Capitolo 8: Deadlock

- Il modello delle risorse di sistema e dei processi.
- Caratterizzazione del deadlock.
- Metodi di gestione dei deadlock.
- Prevenire il deadlock.
- Evitare il deadlock.
- Rilevazione del deadlock.
- Ripristino da un deadlock.

Il problema del deadlock

- I processi che sono in attesa possono permanere indefinitamente in tale stato se le risorse che hanno richiesto sono in possesso di altri processi a loro volta in attesa.
- Esempio:
  - Il sistema ha 2 nastri.
  - P_1 e P_2 utilizzano ciascuno un nastro e ciascuno di loro ne richiede un altro.
- Esempio:
  - semafori *A* e *B*, inizializzati a 1.

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

Risorse di sistema e processi

- Tipi di risorse R_1, R_2, . . ., R_m
  - *cicli di CPU, spazio di memoria, periferiche di I/O.*
- Ogni risorsa di tipo R_i (i=1..m) può avere più istanze identiche W_{ij} (j=1..p) (esemplari).
- Se un processo richiede una risorsa l'allocazione di un qualunque esemplare di essa soddisferà la richiesta a meno di errori nella definizione delle classi relative al tipo di risorsa.
- Ogni processo utilizza una risorsa come segue:
  - Richiesta (se essa non può essere soddisfatta immediatamente il processo deve attendere fino all'acquisizione);
  - Uso (il processo può adoperare la risorsa ed operare su di essa);
  - Rilascio (il processo libera la risorsa).
- Richiesta e rilascio di una risorsa sono chiamate di sistema se la risorsa è controllata dall'O.S. (p. es *open* e *close* su un file), in caso contrario esse possono essere implementate con una coppia acquire()/release() di un semaforo.

Risorse di sistema e processi

- I processi indirizzano al sistema operativo le richieste per le risorse festite da esso.
- Una tabella del sistema operativo registra lo stato (libero/allocato) di ogni risorsa e per ogni risorsa allocata traccia il processo che la impegna.
  - Se un processo richiede una risorsa occupata viene accodato insieme agli processi che ne attendono il rilascio.
- Un gruppo di processi è in uno stato di deadlock quando ciascun processo del gruppo è in attesa di un evento che può essere generato solo da un altro processo del raggruppamento.
- Eventi che generano stalli:
  - Acquisizione/rilascio di risorse (fisiche e logiche [file, semafori, monitor]).
  - IPC

Caratterizzazione del deadlock

Una condizione di deadlock si può verificare se si presentano simultaneamente le quattro condizioni seguenti:

- **Mutua esclusione:** soltanto un processo alla volta può usare la risorsa (almeno una risorsa deve essere usata in modo non condivisibile).
- **Possesso e attesa:** un processo che detiene almeno una risorsa deve attendere per acquisire ulteriori risorse utili alla sua computazione ed allocate ad altri processi.
- **Assenza di preemption:** una risorsa può essere liberata soltanto volontariamente dal processo che la detiene, dopo che ha completato le operazioni su di essa.
- **Attesa circolare:** esiste un gruppo di processi $\{P_0, P_1, \ldots, P_n\}$ in attesa di risorse, tali che P_0 attende una risorsa detenuta da P_1, P_1 ne attende una detenuta da $P_2, \ldots, P_{n-1}$ ne attende una detenuta da P_n, e P_n attende una risorsa detenuta da P_0.

NOTE:
1. Tutte le condizioni devono verificarsi per implicare un deadlock
2. La condizione 4 implica la 2, quindi le condizioni non sono indipendenti.

Grafo di allocazione delle risorse

I deadlock possono essere descritti con l'ausilio di una opportuna rappresentazione grafica costituita da:

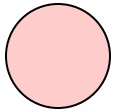
• un insieme di nodi *V*

• un insieme di archi *E*.

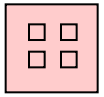
- V è diviso in due tipi di nodi:
  - $P = \{P_1, P_2, \ldots, P_n\}$, insieme di tutti i processi del sistema.
  - $R = \{R_1, R_2, \ldots, R_m\}$, insieme di tutti i tipi di risorse del sistema.
- **arco di richiesta** – arco orientato $P_1 \rightarrow R_j$
- **arco di assegnazione** – arco orientato $R_j \rightarrow P_i$

Grafo di allocazione delle risorse

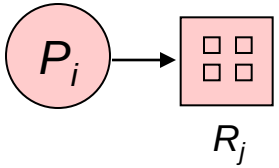
- Processo:



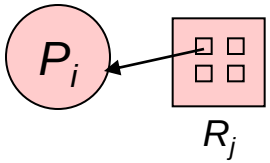
- Tipo di risorsa con 4 istanze (esemplari):



- P_i *ri*chiede un'istanza di risorsa di tipo R_j:

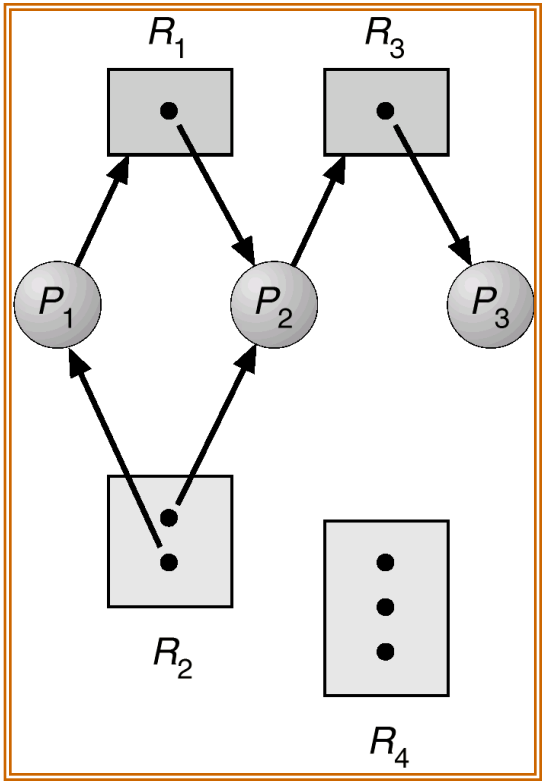


- P_i *detiene* un'istanza di risorsa di tipo R_j:

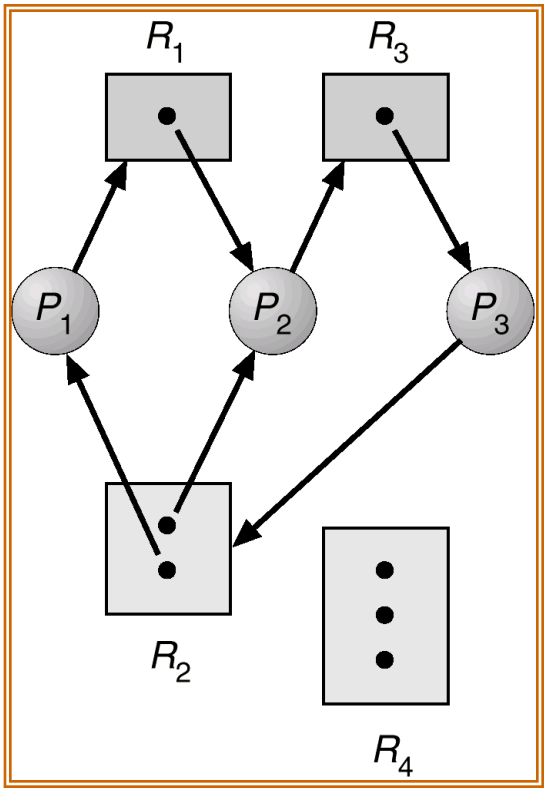


Grafo di allocazione e deadlock

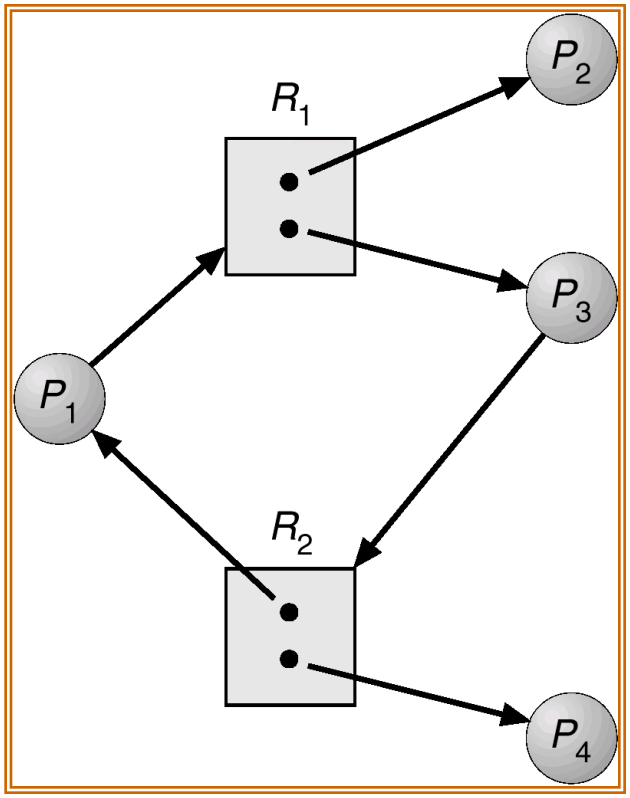
Grafo di allocazione delle risorse



Presenza di cicli con deadlock



Presenza di cicli senza deadlock



- Se il grafo non contiene cicli ⇒ nessun deadlock.
- Se il grafo contiene cicli ⇒
  - se ogni tipo di risorsa ha un'unica istanza, allora si verifica un deadlock.
  - se ogni tipo di risorsa ha parecchie istanze, possibilità di deadlock.

Metodi di gestione dei deadlock

- DEADLOCK PREVENTION
  - Accertarsi che il sistema non entri mai in uno stato di deadlock adoperando strategie di natura preventiva.
- DEADLOCK AVOIDANCE
  - Accertarsi che il sistema non entri mai in uno stato di deadlock mediante uno schema di gestione conservativa deli accessi dei processi alle risorse.
- DETECT AND RECOVER
  - Permettere che il sistema entri in uno stato di stallo, ma lo rilevi e recuperi uno stato corretto.
- IGNORE
  - Ignorare del tutto il problema secondo l'assunzione fondamentale che i deadlock non si presentino mai (o raramente) nel sistema; frequente in molti sistemi operativi, compreso UNIX.

