Introduction sur la concurrence

Cyril Rabat
cyril.rabat@univ-reims.fr

Licence 3 Informatique - Info0601 - Systèmes d'exploitation - concepts avancés

2021-2022

Cours n°8

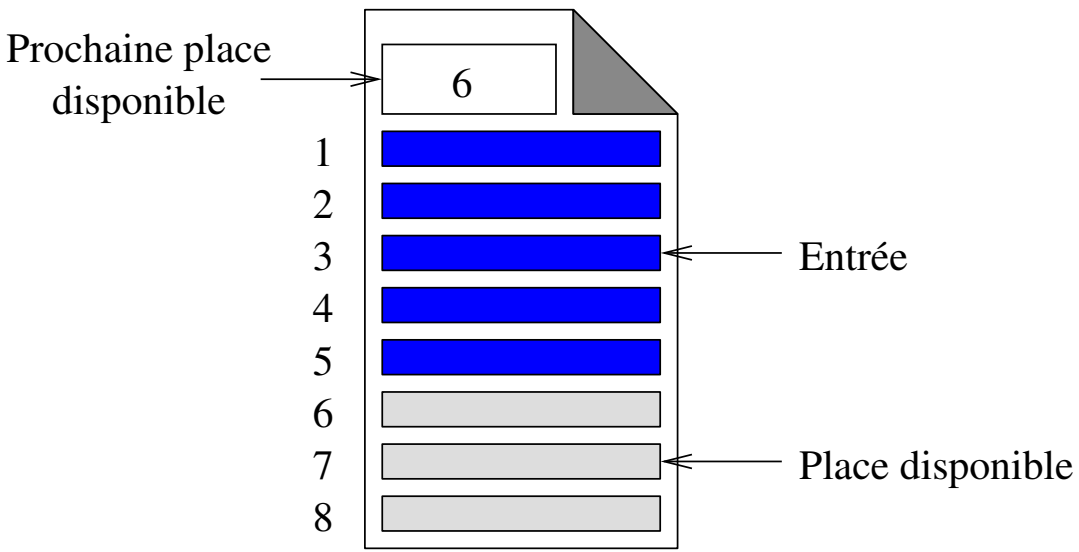
Rappels sur les problèmes de la concurrence

Version 12 janvier 2022

Table des matières

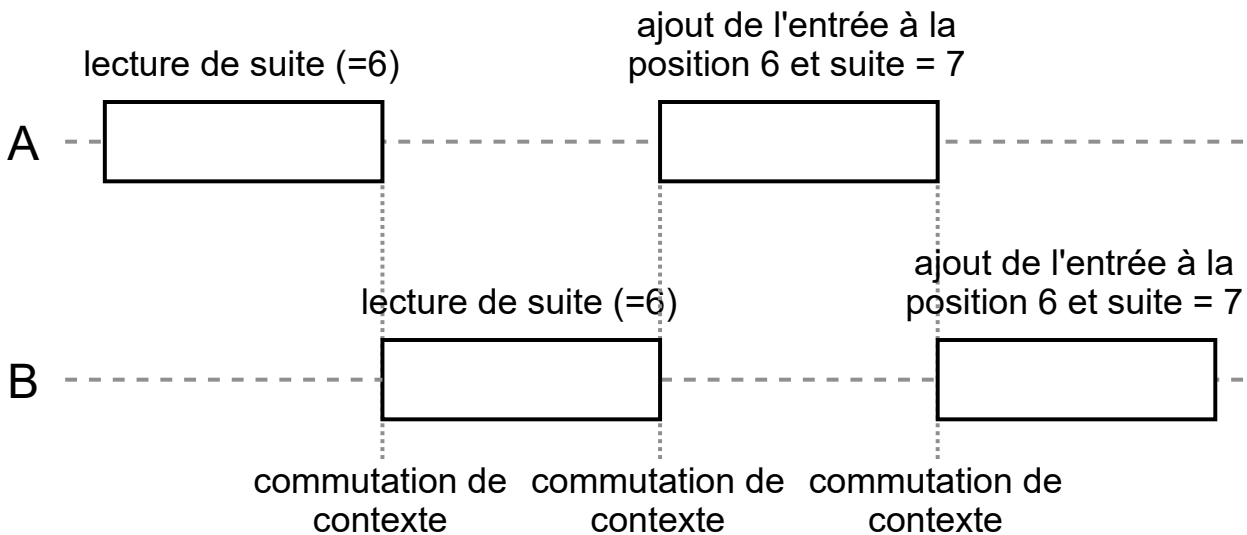
Fichier partagé (1/2)

- Supposons un annuaire stocké dans un fichier contenant plusieurs entrées de taille fixe :
  ↪ L'annuaire est accessible par plusieurs processus
- Chaque entrée est stockée à une position donnée dans le fichier
- En tête de fichier, le numéro de la prochaine place disponible est indiqué (nous l'appellerons *suite*)



Fichier partagé (2/2)

Que se passe-t-il si deux processus A et B désirent ajouter simultanément une nouvelle entrée ?



Les *threads*

- Les *threads* partagent un même espace de mémoire :
  $\hookrightarrow$ Ils ont tous accès aux variables globales
- Que se passe-t-il si plusieurs *threads* tentent de modifier la même variable ?

