Synchronisation Distribuée
Exclusion mutuelle sans datation

Laurent PHILIPPE

Master 2 Informatique ISL
Ingénierie des Systèmes et Logiciels

2021/2022

Sommaire

Sommaire

Exclusion mutuelle : rappels

- Une ressource partagée ou une section critique n'est accédée que par un processus à la fois
- Un processus est dans les trois états possibles, par rapport à l'accès à la ressource :
  - ***demandeur***
  - ***dedans***
  - ***dehors***
- Changement d'états par un processus :
  - de ***dehors*** à ***demandeur***
  - de ***demandeur*** à ***dedans***
  - de ***dedans*** à ***dehors***

Exclusion mutuelle : rappels

Propriétés

Accès en exclusion mutuelle doit respecter 2 propriétés :

- Sûreté : un processus au maximum en section critique (état dedans)
- Vivacité : toute demande d'accès à la section critique est satisfaite en un temps fini

Optionnelles :

- Équité : tous les processus ont la même chance d'accéder à la section critique
- Ordre de passage : l'ordre de passage respecte l'ordre des demandes, dépend de l'ordre de transmission des messages

Mécanismes d'exclusion mutuelle distribuée

En système distribué, nous présentons deux classes de mécanismes d'exclusion mutuelle

- les mécanismes centralisés
- les mécanismes distribués

Sommaire

Mécanisme de synchronisation centralisé

Synchronisation centralisée : définition

Mécanisme centralisé de synchronisation en système distribué regroupe les algorithmes qui utilisent un coordinateur pour gérer l'exclusion mutuelle. Ce coordinateur :

- centralise les requêtes des différents processus de l'application
- réalise l'exclusion mutuelle en accordant la permission d'entrée en Section Critique
- gère une file des requêtes en attente de Section Critique.

L'algorithme

Processus P_i

- Quand un processus P_i veut entrer en Section Critique, il envoie au coordinateur un message de "demande d'entrée en Section Critique". Il attend la permission du coordinateur, avant d'entrer en Section Critique.
- Lorsque P_j reçoit la permission, il entre en Section Critique.
- Quand P_i quitte la Section Critique, il envoie un message de sortie de Section Critique au coordinateur.

L'algorithme

Coordinateur

- Si personne en Section Critique, à la réception d'une demande d'accès de P_i, il retourne "permission d'accès" à P_i.
- Si un processus P_j est déjà en Section Critique. Le coordinateur refuse l'accès. La méthode dépend de l'implantation :
  - soit il n'envoie pas de réponse, le processus P_j est bloqué.
  - soit il envoie une réponse : "Permission non accordée".

  Dans les deux cas, le coordinateur dépose la demande de P_j dans une file d'attente.
- A la réception d'un message de sortie de Section Critique, le coordinateur prend la première requête de la file d'attente de la Section Critique et envoie au processus un message de permission d'entrée en Section Critique.

Mécanisme de synchronisation centralisé

Avantages

- Algorithme garantit l'exclusion mutuelle,
- Algorithme juste (les demandes sont accordées dans l'ordre de réception)
- Pas de famine (aucun processus ne reste bloqué)
- Solution facile à implanter
- Pas de supposition sur l'ordre des messages
- Complexité : maximum 2 ou 3 messages

Mécanisme de synchronisation centralisé

Inconvénients

- Si coordinateur a un problème, tout le système s'effondre
- Si processus bloqués lors de demande d'accès à une Section Critique occupée : impossibilité de détecter la panne du coordinateur
- Dans de grands systèmes, le coordinateur = goulet d'étranglement

Sommaire

Synchronisation distribuée utilisant un jeton

Algorithmes

Différentes structures de communication :

- Anneau : Le Lann
- Arbre : Naimi-Trehel
- Réseau complet : Suzuki-Kazami

Algorithme de Le Lann

Système structuré en anneau

- Algorithme utilisant un jeton pour gérer l'accès à une Section Critique
- Jeton (unique pour l'application) = permission d'entrer en Section Critique
- Un seul processus détient le jeton à un moment donné

Algorithme de Le Lann

Construction de l'anneau

- Construction d'un anneau en attribuant une place à chacun des processus : par ex. numéro du processus
- Chaque processus connaît son successeur dans l'anneau
- Initialisation : jeton attribué à un processus (exemple : Processus P_0)
- Jeton tourne entre les processus
- Accès en SC quand possède le jeton

Algorithme de Le Lann

Avantages de l'algorithme

- Simple à mettre en œuvre
- Intéressant si nombreux demandeurs de la ressource
- Équitable en terme de nombre d'accès et de temps d'attente

Algorithme de Le Lann

Inconvénients (1)

- Nécessite des échanges de messages même si aucun processus ne veut accéder à la Section Critique
- Temps d'accès à la Section Critique peut être long
- Perte du jeton :
  - difficulté de détecter un tel cas
  - impossibilité de prendre en compte le temps écoulé entre deux passages du jeton (temps passé en Section Critique est très variable)
  - des algorithmes gèrent ce problème de perte et régénération de jeton sur un anneau (non vu dans ce cours)

Algorithme de Le Lann

Inconvénients(2)

- Problème de la panne d'un processus
  - plus facile à gérer que dans les algorithmes précédents
  - envoi d'un acquittement à la réception du jeton : permet de détecter la panne de l'un des processus
  - le processus mort peut être retiré de l'anneau et le suivant prend alors sa place
  - pour implanter ceci : chaque processus doit connaître la configuration courante de l'anneau.

Conclusion : structure d'anneau lourde à gérer si virtuelle

Synchronisation distribuée utilisant un jeton

Exercice : synchronisation distribuée des robots

Synchronisation distribuée utilisant un jeton

Algorithme de Naimi-Trehel

Pas de structure d'anneau

Principe

- Les processus sont logiquement organisés en arbre construit à partir des requêtes de demande d'accès en Section Critique
- Seul le processus qui possède le jeton peut accéder à la section critique

Algorithme de Naimi-Trehel

Hypothèses

- Réseau complet, chaque processus peut communiquer avec tous les autres
- Nombre de processus fixé
- Pas de départ ni arrivée de processus
- Chaque processus a un identificateur
- Pas de perte de message

Algorithme de Naimi-Trehel

Structures de données distribuées

- Une file de requêtes : ***next***
- Un arbre de chemins vers le dernier demandeur de SC : ***owner***

Variables pour chaque processus P_i

- ***owner*** : processus qui possède probablement le jeton. Toute demande de P_i d'accès à la Section Critique sera envoyée à ce processus.
- ***next*** : processus à qui P_i devra transmettre le jeton à sa sortie de Section Critique.

