Chapitre 2
Interblocage

Objectifs

- Analyser et comprendre les situations d'interblocage de processus
- Modéliser les situations d'interblocage de processus (graphes, matrices,..)
- Résoudre le problème d'interblocage

Plan

- Introduction
  - Problème de l'interblocage
  - Définition
- Caractérisation de l'interblocage
  - Conditions de l'interblocage
  - Graphe d'allocation des ressources
- Méthodes de traitement de l'interblocage
  - Méthodes de prévention statique
  - L'évitement: Méthode de prévention dynamique
  - Méthodes de détection et guérison

1. Introduction (1/2)

- Dans un système multiprogrammé plusieurs processus partagent un nombre fini de ressources.
  - Un processus demande des ressources, si ces ressources ne sont pas disponibles, ce processus doit se mettre en attente.

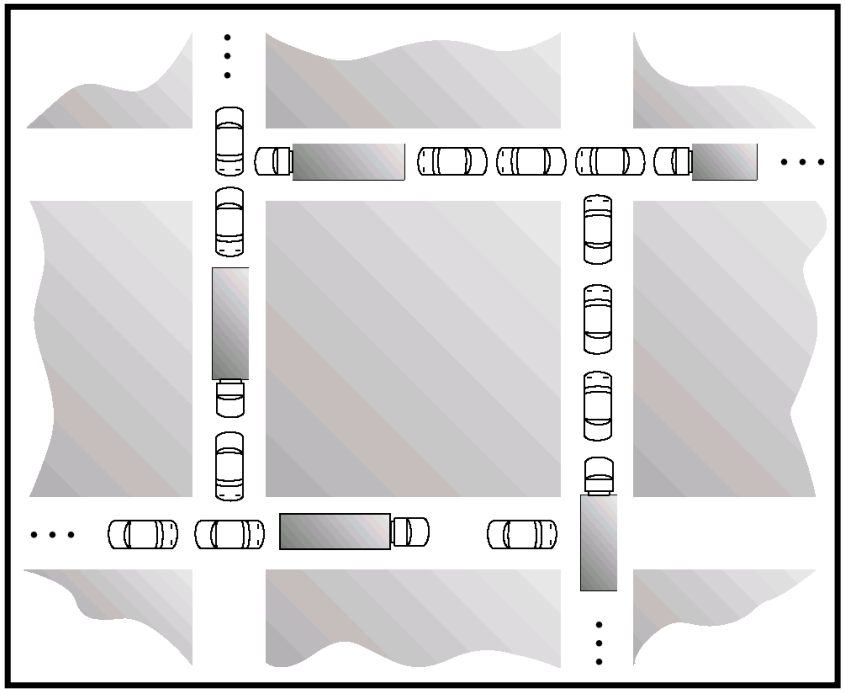
  → Problème de compétition entre les processus pour l'utilisation des ressources.

1. Introduction (2/2)

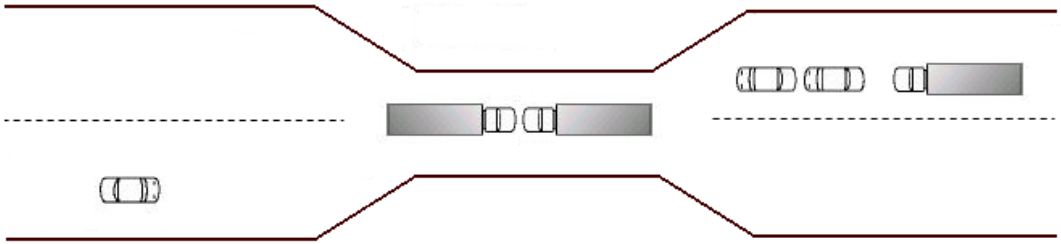
- Il peut arriver qu'un processus reste indéfiniment en attente
  - Les ressources demandées par ce processus sont allouées à des processus qui sont eux-mêmes en attente d'autres ressources.
- Cette situation est appelée interblocage, étreinte fatale ou "deadlock".

Exemples de situations d'interblocage

- Intersection avec priorité à droite.
  - Si 4 véhicules arrivent à cette intersection, aucun ne peut s'engager puisque chaque véhicule possède à sa droite un véhicule plus prioritaire.



- Pont à voie unique



Exemples de situations d'interblocage

- 2 processus A et B qui demandent chacun 2 ressources critiques R1 et R2, mais dans un ordre différent:

Processus A	Processus B
Demander(R1) Demander(R2) <Utiliser R1 et R2> Libérer(R2) Libérer(R1)	Demander(R2) Demander(R1) <Utiliser R1 et R2> Libérer(R1) Libérer(R1)

- Si les 2 processus arrivent en même temps :
  - P1 détient la ressource **R1 et attend R2,**
  - P2 détient la ressource **R2 et attend R1.**

  → Il y a un blocage mutuel infini entre les 2 processus

Exemple (demande de ressources critiques imbriquées) (1/2)

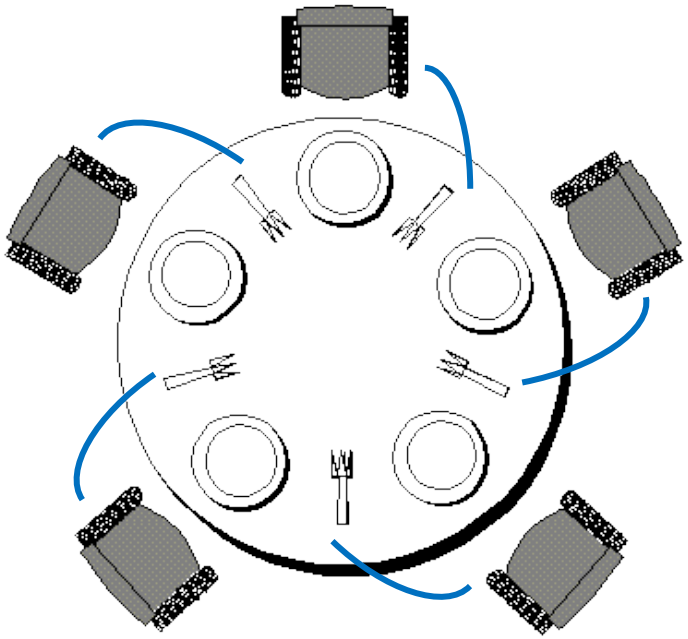
- Deux processus coexistent dans un système, qui a 2 lecteurs-graveurs DVD seulement
- Nous voulons exécuter deux processus de copie DVD en même temps
- Le processus 1 a besoin de
  - Un lecteur-graveur pour démarrer;
  - Puis un deuxième, pour terminer
- Le processus 2 est pareil
- Scénario d'interblocage:
  - Processus 1 demande 1 lect-grav
  - Processus 2 demande 1 lect-grav: les deux sont engagés