Exemple

- Une liste de tâches est partagée entre plusieurs *threads*
- Un *thread* est chargé d'ajouter des tâches (en fonction d'actions de l'utilisateur, par exemple)
- Plusieurs *threads* accèdent à la liste, récupèrent une tâche et la calculent

cf le cours d'INFO0604

Section critique (1/2)

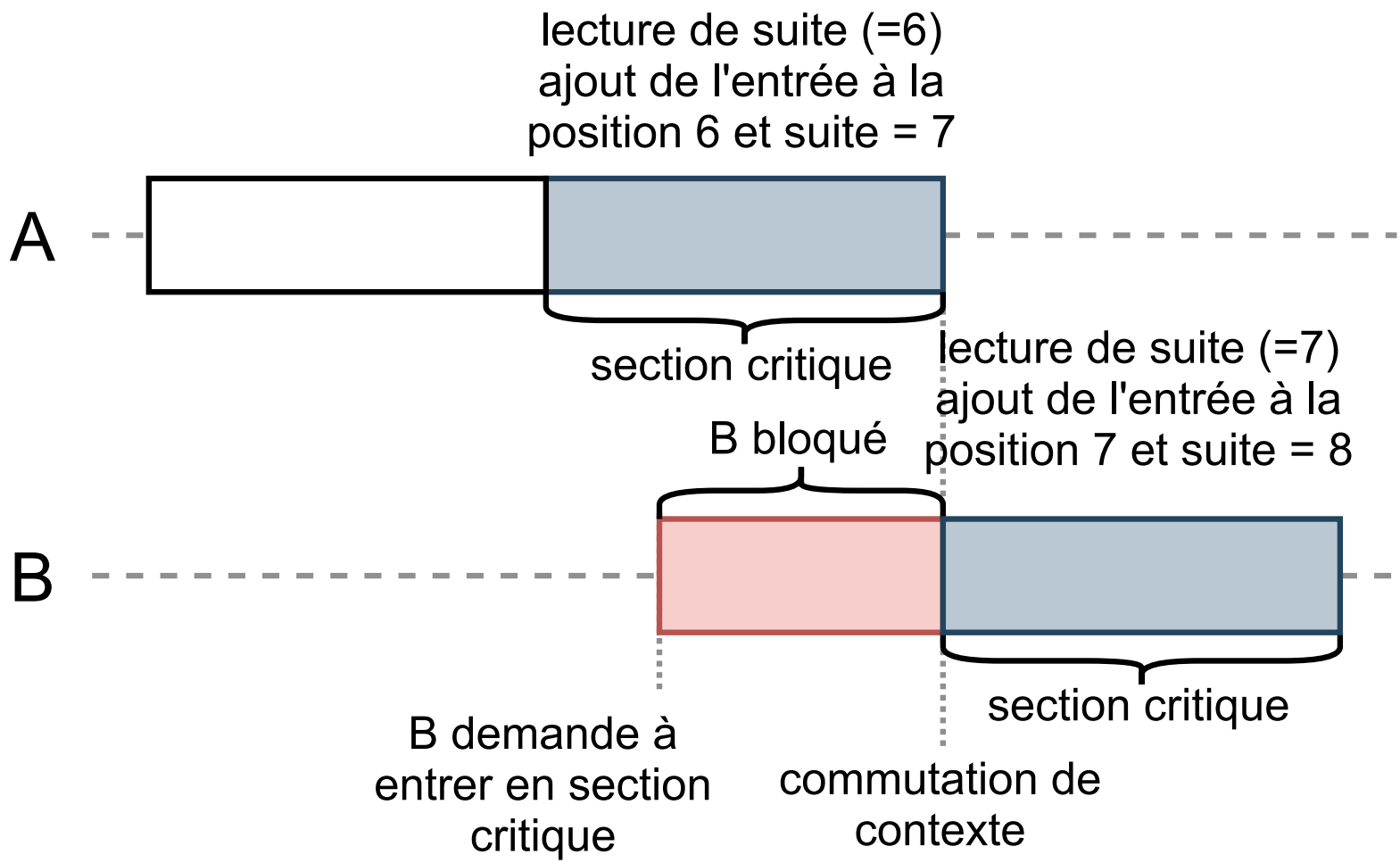
Définition : section critique

Une section critique est une portion de code qui doit être exécutée en exclusion mutuelle.

Quatre conditions

1. Un seul processus peut entrer en section critique
2. Aucune condition ne doit être posée quant à la vitesse ou au nombre de processeurs mis en œuvre
3. Aucun processus s'exécutant à l'extérieur de sa section critique ne doit bloquer d'autres processus
4. Un processus qui demande la section critique l'obtiendra en un temps fini

Section critique (2/2)



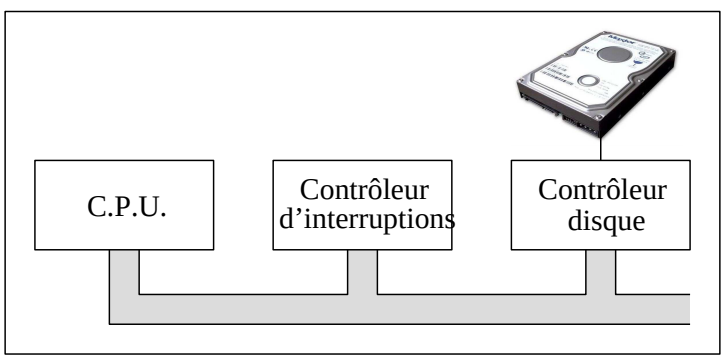
Interruption matérielle

Définition : interruption matérielle

Une *interruption matérielle* est une action déclenchée par le matériel.