Algorithme de Naimi-Trehel

Principe de l'algorithme (1)

Initialisation :

- ***electednode*** : processus désigné comme étant le propriétaire du jeton. C'est la racine de l'arbre.
- ***owner*** : initialisé à ***electednode***
- ***next*** : initialisé à ***nil***

Algorithme de Naimi-Trehel

Principe de l'algorithme (2)

Quand un processus P_i demande l'accès à la Section Critique

- S'il ne possède pas le jeton, il envoie sa requête à ***owner*** et attend de recevoir le jeton
- S'il possède déjà le jeton, il entre en Section Critique

Principe de l'algorithme (3)

Quand un processus P_i reçoit une requête d'entrée en SC de la part de P_j

- S'il possède le jeton et qu'il n'est pas en SC alors il envoie le jeton à P_j et met à jour sa variable ***owner***
- S'il possède le jeton et s'il est en SC, il met à jour ***next***
- S'il ne possède pas le jeton et qu'il attend la section critique : il enregistre la demande dans ***next***
- S'il ne possède pas le jeton et qu'il n'attend pas la section critique : il transmet la requête à ***owner***

Principe de l'algorithme (4)

Quand un processus P_i sort de SC :

- si ***next*** est initialisé il envoie le jeton à ***next*** et réinitialise ***next*** à nil
- sinon garde le jeton

Algorithme de Naimi-Trehel

Exemple des robots

Soient 4 robots R_1, R_2, R_3, R_4

- Initialisation : le robot R_1 possède le jeton (electednode)
- Le robot R_2 demande l'accès à la section critique
- Lorsqu'il est en section critique le robot R_3 demande l'accès à la section critique
- Dérouler l'algorithme jusqu'à la fin

Algorithme de Naimi-Trehel

A l'initialisation :

R_1	0				
owner	*nil*				
next	*nil*				
jeton	T				
SC	F				

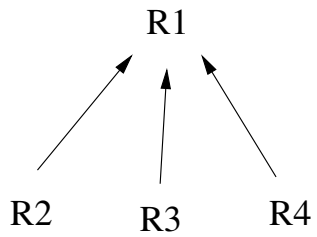
R_2	0				
owner	R_1				
next	*nil*				
jeton	F				
SC	F				

R_3	0				
owner	R_1				
next	*nil*				
jeton	F				
SC	F				

R_4	0				
owner	R_1				
next	*nil*				
jeton	F				
SC	F				

	Évènement
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
...	

Arbre :



Algorithme de Naimi-Trehel

R_1	0				
owner	*nil*				
next	*nil*				
jeton	T				
SC	F				

R_2	0	1			
owner	R_1	*nil*			
next	*nil*	*nil*			
jeton	F	F			
SC	F	T			

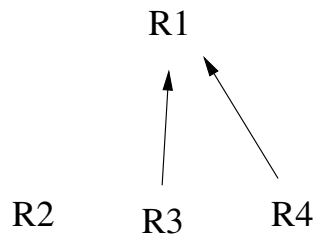
R_3	0				
owner	R_1				
next	*nil*				
jeton	F				
SC	F				

R_4	0				
owner	R_1				
next	*nil*				
jeton	F				
SC	F				

R_2 attend le jeton

	Évènement
1	R_2 demande l'accès à la base
2	R_2 envoie $REQ(2)$ à R_1
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
...	

Arbre :



Algorithme de Naimi-Trehel

R_1	0	3			
owner	*nil*	R_2			
next	*nil*	*nil*			
jeton	T	F			
SC	F	F			

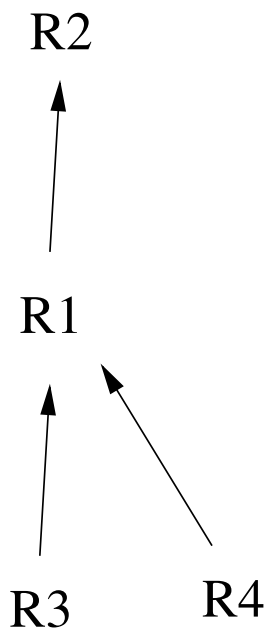
R_2	0	1			
owner	R_1	*nil*			
next	*nil*	*nil*			
jeton	F	F			
SC	F	T			

R_3	0				
owner	R_1				
next	*nil*				
jeton	F				
SC	F				

R_4	0				
owner	R_1				
next	*nil*				
jeton	F				
SC	F				

	Évènement
1	R_2 demande l'accès à la base
2	R_2 envoie $REQ(2)$ à R_1
3	R_1 reçoit $REQ(2)$ de R_2
4	R_1 envoie *JETON* à R_2
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
...	

Arbre :



Algorithme de Naimi-Trehel

R_1	0	3			
owner	*nil*	R_2			
next	*nil*	*nil*			
jeton	T	F			
SC	F	F			

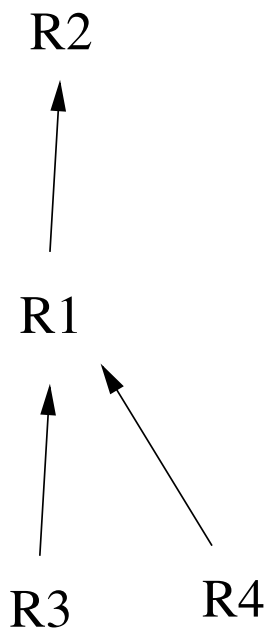
R_2	0	1	5		
owner	R_1	*nil*	*nil*		
next	*nil*	*nil*	*nil*		
jeton	F	F	T		
SC	F	T	T		

R_3	0				
owner	R_1				
next	*nil*				
jeton	F				
SC	F				

R_4	0				
owner	R_1				
next	*nil*				
jeton	F				
SC	F				

	Évènement
1	R_2 demande l'accès à la base
2	R_2 envoie $REQ(2)$ à R_1
3	R_1 reçoit $REQ(2)$ de R_2
4	R_1 envoie *JETON* à R_2
5	R_2 reçoit *JETON* de R_1
6	R_2 accède à la base et se recharge
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
...	

Arbre :



Algorithme de Naimi-Trehel

R_1	0	3			
owner	*nil*	R_2			
next	*nil*	*nil*			
jeton	T	F			
SC	F	F			

R_2	0	1	5		
owner	R_1	*nil*	*nil*		
next	*nil*	*nil*	*nil*		
jeton	F	F	T		
SC	F	T	T		

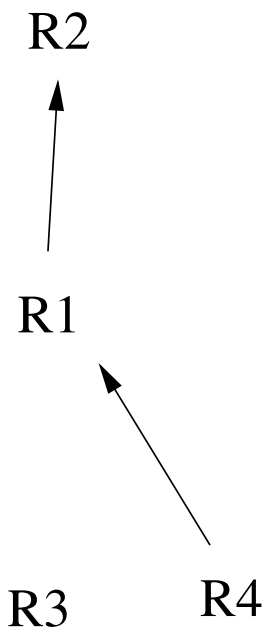
R_3	0	7			
owner	R_1	*nil*			
next	*nil*	*nil*			
jeton	F	F			
SC	F	T			

R_4	0				
owner	R_1				
next	*nil*				
jeton	F				
SC	F				

R_3 attend pour accéder à la base

	Évènement
1	R_2 demande l'accès à la base
2	R_2 envoie $REQ(2)$ à R_1
3	R_1 reçoit $REQ(2)$ de R_2
4	R_1 envoie *JETON* à R_2
5	R_2 reçoit *JETON* de R_1
6	R_2 accède à la base et se recharge
7	R_3 demande l'accès à la base
8	R_3 envoie $REQ(3)$ à R_1
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
...	

