1€
- ✓ Résultat attendu: balance = 9€

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Soient 2 threads en parallèle
 - La première thread ajoute 10 000 000 fois 1€
 - La deuxième thread soustrait 10 000 000 fois 1€
- ✓ Résultat attendu: balance = 9€
- ✗ Résultat obtenu: balance = -98599€

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Comment expliquer ces erreurs de calcul ?

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Comment expliquer ces erreurs de calcul ?
👉 les entrelacements se font au niveau du code binaire

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

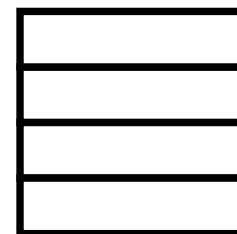
- Comment expliquer ces erreurs de calcul ?
 - 👉 les entrelacements se font au niveau du code binaire
 - 👉 couper **entre chaque instruction assembleur**

PROBLÈME DE LA CONCURRENCE

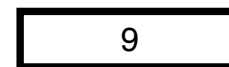
```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

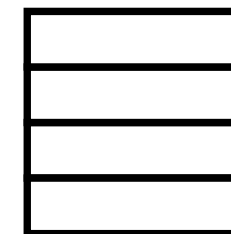
- Comment expliquer ces erreurs de calcul ?
 - 👉 les entrelacements se font au niveau du code binaire
 - 👉 couper **entre chaque instruction assembleur**



PCB t1



balance



PCB t2



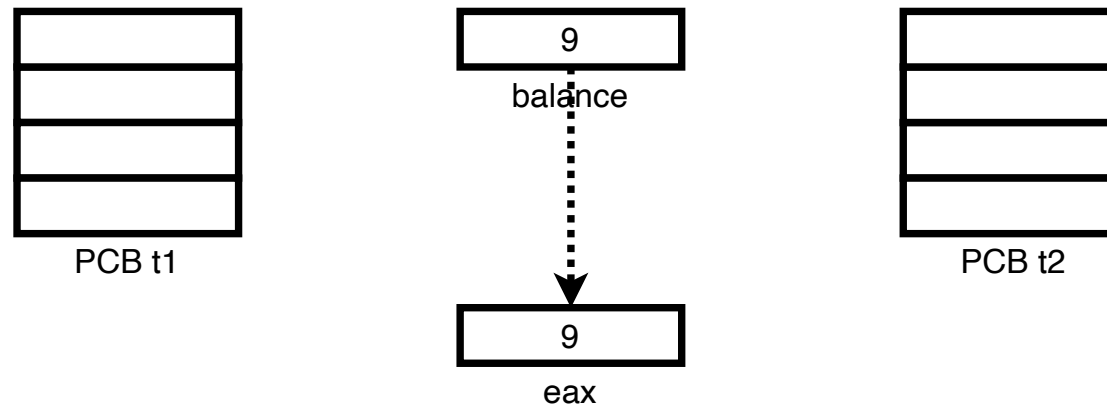
eax

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 📌 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Comment expliquer ces erreurs de calcul ?
 - 📌 les entrelacements se font au niveau du code binaire
 - 📌 couper **entre chaque instruction assembleur**



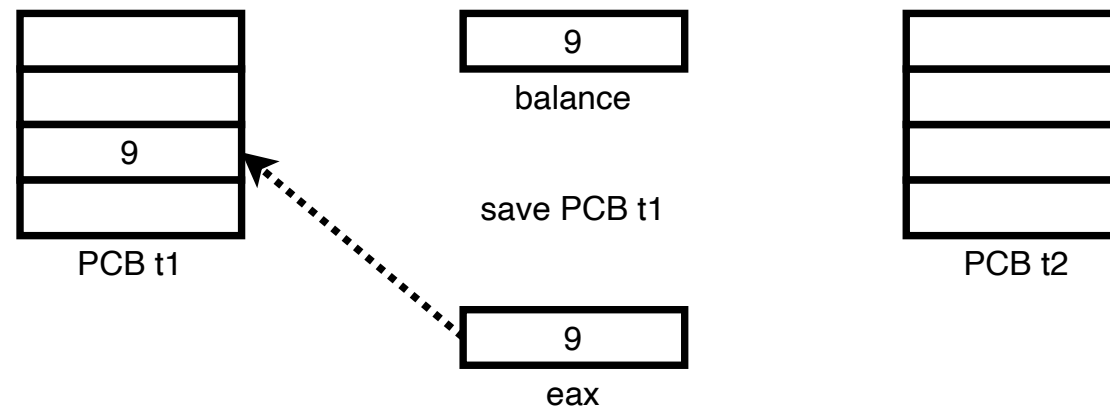
PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```



```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Comment expliquer ces erreurs de calcul ?
 - 👉 les entrelacements se font au niveau du code binaire
 - 👉 couper **entre chaque instruction assembleur**



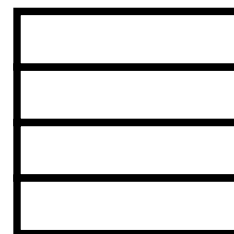
PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

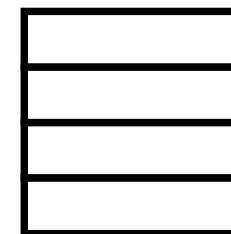
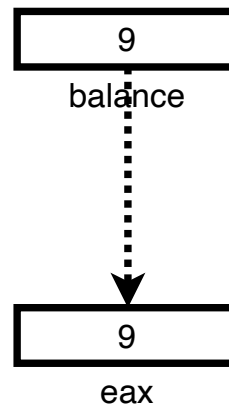


```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Comment expliquer ces erreurs de calcul ?
 - 👉 les entrelacements se font au niveau du code binaire
 - 👉 couper **entre chaque instruction assembleur**



PCB t1



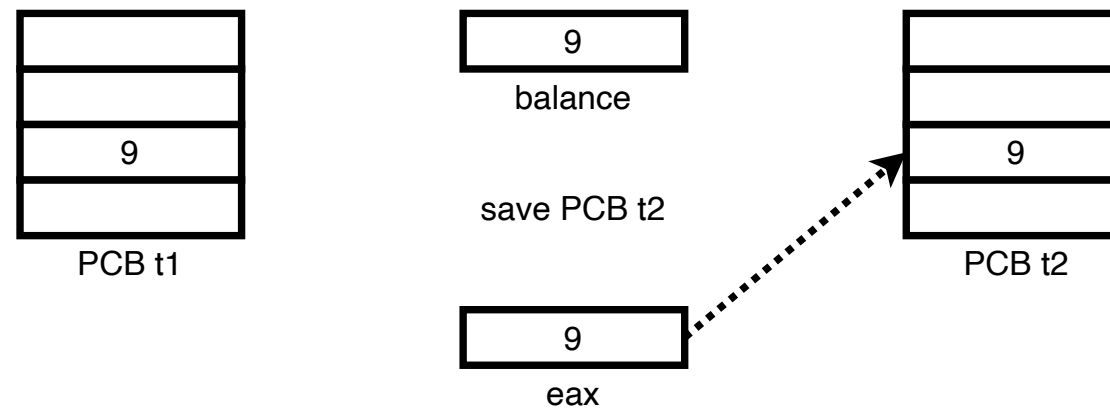
PCB t2

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Comment expliquer ces erreurs de calcul ?
 - 👉 les entrelacements se font au niveau du code binaire
 - 👉 couper **entre chaque instruction assembleur**

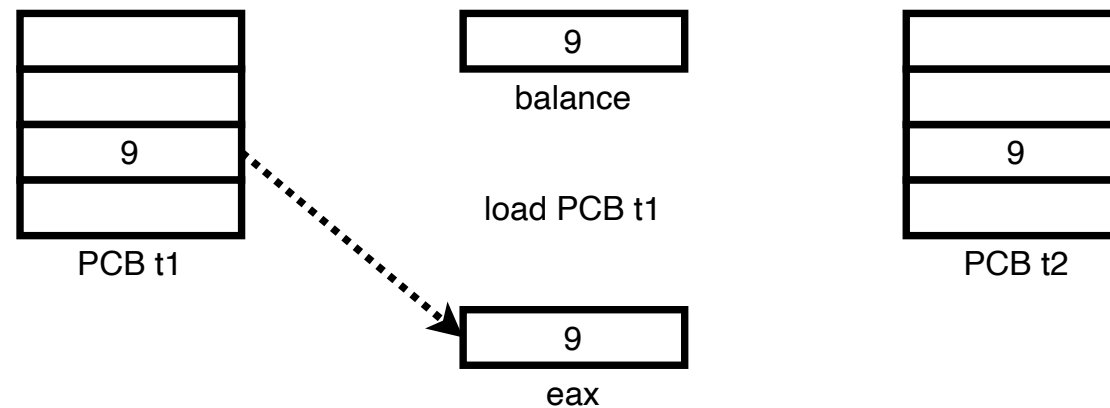


PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Comment expliquer ces erreurs de calcul ?
 - 👉 les entrelacements se font au niveau du code binaire
 - 👉 couper **entre chaque instruction assembleur**

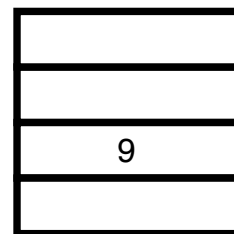


PROBLÈME DE LA CONCURRENCE

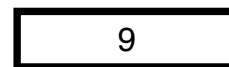
```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