Exemple d'écriture sur un disque dur

- Le pilote informe le contrôleur de périphérique (*CP*) des actions à réaliser
  ↪ Exemple : écriture d'octets
- Le *CP* démarre le périphérique et envoie les actions
- Lorsque les actions sont terminées, le *CP* le signale au contrôleur d'interruptions (*CI*)
- Si le *CI* est prêt, il en informe le *CPU*
- Le *CI* place le numéro du périphérique dans le bus pour avertir le *CPU*
- Le *CPU* décide quand il prend en charge l'interruption



Solution pour l'exclusion mutuelle

Entrée en section critique

- Lorsqu'un processus entre en section critique, il désactive les interruptions matérielles
- Dans ce cas, l'horloge ne peut pas envoyer d'interruption :
  $\Longrightarrow$ Le CPU ne peut plus basculer d'un processus à un autre

- Il faut donner la possibilité aux processus utilisateur de contrôler les interruptions (dangereux)
- Dans le cas d'un système multi-processeurs, la désactivation des interruptions n'affecte qu'un seul processeur
- Que se passe-t-il si le processus ne rend pas la main après la désactivation ?

Variable de verrou (1/2)

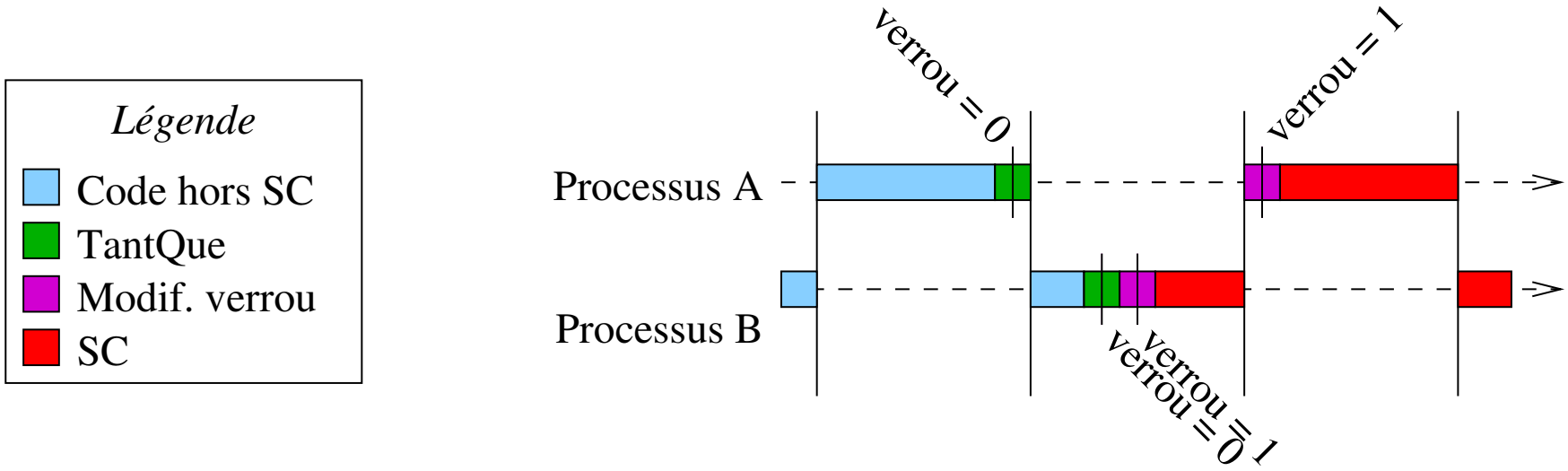
- Solution logicielle au problème de l'exclusion mutuelle
- Une variable unique (*verrou* ou *lock*) est partagée entre tous les processus
- Lorsqu'un processus veut entrer en section critique :
  1. Si le verrou est à 1, le processus attend qu'il passe à 0
  2. Si le verrou est à 0, le processus le place à 1 et entre en section critique
  3. Lorsque la section critique est finie, le verrou est replacé à 0

Variable de verrou (2/2)

```
...
/* Attente (boucle infinie) */
TantQue(verrou == 1) Faire
FinTantQue
/* Section critique */
verrou = 1;
/* Actions en section critique */
verrou = 0;
/* Fin section critique */
...
```

- Le test + la modification de la variable non atomique :
  ↪ Deux processus peuvent être simultanément en section critique

Problème avec la variable de verrou



Exemple de basculement de contexte entre le test et la modification

Alternance stricte

Explications

- Les deux processus n'entrent pas en section critique simultanément
- Ils y rentrent à tour de rôle

Code général

Code du processus A :

```
TantQue(vrai) Faire
  TantQue(verrou != 0) Faire
  FinTantQue
  // Section critique
  verrou = 1;
  // Section non critique
}
```

Code du processus B :

```
TantQue(vrai) Faire
  TantQue(verrou != 1) Faire
  FinTantQue
  // Section critique
  verrou = 0;
  // Section non critique
FinTantQue
```

Un processus ne peut pas entrer deux fois d'affilée en section critique.