Arbre :



Algorithme de Naimi-Trehel

R_1	0	3	9		
owner	*nil*	R_2	R_3		
next	*nil*	*nil*	*nil*		
jeton	T	F	F		
SC	F	F	F		

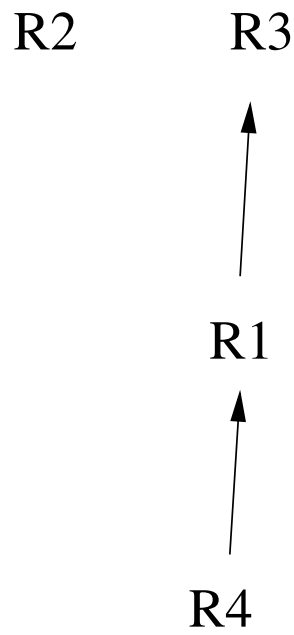
R_2	0	1	5		
owner	R_1	*nil*	*nil*		
next	*nil*	*nil*	*nil*		
jeton	F	F	T		
SC	F	T	T		

R_3	0	7			
owner	R_1	*nil*			
next	*nil*	*nil*			
jeton	F	F			
SC	F	T			

R_4	0				
owner	R_1				
next	*nil*				
jeton	F				
SC	F				

	Évènement
1	R_2 demande l'accès à la base
2	R_2 envoie $REQ(2)$ à R_1
3	R_1 reçoit $REQ(2)$ de R_2
4	R_1 envoie $JETON$ à R_2
5	R_2 reçoit $JETON$ de R_1
6	R_2 accède à la base et se recharge
7	R_3 demande l'accès à la base
8	R_3 envoie $REQ(3)$ à R_1
9	R_1 reçoit $REQ(3)$ de R_3
10	R_1 envoie $REQ(3)$ à R_2
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
...	

Arbre :



Algorithme de Naimi-Trehel

R_1	0	3	9		
owner	*nil*	R_2	R_3		
next	*nil*	*nil*	*nil*		
jeton	T	F	F		
SC	F	F	F		

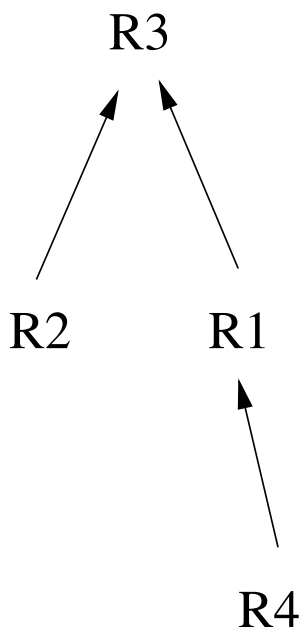
R_2	0	1	5	11	
owner	R_1	*nil*	*nil*	R_3	
next	*nil*	*nil*	*nil*	R_3	
jeton	F	F	T	T	
SC	F	T	T	T	

R_3	0	7			
owner	R_1	*nil*			
next	*nil*	*nil*			
jeton	F	F			
SC	F	T			

R_4	0				
owner	R_1				
next	*nil*				
jeton	F				
SC	F				

	Évènement
1	R_2 demande l'accès à la base
2	R_2 envoie $REQ(2)$ à R_1
3	R_1 reçoit $REQ(2)$ de R_2
4	R_1 envoie *JETON* à R_2
5	R_2 reçoit *JETON* de R_1
6	R_2 accède à la base et se recharge
7	R_3 demande l'accès à la base
8	R_3 envoie $REQ(3)$ à R_1
9	R_1 reçoit $REQ(3)$ de R_3
10	R_1 envoie $REQ(3)$ à R_2
11	R_2 reçoit $REQ(3)$ de R_1
12	
13	
14	
15	
16	
17	
18	
19	
20	
...	

Arbre :



Algorithme de Naimi-Trehel

R_1	0	3	9		
owner	*nil*	R_2	R_3		
next	*nil*	*nil*	*nil*		
jeton	T	F	F		
SC	F	F	F		

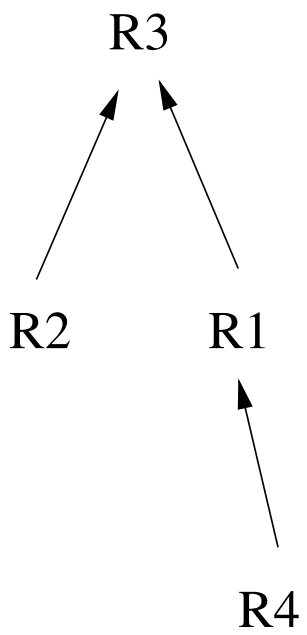
R_2	0	1	5	11	12
owner	R_1	*nil*	*nil*	R_3	R_3
next	*nil*	*nil*	*nil*	R_3	*nil*
jeton	F	F	T	T	F
SC	F	T	T	T	F

R_3	0	7			
owner	R_1	*nil*			
next	*nil*	*nil*			
jeton	F	F			
SC	F	T			

R_4	0				
owner	R_1				
next	*nil*				
jeton	F				
SC	F				

	Évènement
1	R_2 demande l'accès à la base
2	R_2 envoie $REQ(2)$ à R_1
3	R_1 reçoit $REQ(2)$ de R_2
4	R_1 envoie *JETON* à R_2
5	R_2 reçoit *JETON* de R_1
6	R_2 accède à la base et se recharge
7	R_3 demande l'accès à la base
8	R_3 envoie $REQ(3)$ à R_1
9	R_1 reçoit $REQ(3)$ de R_3
10	R_1 envoie $REQ(3)$ à R_2
11	R_2 reçoit $REQ(3)$ de R_1
12	R_2 est rechargé, quitte la base
13	R_2 envoie *JETON* à R_3
14	
15	
16	
17	
18	
19	
20	
...	