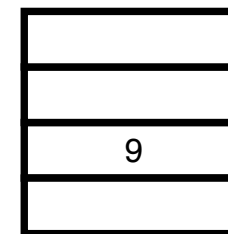
- Comment expliquer ces erreurs de calcul ?
 - 👉 les entrelacements se font au niveau du code binaire
 - 👉 couper **entre chaque instruction assembleur**



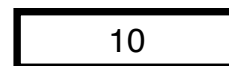
PCB t1



balance



PCB t2



eax

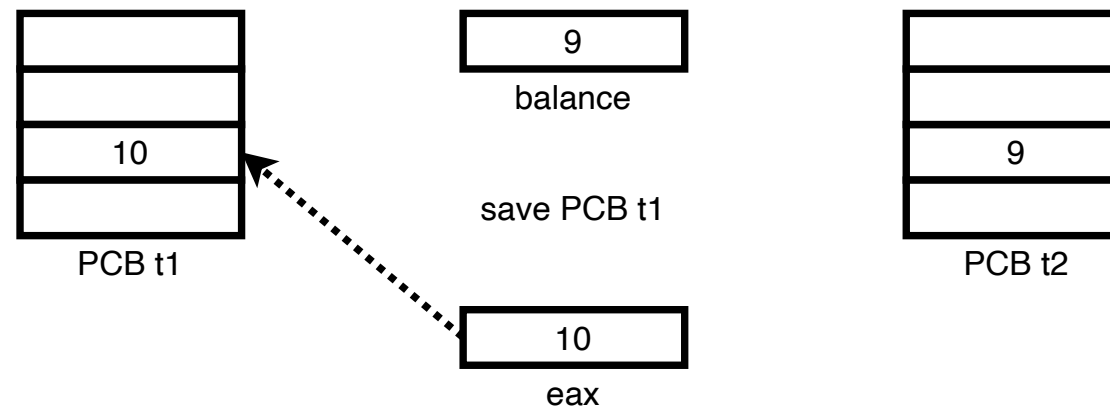
PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```



```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Comment expliquer ces erreurs de calcul ?
 - 👉 les entrelacements se font au niveau du code binaire
 - 👉 couper **entre chaque instruction assembleur**



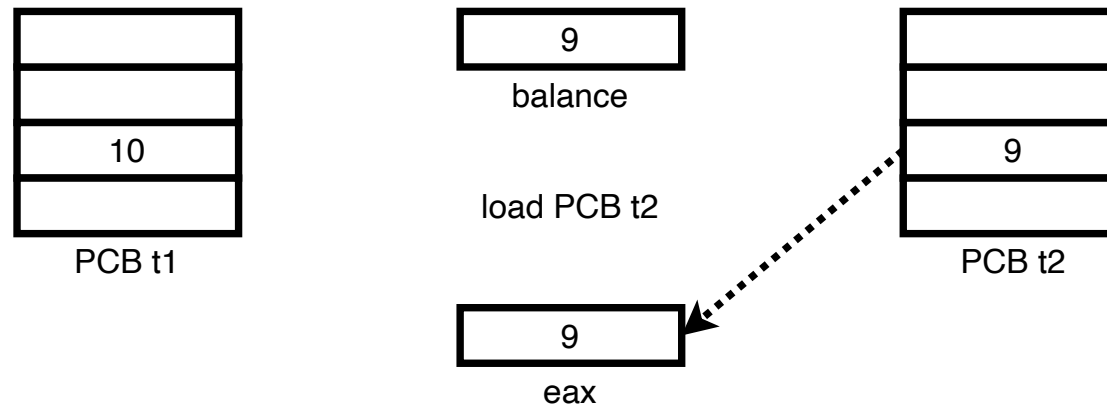
PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```



```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Comment expliquer ces erreurs de calcul ?
 - 👉 les entrelacements se font au niveau du code binaire
 - 👉 couper **entre chaque instruction assembleur**



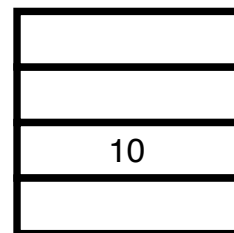
PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

