

Interblocages

- Modèle Système
- Caractérisation d'interblocage
- Méthodes pour Gérer les Interblocages
- Prévention des Interblocages
- Evitement des Interblocages
- Détection des Interblocages
- Recouvrement des Interblocages
- Approche Combinée pour la Gestion des Interblocages

Le Problème d'Interblocage

- Un ensemble de processus bloqués chacun ayant une ressource et attendant l'acquisition d'une ressource déjà utilisée par un autre processus dans l'ensemble.
- Exemple
 - Système a 2 disques
 - P_1 et P_2 chacun utilise un disque et chacun a besoin d'un autre.
- Exemple
 - sémaphores A et B , initialisés à 1

P_0

wait (A);

wait (B);

P_1

wait(B)

wait(A)



Modèle Système

- Types de Ressources R_1, R_2, \dots, R_m

Cycles CPU, Espace Mémoire, Périphériques d'E/S

- Chaque ressource de type R_i a W_i instances.
- Chaque processus utilise une ressource comme suit:
 - requête
 - utilisation
 - libération

Caractérisation de l'Interblocage

Interblocage si les 4 conditions sont vérifiées simultanément.

- **Exclusion Mutuelle:** seulement un processus à un certain moment peut utiliser une ressource.
- **Hold and wait:** un processus ayant au moins une ressource est en attente de ressources additionnelles utilisées par d'autres processus.
- **Pas de préemption:** une ressource ne peut être relâchée que librement par le processus l'utilisant.
- **Attente circulaire:** il existe un ensemble $\{P_0, P_1, \dots, P_n\}$ de processus en attente tel que P_0 est en attente d'une ressource retenue par P_1 , P_1 en attente d'une ressource retenue par P_2 , ..., P_{n-1} en attente d'une ressource retenue par P_n , et P_n en attente d'une ressource retenue par P_0 .

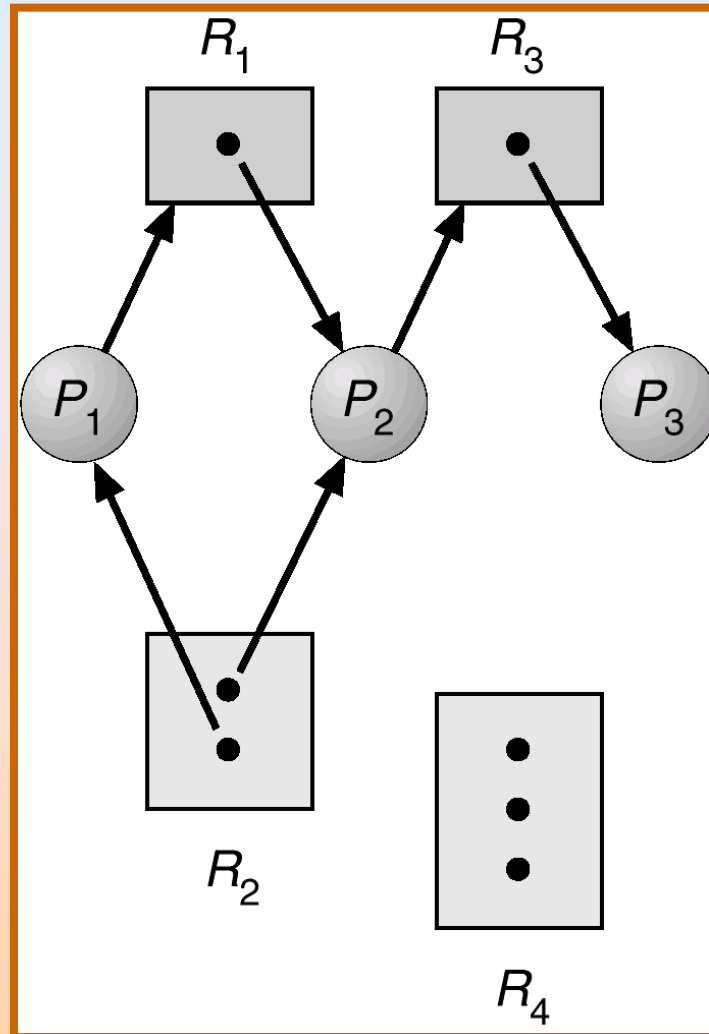
Graphe d'Allocation de Ressources

Un ensemble de noeuds V et un ensemble de vecteurs E .