Arbre :



Algorithme de Naimi-Trehel

R_1	0	3	9		
owner	*nil*	R_2	R_3		
next	*nil*	*nil*	*nil*		
jeton	T	F	F		
SC	F	F	F		

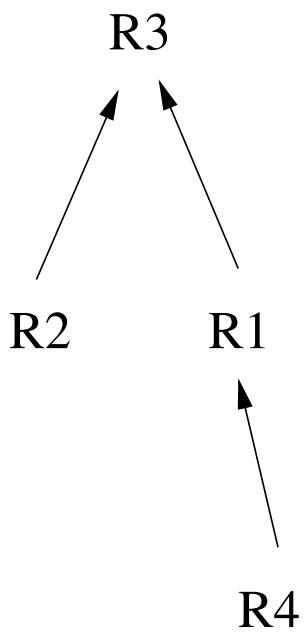
R_2	0	1	5	11	12
owner	R_1	*nil*	*nil*	R_3	R_3
next	*nil*	*nil*	*nil*	R_3	*nil*
jeton	F	F	T	T	F
SC	F	T	T	T	F

R_3	0	7	14		
owner	R_1	*nil*	*nil*		
next	*nil*	*nil*	*nil*		
jeton	F	F	T		
SC	F	T	T		

R_4	0				
owner	R_1				
next	*nil*				
jeton	F				
SC	F				

	Évènement
1	R_2 demande l'accès à la base
2	R_2 envoie $REQ(2)$ à R_1
3	R_1 reçoit $REQ(2)$ de R_2
4	R_1 envoie *JETON* à R_2
5	R_2 reçoit *JETON* de R_1
6	R_2 accède à la base et se recharge
7	R_3 demande la base
8	R_3 envoie $REQ(3)$ à R_1
9	R_1 reçoit $REQ(3)$ de R_3
10	R_1 envoie $REQ(3)$ à R_2
11	R_2 reçoit $REQ(3)$ de R_1
12	R_2 est rechargé, quitte la base
13	R_2 envoie *JETON* à R_3
14	R_3 reçoit *JETON* de R_2
15	R_3 accède à la base et se recharge
16	
17	
18	
19	
20	
...	

Arbre :



Algorithme de Naimi-Trehel

R_1	0	3	9		
owner	*nil*	R_2	R_3		
next	*nil*	*nil*	*nil*		
jeton	T	F	F		
SC	F	F	F		

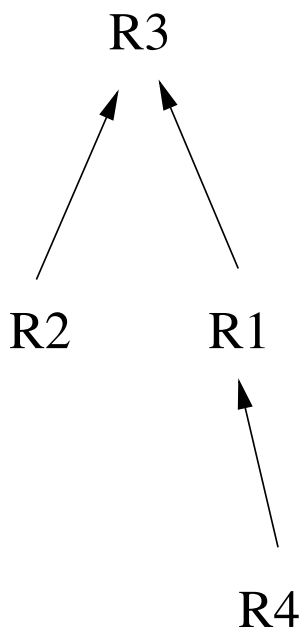
R_2	0	1	5	11	12
owner	R_1	*nil*	*nil*	R_3	R_3
next	*nil*	*nil*	*nil*	R_3	*nil*
jeton	F	F	T	T	F
SC	F	T	T	T	F

R_3	0	7	14	16	
owner	R_1	*nil*	*nil*	*nil*	
next	*nil*	*nil*	*nil*	*nil*	
jeton	F	F	T	T	
SC	F	T	T	F	

R_4	0				
owner	R_1				
next	*nil*				
jeton	F				
SC	F				

	Évènement
1	R_2 demande l'accès à la base
2	R_2 envoie $REQ(2)$ à R_1
3	R_1 reçoit $REQ(2)$ de R_2
4	R_1 envoie *JETON* à R_2
5	R_2 reçoit *JETON* de R_1
6	R_2 accède à la base et se recharge
7	R_3 demande l'accès à la base
8	R_3 envoie $REQ(3)$ à R_1
9	R_1 reçoit $REQ(3)$ de R_3
10	R_1 envoie $REQ(3)$ à R_2
11	R_2 reçoit $REQ(3)$ de R_1
12	R_2 est rechargé, quitte la base
13	R_2 envoie *JETON* à R_3
14	R_3 reçoit *JETON* de R_2
15	R_3 accède à la base pour se recharger
16	R_3 est rechargé, quitte la base
17	
18	
19	
20	
...	

Arbre :



Algorithme de Naimi-Trehel

File de requêtes : *next*

- Processus en tête de file possède le jeton
- Possède une variable ***next*** qui mémorise le processus suivant en attente
- Chaque nouvelle requête de SC est placée en queue de file
- Processus en queue de file est le dernier processus qui a fait une demande d'entrée en SC

Algorithme de Naimi-Trehel

Arbre de chemins vers le dernier demandeur de SC : ***owner***

- Une nouvelle requête est transmise suivant le chemin des pointeurs de ***owner*** jusqu'à la racine de l'arbre (***owner*** = nil)
  - reconfiguration dynamique de l'arbre : nouveau demandeur devient la racine de l'arbre
  - les processus faisant partie du chemin entre la nouvelle racine et les anciennes racines modifient leurs pointeurs ***owner*** :
    owner = nouvelle racine

Algorithme de Naimi-Trehel

Complexité

Le nombre de messages requis pour une utilisation de la Section Critique est :

- Pire cas : N, chaque processus demande à son tour (arbre linéaire)
- Meilleur cas : 0, le processus possède le jeton
- Moyenne : Log(N)

Algorithme de Naimi-Trehel

Exercice : déroulement de l'algorithme

Algorithme de Naimi-Trehel

Un algorithme de section critique doit garantir la sûreté et l'absence de famine

Preuve de la sûreté... Informelle

Il n'y a qu'un seul jeton, il y a deux cas où le processus envoi le jeton (quand il sort de SC et que la variable *next* n'est pas à *nil* et quand il reçoit une demande et que sa variable *sc* est à *false*). Dans les deux cas il ne transmet le jeton à un seul processus et le perd. Il n'y a donc qu'un seul processus qui possède le jeton et donc un seul processus qui peut être en section critique à un instant t puisqu'il faut le jeton pour entrer en section critique.

Algorithme de jeton

Limitations

- Accès à la section critique uniquement basé sur la circulation d'un jeton
- Garantissent la sûreté, l'absence de famine et de deadlock
- **MAIS** ne garantissent pas l'ordre d'accès par rapport aux messages échangés entre les processus

Illustration

- Affichage de log pour un programme distribué
- Accès à la console de log en section critique
- Les processus échangent des informations d'avancement

Exemple

Exemple

C = A * B
F = D * E
I = C * F
R = Z * I

V = T * S
W = X * Y
Z = W * V

4 processus P1 à P4

Exemple

Exemple

Exécution prévue :

P4 : $C = A * B$ en parallèle avec P2

P2 : $F = D * E$ en parallèle avec P4

P2 : $I = C * F$

P1, P3 : W et $Z \Rightarrow$ ne participent pas à cette partie de calcul mais partagent la ressource d'affichage

Exemple

Exemple

Exécution prévue :

