PROCESSES DEADLOCK (OR DEADLY EMBRACE)

- Molte ***risorse*** dei sistemi di calcolo possono essere ***usate in modo esclusivo***.
- I sistemi operativi devono ***assicurare l'uso consistente di tali risorse***: le risorse vengono allocate ai processi in modo esclusivo, per un certo periodo di tempo. Gli altri richiedenti vengono messi in attesa.
- Ma un processo può ***avere bisogno di molte risorse contemporaneamente***.
- Questo può portare ad ***attese circolari → il deadlock*** (stallo).
- Situazioni di stallo si possono verificare su risorse sia locali sia distribuite, sia software che hardware.
- È necessario avere dei metodi per ***prevenire***, ***riconoscere o almeno risolvere i deadlock***.

RISORSE E DEADLOCK

Una ***risorsa*** è una componente del sistema di calcolo a cui i processi possono accedere in modo esclusivo, per un certo periodo di tempo.

Risorse prerilasciabili: possono essere tolte al processo allocante, senza effetti dannosi.
Esempio: memoria centrale.

Risorse non prerilasciabili: non possono essere cedute dal processo allocante, pena il fallimento dell'esecuzione.
Esempio: stampante.

I deadlock si hanno con le risorse non prerilasciabili.

ALLOCAZIONE DELLE RISORSE

Allocazione di una risorsa

➔ Si può disciplinare l'allocazione mediante dei semafori, uno per ogni risorsa

Allocazione di più risorse

➔ come allocarle?

Non è detto che i due o più programmi che intendono allocare la stessa risorsa siano scritti dallo stesso utente: come coordinarsi?

Con decine, centinaia di risorse (come quelle che deve gestire il kernel stesso), come determinare se una sequenza di allocazioni è sicura?

Sono necessari dei metodi per:

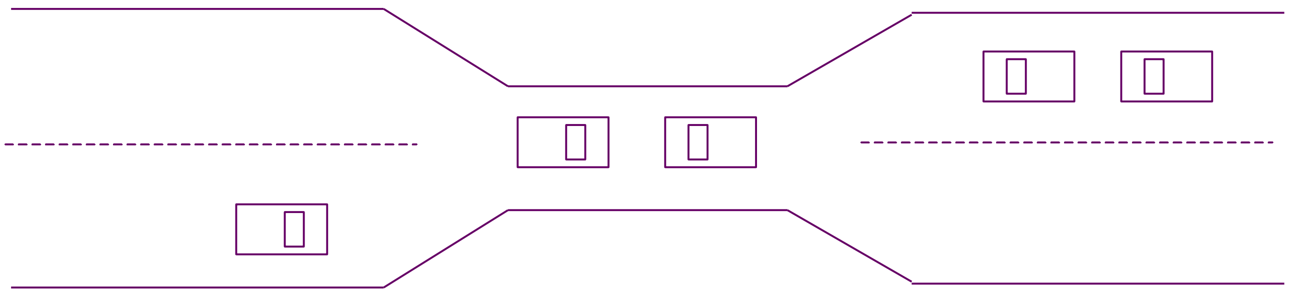
- riconoscere la possibilità di deadlock (prevenzione),
- riconoscere un deadlock,
- risolvere una situazione di deadlock.

PROCESSES DEADLOCK (OR DEADLY EMBRACE)

- ***DEADLOCK DEFINITION***: two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes.
  - Tape drives example
    - System has 2 tape drives.
    - P_1 and P_2 each hold one tape drive and each needs another one.
    - semaphores *A* and *B*, initialized to 1

P_0	P_1
wait (A);	*wait(B)*
wait (B);	*wait(A)*

  - **Bridge Crossing Example**



- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.

DEADLOCK SYSTEM MODEL

- Resource types $R_1, R_2, \ldots, R_m$

  CPU cycles, memory space, I/O devices

- Each resource type R_i has W_i instances.

- Each process utilizes a resource as follows:
  - request
  - use
  - release

- ***Deadlock can arise if four conditions hold simultaneously*** (Coffman theorem, 1971)
  - **Mutual exclusion:** only one process at a time can use a resource.
  - **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
  - **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.
  - **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

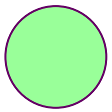
MODEL REPRESENTATION

Resource-Allocation Graph

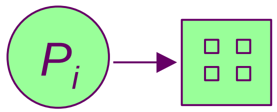
A set of vertices V and a set of edges E.

- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system.
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system.
- request edge: directed edge $P_1 \rightarrow R_j$
- assignment edge: directed edge $R_j \rightarrow P_i$

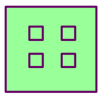
Process



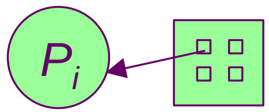
P_i requests instance of R_j



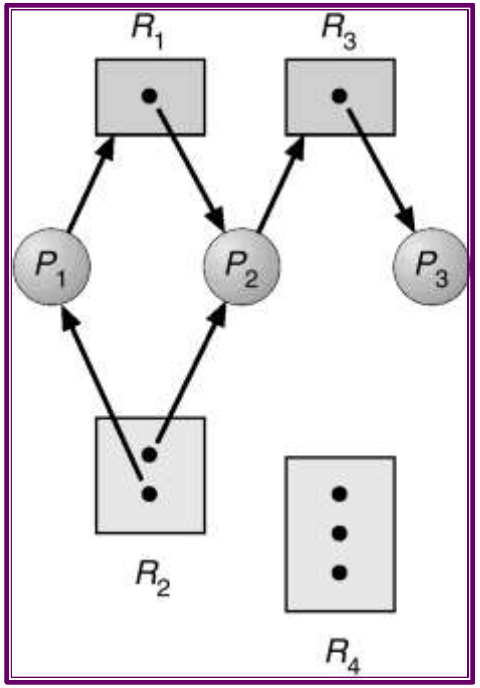
Resource Type with 4 instances

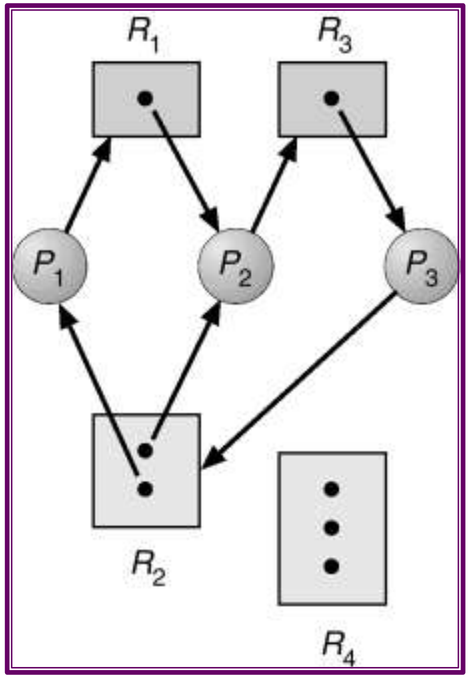


P_i is holding an instance of R_j



Resource-Allocation Graph examples

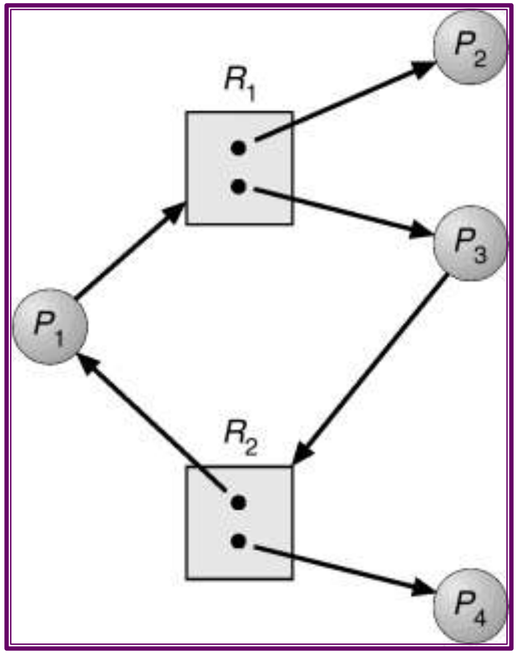




Detecting a deadlock in a Resource-Allocation Graph

☞ If graph contains no cycles ⇒ no deadlock.