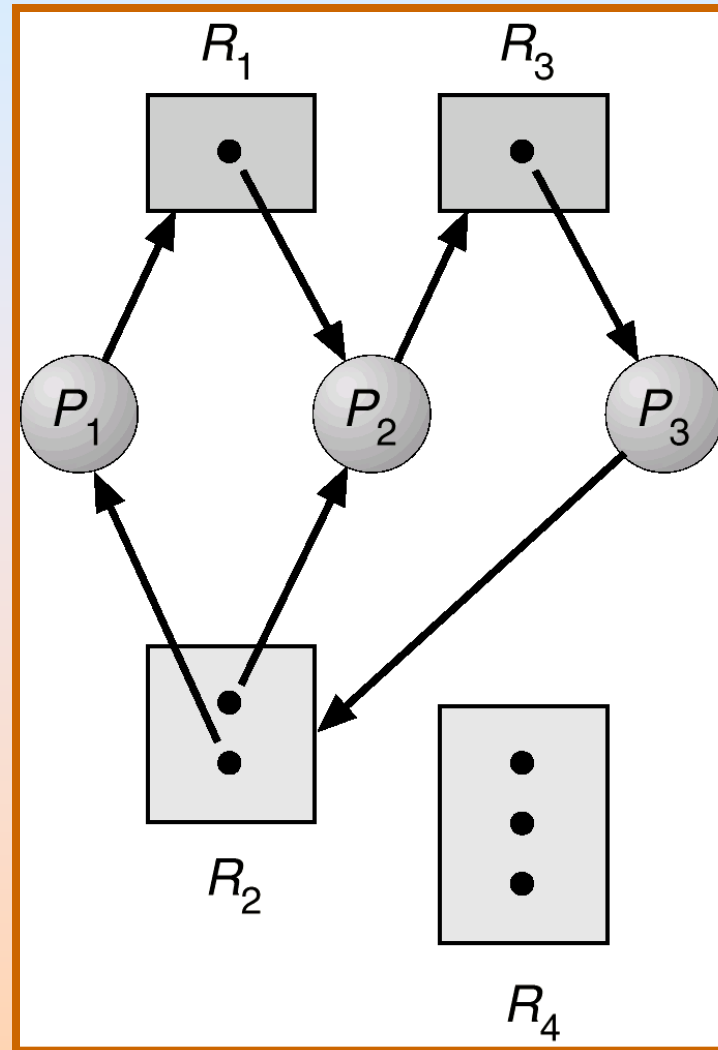
- V est partitionné entre 2 types:
 - $P = \{P_1, P_2, \dots, P_n\}$, l'ensemble consistant de tous les processus dans le système.
 - $R = \{R_1, R_2, \dots, R_m\}$, l'ensemble des ressources dans le système.
- Vecteur requête – vecteur orienté $P_i \rightarrow R_j$
- Vecteur d'affectation – vecteur dirigé $R_j \rightarrow P_i$