Exemple (demande de ressources critiques imbriquées) (2/2)

- interblocage! aucun processus ne peut compléter à moins qu'un des processus ne puisse être suspendu ou puisse retourner en arrière
- Observez que l 'interblocage n 'est pas inévitable, p.ex. si P1 complète avant le début de P2

Exemples de situations d'interblocage

Problème des philosophes



```
semémaphore fourch[ N ] = {1,.....1}

Philosophe i
{
 tant que (vrai)
   {
   penser();
   P(fourch[i]);
   P(fourch[(i + 1) % N]);
   manger()
   V(fourch[i]);
   V(fourch[(i + 1) % N]);
}
}
```

- Interblocage possible si tous les philosophes prennent la fourchette de gauche, personne ne pourra prendre la fourchette a sa droite

Exemples de situations d'interblocage

- Allocation partielle d'une ressource banalisée existant en plusieurs exemplaires.
  - Nombre d'unités disponibles : 4.
  - Rlibre= 4.

P1	P2
Besoins : 3 unités Demander et obtenir : 2 unités Demander et Attendre : 1 unité	Besoins : 3 unités Demander et obtenir : 2 unités Demander et Attendre : 1 unité

- Chaque processus détient 2 unités de la ressource et attend une unité → interblocage.

Définition de l'interblocage [Tanenbeaum]

- Un ensemble de processus est dans une situation d'interblocage si chaque processus attend un événement que seul un autre processus de l'ensemble peut provoquer.
  - Evénement: libération d'une ressource

Modèle du système

- Un système comporte un nombre fini de ressources devant être distribué parmi un certain nombre de processus concurrents.
  - Les ressources sont groupées en classes (types : Imprimantes, mémoires, lecteurs CD registres, fichiers,...).
  - Chaque classe de ressources comporte un nombre fini d'exemplaires (copies identiques).

Protocole d'accès aux ressources

- Pour acquérir une ressource chaque processus suit le protocole suivant :

```
Demander (Ri)
<Utilisation>
Liberer (Ri)
```

Demander() et Liberer() sont généralement des appels systèmes (Ex: Open et Close File)

- Demande: Si la demande n'est pas satisfaite immédiatement, le processus doit se mettre en attente, il ne pourra passer à l'étape suivante (utilisation) que si la ressource lui a été allouée.
- Libération: Toute ressource doit être libérée(restituée), au bout d'un temps fini, après son utilisation par un processus.

Conditions d'interblocage.

- L'interblocage demande la présence simultanée de 4 conditions (conditions nécessaires et suffisantes)

1. Exclusion mutuelle: les ressources ne sont pas partageables, un seul processus à la fois peut utiliser la ressource.
2. Possession et attente: un processus qui détient des ressources peut demander de nouvelles ressources (sans relâcher celles qu'il détient)

Conditions d'interblocage.

3. Pas de réquisition : Les ressources déjà détenues ne peuvent être retirées de force à un processus. Elles doivent être explicitement libérées par le processus qui les détient.
4. Attente circulaire : Il existe un ensemble de k processus P_1, P_2, ..., P_k tels que :
   - P_1 attend une ressource détenue par P_2
   - P_2 attend une ressource détenue par P_3
   - ...
   - P_k attend une ressource détenue par P_1

Conditions d'interblocage.

- En présence des 3 premières conditions, une attente circulaire est un interblocage
- Les 3 premières conditions n'impliquent pas nécessairement interblocage, car l'attente circulaire pourrait ne pas se vérifier

Exercices

- Exercice1: montrer que si l'une des condition n'est pas vérifiée alors il ne peut y avoir d'interblocage.

- Exercice 2: Considérez un système dans lequel chaque processus n'a besoin que d'une seule ressource pendant toute son existence. L'interblocage, est-il possible?

Exercice 3

- Vérifier que si dans un système il y a toujours suffisamment de ressources pour tous, il n'y aura jamais d'interblocages
- Cependant ceci est souvent impossible, notamment dans le cas de *sections critiques*, car les données partagées existent normalement dans un seul exemplaire

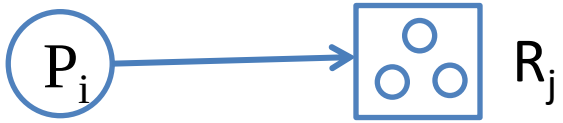
Graphe d'allocation ressources

- Un ensemble de sommets V et d'arêtes E
- V est partitionné en 2 ensembles:
  - $P = \{P_1, P_2, \ldots, P_n\}$, l'ensemble qui consiste de tous les processus dans le système
  - $R = \{R_1, R_2, \ldots, R_m\}$, l'ensemble qui consiste de tous les types de ressources dans le système
- Arêtes requête : arêtes dirigées $P_i \rightarrow R_j$
- Arêtes affectation : arêtes dirigées $R_k \rightarrow P_l$

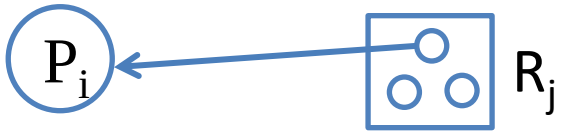
Graphe d'allocation ressources: notation

- Processus: P_i
- Ressource dont il y a un seul exemplaire: R_j
- Ressource dont il y a N exemplaires (instances) (ex: 4) :

- P_i demande un exemplaire de R_j, dont il y en a 3:



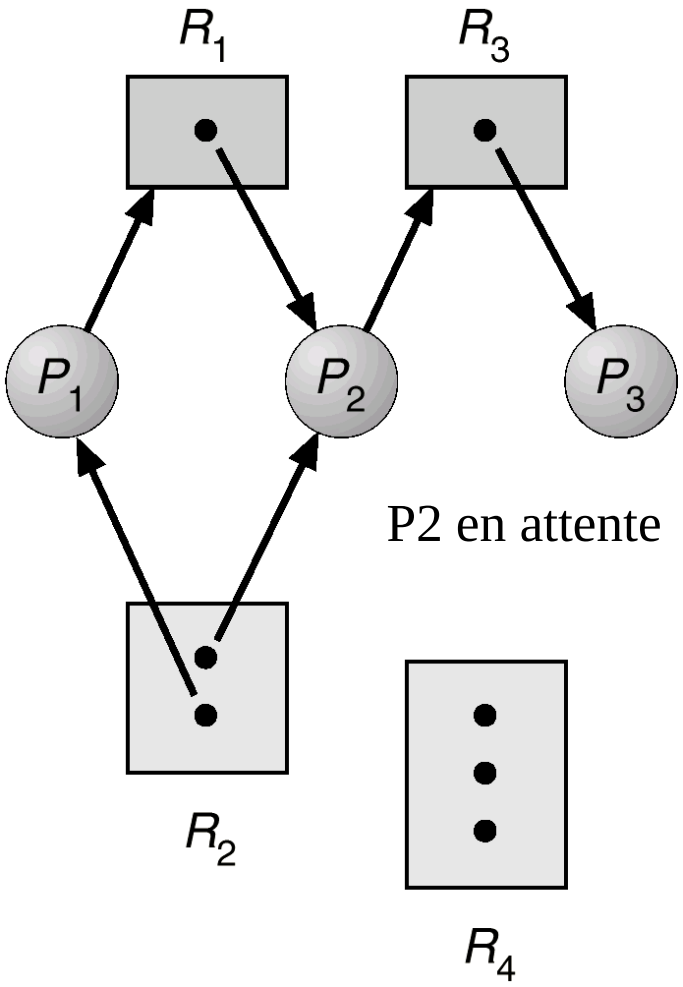
- Pj détient (et utilise) un exemplaire de Rj



Exercice

- Dessiner le graphe d'allocation ressources pour les premiers exemples:
  - Sections critiques imbriquées avec interblocage
  - Problème des philosophes avec interblocage

Exemple de graphe allocation ressources



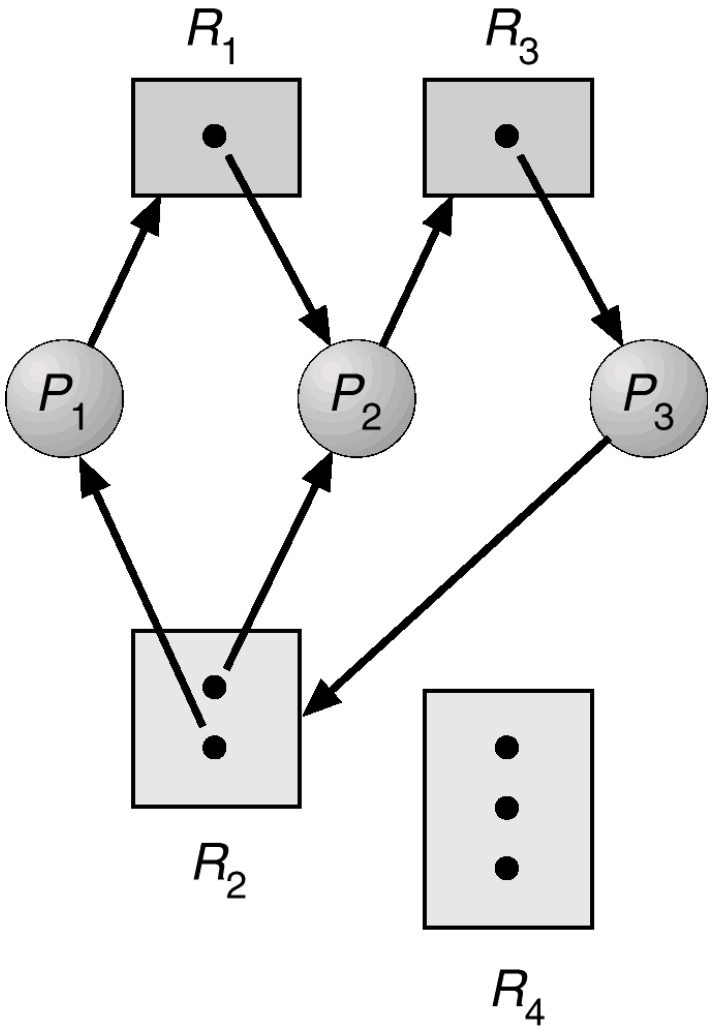
P1 en attente

P3 pas en attente

P2 en attente

Y-a-t-il un interblocage?

Graphe allocation ressources avec interblocage



Nous avons deux cycles:

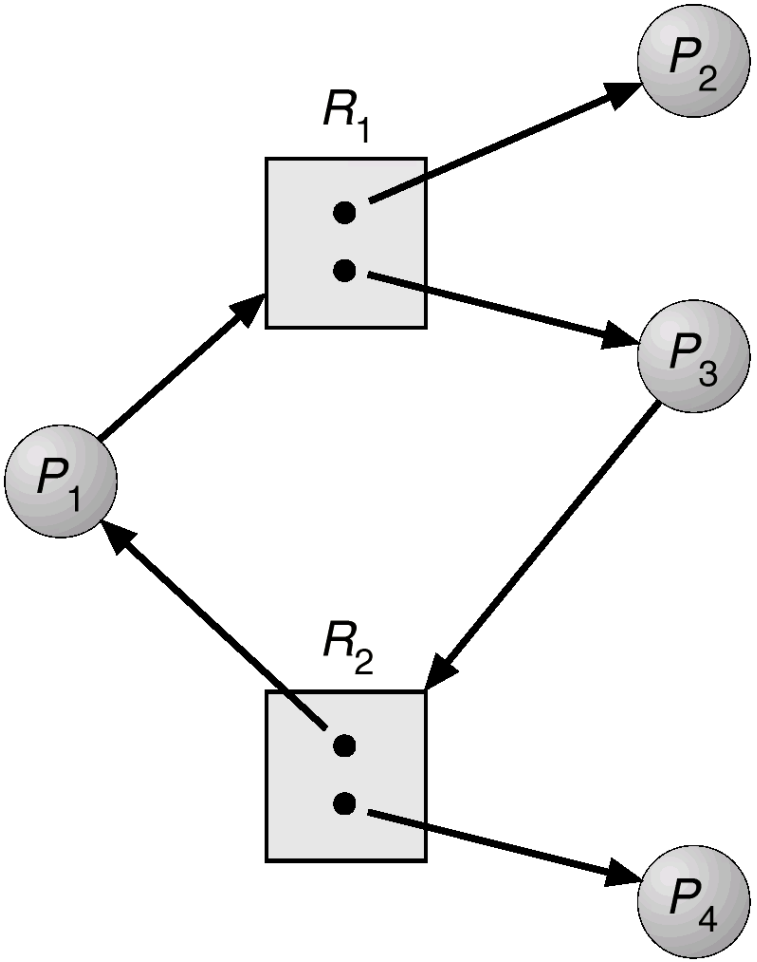
P1 → R1 → P2 → R3 → P3 → R2 → P1

P2 → R3 → P3 → R2 → P2

aucun Processus ne peut terminer

aucune possibilité d'en sortir

Graphe allocation ressources avec cycle, mais pas d' interblocage (pourquoi?)



Pas d'attente circulaire, les ressources peuvent devenir disponibles

Terminaison de processus et libération de ressources

- Hypothèse: Un processus qui a saisi toutes les ressources dont il a besoin peut terminer
  - Cette terminaison pourrait conduire à la libération de ressources
  - Qui pourraient être saisies par d'autre processus qui les attendent
  - Ce qui pourrait conduire à la terminaison d'autres processus
- Il n'y a pas d'interblocage si tous les processus peuvent terminer de cette manière

Constatations

- Les cycles dans le graphe d'allocation de ressources ne signalent pas nécessairement une attente circulaire
  - S'il n'y a pas de cycles dans le graphe, aucun interblocage
  - S'il y a de cycles:
    - Si seulement une ressource par type, interblocage (pourquoi?!)
    - Si plusieurs ressources par type, possibilité d'interblocage
- Il faut se poser la question:
  - Y-a-t-il au moins un processus qui peut terminer et si oui, quels autres processus peuvent terminer en conséquence?

Approches de gestion de l'interblocage

1. Ignorer le problème (Politique de l'autruche)
2. Détection et guérison
   - Réagir en cas d'interblocage
3. Évitement: en allouant les ressources avec précaution
   - Les interblocages sont possibles, mais sont évités (avoidance)
4. Prévention: en empêchant l'apparition d'une condition nécessaire (Exclusion mutuelle, …)
   - Concevoir le système de façon qu'un interblocage soit impossible

1. La politique de l'Autruche

- « Plonger la tête dans le sable en prétendant qu'il n'y a aucun problème ! »
  - Ignorer le problème, qui donc doit être résolu par l'utilisateur.
- Malheureusement, c'est l'approche la plus utilisée!
  - Windows et UNIX adoptent cette stratégie
- Arguments:
  - Les interblocages surviennent rarement
  - Le coût de la prévention ou de la détection est élevé.

2. Détection

- Cas 1: Une seule instance par ressource
  - Interblocage s'il existe un cycle dans le graphe
- Cas 2: Plusieurs instances par ressource
  - L'existence d'un cycle n'implique pas forcément un interblocage;
  - La non-existence de cycle implique qu'il n'y a pas interblocage

Cas d'une seule instance par ressource

- Représenter les demandes et les allocations de ressources par un graphe
  - Le graphe représente l'état du système à un instant donné.
- Utiliser un algorithme de recherche d'un cycle dans un graphe
  - Algorithme coûteux : En général, nombreux processus et ressources

Cas de plusieurs instances (exemplaires) par ressource

- Modélisation de l'allocation de ressources à travers des matrices on considérons que:
  - Le système est composé de *n* processus P1, P2, ...,Pn
  - Et de *m* classes (types) de ressources R1, R2, ...,Rm
- Représenter les demandes et les allocations de ressources par 4 matrices:
  - Cette représentation fait intervenir les structures de données suivantes (qui sont une implémentation du graphes d'allocation) :

Matrices de modélisation d'allocations de ressources

1. Vecteur de ressources existantes (Existing resources):
   - E: Tableau[1..M] d'entiers
   - E[i] = le nombre maximum d'exemplaires de la ressource Ri (disponible au départ). Ex: E=(4,2,3,1)
2. Vecteur des ressources disponibles (Available):
   - Available: Tableau[1..M] d'entiers
   - Available[j]=nombre d'exemplaires libres de la ressource Rj. Ex: Available =(2,1,0,0)
   - Au départ ce vecteur est égal au vecteur de de ressources existantes E.

Matrices de modélisation d'allocations de ressources

3. Matrice d'allocation (Allocation) :
   - Allocation:Tableau[1..N, 1..M] d' entiers;
   - Allocation[i,j]=nombre d'exemplaires de la ressource Rj alloués au processus Pi.

Ex: Allocation =

	R1	R2	R3	R4
P1	0	0	1	0
P2	2	0	0	1
P3	0	1	2	0

Matrices de modélisation d'allocations de ressources

4. Matrice des demandes en attente(Requests) :
   - Request: Tableau[1..N, 1..M] De Entier ;
   - Request[i,j] = nombre d'exemplaires de la ressource Rj demandés et non obtenus par le processus Pi.

Ex: Request =

	R1	R2	R3	R4
P1	2	0	0	1
P2	1	0	1	1
P3	2	1	0	0

Algorithme de détection d'interblocage

Début
Work: Tableau[m] d'Entier ;
Finish: Tableau[n] de booléen ;
Etape 1. Initialisation :
(a) Work = Available.
(b) Pour i = 1; 2; … ; n, Si $Allocation_i$!= 0, Alors Finish[i] = FALSE; Sinon, Finish[i] = TRUE.
Etape 2. Trouver un indice i tel que :
(a) Finish[i] == FALSE, et
(b) $Request_i$ <= Work.
Si un tel i n'existe pas aller à **l'étape 4**.
Etape 3. Work = Work + $Allocation_i$
Finish[i] = TRUE
Aller à l'étape 2
Etape 4. Si Finish[i]=FALSE (pour un certain i)
Alors Le système est dans un état d'interblocage
Sinon Le système n'est pas dans un état d'interblocage