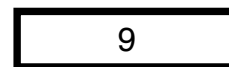


```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

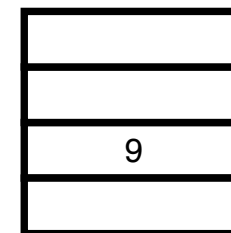
- Comment expliquer ces erreurs de calcul ?
 - 👉 les entrelacements se font au niveau du code binaire
 - 👉 couper **entre chaque instruction assembleur**



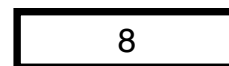
PCB t1



balance



PCB t2



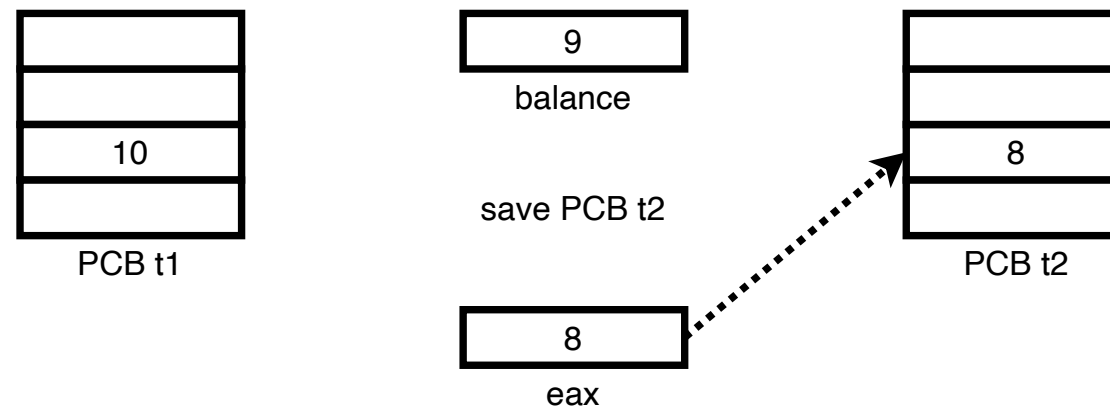
eax

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Comment expliquer ces erreurs de calcul ?
 - 👉 les entrelacements se font au niveau du code binaire
 - 👉 couper **entre chaque instruction assembleur**

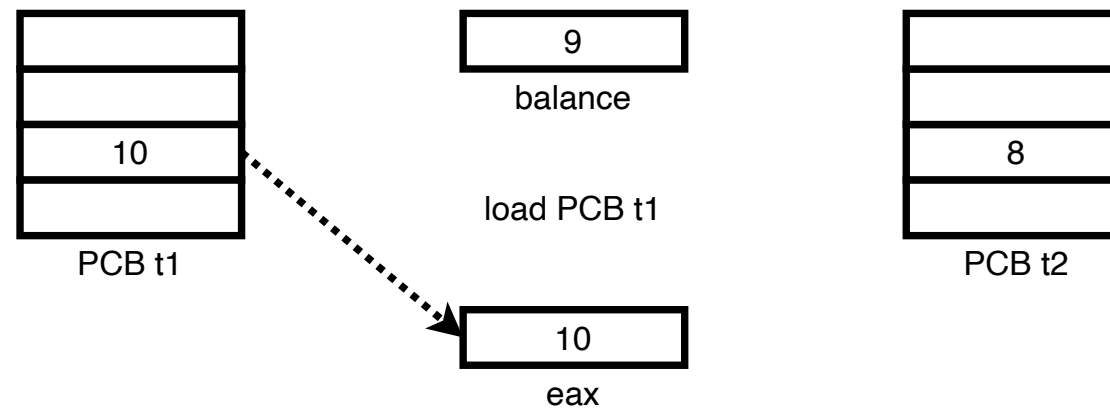


PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Comment expliquer ces erreurs de calcul ?
 - 👉 les entrelacements se font au niveau du code binaire
 - 👉 couper **entre chaque instruction assembleur**

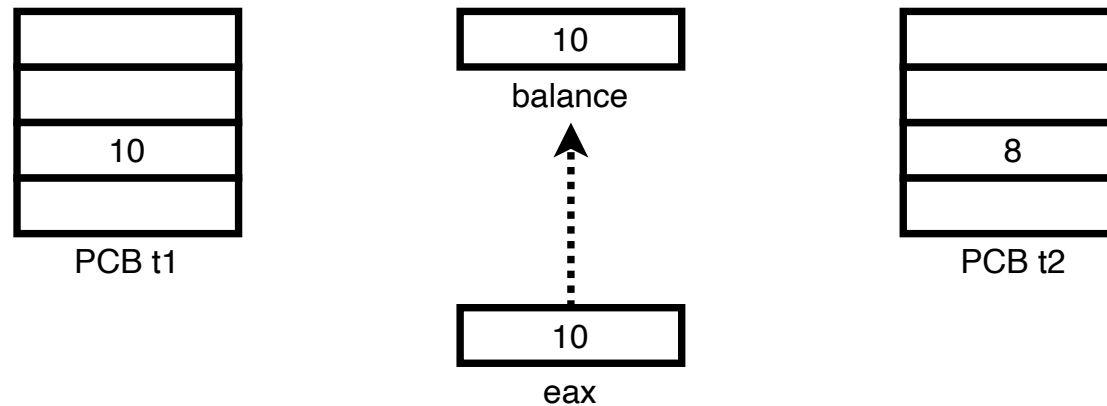


PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Comment expliquer ces erreurs de calcul ?
 - 👉 les entrelacements se font au niveau du code binaire
 - 👉 couper **entre chaque instruction assembleur**



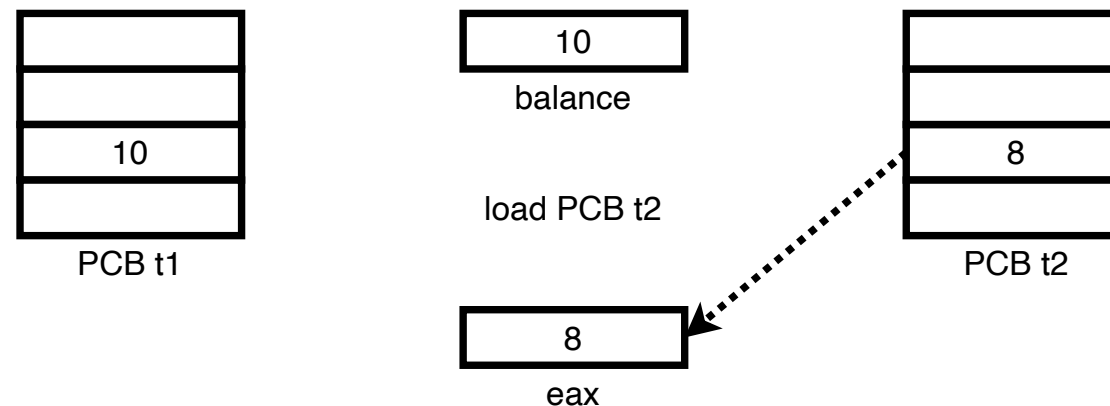
PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```



```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Comment expliquer ces erreurs de calcul ?
 - 👉 les entrelacements se font au niveau du code binaire
 - 👉 couper **entre chaque instruction assembleur**



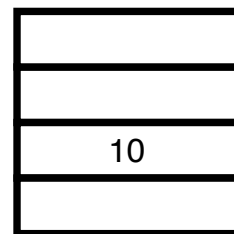
PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

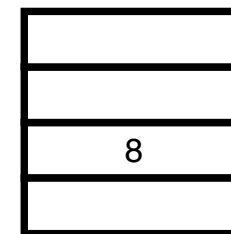
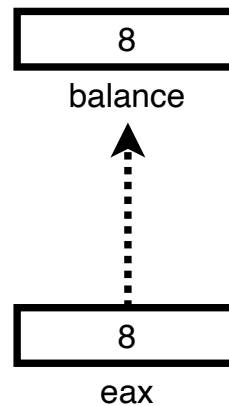


```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

- Comment expliquer ces erreurs de calcul ?
 - 👉 les entrelacements se font au niveau du code binaire
 - 👉 couper **entre chaque instruction assembleur**



PCB t1



PCB t2

PROBLÈME DE LA CONCURRENCE

```
1 ;balance = balance + 1
2 mov eax, balance
3 add eax, 1
4 mov balance, eax
```