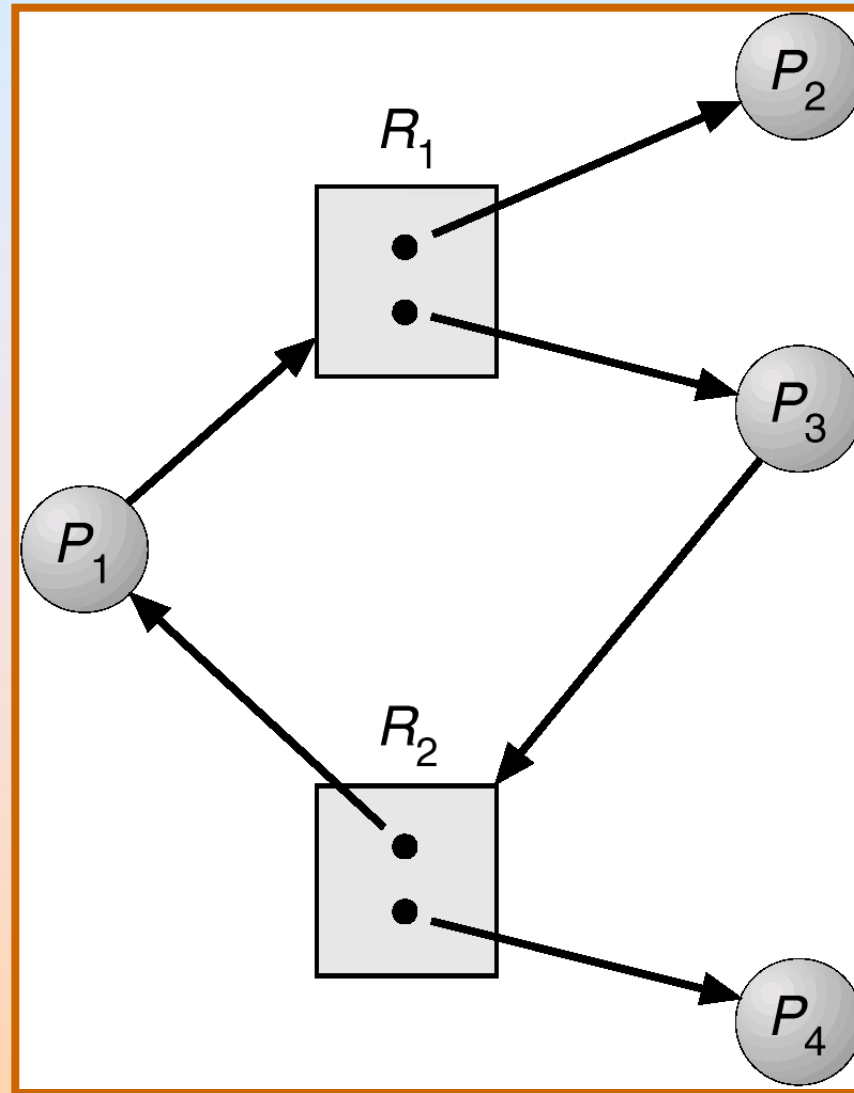
Exemple d'un Graphe d'Allocation de Ressources





Graphe d'Allocations de Ressources avec Interblocage



Graphes d'Allocation Ressources avec cycle mais sans Interblocage



Faits Basiques

- Si le graphe ne contient pas de cycles  pas d'interblocage.
- Si le graphe contient un cycle 
 - Si seulement on a une instance par ressource, alors interblocage.
 - Si plusieurs instances par ressource, possibilité d'interblocage.

Méthodes pour Gérer les Interblocages

- Assurer que le système n'entre jamais dans un état d'interblocage.
- Permettre au système d'entrer en interblocage et récupérer.
- Ignorer le problème et prétendre que les interblocages n'arrivent jamais sur le système; utilisé par la plupart des OSs, UNIX compris.

Prévention des Interblocages

Restreindre les façons dont les requêtes peuvent être émises.

- **Exclusion Mutuelle** – pas demandée pour des ressources partagées; doit tenir pour des ressources non partagées.
- **Hold and Wait** – doit garantir que quand un processus demande une ressource, il ne doit pas avoir une autre ressource.
 - Requièrè qu'un processus demande et ait toutes ses ressources avant le début de son exécution, ou permettre au processus de demander des ressources seulement si le processus n'en n'a pas déjà.
 - Utilisation des ressources basse; famine possible.

Prévention des Interblocages (Cont.)

■ Pas de préemption –

- Si un processus qui a déjà des ressources demande d'autres ressources qui ne sont pas directement allouables, alors toutes ses ressources sont libérées.
- Les ressources préemptées sont ajoutées à la liste de ressources que le processus attend.
- Le processus sera relancé seulement quand il pourra avoir l'ensemble de ses anciennes ressources, ainsi que les nouvelles demandées.

■ Attente Circulaire – impose un ordre total sur tous les types de ressources et demande que chaque processus fasse des requêtes de ressources dans un ordre.

Evitement des Interblocages

Demande que le système ait des informations disponibles.

- Le plus simple et le plus utile des modèles demande que chaque processus déclare le nombre maximum de ressources de chaque type qu'il pourrait utiliser.
- L'algorithme d'évitement examine dynamiquement l'état des allocations de ressources pour s'assurer qu'il ne peut pas y avoir d'attente circulaire.
- L'état des allocations de ressources est défini par le nombre de ressources libres et allouées, et le maximum de requêtes de chaque processus.