*P4 : C = A * B en parallèle avec P2*

*P2 : F = D * E en parallèle avec P4*

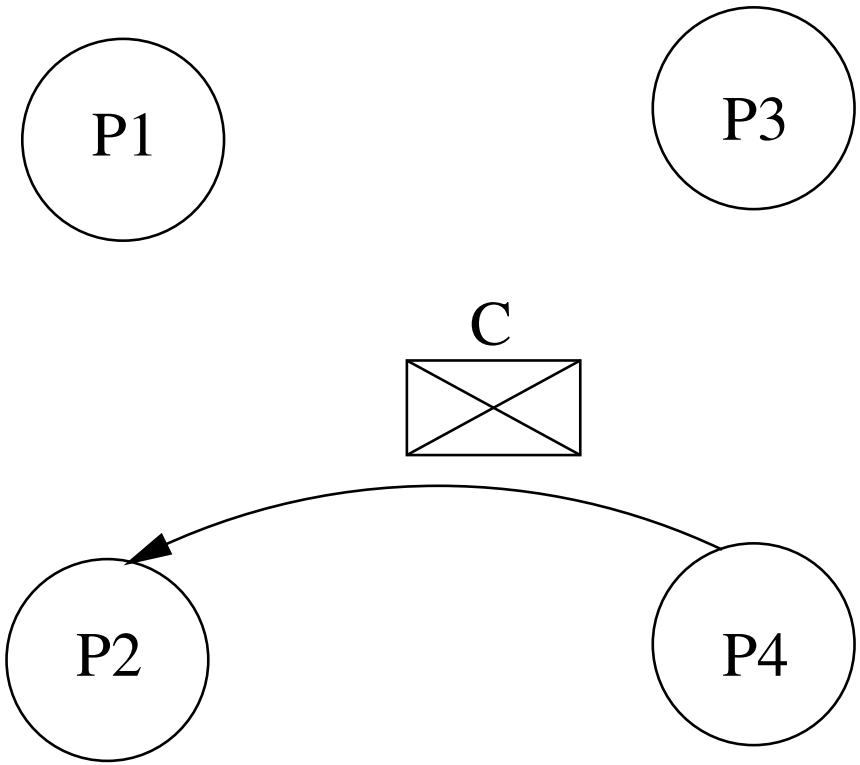
*P2 : I = C * F*

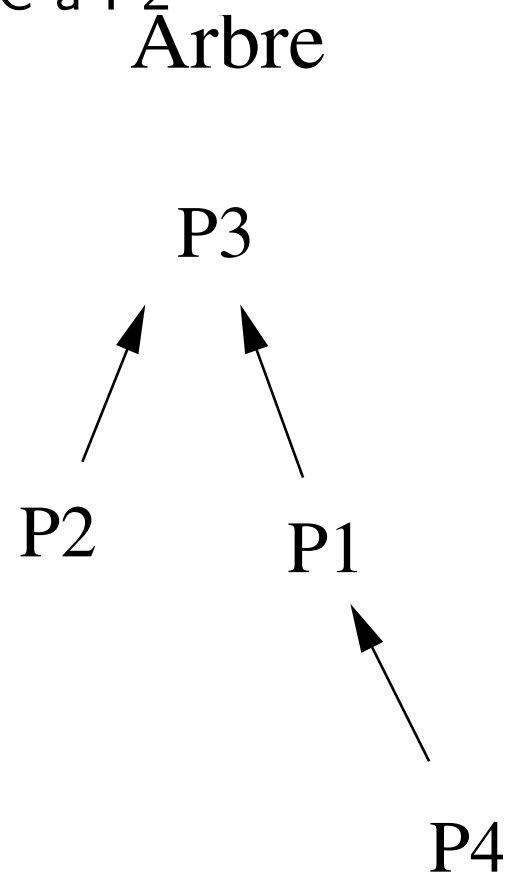
P1, P3 : W et Z ⇒ ne participent pas à cette partie de calcul mais partagent la ressource d'affichage

I ne peut être calculé qu'une fois que C et F le sont

Exemple

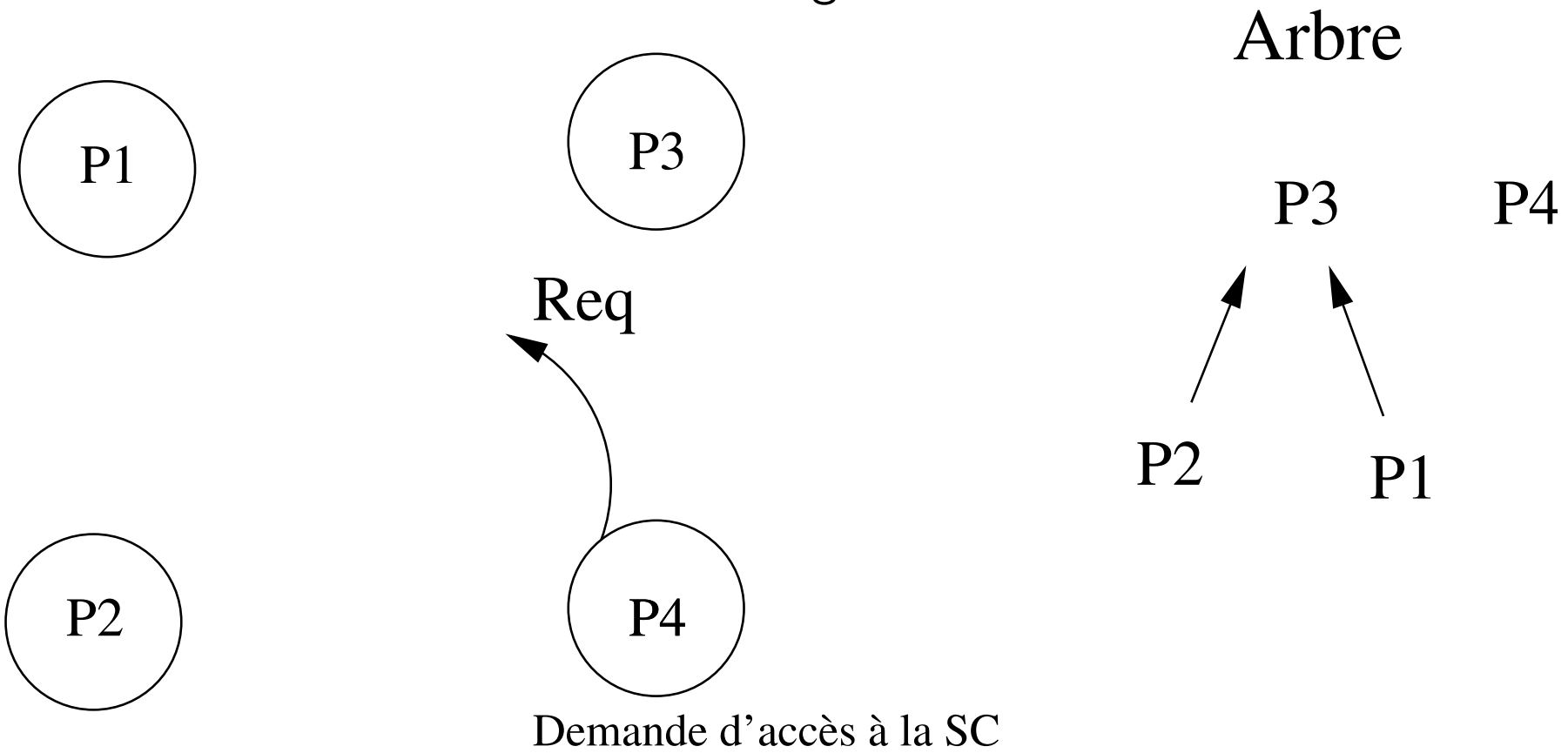
P4 a fini son calcul envoie le résultat C à P2