```
1 ;balance = balance - 1
2 mov eax, balance
3 sub eax, 1
4 mov balance, eax
```

✗ Après une itération → balance = 8

PROBLÈME DE LA CONCURRENCE

- **Situation de compétition**
👉 **erreur** dépendant de l'enchaînement temporel d'événements impliquant une ressource partagée

PROBLÈME DE LA CONCURRENCE

- **Situation de compétition**
 - 👉 **erreur** dépendant de l'enchaînement temporel d'événements impliquant une ressource partagée
 - 👉 non déterministe

PROBLÈME DE LA CONCURRENCE

- **Situation de compétition**

👉 **erreur** dépendant de l'enchaînement temporel d'événements impliquant une ressource partagée

👉 non déterministe

❌ difficile à détecter (**tests**)

PROBLÈME DE LA CONCURRENCE

- **Situation de compétition**

👉 **erreur** dépendant de l'enchaînement temporel d'événements impliquant une ressource partagée

👉 non déterministe

✗ difficile à détecter (**tests**)

✗ difficile à corriger (**debug**)

PLAN

- > Sections critiques
- > Solution programmée
- > Méthodes de synchronisation
- > Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

SECTION CRITIQUE

SECTION CRITIQUE

- Lorsqu'il y a des **variables partagées**, il existe des **portions de code** qu'on ne veut pas pouvoir interrompre.

SECTION CRITIQUE

- Lorsqu'il y a des **variables partagées**, il existe des **portions de code** qu'on ne veut pas pouvoir interrompre.
👉 des **zones du code** qui manipulent des **ressources partagées**

SECTION CRITIQUE

- Lorsqu'il y a des **variables partagées**, il existe des **portions de code** qu'on ne veut pas pouvoir interrompre.
 - 👉 des **zones du code** qui manipulent des **ressources partagées**
 - 👉 ces zones sont appelées **sections critiques**

EXCLUSION MUTUELLE

EXCLUSION MUTUELLE

- Lorsqu'on déclare une **section critique**, on doit garantir qu'**au plus un seul** processus/thread est dans la section critique.

EXCLUSION MUTUELLE

- Lorsqu'on déclare une **section critique**, on doit garantir qu'**au plus un seul** processus/thread est dans la section critique.
👉 besoin de gérer l'**exclusion mutuelle** des **sections critiques**.

EXCLUSION MUTUELLE

- Lorsqu'on déclare une **section critique**, on doit garantir qu'**au plus un seul** processus/thread est dans la section critique.
 - 👉 besoin de gérer l'**exclusion mutuelle** des **sections critiques**.
 - 👉 **une seule section critique** peut être exécutée à la fois.

EXCLUSION MUTUELLE

- Lorsqu'on déclare une **section critique**, on doit garantir qu'**au plus un seul** processus/thread est dans la section critique.
 - 👉 besoin de gérer l'**exclusion mutuelle** des **sections critiques**.
 - 👉 **une seule section critique** peut être exécutée à la fois.
- Pour résoudre ce problème il faut un **système de verrou**.

LES PROPRIÉTÉS À RESPECTER

LES PROPRIÉTÉS À RESPECTER

1. **Exclusion mutuelle** : si une thread effectue sa section critique, alors **aucune autre thread** ne peut entrer en section critique.

LES PROPRIÉTÉS À RESPECTER

1. **Exclusion mutuelle** : si une thread effectue sa section critique, alors **aucune autre thread** ne peut entrer en section critique.
2. **Déroulement** : une thread qui souhaite entrer en section critique **ne peut pas décider** qui doit rentrer en section critique.

LES PROPRIÉTÉS À RESPECTER

1. **Exclusion mutuelle** : si une thread effectue sa section critique, alors **aucune autre thread** ne peut entrer en section critique.
2. **Déroulement** : une thread qui souhaite entrer en section critique **ne peut pas décider** qui doit rentrer en section critique.
3. **Vivacité** : une thread qui souhaite entrer en section critique **y rentre en temps borné**.

LES PROPRIÉTÉS À RESPECTER

1. **Exclusion mutuelle** : si une thread effectue sa section critique, alors **aucune autre thread** ne peut entrer en section critique.
2. **Déroulement** : une thread qui souhaite entrer en section critique **ne peut pas décider** qui doit rentrer en section critique.
3. **Vivacité** : une thread qui souhaite entrer en section critique **y rentre en temps borné**.

Définir des **mécanismes** qui garantissent ces trois propriétés

PLAN

- Sections critiques
- Solution programmée
- Méthodes de synchronisation
- Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

UNE SOLUTION POUR 2 THREADS

UNE SOLUTION POUR 2 THREADS

- Principes

UNE SOLUTION POUR 2 THREADS

- Principes
 - un contrôleur central

UNE SOLUTION POUR 2 THREADS

- **Principes**
 - un contrôleur central
 - met en attente

UNE SOLUTION POUR 2 THREADS

- **Principes**
 - un contrôleur central
 - met en attente
 - autorise les SC chacune à son tour

UNE SOLUTION POUR 2 THREADS

- **Principes**
 - un contrôleur central
 - met en attente
 - autorise les SC chacune à son tour
- 👉 respecte les 3 objectifs de la synchronisation

UNE SOLUTION POUR 2 THREADS

```
1 interface Mutex {  
2     abstract void commencerSC(int id);  
3     abstract void finirSC(int id);  
4     // id dans {0,1} numéro de la thread  
5 }
```

```
1 //... code non critique  
2 my_mutex.commencerSC(my_id);  
3 //... code critique  
4 my_mutex.finirSC(my_id);  
5 //... code non critique
```

UNE SOLUTION POUR 2 THREADS

```
1 interface Mutex {  
2     abstract void commencerSC(int id);  
3     abstract void finirSC(int id);  
4     // id dans {0,1} numéro de la thread  
5 }
```

```
1 //... code non critique  
2 my_mutex.commencerSC(my_id);  
3 //... code critique  
4 my_mutex.finirSC(my_id);  
5 //... code non critique
```

- Lorsque t_i invoque **commencerSC(i)**:
 - Vérifier que t_{1-i} n'est pas en SC (sinon, attendre).
 - Noter que t_i est en SC

UNE SOLUTION POUR 2 THREADS

```
1 interface Mutex {  
2     abstract void commencerSC(int id);  
3     abstract void finirSC(int id);  
4     // id dans {0,1} numéro de la thread  
5 }
```

```
1 //... code non critique  
2 my_mutex.commencerSC(my_id);  
3 //... code critique  
4 my_mutex.finirSC(my_id);  
5 //... code non critique
```

- Lorsque t_i invoque **commencerSC(i)**:
 - Vérifier que t_{1-i} n'est pas en SC (sinon, attendre).
 - Noter que t_i est en SC
- Lorsque t_i invoque **finirSC(i)**:
 - Noter que t_i n'est plus en SC

UNE PREMIÈRE IMPLÉMENTATION

```
1 class Mutex1 implements Mutex{
2     boolean[] est_SC = {false,false}; //pour noter qui est en SC
3
4     void commencerSC(int id){
5         while (est_SC[1-id])
6             ; //attendre ...
7         est_SC[id]=true;
8     }
9
10    void finirSC(int id){
11        est_SC[id]=false;
12    }
13 }
```

UNE PREMIÈRE IMPLÉMENTATION

```
1 class Mutex1 implements Mutex{
2     boolean[] est_SC = {false,false}; //pour noter qui est en SC
3
4     void commencerSC(int id){
5         while (est_SC[1-id])
6             ; //attendre ...
7         est_SC[id]=true;
8     }
9
10    void finirSC(int id){
11        est_SC[id]=false;
12    }
13 }
```

UNE PREMIÈRE IMPLÉMENTATION

```
1 class Mutex1 implements Mutex{
2     boolean[] est_SC = {false,false}; //pour noter qui est en SC
3
4     void commencerSC(int id){
5         while (est_SC[1-id])
6             ; //attendre ...
7         est_SC[id]=true;
8     }
9
10    void finirSC(int id){
11        est_SC[id]=false;
12    }
13 }
```

UNE PREMIÈRE IMPLÉMENTATION

```
1 class Mutex1 implements Mutex{
2     boolean[] est_SC = {false,false}; //pour noter qui est en SC
3
4     void commencerSC(int id){
5         while (est_SC[1-id])
6             ; //attendre ...
7         est_SC[id]=true;
8     }
9
10    void finirSC(int id){
11        est_SC[id]=false;
12    }
13 }
```

ÉVALUATION

```
void commencerSC(int id){  
    while (est_SC[1-id])  
        ; //attendre ...  
    est_SC[id]=true;  
}
```


ÉVALUATION

```
void commencerSC(int id){  
    while (est_SC[1-id])  
        ; //attendre ...  
    est_SC[id]=true;  
}
```

est_SC[0]	false
est_SC[1]	false

ÉVALUATION

id=0 🖱️

```
void commencerSC(int id){  
    while (est_SC[1-id])  
        ; //attendre ...  
    est_SC[id]=true;  
}
```

est_SC[0]	false
est_SC[1]	false

ÉVALUATION

id=0 🖱️

```
void commencerSC(int id){  
    while (est_SC[1-id])  
        ; //attendre ...  
    est_SC[id]=true;  
}
```

est_SC[0]

false

est_SC[1]

false

ÉVALUATION

id=0 🖱️

```
void commencerSC(int id){  
    while (est_SC[1-id])  
        ; //attendre ...  
    est_SC[id]=true;  
}
```

est_SC[0]

false

est_SC[1]

false

ÉVALUATION



```
void commencerSC(int id){  
    while (est_SC[1-id])  
        ; //attendre ...  
    est_SC[id]=true;  
}
```

id=0 🙋

est_SC[0]	false
est_SC[1]	false

ÉVALUATION



```
void commencerSC(int id){  
    while (est_SC[1-id])  
        ; //attendre ...  
    est_SC[id]=true;  
}
```

id=0   id=1

est_SC[0]	false
est_SC[1]	false

ÉVALUATION