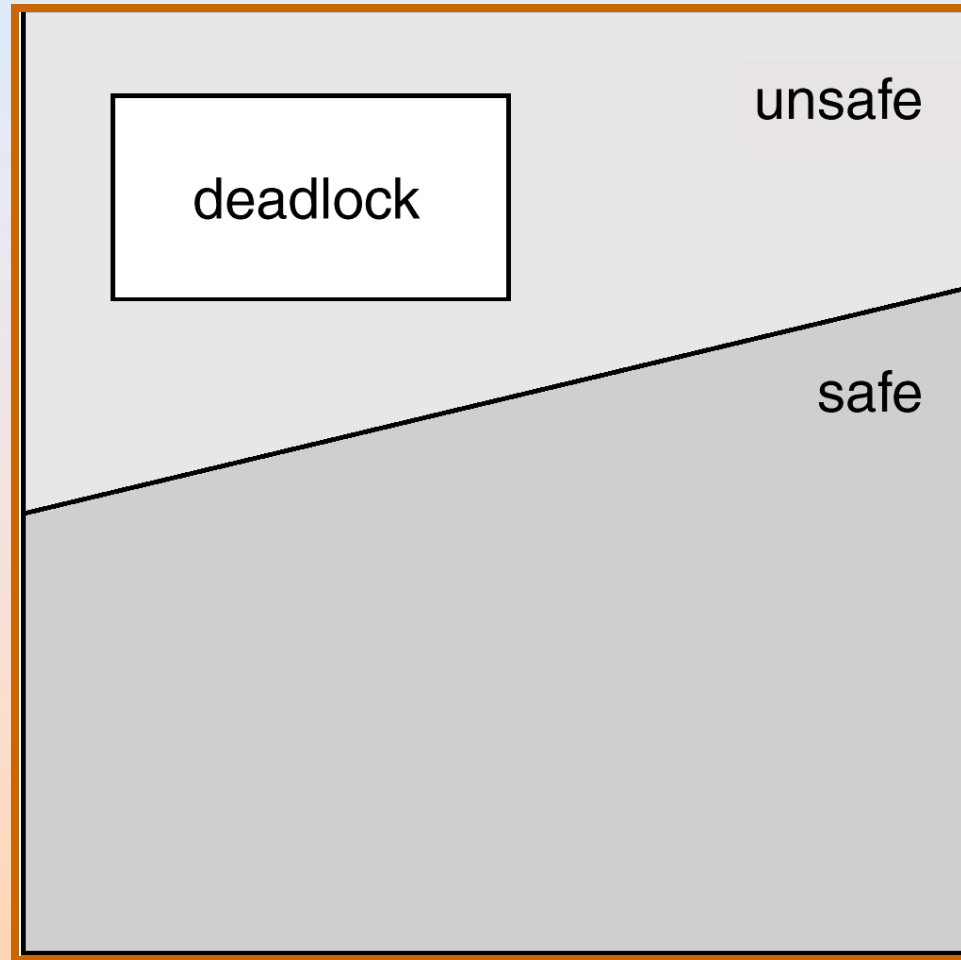
Etat Sûr

- Quand un processus demande une ressource disponible, le système doit décider si l'allocation directe laisse le système dans un état sûr.
- Un système est dans un état sûr s'il existe une séquence sûre pour tous les procesus.
- La séquence $\langle P_1, P_2, \dots, P_n \rangle$ est sûre si pour chaque P_i , les ressources que P_i peut encore demander peuvent être satisfaites par les ressources disponibles + ressources allouées aux processus P_j , avec $j < i$.
 - Si les besoins de P_i en ressources ne sont pas immédiatement disponibles, alors P_i peut attendre que tous les P_j soient terminés.
 - Quand P_j se termine, P_i peut obtenir les ressources, s'exécuter, retourner les ressources allouées, puis se terminer.
 - Quand P_j se termine, P_{j+1} peut obtenir ses ressources, et ainsi de suite.

Faits de Base

- Si un système est dans un état sûr \Rightarrow pas d'interblocage.
- Si un système est dans un état non sûr \Rightarrow possibilité d'interblocage.
- Evitement \Rightarrow assurer que le système n'entrera jamais dans un état non sûr.