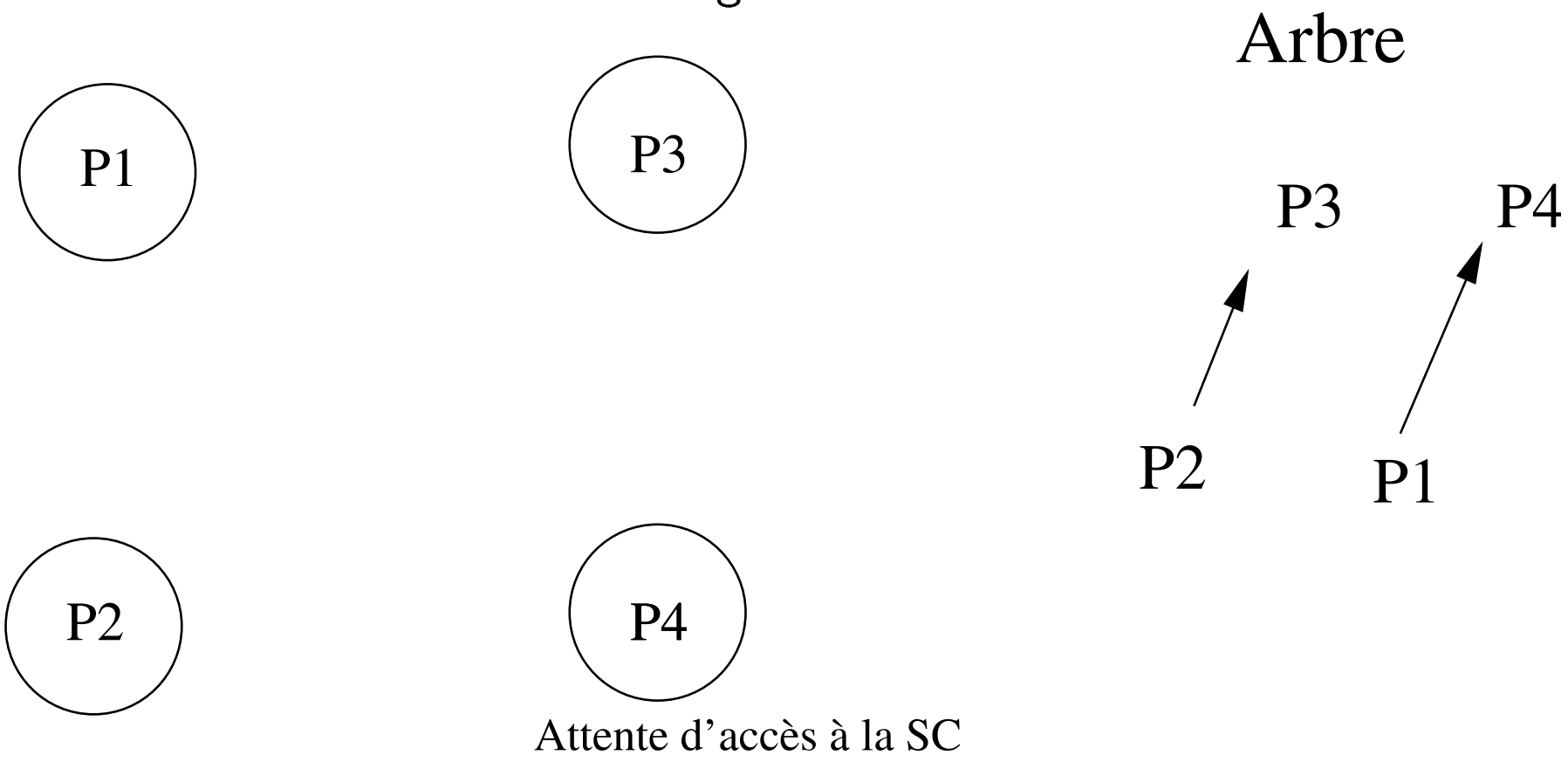
Exemple

P4 veut afficher un message de fin sur la console



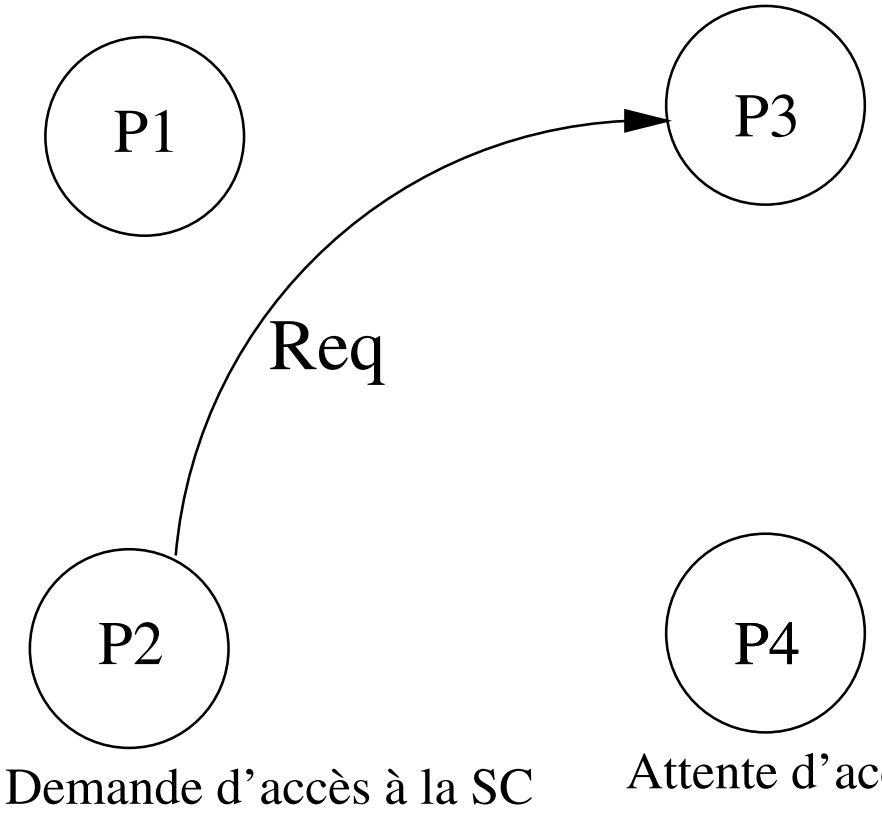
Exemple

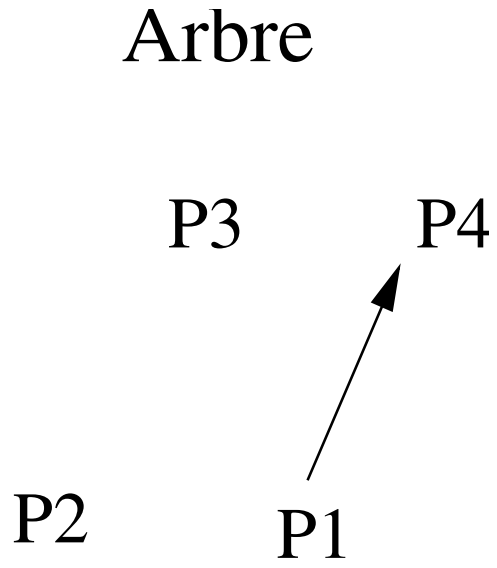
Le message arrive en P1



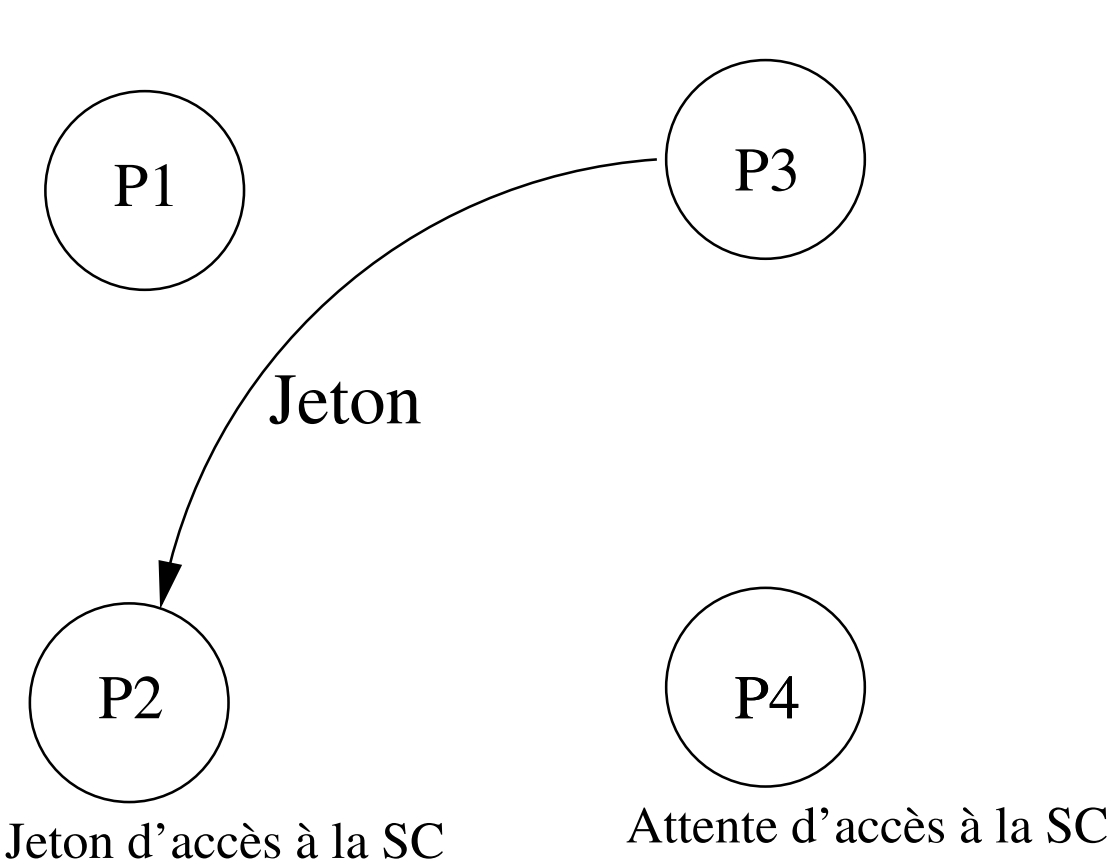
Exemple

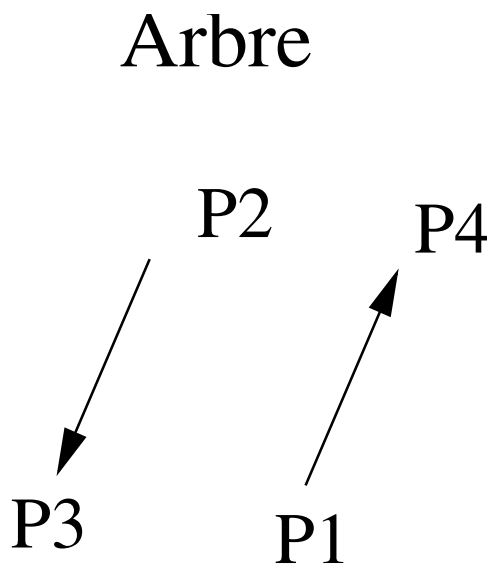
P2 termine, il veut afficher un message de fin sur la console