Solution de Peterson

- Chaque processus est identifié par un numéro (ici 0 ou 1)
- Données partagées par tous les processus :
  - Variable `tour`
  - Tableau `etats` : indique si le processus désire entrer en section critique

Entrée en section critique :

```
Procédure entrée(numPro : entier)
Variables
  autre, tour : entiers
Début
  autre = 1 - numPro
  etats[numPro] = vrai
  tour = numPro
  TantQue (tour == numPro et
           etats[autre] == vrai) Faire
  FinTantQue
Fin
```

Sortie de section critique :

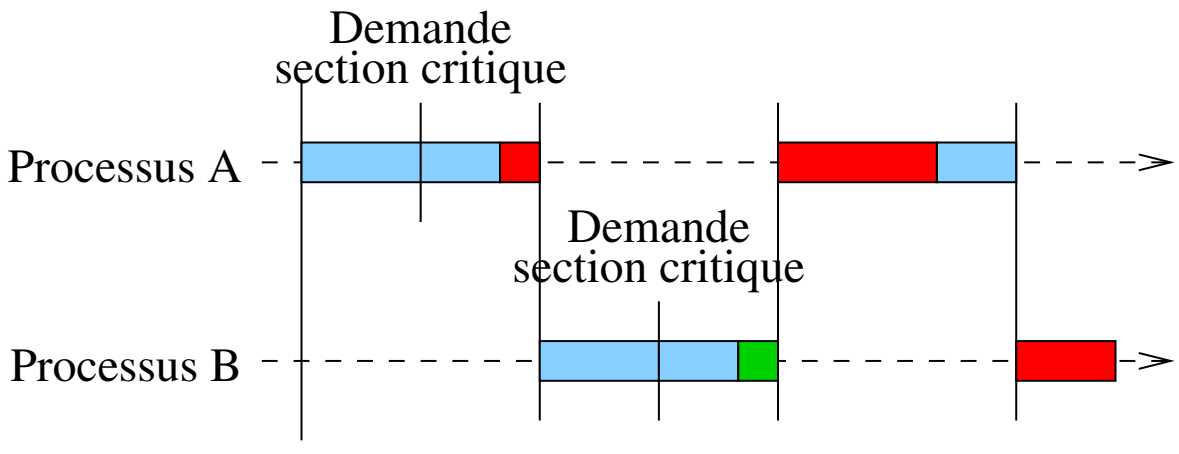
```
Procédure sortie(numPro : entier)
Début
  etats[numPro] = faux
Fin
```

Problème

Les processus sont en attente active !