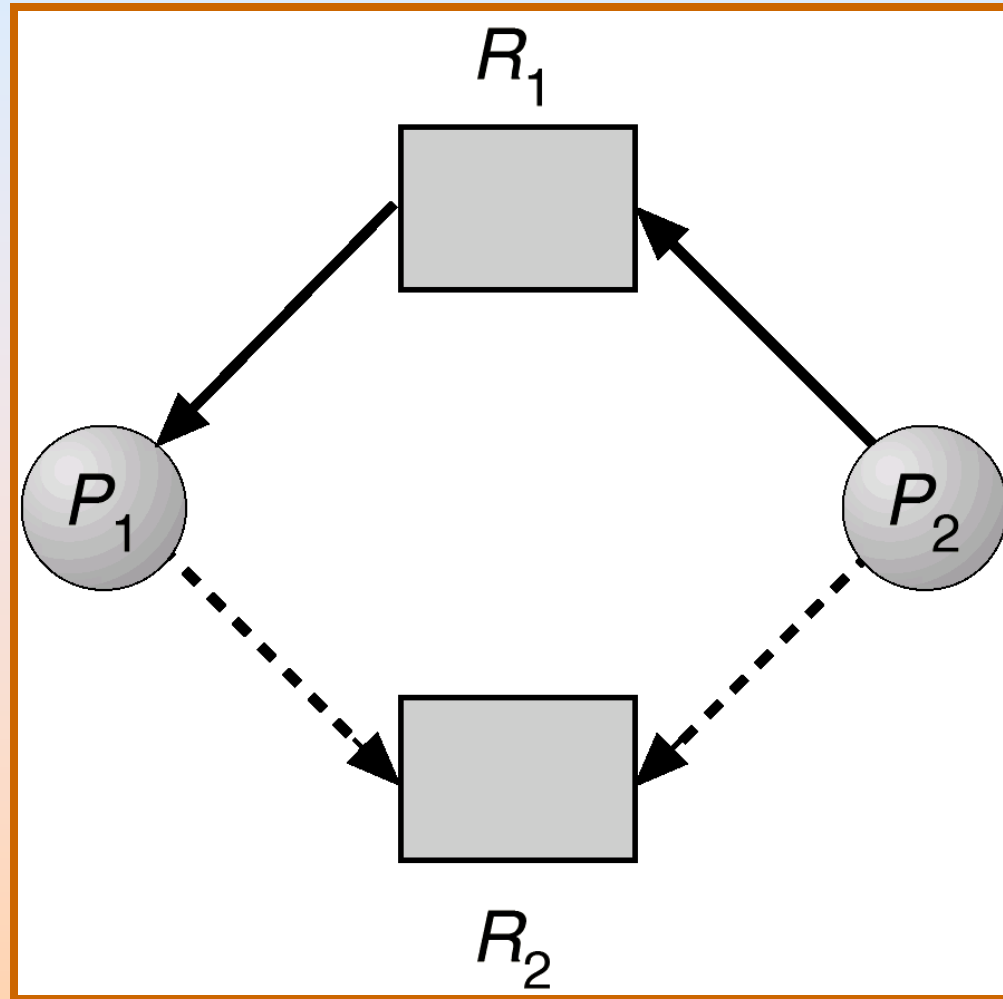
Etats Sûrs, Non Sûrs , Interblocage s



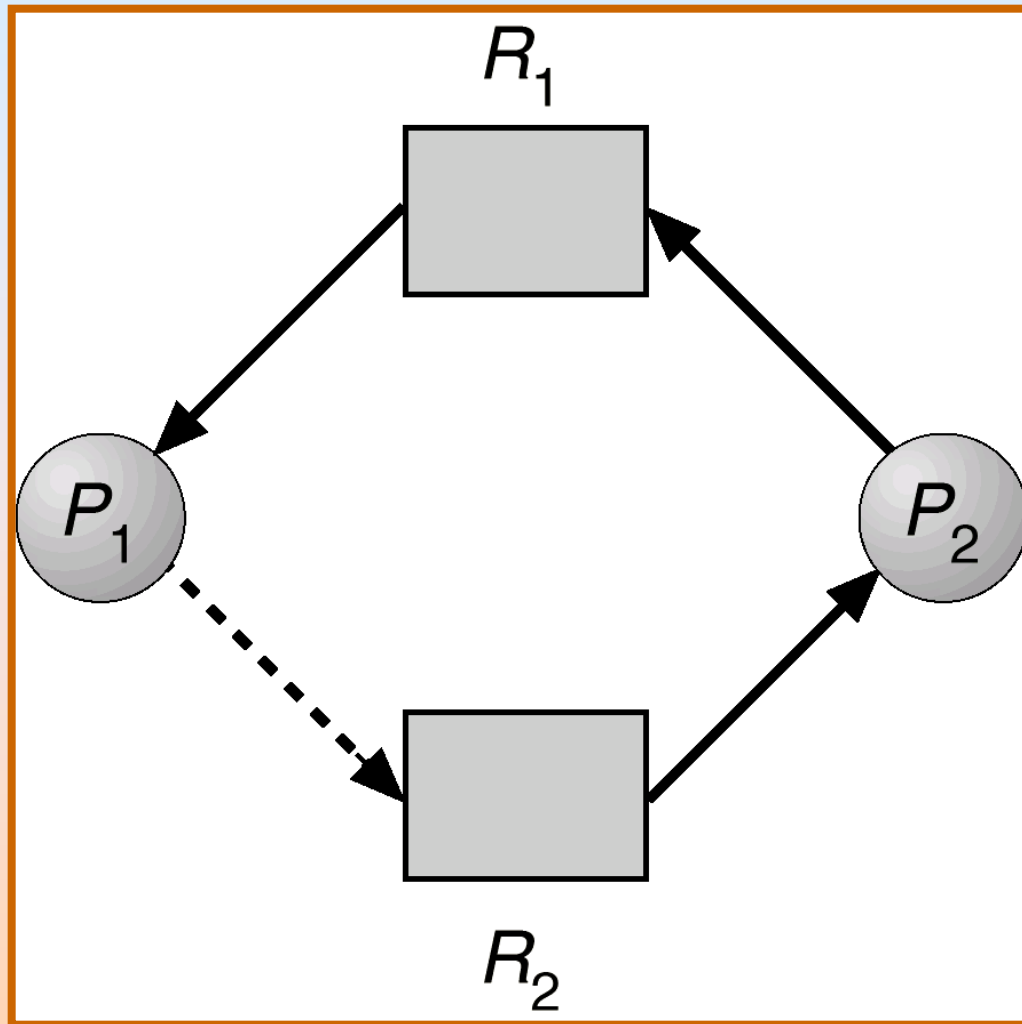
Resource-Allocation Graph Algorithm

- Claim edge $P_i \rightarrow R_j$ indicated that process P_i may request resource R_j ; represented by a dashed line.
- Claim edge converts to request edge when a process requests a resource.
- When a resource is released by a process, assignment edge reconverts to a claim edge.
- Resources must be claimed *a priori* in the system.

Resource-Allocation Graph For Deadlock Avoidance



Unsafe State In Resource-Allocation Graph



Banker's Algorithm

- Multiple instances.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.

Data Structures for the Banker's Algorithm

Let n = number of processes, and m = number of resources types.

- *Available*: Vector of length m . If available $[j] = k$, there are k instances of resource type R_j available.
- *Max*: $n \times m$ matrix. If $Max[i, j] = k$, then process P_i may request at most k instances of resource type R_j .
- *Allocation*: $n \times m$ matrix. If $Allocation[i, j] = k$ then P_i is currently allocated k instances of R_j .
- *Need*: $n \times m$ matrix. If $Need[i, j] = k$, then P_i may need k more instances of R_j to complete its task.

$$Need[i, j] = Max[i, j] - Allocation[i, j].$$

Safety Algorithm

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively.
Initialize:

Work = *Available*

Finish [*i*] = *false* for *i* = 1, 3, ..., *n*.