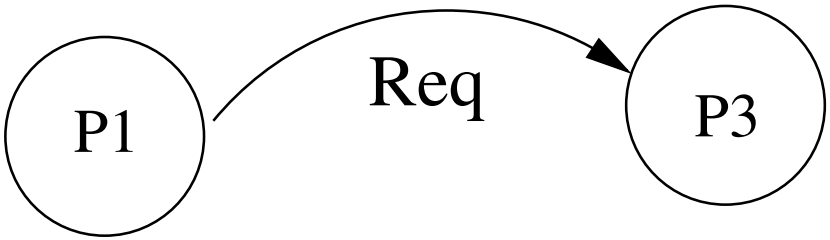
Exemple

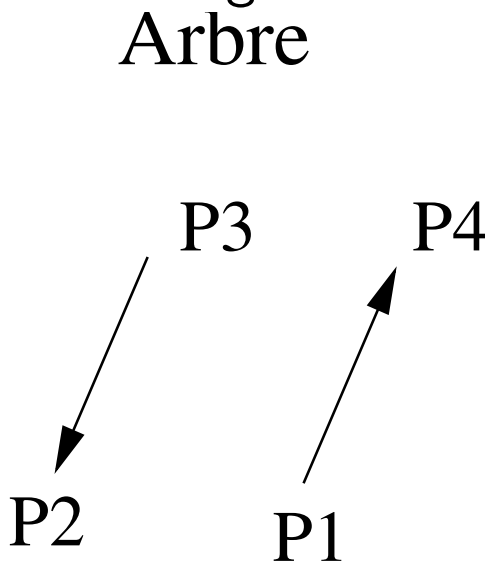

P1
P3
Jeton
P2
P4
Jeton d'accès à la SC
Attente d'accès à la SC

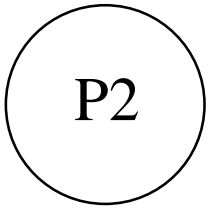

Arbre
P2
P4
P3
P1

Exemple

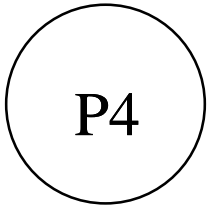
P1 transmet le message de P4, P2 affiche son message