```
void commencerSC(int id){  
    while (est_SC[1-id])  
        ; //attendre ...  
    est_SC[id]=true;  
}
```

id=0   id=1

est_SC[0]	false
est_SC[1]	false

ÉVALUATION



```
void commencerSC(int id){  
    while (est_SC[1-id])  
        ; //attendre ...  
    est_SC[id]=true;  
}
```

id=0   id=1

est_SC[0]	false
est_SC[1]	false

ÉVALUATION


```
void commencerSC(int id){  
    while (est_SC[1-id])  
        ; //attendre ...  
    est_SC[id]=true;  
}
```

id=0   id=1

est_SC[0]	false
est_SC[1]	false

ÉVALUATION

```
void commencerSC(int id){  
    while (est_SC[1-id])  
        ; //attendre ...  
    est_SC[id]=true;  
}
```

id=0 

 id=1

est_SC[0]

true

est_SC[1]

false

ÉVALUATION

```
void commencerSC(int id){  
    while (est_SC[1-id])  
        ; //attendre ...  
    est_SC[id]=true;  
}
```

id=0 🙋

🙋 id=1

est_SC[0]	true
est_SC[1]	true

ÉVALUATION

```
void commencerSC(int id){  
    while (est_SC[1-id])  
        ; //attendre ...  
    est_SC[id]=true;  
}
```

id=0 🙋

🙋 id=1

❌ Les deux threads sont en SC!

ÉVALUATION

```
void commencerSC(int id){  
    while (est_SC[1-id])  
        ; //attendre ...  
    est_SC[id]=true;  
}
```

id=0 🖱️

🖱️ id=1

- ❌ Les deux threads sont en SC!
- ❌ Ce code contient lui-même des SC!

UNE DEUXIÈME IMPLÉMENTATION

```
1 //Mettre en attente la thread demandeuse si l'autre est déjà en SC
2 class Mutex2 implements Mutex{
3     boolean[] est_SC = {false, false};
4
5     void commencerSC(int id){
6         est_SC[id]=true; //on commence par noter la SC !
7         while (est_SC[1-id])
8             ;
9     }
10
11     void finirSC(int id){
12         est_SC[id]=false;
13     }
14 }
```

UNE DEUXIÈME IMPLÉMENTATION

```
1 //Mettre en attente la thread demandeuse si l'autre est déjà en SC
2 class Mutex2 implements Mutex{
3     boolean[] est_SC = {false,false};
4
5     void commencerSC(int id){
6         est_SC[id]=true; //on commence par noter la SC !
7         while (est_SC[1-id])
8             ;
9     }
10
11     void finirSC(int id){
12         est_SC[id]=false;
13     }
14 }
```

ÉVALUATION

```
void commencerSC(int id){  
    est_SC[id]=true;  
    while (est_SC[1-id])  
        ;  
}
```


ÉVALUATION

```
void commencerSC(int id){  
    est_SC[id]=true;  
    while (est_SC[1-id])  
        ;  
}
```

est_SC[0]	false
est_SC[1]	false

ÉVALUATION

id=0 🖐️

```
void commencerSC(int id){  
    est_SC[id]=true;  
    while (est_SC[1-id])  
        ;  
}
```

est_SC[0]	false
est_SC[1]	false

ÉVALUATION

id=0 📌

```
void commencerSC(int id){  
    est_SC[id]=true;  
    while (est_SC[1-id])  
        ;  
}
```

est_SC[0]	false
est_SC[1]	false

ÉVALUATION

id=0 🖱️

```
void commencerSC(int id){  
    est_SC[id]=true;  
    while (est_SC[1-id])  
        ;  
}
```

est_SC[0]



true

est_SC[1]

false

ÉVALUATION



```
void commencerSC(int id){  
    est_SC[id]=true;  
    while (est_SC[1-id])  
        ;  
}
```

id=0   id=1

est_SC[0]	true
est_SC[1]	false


ÉVALUATION

```
void commencerSC(int id){  
    est_SC[id]=true;  
    while (est_SC[1-id])  
        ;  
}
```


id=0   id=1

est_SC[0]	true
est_SC[1]	false

ÉVALUATION

id=0 



```
void commencerSC(int id){  
    est_SC[id]=true;  
    while (est_SC[1-id])  
        ;  
}
```

 id=1

est_SC[0]	true
est_SC[1]	true

ÉVALUATION



```
void commencerSC(int id){  
    est_SC[id]=true;  
    while (est_SC[1-id])  
        ;  
}
```

id=0   id=1

est_SC[0]	true
est_SC[1]	true

ÉVALUATION



```
void commencerSC(int id){  
    est_SC[id]=true;  
    while (est_SC[1-id])  
        ;  
}
```

id=0   id=1

est_SC[0]	true
est_SC[1]	true

ÉVALUATION

```
void commencerSC(int id){  
    est_SC[id]=true;  
    while (est_SC[1-id])  
        ;  
}
```

id=0   id=1

✗ Problème → interblocage

UNE TROISIÈME IMPLÉMENTATION

```
1 //Ajout de tours de priorité
2 class Mutex3 implements Mutex{
3     boolean[] est_SC = {false,false};
4     int tour = 0;
5
6     void commencerSC(int id){
7         tour=1-id;
8         est_SC[id]=true;
9         while (est_SC[1-id] && tour==1-id)
10             ;
11     }
12
13     void finirSC(int id){
14         est_SC[id]=false;
15     }
16 }
```

UNE TROISIÈME IMPLÉMENTATION

```
1 //Ajout de tours de priorité
2 class Mutex3 implements Mutex{
3     boolean[] est_SC = {false,false};
4     int tour = 0;
5
6     void commencerSC(int id){
7         tour=1-id;
8         est_SC[id]=true;
9         while (est_SC[1-id] && tour==1-id)
10             ;
11     }
12
13     void finirSC(int id){
14         est_SC[id]=false;
15     }
16 }
```

UNE TROISIÈME IMPLÉMENTATION

```
1 //Ajout de tours de priorité
2 class Mutex3 implements Mutex{
3     boolean[] est_SC = {false,false};
4     int tour = 0;
5
6     void commencerSC(int id){
7         tour=1-id;
8         est_SC[id]=true;
9         while (est_SC[1-id] && tour==1-id)
10             ;
11     }
12
13     void finirSC(int id){
14         est_SC[id]=false;
15     }
16 }
```

UNE TROISIÈME IMPLÉMENTATION

```
1 //Ajout de tours de priorité
2 class Mutex3 implements Mutex{
3     boolean[] est_SC = {false,false};
4     int tour = 0;
5
6     void commencerSC(int id){
7         tour=1-id;
8         est_SC[id]=true;
9         while (est_SC[1-id] && tour==1-id)
10             ;
11     }
12
13     void finirSC(int id){
14         est_SC[id]=false;
15     }
16 }
```

ÉVALUATION

```
void commencerSC(int id){  
    tour=1-id;  
    est_SC[id]=true;  
    while (est_SC[1-id] && tour==1-id)  
        ;  
}
```

ÉVALUATION

```
void commencerSC(int id){  
    tour=1-id;  
    est_SC[id]=true;  
    while (est_SC[1-id] && tour==1-id)  
        ;  
}
```

est_SC[0]	false
est_SC[1]	false
tour	0

ÉVALUATION

id=0 🖱️

```
void commencerSC(int id){  
    tour=1-id;  
    est_SC[id]=true;  
    while (est_SC[1-id] && tour==1-id)  
        ;  
}
```

est_SC[0]	false
est_SC[1]	false
tour	0

ÉVALUATION

id=0 🖱️

```
void commencerSC(int id){  
    tour=1-id;  
    est_SC[id]=true;  
    while (est_SC[1-id] && tour==1-id)  
        ;  
}
```

est_SC[0]	false
est_SC[1]	false
tour	0

ÉVALUATION

id=0 🖱️