☞ If graph contains a cycle ⇒
- **if only one instance per resource type, then deadlock.**
- if several instances per resource type, possibility of deadlock.



METHODS FOR HANDLING DEADLOCKS

- Ensure that the system will *never* enter a deadlock state
  - **Prevention**
  - **Avoidance**

- Allow the system to enter a deadlock state, detect the deadlock with suitable identification algorithms and then recover
  - **Detect and Recover**

- **Ignore the problem** and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.
  Assicurare l'assenza di deadlock impone costi (in prestazioni, funzionalità) molto alti.
  Tali costi sono necessari in alcuni contesti, ma insopportabili in altri.
  Si considera il rapporto costo/benefici: se la probabilità che accada un deadlock è sufficientemente bassa, non giustifica il costo per evitarlo.

DEADLOCK PREVENTION

Resources must be claimed *a priori* in the system (resources pre-allocation)

or

Restrain the ways request can be made (Coffman condition denial).

Mutual Exclusion – not required for sharable resources; must hold for nonsharable resources. Regola di buona programmazione: allocare le risorse per il minor tempo possibile.

- **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources.
  - Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none.
  - Low resource utilization; starvation possible.
  - Negare la mancanza di prerilascio: impraticabile per molte risorse.

- **No Preemption** –
  - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
  - Preempted resources are added to the list of resources for which the process is waiting.
  - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

- **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration. Teoricamente fattibile, ma difficile da implementare

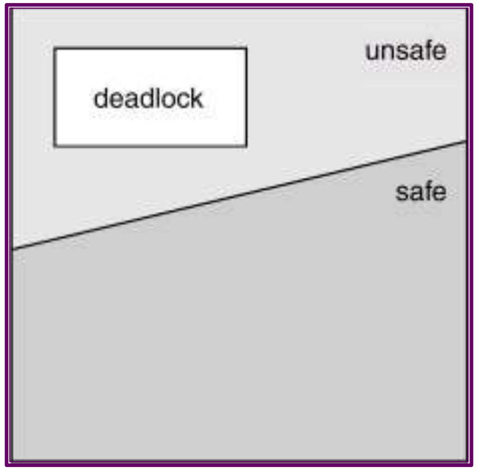
DEADLOCK AVOIDANCE

- Requires that the system has some additional *a priori* information available.
- Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need.
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
- Resource-allocation ***state*** is defined by the number of available and allocated resources, and the maximum demands of the processes.

SAFE STATE – SAFE SEQUENCE

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a *safe state*.
- System is in safe state if there exists a safe sequence of all processes.
- Sequence $<P_1, P_2, \ldots, P_n>$ is safe if for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j \neq i$.
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.
  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
  - When P_i terminates, P_{i+1} can obtain its needed resources, and so on.
- If a system is in safe state $\Rightarrow$ no deadlocks.
- If a system is in unsafe state $\Rightarrow$ possibility of deadlock.

Avoidance $\Rightarrow$ ensure that a system will never enter an unsafe state.



BANKER'S ALGORITHM (HABERMANN THEOREM)

- 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.

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 into the system.

Max: n x m matrix. If $Max\,[i,j] = k$, then process P_i may request at most k instances of resource type R_j.

Allocation: n x m matrix. If Allocation$[i,j] = k$ then P_i is currently allocated k instances of R_j.

Need: n x m matrix. If $Need[i,j] = k$, then P_i may need k more instances of R_j to complete its task.

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

THE BANKER'S ALGORITHM

Assumptions

$Request_i$ = request vector for process P_i.

$Request_i[j] = k$ means the process P_i wants k instances of resource type R_j.

1. If $Request_i \leq Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $Request_i \leq Available$, 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 state* $\Rightarrow$ *the resources are allocated to* P_i.

☞ *If unsafe state* $\Rightarrow$ P_i *must wait, and the old resource-allocation state is restored*

THE BANKER'S ALGORITHM

Safety algorithm

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively Initialize:
   (a) *Work* = *Available*
   (b) For $i = 1,2, \ldots, n$,
   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.
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.