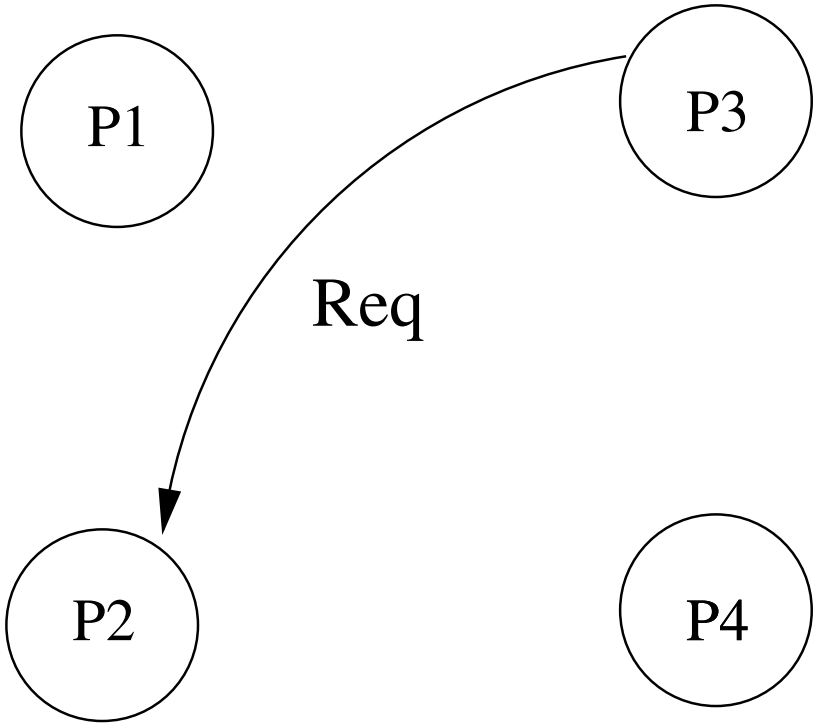
Sort de SC



Attente d'accès à la SC

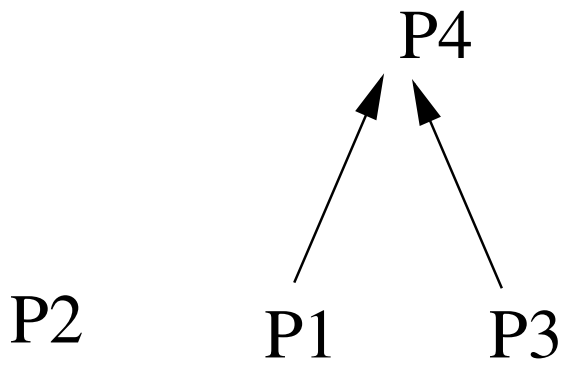
Exemple

P3 transmet le message de P4



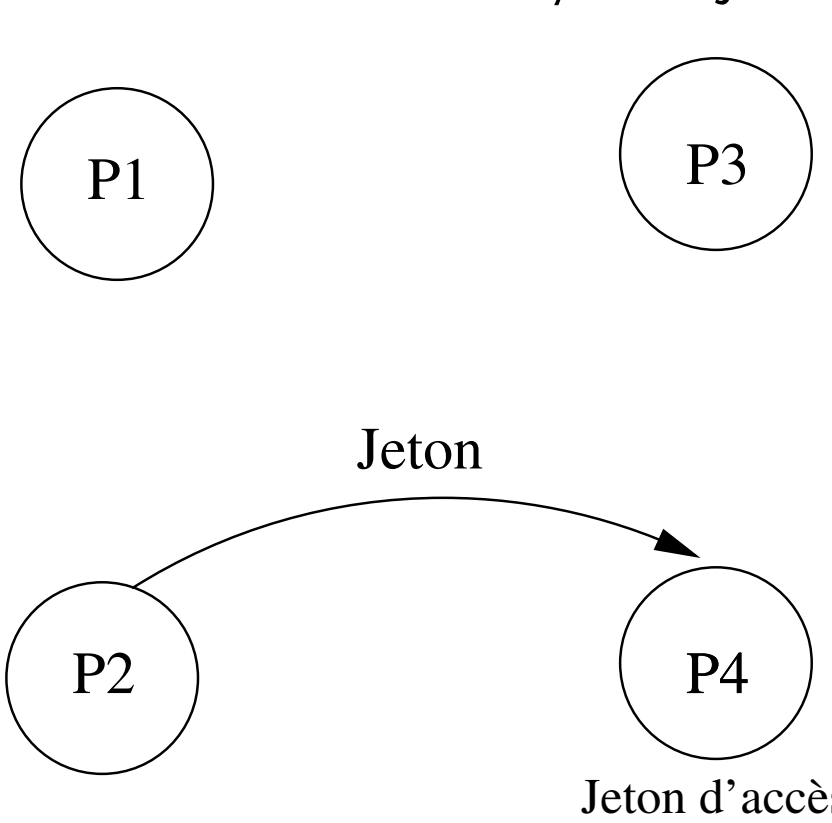
Attente d'accès à la SC

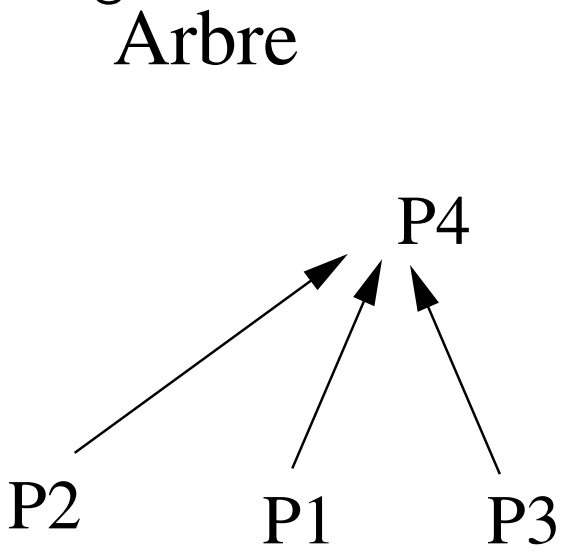
Arbre



Exemple

P4 reçoit le jeton et affiche le message





Exemple

Exécution réelle :	Affichage console :
P4 : affiche "`P4 commence C = A * B`"	`P4 commence C = A * B`
P2 : affiche "`P2 commence F = D * E`"	`P2 commence F = D * E`
P4 : termine en 1er, envoie C à P2	
P4 : demande à accéder à la console (SC)	
P2 : reçoit message de P4	
P2 : termine en 2nd	
P2 : demande l'accès à la console	
P2 : obtient l'accès à la console	
P2 : Affiche "`P2 finit son calcul`"	`P2 finit son calcul`
P2 : affiche "`P2 commence I = C * F`"	`P2 commence I = C * F`
P2 : relâche l'accès à la console	
P2 : calcule I = C * F	
P4 : obtient l'accès à la console	
P4 : Affiche "`P4 finit son calcul`"	`P4 finit son calcul`