Exercice 1

- Soit les matrices d'allocation des ressources suivantes:

Available =(2,1,0,0)

Allocation =

	R1	R2	R3	R4
P1	0	0	1	0
P2	2	0	0	1
P3	0	1	2	0

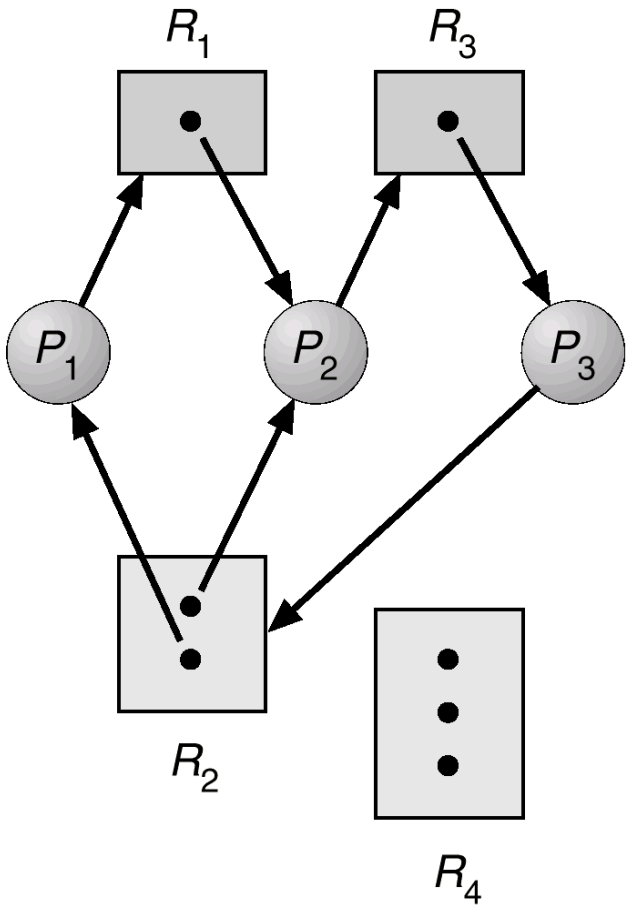
Request =

	R1	R2	R3	R4
P1	2	0	0	1
P2	1	0	1	0
P3	2	1	0	0

- Est-ce qu'il y a un interblocage?
- Dessiner le graphe d'allocation de ressources correspondant

Exercice 2

- Représenter ce graphe d'allocation de ressources par des matrices
- Est qu'il ya un inerblocage?



Exercice 3

- Est-ce qu'on peut appliquer l'algorithme de détection d'interblocage dans le cas d'un seul exemplaire par ressource?
- Appliquer l'algorithme au problème des philosophes avec interblocage

Algorithme de détection d'interblocage

Critique de l'algorithme de détection :

- L'algorithme de détection des interblocages est très coûteux s'il est exécuté après chaque demande de ressources.
- L'idée donc c'est de le lancer périodiquement,
  - Mais comment choisir cette période ?

Mise en œuvre de la détection

- Quand lancer l'algorithme de détection d'interblocage?
- La réponse à cette question dépend de deux facteurs:
  - La fréquence d'apparition des interblocages;
  - Le nombre de processus et de ressources concernés par l'interblocage.

Mise en œuvre de la détection

3 possibilités:

1. L'algorithme de détection est exécuté à chaque changement du graphe d'allocation de ressources
   - Coût élevé en temps processeur
2. L'algorithme est exécuté périodiquement
   - Coût dépond de la période.
3. L'algorithme est exécuté à chaque fois qu'une requête n'est pas satisfaite (blocage).

Principaux inconvénients de la détection

- Le coût de l'entretient des graphes
  - Les graphes doivent être mis à jour à chaque allocation ou libération de ressource.
- Le coût de la détection d'interblocage
  - Le nombre de nœuds dans les graphes peut être important.

Guérison d’interblocage

- La guérison de l'interblocage vise à reprendre l’exécution du système dans un état cohérent.
- Deux solutions existes:
  - Préemption de ressource (sans destruction de processus).
  - Destruction de processus

Préemption de ressource

- Retirer temporairement une ressource à un processus pour l'attribuer à un autre.
  - Ce type de solution n'est pas souvent possible
  - Dépend du type de ressource (les ressources n'étant pas toutes réquisitionnables),
- Comment reprendre l'exécution d'un processus suspendu?

Préemption avec rollback

- Il est possible de placer des points de reprise sur les processus:
  - En sauvegardant l'état du processus dans un fichier.
  - Le point de reprise contient l'image mémoire du processus, ainsi que l'état des ressources.
- Ce type de reprise est extrêmement coûteux et lourd à mettre en œuvre.

Destruction de processus

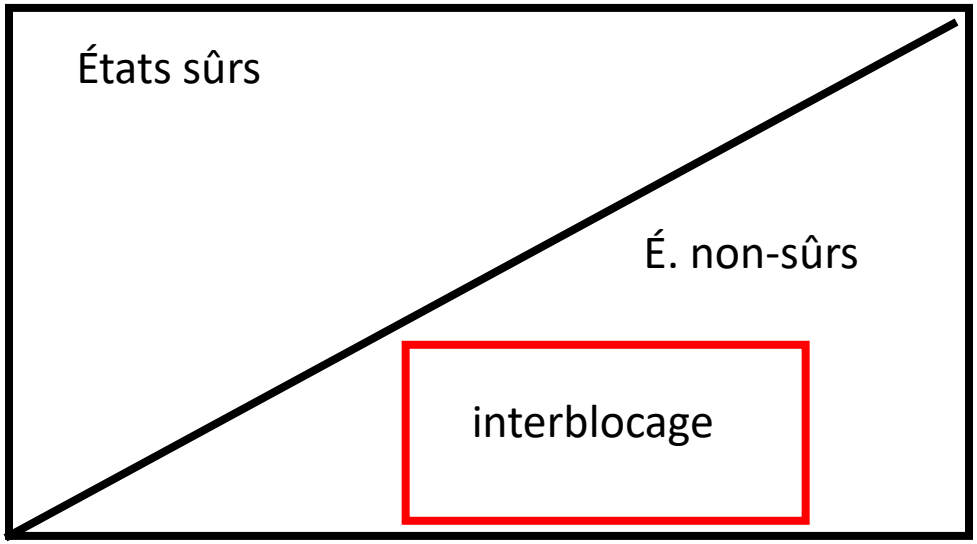
- Tuer tous les processus bloqués
  - Solution radicale
- Si nous tuons un des processus, il libèrera peut-être des ressources nécessaires
  - Problème du choix du processus à tuer.
- Faire repartir le traitement à partir d'un point de contrôle antérieur (rollback),

3. Éviter les interblocages (deadlock avoidance)

- A chaque demande d'allocation de ressources, le système vérifie si cette allocation peut mener à un état non-sûr.
  - Si c'est le cas la demande d'allocation est refusée
- Le système examine les séquences d 'exécution possibles pour voir si un blocage est possible

État sûr (prudent, sain, certain,..)