```
void commencerSC(int id){  
    tour=1-id;  
    est_SC[id]=true;  
    while (est_SC[1-id] && tour==1-id)  
        ;  
}
```

est_SC[0]

false

est_SC[1]

false

tour

1

ÉVALUATION

```
void commencerSC(int id){  
    tour=1-id;  
    est_SC[id]=true;  
    while (est_SC[1-id] && tour==1-id)  
        ;  
}
```

id=0 🖱️

est_SC[0]	true
est_SC[1]	false
tour	1

ÉVALUATION

```
void commencerSC(int id){  
    tour=1-id;  
    est_SC[id]=true;  
    while (est_SC[1-id] && tour==1-id)  
        ;  
}
```

id=0 →

→ id=1

est_SC[0]	true
est_SC[1]	false
tour	1

ÉVALUATION

```
void commencerSC(int id){  
    tour=1-id;  
    est_SC[id]=true;  
    while (est_SC[1-id] && tour==1-id)  
        ;  
}
```



id=0 →

→ id=1

est_SC[0]	true
est_SC[1]	false
tour	1

ÉVALUATION



```
void commencerSC(int id){  
    tour=1-id;  
    est_SC[id]=true;  
    while (est_SC[1-id] && tour==1-id)  
        ;  
}
```

id=0   id=1

est_SC[0]	true
est_SC[1]	false
tour	0

ÉVALUATION



```
void commencerSC(int id){  
    tour=1-id;  
    est_SC[id]=true;  
    while (est_SC[1-id] && tour==1-id)  
        ;  
}
```

id=0   id=1

est_SC[0]	true
est_SC[1]	true
tour	0

ÉVALUATION



```
void commencerSC(int id){  
    tour=1-id;  
    est_SC[id]=true;  
    while (est_SC[1-id] && tour==1-id)  
        ;  
}
```

id=0   id=1

est_SC[0]	true
est_SC[1]	true
tour	0

ÉVALUATION

```
void commencerSC(int id){  
    tour=1-id;  
    est_SC[id]=true;  
    while (est_SC[1-id] && tour==1-id)  
        ;  
}
```

id=0   id=1

est_SC[0]	true
est_SC[1]	true
tour	0

ÉVALUATION

```
void commencerSC(int id){  
    tour=1-id;  
    est_SC[id]=true;  
    while (est_SC[1-id] && tour==1-id)  
        ;  
}
```

id=0



id=1

est_SC[0]

true

est_SC[1]



true

tour

0

ÉVALUATION

```
void commencerSC(int id){  
    tour=1-id;  
    est_SC[id]=true;  
    while (est_SC[1-id] && tour==1-id)  
        ;  
}
```

id=0   id=1

✓ Une seule thread est passée, l'autre est en attente

ATTENTE ACTIVE

ATTENTE ACTIVE

✗ Les processus font de l'attente active.

```
while (est_SC[1-id] && tour==1-id)  
    ;
```

ATTENTE ACTIVE

✗ Les processus font de l'attente active.

```
while (est_SC[1-id] && tour==1-id)  
    ;
```

👉 Utilisation inutile du processeur

ATTENTE ACTIVE

✗ Les processus font de l'attente active.

```
while (est_SC[1-id] && tour==1-id)  
    ;
```

👉 Utilisation inutile du processeur

✓ Des **mécanismes de haut niveau** sont offerts par les **OS**

MÉCANISMES DE HAUT NIVEAU

MÉCANISMES DE HAUT NIVEAU

Méthode	Description
Signal	Un message système est envoyé d'un processus à un autre.
Pipe	Un canal unidirectionnel ; les données émises sont accumulées dans une mémoire tampon (FIFO).
File	Lecture/Écriture dans un fichier.
Socket	Un flux de données envoyé à travers une interface réseau à un autre processus.
Mémoire Partagée	Espace de mémoire alloué à plusieurs processus.
Moniteur/Sémaphore	Une structure de synchronisation pour les processus travaillant sur des ressources partagées.

MÉCANISMES DE HAUT NIVEAU

Méthode	Description
Signal	Un message système est envoyé d'un processus à un autre.
Pipe	Un canal unidirectionnel ; les données émises sont accumulées dans une mémoire tampon (FIFO).
File	Lecture/Écriture dans un fichier.
Socket	Un flux de données envoyé à travers une interface réseau à un autre processus.
Mémoire Partagée	Espace de mémoire alloué à plusieurs processus.
Moniteur/Sémaphore	Une structure de synchronisation pour les processus travaillant sur des ressources partagées.

PLAN

- Sections critiques
- Solution programmée
- Méthodes de synchronisation
- Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

MONITEUR

MONITEUR

- **Un moniteur** est un module constitué de:
 - objets inaccessibles de l'extérieur
 - fonctions manipulant l'objet en exclusion mutuelle

MONITEUR

- **Un moniteur** est un module constitué de:
 - objets inaccessibles de l'extérieur
 - fonctions manipulant l'objet en exclusion mutuelle
- Dans la **JVM** de **Java**, on peut définir que des méthodes d'une classe sont en exclusion mutuelle.
 - mot clef **synchronized** : utilisation d'un verrou (lock)

MONITEUR

- **Un moniteur** est un module constitué de:
 - objets inaccessibles de l'extérieur
 - fonctions manipulant l'objet en exclusion mutuelle
 - Dans la **JVM** de **Java**, on peut définir que des méthodes d'une classe sont en exclusion mutuelle.
 - mot clef **synchronized** : utilisation d'un verrou (lock)
- 👉 une seule thread dans un bloc **synchronized** pour un objet donné

MONITEURS EN JAVA

MONITEURS EN JAVA

- Une thread qui possède un verrou peut rentrer dans n'importe quelle méthode (**verrou récursif**).

MONITEURS EN JAVA

- Une thread qui possède un verrou peut rentrer dans n'importe quelle méthode (**verrou récursif**).
- Une thread peut **verrouiller plusieurs objets** (risque d'interblocage).

MONITEURS EN JAVA

- Une thread qui possède un verrou peut rentrer dans n'importe quelle méthode (**verrou récursif**).
- Une thread peut **verrouiller plusieurs objets** (risque d'interblocage).
- Tout bloc **non synchronized** peut être appelé par n'importe qui n'importe quand.

EXAMPLE

```
1 public class Account {  
2     private int value;  
3  
4     public Account(int i) {  
5         this.value = i;  
6     }  
7  
8     synchronized public void add(int v){  
9         this.value = this.value + v;  
10    }  
11  
12    synchronized public void sub(int v){  
13        this.value = this.value - v;  
14    }  
15 }
```

EXAMPLE

```
1 public class Account {  
2     private int value;  
3  
4     public Account(int i) {  
5         this.value = i;  
6     }  
7  
8     synchronized public void add(int v){  
9         this.value = this.value + v;  
10    }  
11  
12    synchronized public void sub(int v){  
13        this.value = this.value - v;  
14    }  
15 }
```

EXAMPLE

```
1 public class Account {  
2     private int value;  
3  
4     public Account(int i) {  
5         this.value = i;  
6     }  
7  
8     synchronized public void add(int v){  
9         this.value = this.value + v;  
10    }  
11  
12    synchronized public void sub(int v){  
13        this.value = this.value - v;  
14    }  
15 }
```

EXAMPLE