2. Find an *i* such that both:

(a) *Finish* [*i*] = *false*

(b) *Need*_{*i*} ≤ *Work*

If no such *i* exists, go to step 4.

3. *Work* = *Work* + *Allocation*_{*i*}
Finish [*i*] = *true*
go to step 2.

4. If *Finish* [*i*] == *true* for all *i*, then the system is in a safe state.

Resource-Request Algorithm for Process P_i

$Request$ = request vector for process P_i . If $Request_i[j] = k$ then process P_i wants k instances of resource type R_j .

1. If $Request_i \nlessgtr Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $Request_i \nlessgtr Available$, go to step 3. Otherwise P_i must wait, since resources are not available.
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

$Available = Available - Request_i$

$Allocation_i = Allocation_i + Request_i$

$Need_i = Need_i - Request_i$

- If safe \Rightarrow the resources are allocated to P_i .
- If unsafe $\Rightarrow P_i$ must wait, and the old resource-allocation state is restored

Example of Banker's Algorithm

- 5 processes P_0 through P_4 ; 3 resource types A (10 instances), B (5 instances, and C (7 instances).
- Snapshot at time T_0 :

| | <u>Allocation</u> | | | <u>Max</u> | | | <u>Available</u> |
|-------|-------------------|---|---|------------|---|---|------------------|
| | A | B | C | A | B | C | A B C |
| P_0 | 0 | 1 | 0 | 7 | 5 | 3 | 3 3 2 |
| P_1 | 2 | 0 | 0 | 3 | 2 | 2 | |
| P_2 | 3 | 0 | 2 | 9 | 0 | 2 | |
| P_3 | 2 | 1 | 1 | 2 | 2 | 2 | |
| P_4 | 0 | 0 | 2 | 4 | 3 | 3 | |

Example (Cont.)