- Un état est sûr si le système peut en sortir sans interblocages:
  - Il existe une séquence sûre à partir de cet état.
- Ne pas allouer une ressource à un processus si l'état qui en résulte n'est pas sûr



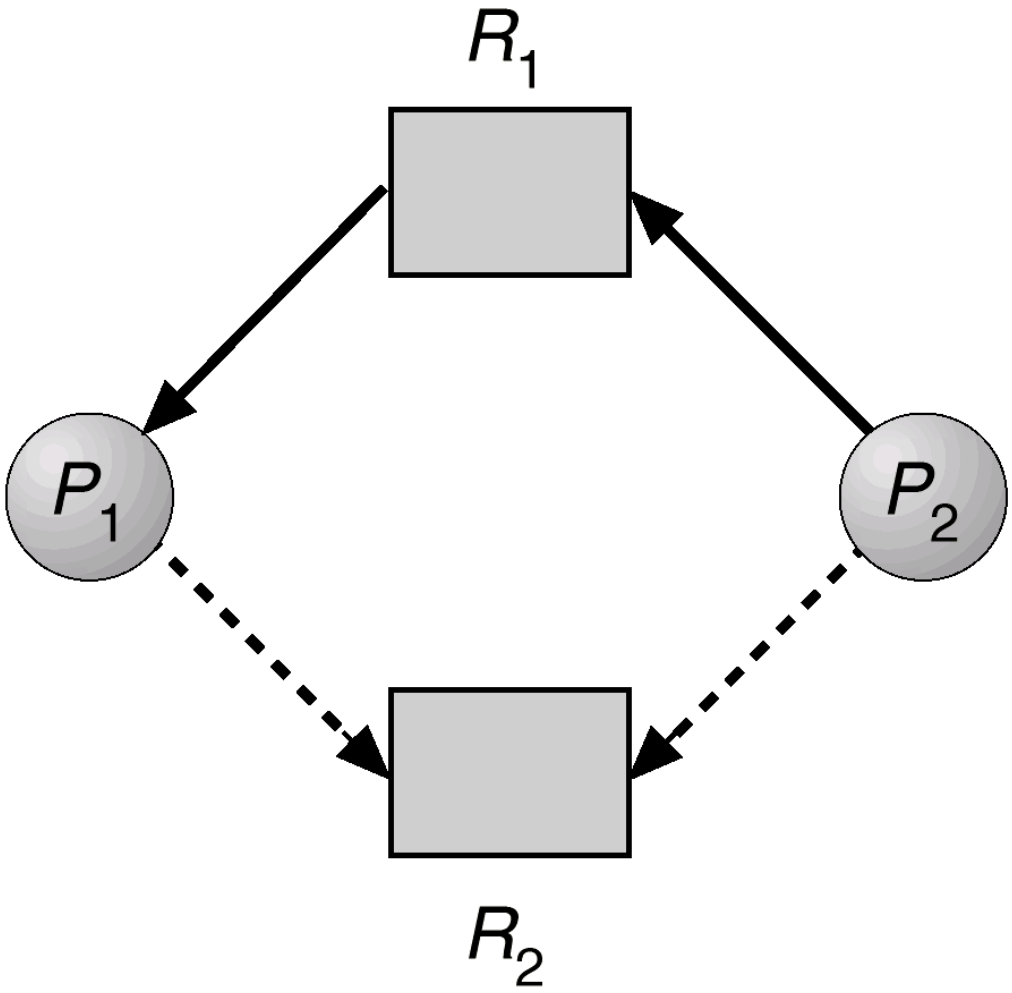
Séquence sûre

- Une séquence de processus <P_1, P_2, …, P_n> est sûre si pour chaque Pi, les ressources que Pi peut encore demander peuvent être satisfaites par les ressources couramment disponibles + ressources utilisées par *les Pj qui les précèdent*.
  - Quand Pi aboutit, Pi+1 peut obtenir les ressources dont il a besoin, terminer, donc
- <P_1, P_2, …, P_n> est un ordre de terminaison de processus: tous peuvent se terminer dans cet ordre.

Cas d'un seul exemplaire par ressource: Algorithme d'allocation de ressources

- Il faut maintenant prendre en considération:
  - les requêtes possibles dans le futur (chaque processus doit déclarer ça)
- Arête demande P_i - - - - - - → R_j indique que le processus P_i peut demander la ressource R_j (ligne à tirets)