Premier exemple d'exécution

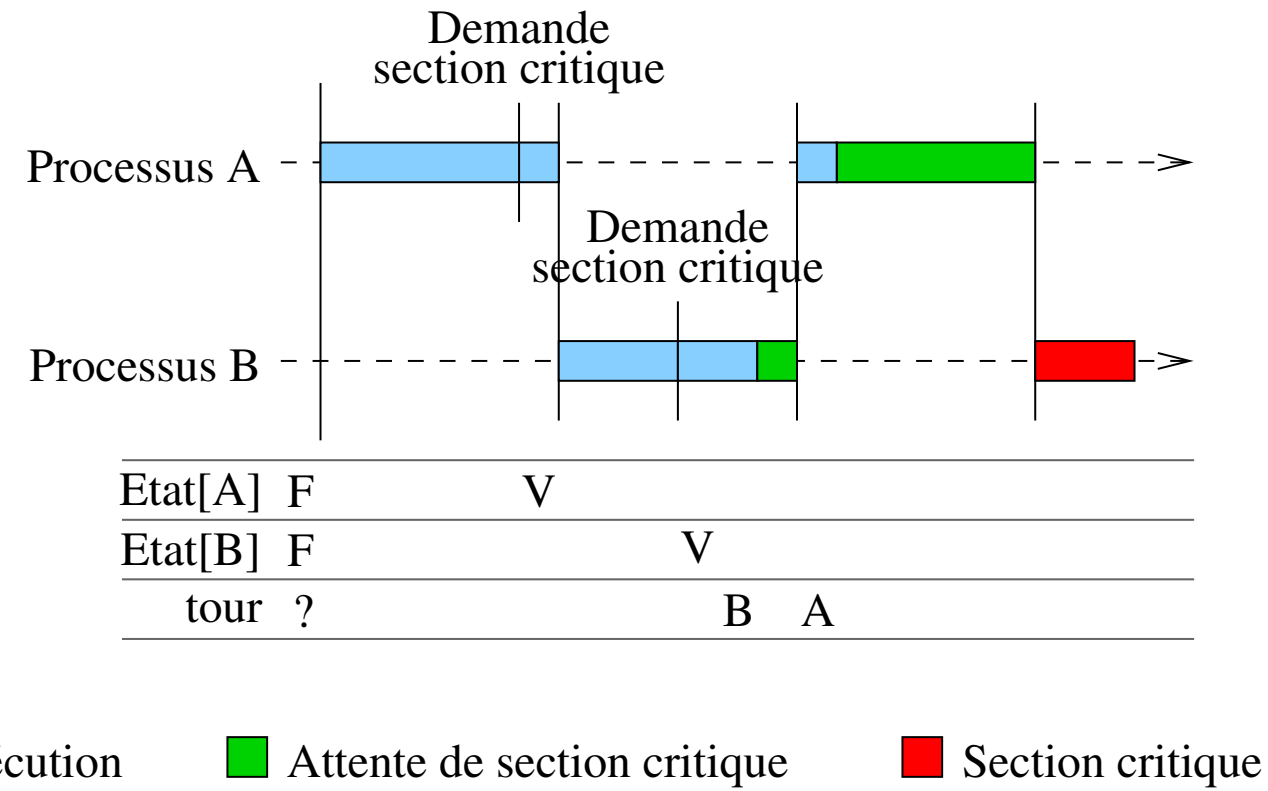
Exemples d'exécution



Exécution Attente de section critique Section critique

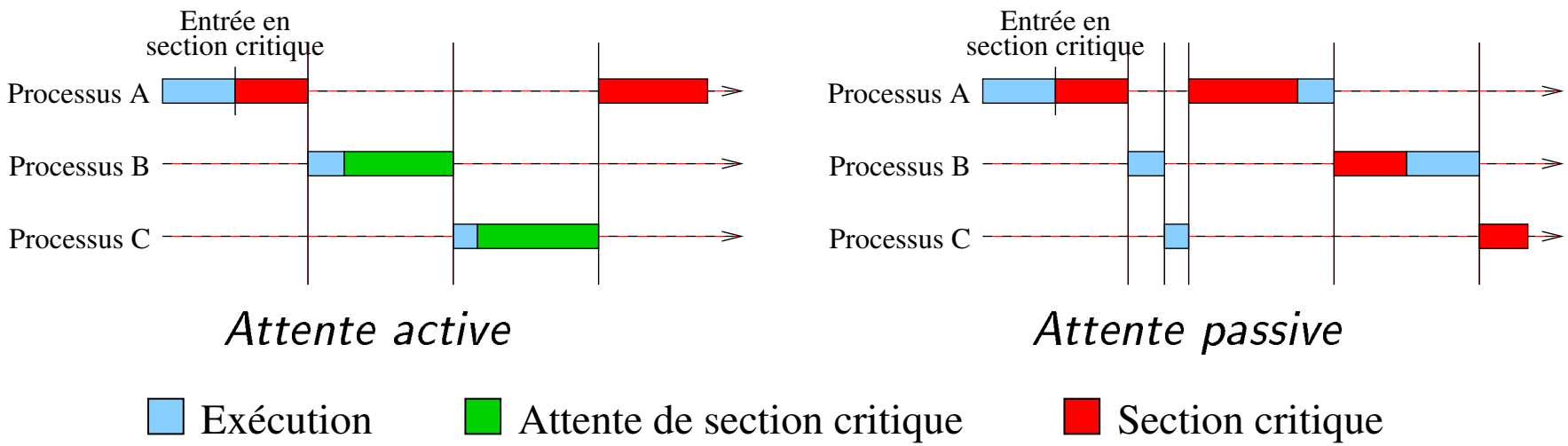
Deuxième exemple d'exécution

Exemples d'exécution



Attente active *vs* attente passive

- *Attente active* : le processus est en attente de section critique mais consomme du CPU
- *Attente passive* : le processus est en attente de section critique mais ne consomme pas de CPU



Les sémaphores

- Les solutions précédentes font appel à l'attente active :
  $\hookrightarrow$ Tant que les processus ne peuvent pas entrer en section critique, ils entrent dans une boucle qui consomme du CPU
- En 1965, *Edsger Dijkstra* propose de nouvelles variables nommées *sémaphores*
- Deux opérations *atomiques* possibles pour ces variables :
  - P(S) : test de la valeur du sémaphore S
    - Si la valeur est $\leq$ à 0, le processus est bloqué
    - Sinon, la valeur est décrémentée.
  - V(S) : incrémentation de la valeur du sémaphore S
- Initialisation avec `init(S, valeur)`

Moyen mnémotechnique

P pour *Proberen* et V pour *Verhogen*

Gérer l'ordre d'exécution avec les sémaphores

Problème

- Deux processus qui exécutent respectivement les blocs de code A et B
- Le bloc A doit être exécuté avant le bloc B
- Comment résoudre ce problème avec un ou plusieurs sémaphores ?

Indications

- Le processus 2 doit être bloqué avant le bloc B :
  $\hookrightarrow$ Débloqué dès que le bloc A est terminé

Solution

- Un seul sémaphore S initialisé à 0
  - V(S) par le processus 1 après le bloc A
  - P(S) par le processus 2 avant le bloc B
    $\hookrightarrow$ Bloqué avant le bloc B

Gérer l'exclusion mutuelle avec les sémaphores

Problème

- Deux processus qui exécutent respectivement les blocs de code A et B
- Les blocs A et B ne doivent pas être exécutés simultanément
- Comment résoudre ce problème avec un ou plusieurs sémaphores ?

Indications

- Le premier processus qui exécute son bloc ne doit pas être bloqué
- Mais le suivant si !

Solution

- Un seul sémaphore S initialisé à 1
- P(S) avant chaque bloc
- V(S) après chaque bloc

Les sémaphores : l'annuaire partagé

- On reprend l'exemple de l'annuaire partagé
- Une section critique : l'ajout d'une entrée dans l'annuaire
- Comment le résoudre avec des sémaphores ?