THE BANKER'S ALGORITHM

Example

5 processes P_1 through P_5; 3 resource types:
A (10 instances), B (5 instances), and C (7 instances).
Snapshot at time T_0:

	Allocation	Max	Available
	A B C	A B C	A B C
P_1	0 1 0	7 5 3	3 3 2
P_2	2 0 0	3 2 2	
P_3	3 0 2	9 0 2	
P_4	2 1 1	2 2 2	
P_5	0 0 2	4 3 3	

	Need
	A B C
P_1	7 4 3
P_2	1 2 2
P_3	6 0 0
P_4	0 1 1
P_5	4 3 1

The system is in a safe state since the sequence $< P_2, P_4, P_5, P_3, P_1 >$ satisfies safety criteria.

THE BANKER'S ALGORITHM

Example

P_2 makes a Request$_1$ = (1,0,2) $\leq$ Available = (3,3,2) $\Rightarrow$ *true*.

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_1	0 1 0	7 4 3	2 3 0
P_2	3 0 2	0 2 0	
P_3	3 0 2	6 0 0	
P_4	2 1 1	0 1 1	
P_5	0 0 2	4 3 1	

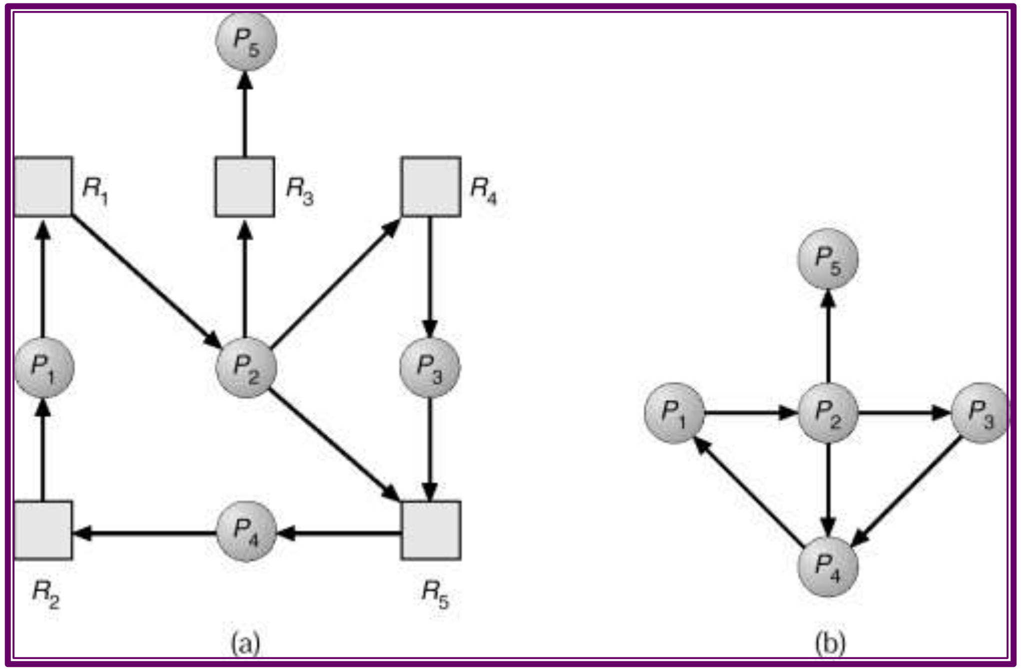
Executing safety algorithm shows that sequence $<P_2, P_4, P_5, P_3, P_1>$ satisfies safety requirement.

Can request for (3,3,0) by P_5 be granted?

Can request for (0,2,0) by P1 be granted?

DEADLOCK DETECTION

- Allow system to enter deadlock state
- Detection algorithm
  - 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.
- Recovery scheme
  - **Process termination**
  - **Resource Preemption**



DETECTION-ALGORITHM USAGE

If a system does not use either a deadlock prevention or a deadlock avoidance approach, it

↳ can provide a detection algorithm

↳ must anyway provide a deadlock recovery algorithm.

Detection algorithm

When and how often to invoke the detection algorithm depends on:

Φ How often a deadlock is likely to occur?

Φ How many processes will need to be rolled back?

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?

Resource Preemption

- Selecting a victim – minimize cost.

- Rollback – return to some safe state, restart process for that state.