```
1 public class Account {  
2     private int value;  
3  
4     public Account(int i) {  
5         this.value = i;  
6     }  
7  
8     synchronized public void add(int v){  
9         this.value = this.value + v;  
10    }  
11  
12    synchronized public void sub(int v){  
13        this.value = this.value - v;  
14    }  
15 }
```


SÉMAPHORE - DIJKSTRA - 1962

SÉMAPHORE - DIJKSTRA - 1962

- Un sémaphore définit un objet partagé

SÉMAPHORE - DIJKSTRA - 1962

- Un **sémaphore** définit un objet partagé
 - Qu'on peut acquérir;

SÉMAPHORE - DIJKSTRA - 1962

- **Un sémaphore** définit un objet partagé
 - Qu'on peut acquérir;
 - Qui met en attente ceux qui le demandent;

SÉMAPHORE - DIJKSTRA - 1962

- **Un sémaphore** définit un objet partagé
 - Qu'on peut acquérir;
 - Qui met en attente ceux qui le demandent;
 - Qui donne la main dans l'ordre des demandes.

SÉMAPHORE - DIJKSTRA - 1962

- **Un sémaphore** définit un objet partagé
 - Qu'on peut acquérir;
 - Qui met en attente ceux qui le demandent;
 - Qui donne la main dans l'ordre des demandes.
- Toutes les threads en **concurrence sur une ressource** partagent **un même sémaphore**

SÉMAPHORE - DIJKSTRA - 1962

- **Un sémaphore** définit un objet partagé
 - Qu'on peut acquérir;
 - Qui met en attente ceux qui le demandent;
 - Qui donne la main dans l'ordre des demandes.
- Toutes les threads en **concurrence sur une ressource** partagent **un même sémaphore**
 - On **acquiert** le sémaphore avant d'entrer en SC;

SÉMAPHORE - DIJKSTRA - 1962

- **Un sémaphore** définit un objet partagé
 - Qu'on peut acquérir;
 - Qui met en attente ceux qui le demandent;
 - Qui donne la main dans l'ordre des demandes.
- Toutes les threads en **concurrence sur une ressource** partagent **un même sémaphore**
 - On **acquiert** le sémaphore avant d'entrer en SC;
 - On **relâche** le sémaphore en sortant de la SC.

UTILISATION DU SÉMAPHORE

```
1 Semaphore s = new Semaphore();  
2 // ... code non-critique ...  
3 s.acquire();  
4 // ... code critique ...  
5 s.release();  
6 // ... code non-critique ...
```

UTILISATION DU SÉMAPHORE

```
1 Semaphore s = new Semaphore();  
2 // ... code non-critique ...  
3 s.acquire();  
4 // ... code critique ...  
5 s.release();  
6 // ... code non-critique ...
```

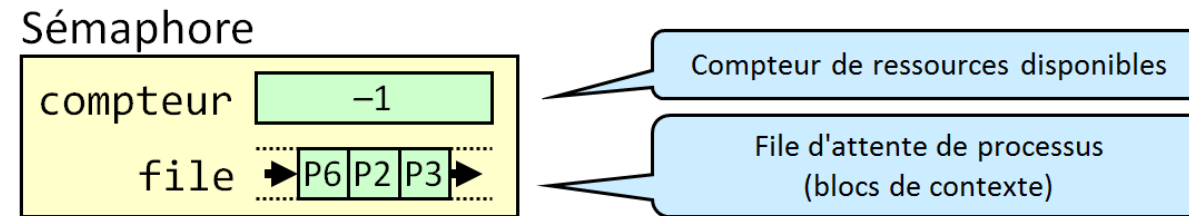
- **Vivacité** : on veut passer la main dans le bon ordre
👉 utilisation d'une file d'attente

UTILISATION DU SÉMAPHORE

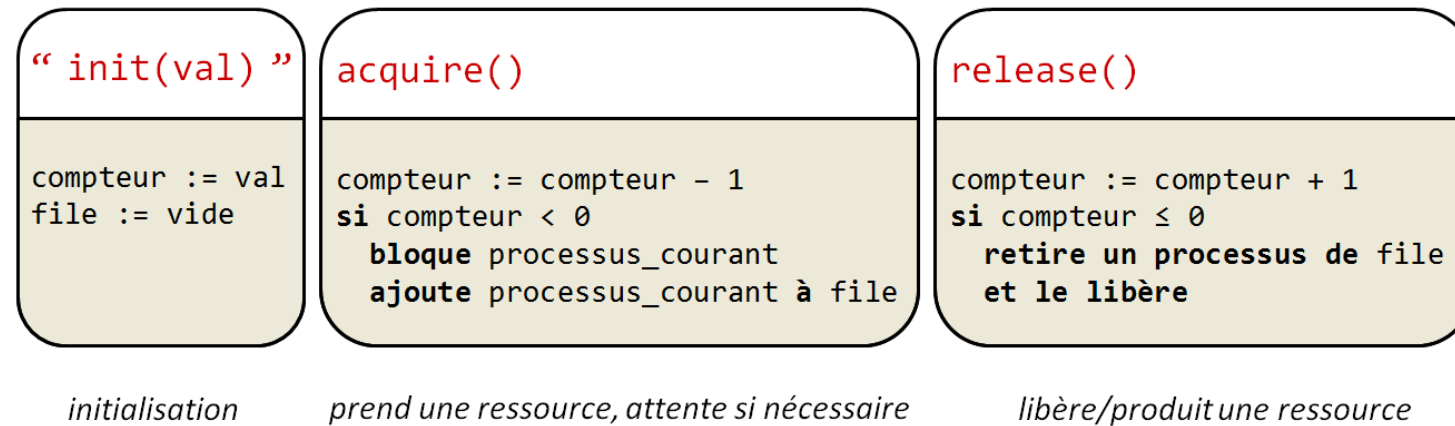
```
1 Semaphore s = new Semaphore();  
2 // ... code non-critique ...  
3 s.acquire();  
4 // ... code critique ...  
5 s.release();  
6 // ... code non-critique ...
```

- **Vivacité** : on veut passer la main dans le bon ordre
👉 utilisation d'une file d'attente
- **acquire** et **release** doivent être atomiques !
👉 elles sont elles-mêmes des SC pour le sémaphore ...

IMPLÉMENTATION DU SÉMAPHORE



3 primitives (opérations **non interruptibles = atomiques**) :



L'EXCLUSION MUTUELLE

L'EXCLUSION MUTUELLE

Un sémaphore permettant de faire de l'**exclusion mutuelle** est un sémaphore dont la valeur initiale est 1.

```
1 int cpt = 0;  
2 Semaphore mutex = new Semaphore(1);
```

L'EXCLUSION MUTUELLE

Un sémaphore permettant de faire de l'**exclusion mutuelle** est un sémaphore dont la valeur initiale est 1.

```
1 int cpt = 0;  
2 Semaphore mutex = new Semaphore(1);
```

```
1 void Processus_1(){  
2     mutex.acquire();  
3     cpt = cpt + 1;  
4     mutex.release();  
5 }
```

```
1 void Processus_2(){  
2     mutex.acquire();  
3     cpt = cpt - 1;  
4     mutex.release();  
5 }
```

DEMI RENDEZ-VOUS

DEMI RENDEZ-VOUS

Quand un processus veut s'assurer qu'un traitement a été réalisé par un autre processus avant de réaliser le sien.

DEMI RENDEZ-VOUS

Quand un processus veut s'assurer qu'un traitement a été réalisé par un autre processus avant de réaliser le sien.

```
1 Semaphore sem = new Semaphore(0);
```

DEMI RENDEZ-VOUS

Quand un processus veut s'assurer qu'un traitement a été réalisé par un autre processus avant de réaliser le sien.

```
1 Semaphore sem = new Semaphore(0);
```

```
1 void Processus_1(){  
2     traitement_1();  
3     sem.release();  
4 }
```

```
1 void Processus_2(){  
2     sem.acquire();  
3     traitement_2();  
4 }
```

RENDEZ-VOUS

RENDEZ-VOUS

Permet à deux processus de définir un point de synchronisation.

RENDEZ-VOUS

Permet à deux processus de définir un point de synchronisation.

```
1 Semaphore sem1 = new Semaphore(0);  
2 Semaphore sem2 = new Semaphore(0);
```

RENDEZ-VOUS

Permet à deux processus de définir un point de synchronisation.