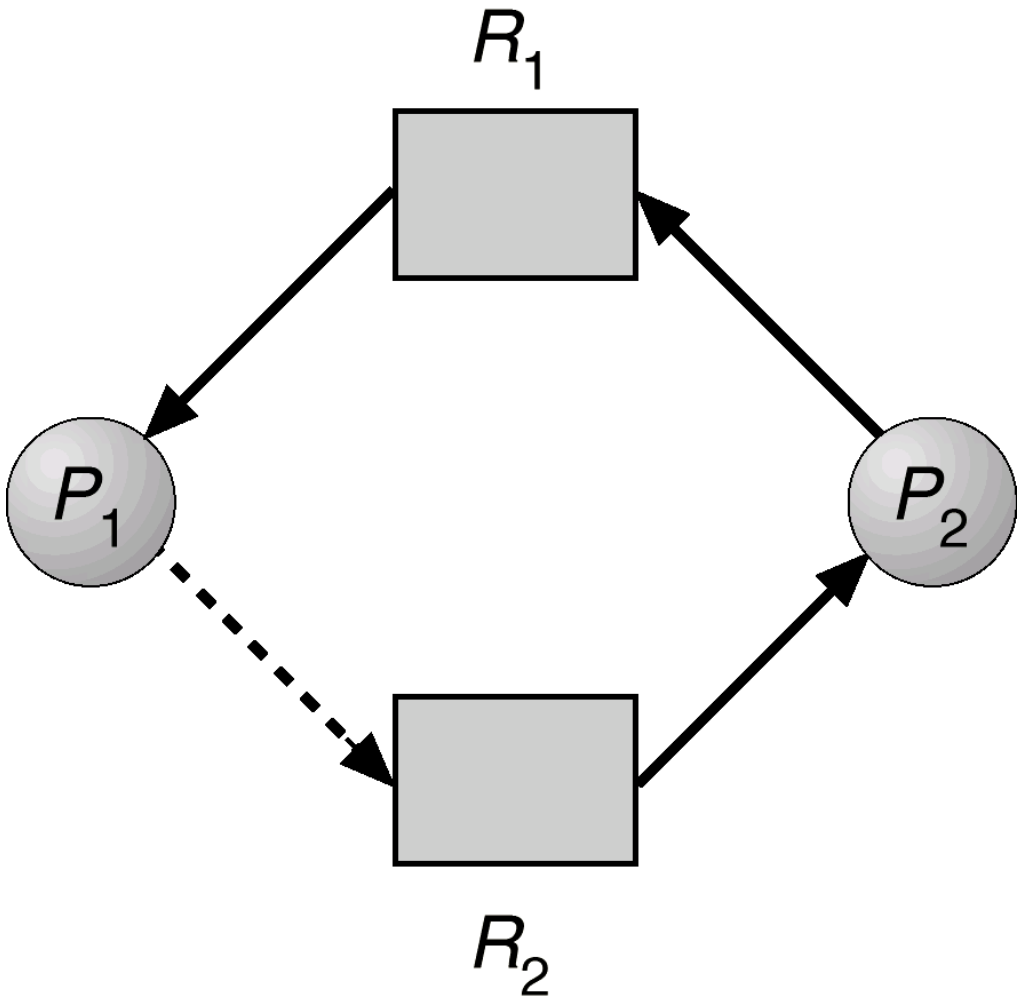
Graphe d'allocation ressources



Ligne continue: requête courante;

tirets: requête possible dans le futur

Exemple: Un état non-sûr



Si P2 demande R2, ce dernier ne peut pas lui être donné, car ceci peut causer un cycle dans le graphe: P1 req R2,

Cas de plusieurs exemplaire par ressource: l'algorithme du banquier

- Les processus sont comme des clients qui désirent emprunter de l'argent (ressources) à la banque...
- Un banquier ne devrait pas prêter de l'argent s'il ne peut pas satisfaire les besoins de tous ses clients

Algorithme du banquier

- Cet algorithme (Dijistra, Habermann) consiste à examiner chaque nouvelle requête pour voir si elle conduit à un état sûr.
  - Si c'est le cas, la ressource est allouée,
  - sinon la requête est mise en attente.
- L'algorithme détermine donc si un état est ou non sûr

Structures de données

- Available : Vecteur de longueur m indiquant le nombre de ressources disponibles de chaque type.
  - Available[j]=k: veut dire que le type de ressources Rj possède k instances disponibles.
- Max : Matrice n×m définissant la demande maximale de chaque processus.
  - Si Max[i, j]=k: cela veut dire que le processus Pi peut demander au plus k instances du type de ressources Rj.

Structures de données

- Allocation : Matrice nxm définissant le nombre de ressources de chaque type de ressources actuellement alloué à chaque processus.
  - Si Allocation[i, j]=k: cela veut dire que l'on a alloué au processus Pi k instances du type de ressources Rj.
- Need : Matrice nxm indiquant les ressources restant à satisfaire à chaque processus.
  - Si Need[i, j]=k: cela veut dire que le processus Pi peut avoir besoin de k instances au plus du type de ressources Rj pour achever sa tâche.
- Request : Matrice nxm indiquant les ressources supplémentaires que les processus viennent de demander.
  - Si Request[i, j]=k: cela veut dire que le processus Pi vient de demander k instances supplémentaires du type de ressources Rj.

Algorithme du banquier

- Pour décider si la requête doit être accordée, l'algorithme du banquier teste si cette allocation conduira le système dans un état sûr:
  - accorder la requête si l'état est sûr
  - sinon refuser la requête
- Un état est sûr ssi il existe une séquence {P1..Pn} où chaque Pi est alloué toutes les ressources dont il a besoin pour terminer
  - ie: nous pouvons toujours exécuter tous les processus jusqu'à terminaison à partir d'un état sûr

Algorithme du banquier

❑ Cet algorithme est appelé à chaque fois qu'un processus fait une demande de ressources

Début

Etape 1 : Si $Request_i <= Need_i$ Alors Aller à l'étape 2

Sinon erreur : le processus a excédé ses besoins maximaux

Etape 2 : Si $Request_i <= Available$ Alors Aller à l'étape 3

Sinon Attendre : les ressources ne sont pas disponibles.

Etape 3 : Sauvegarder l'état du système (les matrices Available, Allocation et Need).

Allouer les ressources demandées par le processus P_i en modifiant l'état du système de la manière suivante :

$Available := Available - Request_i$

$Allocation_i := Allocation_i + Request_i$

$Need_i := Need_i - Request_i$

Si Verification_Etat_Sain=Vrai

Alors L'allocation est validée

Sinon L'allocation est annulée ; Restaurer l'ancien Etat du système

Fin.

Algorithme Verification_Etat_Sain

Début
Work : Tableau[m] de Entier ;
Finish : Tableau[n] de Logique ;
Etape 1 : **Initialisation**
Work :=Available
Finish :=Faux ;
Etape 2 : Trouver i tel que : Finish[i]=faux et $Need_i$<=Work
Si un tel i n'existe pas aller à l'étape 4.
Etape 3 : Work :=Work + $Allocation_i$
Finish[i] :=Vrai
Aller à l'étape 2
Etape 4 : Si Finish=Vrai (pour tout i)
Alors Verification_Etat_Sain :=Vrai
Sinon Verification_Etat_Sain :=Faux
Finsi
Fin.