Code original :

```
...
Lecture de Suite
Écriture de la nouvelle entrée
Modification de Suite
...
```

Un sémaphore `S` initialisé à 1 :

```
...
P(S)
Lecture de Suite
Écriture de la nouvelle entrée
Modification de Suite
V(S)
...
```

Le graphe de précédence

- Chaque processus est représenté verticalement
- Les processus sont représentés par leurs sections : $\boxed{\text{A}}$
- Deux sections de processus exécutées en exclusion mutuelle : $\longleftrightarrow$
- Un ordre d'exécution entre deux sections : $\longrightarrow$

Exemples

- Si A doit être exécuté avant B :

$$\boxed{\text{A}} \longrightarrow \boxed{\text{B}}$$

- Si A et B doivent être exécutés en exclusion mutuelle :

$$\boxed{\text{A}} \longleftrightarrow \boxed{\text{B}}$$

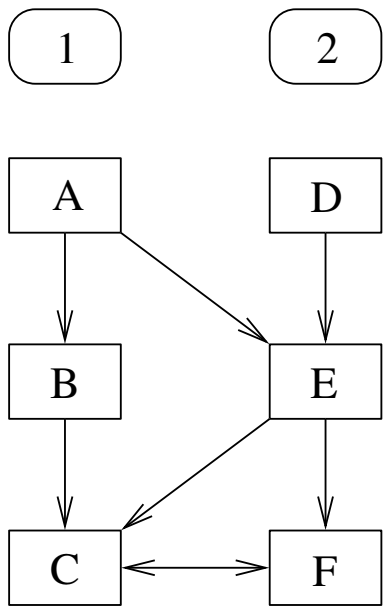
- Soit un processus contenant 3 sections A, B et C :

$$\boxed{\text{A}} \longrightarrow \boxed{\text{B}} \longrightarrow \boxed{\text{C}}$$

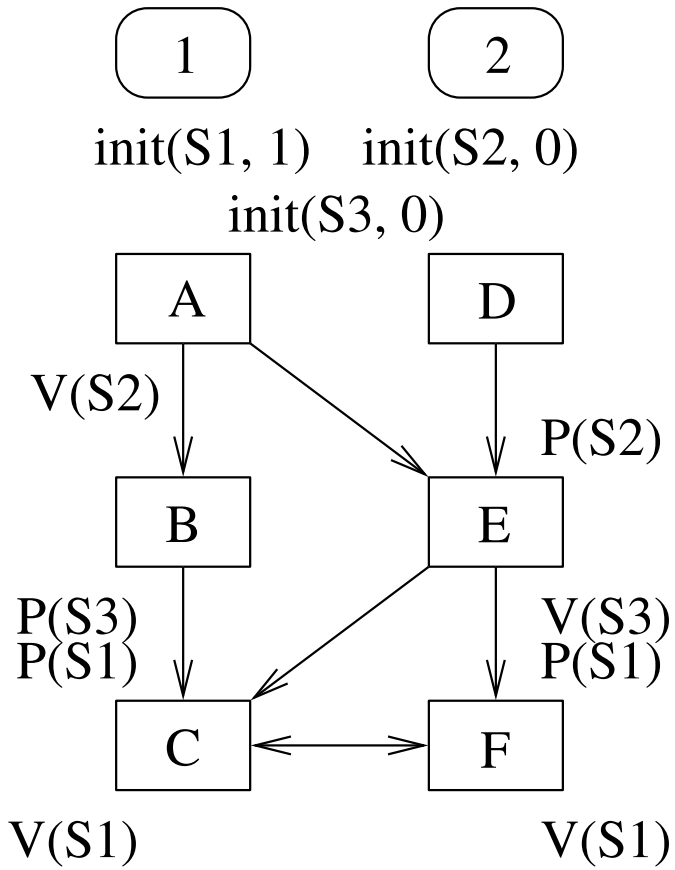
Exemple d'un graphe de précédence

Exemple

- Soient deux processus 1 et 2 ayant respectivement trois sections :
  $\hookrightarrow$ A, B et C pour le processus 1 et D, E et F pour le processus 2
- E doit être exécutée après A, C après E
- C et F ne doivent pas être exécutées simultanément



Résolution avec les sémaphores

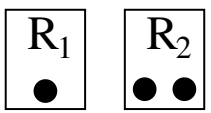


Ressources et processus

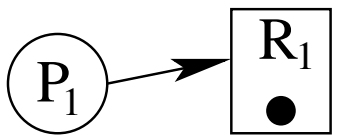
- Tout au long de leur exécution, les processus exploitent des **ressources**
- Ces ressources peuvent exister en un ou plusieurs exemplaires identiques
- Exemple : un ordinateur possède 4 processeurs, le type de ressource processeur existe en 4 exemplaires
- Pour utiliser une ressource, le processus doit faire une demande : la **requête**
  - S'il ne peut avoir la ressource, il est en attente
  - Dès qu'il peut l'obtenir, il l'utilise
    $\hookrightarrow$ La ressource lui est affectée : l'**affectation**
  - Une fois utilisée, le processus libère la ressource

Graphe d'allocation des ressources

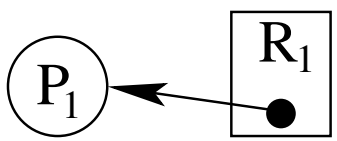
- L'état du système est représenté à un instant t par un **graphe d'allocation des ressources**
- Les processus sont représentés par des ronds : 