- The content of the matrix. Need is defined to be Max – Allocation.

| | <u>Need</u> |
|-------|-------------|
| | A B C |
| P_0 | 7 4 3 |
| P_1 | 1 2 2 |
| P_2 | 6 0 0 |
| P_3 | 0 1 1 |
| P_4 | 4 3 1 |

- The system is in a safe state since the sequence $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ satisfies safety criteria.

Example P_1 Request (1,0,2) (Cont.)

- Check that Request $\not\leq$ Available (that is, $(1,0,2) \not\leq (3,3,2)$  true.

| | <u>Allocation</u> | | | <u>Need</u> | | | <u>Available</u> | | |
|-------|-------------------|---|---|-------------|---|---|------------------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P_0 | 0 | 1 | 0 | 7 | 4 | 3 | 2 | 3 | 0 |
| P_1 | 3 | 0 | 2 | 0 | 2 | 0 | | | |
| P_2 | 3 | 0 | 1 | 6 | 0 | 0 | | | |
| P_3 | 2 | 1 | 1 | 0 | 1 | 1 | | | |
| P_4 | 0 | 0 | 2 | 4 | 3 | 1 | | | |

- Executing safety algorithm shows that sequence $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ satisfies safety requirement.
- Can request for (3,3,0) by P_4 be granted?
- Can request for (0,2,0) by P_0 be granted?

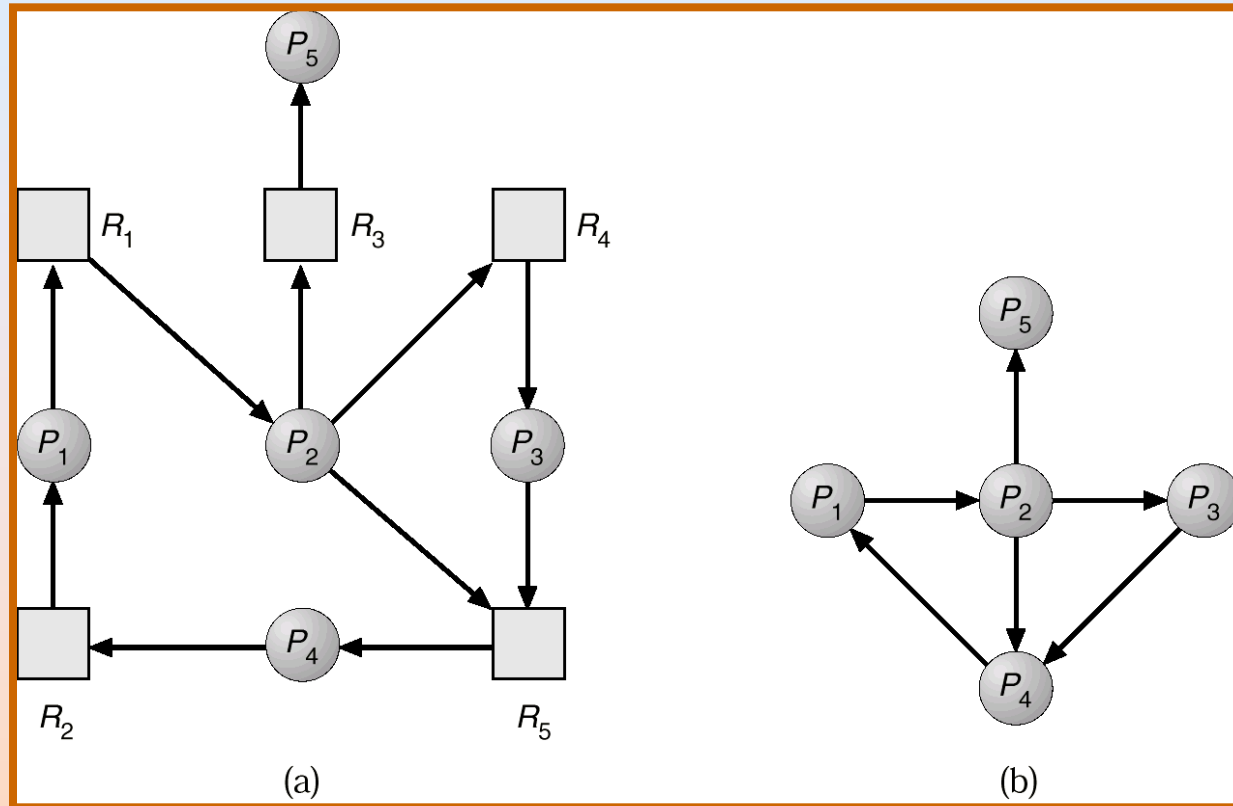
Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