```
1 Semaphore sem1 = new Semaphore(0);  
2 Semaphore sem2 = new Semaphore(0);
```

```
1 void Processus_1(){  
2     traitement_11();  
3     sem1.release();  
4     sem2.acquire();  
5     traitement_12();  
6 }
```

```
1 void Processus_2(){  
2     traitement_21();  
3     sem2.release();  
4     sem1.acquire();  
5     traitement_22();  
6 }
```

PARTAGE DE N RESSOURCES

PARTAGE DE N RESSOURCES

Soient N ressources disponibles et P processus voulant avoir accès à au moins l'une de ces ressources ($P > N$).

PARTAGE DE N RESSOURCES

Soient N ressources disponibles et P processus voulant avoir accès à au moins l'une de ces ressources ($P > N$).

```
1 Semaphore sem = new Semaphore(N);
```

PARTAGE DE N RESSOURCES

Soient N ressources disponibles et P processus voulant avoir accès à au moins l'une de ces ressources ($P > N$).

```
1 Semaphore sem = new Semaphore(N);
```

```
1 void Processus_i(){  
2     debutTraitement();  
3     sem.acquire();  
4     utiliserLaRessource();  
5     sem.release();  
6     finTraitement();  
7 }
```

INTER-BLOCAGE (DEADLOCK)

INTER-BLOCCAGE (DEADLOCK)

```
1 Semaphore sem1 = new Semaphore(1);  
2 Semaphore sem2 = new Semaphore(1);
```

INTER-BLOCAGE (DEADLOCK)


```
1 Semaphore sem1 = new Semaphore(1);  
2 Semaphore sem2 = new Semaphore(1);
```

```
1 void Processus_1(){  
2     sem1.acquire();  
3     sem2.acquire();  
4     utiliserRessource1();  
5     utiliserRessource2();  
6     sem1.release();  
7     sem2.release();  
8 }
```

```
1 void Processus_2(){  
2     sem2.acquire();  
3     sem1.acquire();  
4     utiliserRessource2();  
5     utiliserRessource1();  
6     sem1.release();  
7     sem2.release();  
8 }
```

INTER-BLOCAGE (DEADLOCK)

```
1 Semaphore sem1 = new Semaphore(1);  
2 Semaphore sem2 = new Semaphore(1);
```

```
1  void Processus_1(){  
2     sem1.acquire();  
3     sem2.acquire();  
4     utiliserRessource1();  
5     utiliserRessource2();  
6     sem1.release();  
7     sem2.release();  
8 }
```



```
1 void Processus_2(){  
2     sem2.acquire();  
3     sem1.acquire();  
4     utiliserRessource2();  
5     utiliserRessource1();  
6     sem1.release();  
7     sem2.release();  
8 }
```

INTER-BLOCAGE (DEADLOCK)

```
1 Semaphore sem1 = new Semaphore(1);  
2 Semaphore sem2 = new Semaphore(1);
```

```
1 void Processus_1(){  
2  🖱 sem1.acquire();  
3  sem2.acquire();  
4  utiliserRessource1();  
5  utiliserRessource2();  
6  sem1.release();  
7  sem2.release();  
8 }
```



```
1 void Processus_2(){  
2  sem2.acquire();  
3  sem1.acquire();  
4  utiliserRessource2();  
5  utiliserRessource1();  
6  sem1.release();  
7  sem2.release();  
8 }
```


INTER-BLOCAGE (DEADLOCK)

```
1 Semaphore sem1 = new Semaphore(1);  
2 Semaphore sem2 = new Semaphore(1);
```

```
1 void Processus_1(){  
2  🖱️ sem1.acquire();  
3  sem2.acquire();  
4  utiliserRessource1();  
5  utiliserRessource2();  
6  sem1.release();  
7  sem2.release();  
8 }
```



```
1 void Processus_2(){  
2  sem2.acquire();  
3  sem1.acquire();  
4  utiliserRessource2();  
5  utiliserRessource1();  
6  sem1.release();  
7  sem2.release();  
8 }
```

INTER-BLOCAGE (DEADLOCK)

```
1 Semaphore sem1 = new Semaphore(1);  
2 Semaphore sem2 = new Semaphore(1);
```

```
1 void Processus_1(){  
2     sem1.acquire();  
3     sem2.acquire();  
4     utilisRessource1();  
5     utilisRessource2();  
6     sem1.release();  
7     sem2.release();  
8 }
```



```
1 void Processus_2(){  
2     sem2.acquire();  
3     sem1.acquire();  
4     utilisRessource2();  
5     utilisRessource1();  
6     sem1.release();  
7     sem2.release();  
8 }
```

INTER-BLOCAGE (DEADLOCK)

```
1 Semaphore sem1 = new Semaphore(1);  
2 Semaphore sem2 = new Semaphore(1);
```

```
1 void Processus_1(){  
2     sem1.acquire();  
3     🖱️ sem2.acquire();  
4     utiliserRessource1();  
5     utiliserRessource2();  
6     sem1.release();  
7     sem2.release();  
8 }
```

```
1 void Processus_2(){  
2     sem2.acquire();  
3     sem1.acquire();  
4     utiliserRessource2();  
5     utiliserRessource1();  
6     sem1.release();  
7     sem2.release();  
8 }
```

INTER-BLOCCAGE (DEADLOCK)

```
1 Semaphore sem1 = new Semaphore(1);  
2 Semaphore sem2 = new Semaphore(1);
```

```
1 void Processus_1(){  
2     sem1.acquire();  
3     sem2.acquire();  
4     utiliserRessource1();  
5     utiliserRessource2();  
6     sem1.release();  
7     sem2.release();  
8 }
```

```
1 void Processus_2(){  
2     sem2.acquire();  
3     sem1.acquire();  
4     utiliserRessource2();  
5     utiliserRessource1();  
6     sem1.release();  
7     sem2.release();  
8 }
```

✗ Le programmeur doit s'assurer qu'il ne crée pas d'**interblocage**

INTER-BLOCAGE (SOLUTION)

```
1 Semaphore sem1 = new Semaphore(1);  
2 Semaphore sem2 = new Semaphore(1);
```

```
1 void Processus_1(){  
2     sem1.acquire();  
3     utiliserRessource1();  
4     sem1.release();  
5     sem2.acquire();  
6     utiliserRessource2();  
7     sem2.release();  
8 }
```

```
1 void Processus_2(){  
2     sem2.acquire();  
3     utiliserRessource2();  
4     sem2.release();  
5     sem1.acquire();  
6     utiliserRessource1();  
7     sem1.release();  
8 }
```

PLAN

- Sections critiques
- Solution programmée
- Méthodes de synchronisation
- Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

SYNTHÈSE

SYNTHÈSE

- Problème d'utilisation des **ressources partagées**

SYNTHÈSE

- Problème d'utilisation des **ressources partagées**
- **Exemple** : accès à une variable partagée

SYNTHÈSE

- Problème d'utilisation des **ressources partagées**
- **Exemple** : accès à une variable partagée
- Notion de **section critique**

SYNTHÈSE

- Problème d'utilisation des **ressources partagées**
- **Exemple** : accès à une variable partagée
- Notion de **section critique**
- Propriétés d'**exclusion mutuelle**, de **déroulement** et de **vivacité**

SYNTHÈSE

- Problème d'utilisation des **ressources partagées**
- **Exemple** : accès à une variable partagée
- Notion de **section critique**
- Propriétés d'**exclusion mutuelle**, de **déroulement** et de **vivacité**
- Problème de l'**attente active**

SYNTHÈSE

- Problème d'utilisation des **ressources partagées**
- **Exemple** : accès à une variable partagée
- Notion de **section critique**
- Propriétés d'**exclusion mutuelle**, de **déroulement** et de **vivacité**
- Problème de l'**attente active**
- **Moniteurs**

SYNTHÈSE

- Problème d'utilisation des **ressources partagées**
- **Exemple** : accès à une variable partagée
- Notion de **section critique**
- Propriétés d'**exclusion mutuelle**, de **déroulement** et de **vivacité**
- Problème de l'**attente active**
- **Moniteurs**
- **Sémaphores**

FIN

- [Retour à l'accueil](#)
- [Retour au plan](#)