- Les ressources sont représentées par des carrés contenant autant de points que d'exemplaires : 

- La requête correspond à une flèche du processus vers la ressource :



  $\hookrightarrow$ Le processus P_1 attend la ressource R_1
- L'affectation correspond à une flèche d'un exemplaire de la ressource au processus :



  $\hookrightarrow$ L'unique exemplaire de la ressource R_1 est allouée au processus P_1

Qu'est-ce que un interblocage

Définition : interblocage

"*Un ensemble de processus est en interblocage si chaque processus attend un événement que seul un autre processus de l'ensemble peut provoquer*"

- Les 4 conditions pour avoir un interblocage
  1. *Condition d'exclusion mutuelle* : chaque ressource est attribuée à un seul processus ou est disponible
  2. *Condition de détention et d'attente* : un processus ayant déjà obtenu une ressource peut en demander une nouvelle
  3. *Pas de réquisition* : si un processus possède une ressource, elle ne peut lui être retirée de force
  4. *Condition d'attente circulaire* : il doit y avoir un cycle d'au moins 2 processus, chacun attendant une ressource détenue par un autre processus du cycle

Interblocage dans un grape de précédence

Processus A :

- $P(S_1)$
- $P(S_2)$
- *Section critique*
- $V(S_2)$
- $V(S_1)$

Processus B :

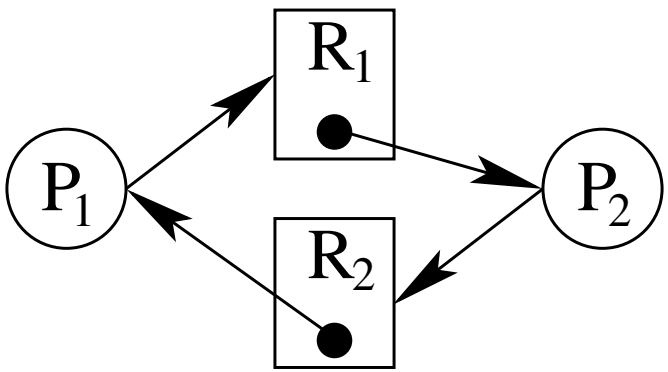
- $P(S_2)$
- $P(S_1)$
- *Section critique*
- $V(S_1)$
- $V(S_2)$

Attention !

Chaque opération est atomique, mais pas une suite de plusieurs opérations !

Interblocage dans un graphe d'allocation des ressources (1/2)

- Un interblocage *peut* être représenté par un circuit dans le graphe

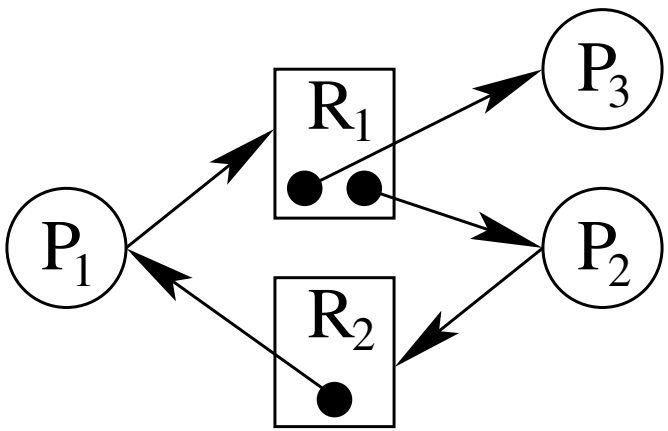


- P_1 attend R_1 dont l'exemplaire est détenu par P_1
- P_2 attend R_2 dont l'exemplaire est détenu par P_1

$\Rightarrow$ Les deux processus ne peuvent continuer leur exécution tant qu'ils n'obtiennent pas la ressource demandée

Interblocage dans un graphe d'allocation des ressources (2/2)

- Un circuit dans le graphe ne signifie pas forcément un interblocage



- P_1 attend R_1 dont les exemplaires sont détenus par P_2 et P_3
- P_2 attend R_2 dont l'exemplaire est détenu par P_1
- Quand P_3 libère un exemplaire de R_1, celui-ci est affecté à P_1 qui peut poursuivre son exécution

Le problème du dîner des philosophes (1/2)

- Problème proposé par Edsger Dijkstra en 1965 :
  ↪ Résolu par lui-même à l'aide de sémaphores
- 5 philosophes sont assis autour d'une table :
  - 5 plats de spaghettis (un devant chaque philosophe)
  - 5 fourchettes situées entre chaque plat
- Pour manger, un philosophe a besoin de deux fourchettes

Le problème du dîner des philosophes (2/2)