Single Instance of Each Resource Type

- Maintain *wait-for* graph
 - Nodes are processes.
 - $P_i \rightarrow P_j$ if P_i is waiting for P_j .
- Periodically invoke an algorithm that searches for a cycle in the graph.
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.

Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph

Corresponding wait-for graph

Several Instances of a Resource Type

- *Available*: A vector of length m indicates the number of available resources of each type.
- *Allocation*: An $n \times m$ matrix defines the number of resources of each type currently allocated to each process.
- *Request*: An $n \times m$ matrix indicates the current request of each process. If $Request[i_j] = k$, then process P_i is requesting k more instances of resource type R_j .

Detection Algorithm

1. Let *Work* and *Finish* be vectors of length m and n , respectively
Initialize:

(a) *Work* = *Available*

(b) For $i = 1, 2, \dots, n$, if $Allocation_i \neq 0$, then
Finish[i] = false; otherwise, *Finish*[i] = true.

2. Find an index i such that both:

(a) *Finish*[i] == false

(b) $Request_i \leq Work$

If no such i exists, go to step 4.

Detection Algorithm (Cont.)

3. $Work = Work + Allocation_i$
 $Finish[i] = true$
go to step 2.
4. If $Finish[i] == false$, for some i , $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if $Finish[i] == false$, then P_i is deadlocked.

Algorithm requires an order of $O(m \times n^2)$ operations to detect whether the system is in deadlocked state.

Example of Detection Algorithm

- Five processes P_0 through P_4 ; three resource types A (7 instances), B (2 instances), and C (6 instances).
- Snapshot at time T_0 :

| | <u>Allocation</u> | | | <u>Request</u> | | | <u>Available</u> |
|-------|-------------------|---|---|----------------|---|---|------------------|
| | A | B | C | A | B | C | A B C |
| P_0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 0 0 |
| P_1 | 2 | 0 | 0 | 2 | 0 | 2 | |
| P_2 | 3 | 0 | 3 | 0 | 0 | 0 | |
| P_3 | 2 | 1 | 1 | 1 | 0 | 0 | |
| P_4 | 0 | 0 | 2 | 0 | 0 | 2 | |

- Sequence $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ will result in $Finish[i] = \text{true}$ for all i .

Example (Cont.)

- P_2 requests an additional instance of type C.

| | <u>Request</u> | | |
|-------|----------------|---|---|
| | A | B | C |
| P_0 | 0 | 0 | 0 |
| P_1 | 2 | 0 | 1 |
| P_2 | 0 | 0 | 1 |
| P_3 | 1 | 0 | 0 |
| P_4 | 0 | 0 | 2 |

- State of system?
 - Can reclaim resources held by process P_0 , but insufficient resources to fulfill other processes; requests.
 - Deadlock exists, consisting of processes P_1 , P_2 , P_3 , and P_4 .

Detection-Algorithm Usage

- When, and how often, to invoke depends on:
 - How often a deadlock is likely to occur?
 - How many processes will need to be rolled back?
 - ▶ one for each disjoint cycle
- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes “caused” the deadlock.

Recovery from Deadlock: Process Termination

- Abort all deadlocked processes.
- Abort one process at a time until the deadlock cycle is eliminated.
- In which order should we choose to abort?
 - Priority of the process.
 - How long process has computed, and how much longer to completion.
 - Resources the process has used.
 - Resources process needs to complete.
 - How many processes will need to be terminated.
 - Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

- Selecting a victim – minimize cost.
- Rollback – return to some safe state, restart process for that state.
- Starvation – same process may always be picked as victim, include number of rollback in cost factor.

Combined Approach to Deadlock Handling

- Combine the three basic approaches

- prevention
- avoidance
- detection

allowing the use of the optimal approach for each of resources in the system.

- Partition resources into hierarchically ordered classes.

- Use most appropriate technique for handling deadlocks within each class.

Traffic Deadlock for Exercise 8.4