Exercice 1

- Nous avons 3 types de ressources avec les quantités suivantes:
  - R(1) = 9, R(2) = 3, R(3) = 6
- et 4 processus avec l'état initial:

	Max			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	1	1	2
P2	6	1	3	5	1	1			
P3	3	1	4	2	1	1			
P4	4	2	2	0	0	2			

  - Supposer que la requête de P2 est Q = (1,0,1). Doit-elle être accordée?

Exercice 2

- A partir de l'état initial de l'exercice 1, supposer que la requête de P1 est Q = (1,0,1). Doit-elle être accordée?

Critique de l'algorithme du Banquier

- Coûteux : L'algorithme est très coûteux en temps d'exécution et en mémoire
  - Il faut maintenir plusieurs matrices, et déclencher à chaque demande de ressource, l'algorithme de vérification de l'état sain qui demande mxn^2 opérations.
- Théorique : L'algorithme exige que chaque processus déclare à l'avance les ressources qu'il doit utiliser, en type et en nombre.
  - Cette contrainte est difficile à réaliser dans la pratique.
- Pessimiste : L'algorithme peut retarder une demande de ressources dès qu'il y a risque d'interblocage (mais en réalité l'interblocage peut ne pas se produire).

4. Prévention de l'interblocage

- Pour qu'un interblocage puisse se produire, il faudrait que les 4 conditions d'interblocage soient vérifiées:
  - Exclusion Mutuelle
  - Possession et attente
  - Pas de réquisition
  - Attente circulaire
- Pour prévenir l'interblocage il suffit d'éliminer une condition d'entre elles.

A. Éliminer l'exclusion mutuelle

- Exemple : deux processus souhaitent imprimer le contenu du même fichier (avec interblocage) :
  - L'un verrouille l'imprimante puis le fichier
  - L'autre verrouille le fichier puis l'imprimante
- Solution : au lieu d'imprimer directement, mettre la requête d'impression dans une file d'impression (Spooling)
  - Donne l'illusion que l'attribution a été possible alors que ce n'est pas forcément le cas …
- Rarement faisable dans le cas général (ressources critiques)

B. Éliminer « Possession et attente »

- Pour éliminer cette condition, on doit s'assurer que le processus ayant fait la requête ne dispose pas d'autres ressources.
- Deux approches possibles:
  1. Allouer toutes les ressources demandées d'un seul coup (ou rien);
  2. Imposer la Libération des ressources.

1) L'allocation globale

- Demander toutes les ressources dont on a besoin à accès exclusif d'un seul coup
  - Une seule SC « utilisant » toutes les ressources
- Principe: rendre atomique (indivisible) la séquence d'acquisition des ressources
  - On obtient toutes les ressources ou aucune ("tout ou rien"),
  - La phase d'acquisition étant elle même une section critique

Inconvénients

- Il est généralement impossible de prédire les ressources effectivement utilisées.
- Un processus risque d'attendre longtemps avant d'obtenir ses ressources.
- Il peut y avoir un gaspillage éventuel de ressources.
  - On oblige le processus à demander des ressources à un moment où il n'en a pas besoin.

2) L'allocation partielle (deux possibilités)

1. Un processus doit d'abord libérer la ressource acquise avant de demander une autre.
   - Cette solution n'est pas toujours possible.
2. Si un processus détient certaines ressources et si on lui refuse une ressource supplémentaire, il doit libérer les ressources détenus.
   - Cette solution n'est pas toujours possible.

C. Éliminer « Pas de réquisition »

- Suppose de récupérer de force certaines ressources utilisées par un processus
  - Soit en terminant le processus
  - Soit en étant capable de le faire revenir en arrière
  - Nécessite de prévoir des « points de reprise » (rollback)
- Autoriser la réquisition n'est pas toujours souhaitable ou bien est délicate …

Exemples de problèmes de la réquisition

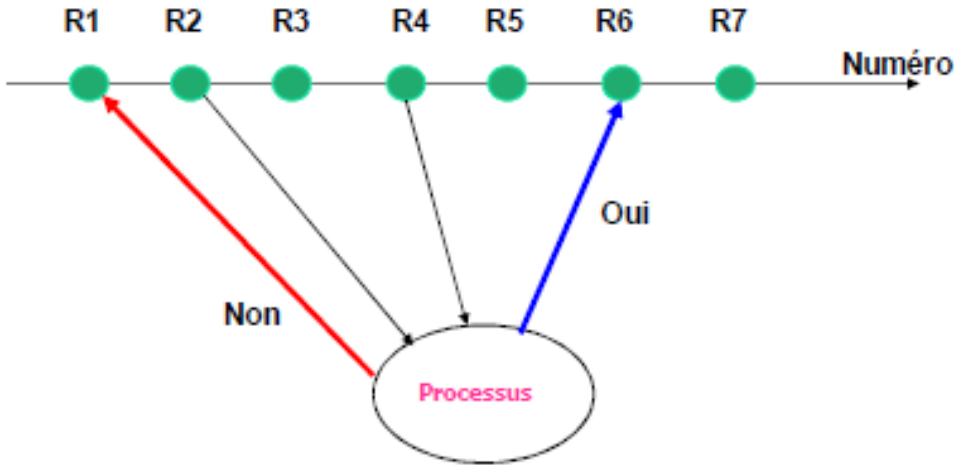
- Difficile d'interrompre un processus qui en train d'utiliser une imprimante !
  - Car annuler les effets de l'utilisation de la ressource supposerait d'effacer des lignes déjà imprimées!!!
- Difficile d'annuler une transaction sur une BD (avec verrouillage d'enregistrements)
  - Le « rollback » nécessite l'usage d'un journal pour enregistrer les modifications effectuées.

D. Éliminer l’attente circulaire

- Solution possible : numérotation des ressources
  - Les processus peuvent demander toutes les ressources dont ils ont besoin, à condition de respecter l'ordre croissant de la numérotation des ressources.
  - Permet de briser la condition d'attente circulaire.

Numérotation des ressources

- Si un processus détient des ressources d'un type donné, il ne peut demander que des ressources dont les numéros sont plus élevés.



Numérotation des ressources

- Avantage: évite les inconvénients de la méthode d'allocation globale et améliore, donc, la gestion des ressources.
- Inconvénient: Priorité fixée de manière statique entre les classes de ressources
  - Rigidité dans la programmation des demandes de ressources des processus.