Etats des philosophes

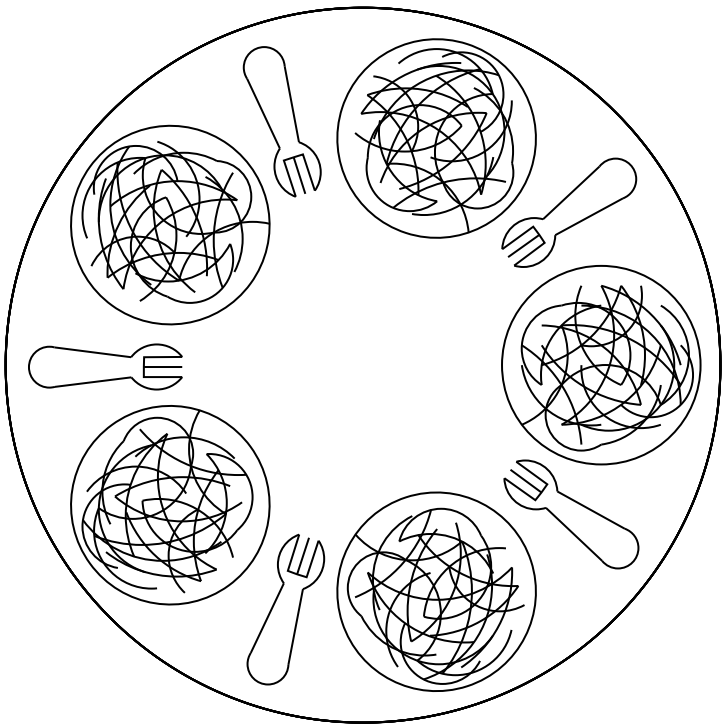
- Deux états définis :
  - Le philosophe mange
  - Le philosophe réfléchit
- Quand le philosophe ne mange pas, il repose ses fourchettes

Changements d'état

- Quand un philosophe désire manger :
  - Il tente de prendre la fourchette à sa droite
  - Il tente de prendre la fourchette à sa gauche
  - Dans un ordre quelconque !
- Un philosophe mange pendent un moment fini
- Quand un philosophe veut réfléchir, il repose ses fourchettes

Routine générale

Configuration de la table



Routine générale d'un philosophe

```
Procédure philosophe(i : entier)
Début
  TantQue(vrai) Faire
    penser()
    prendre_fourchettes(i)
    manger()
    poser_fourchettes(i)
  FinTantQue
Fin
```

Un sémaphore par fourchette

Une (mauvaise) solution

- Les ressources critiques = les fourchettes
- Solution naïve : un sémaphore par fourchette
- Chaque philosophe tente de prendre d'abord la fourchette de droite
- Une fois qu'il l'a prise, il essaie de prendre celle de gauche

Que se passe-t-il si tous les philosophes ont faim en même temps ?

Explications

- Le test et la modification de la valeur d'un sémaphore se font de manière atomique
- Entre deux test/modification, il peut y avoir commutation de contexte

Utilisation d'un sémaphore

- On utilise donc un sémaphore correspondant à toutes les fourchettes :
  - Quand un philosophe a faim, il se met en attente sur le sémaphore P()
  - Lorsque le philosophe a la main, il prend ses deux fourchettes et mange
  - Une fois terminé, il libère le sémaphore V()

Problème

Un seul philosophe mange à la fois.

- Tant que le philosophe ne relâche pas le sémaphore, les autres sont bloqués
- Les ressources utilisées par un philosophe ne sont pas forcément nécessaires pour un autre !

Description générale de la solution de Dijkstra (1/2)

- On définit 3 états pour les philosophes :
  - *Pense*
  - *Faim* (en attente de manger)
  - *Mange*
- Les états des philosophes sont placés dans un tableau "`états`"
- Lorsqu'un philosophe veut manger :
  1. Il passe dans l'état *Faim*
  2. Vérifie qu'aucun de ses voisins ne mange
  3. Si c'est le cas, il passe lui-même dans l'état *Mange*
  4. Sinon, il est bloqué
- Pour s'assurer de ne pas être interrompu pendant cette phase : utilisation d'un sémaphore "`S`"

Description générale (2/2)

- Lorsqu'un philosophe a faim et n'est pas en mesure de manger, il est bloqué
- Pour bloquer les philosophes, on utilise un sémaphore pour chacun :
  ↪ Tableau de sémaphores "`T`"
- Lorsqu'un philosophe termine de manger :
  - Il passe dans l'état *Pense*
  - Vérifie si son voisin de gauche est bloqué :
    ↪ Si c'est le cas, il vérifie s'il peut le débloquer
  - Idem avec le voisin de droite

Dans cette solution, c'est les philosophes qui débloquent leurs voisins

Algorithme (1/2)

Procédure philosophe(i : entier)

```
TantQue(vrai) Faire
  penser()
  prendre_fourchettes(i)
  manger()
  poser_fourchettes(i)
FinTantQue
```

Procédure test(i : entier)

```
gauche <- (i + N - 1) modulo N
droite <- (i + 1) modulo N
Si(états(gauche) != mange) et (états(droite) != mange) et
  (états(i) == faim) Alors
  états(i) <- mange
  V(T[i])
FinSi
```

Algorithme (2/2)

Procédure prendre_fourchette(i : entier)

```
P(S)                // Entrée en section critique
états(i) <- faim    // On passe dans l'état faim
test(i)             // On teste si on peut passer dans l'état manger
V(S)                // Sortie de section critique
P(T[i])             // On bloque si on n'a pu prendre les fourchettes
```

Procédure poser_fourchette(i : entier)

```
P(S)                // Entrée en section critique
états(i) <- pense   // On repasse dans l'état pense

// On teste si le voisin de gauche peut manger
gauche <- (i + N - 1) modulo N
test(gauche)
// On teste si le voisin de droite peut manger
droite <- (i + 1) modulo N
test(droite)

V(S)                // Sortie de section critique
```