Prevenire il deadlock (1/2)

Accertandosi che almeno una delle condizioni necessarie non possa essere soddifatta è possibile prevenire il nascere di un deadlock.

- **Mutua esclusione** – non necessaria per le risorse condivisibili; deve essere soddifatta per le risorse non condivisibili.
  - In generale non si può impedire un deadlock negando la mutua esclusione dato che alcune risorse sono intrinsecamente non condivisibili.
- **Possesso ed attesa** – bisogna garantire che ogni volta che un processo chiede una risorsa, non detenga già qualche altra risorsa.
  - Implica che ogni processo chieda e ottenga in assegnazione tutte le risorse prima di iniziare l'esecuzione o permette che un processo richieda risorse soltanto quando non ne detiene altre.
  - Basso utilizzo delle risorse; possibile starvation.

Prevenire il deadlock (2/2)

- **Assenza di preemption**
  - Si applica un protocollo di controllo sull'assegnazione di risorse (nel caso in cui il relativo stato possa essere facilmente salvato e ripristinato):
    - Se un processo sta detenendo alcune risorse e ne richiede un'altra che non può essergli assegnata immediatamente, allora tutte le risorse occupate dovranno essere rilasciate.
    - Le risorse rilasciate vengono aggiunte alla lista delle risorse per cui il processo è in attesa.
    - Il processo sarà fatto ripartire soltanto quando può riguadagnare sia risorse precedentemente possedute che quelle che sta richiedendo.
  - Alternativa:
    - Se un processo richiede risorse disponibili, queste gli vengono assegnate.
    - Se non lo sono, si controlla che non siano assegnate a processi in attesa di ulteriori risorse, nel qual caso su di esse verrà applicata preemption.
    - Se le risorse richieste non sono né disponibili né assegnate a processi in attesa, il processo richiedente dovrà attendere.
- **Attesa circolare**
  - Si impone un ordinamento globale di tutti i tipi di risorsa. Se $R = \{R_1, R_2, \ldots, R_m\}$, si definisce $F: R \rightarrow N$ in base alla normale sequenza d'uso di risorse in un sistema. Un processo può alternativamente:
    1. Richiesta un'istanza di R_m , richiedere esemplari di R_n tali che $F(R_n) > F(R_m)$
    2. Richiesta un'istanza di R_j , liberare tutte le istanze di R_i tali che $F(R_i) \geq F(R_j)$
  - Per assurdo si dimostra che questo metodo previene la circular wait.
  - Un *witness* è un software che verifica l'acquisizione delle risorse nel giusto ordine.

Evitare il deadlock

Esigere informazioni aggiuntive *a priori* su come vengono richieste le risorse.

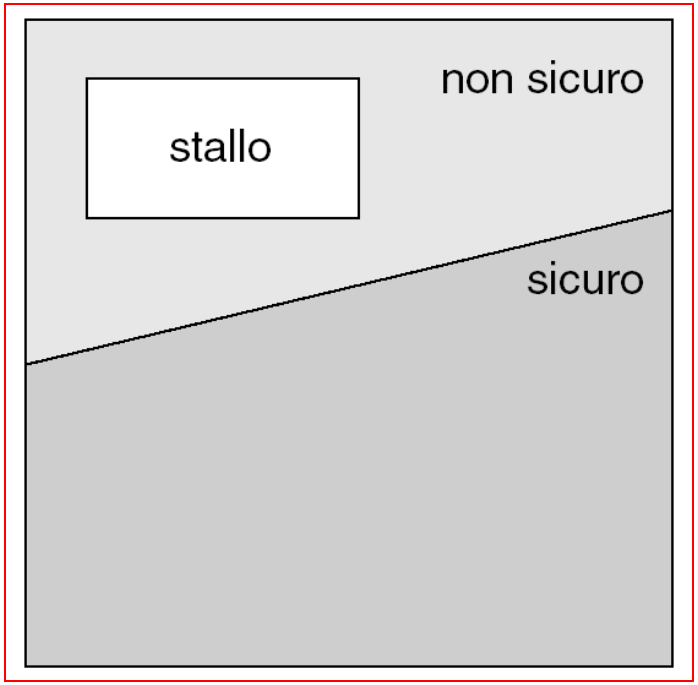
- Mediante la conoscenza totale della sequenza completa delle richieste e dei rilasci per ciascun processo, è possibile stabilire se:
  - in corrispondenza di ogni richiesta un processo deve attendere o meno per evitare un presumibile deadlock.
- Nel decidere sull'assegnazione di una risorsa il sistema tiene conto di:
  - Risorse disponibili all'istante t_0 della richiesta
  - Risorse assegnate a ciascun processo all'istante t_0 della richiesta
  - Richieste e rilasci per $t > t_0$ per ciascun processo
- Il modo più semplice richiede che ogni processo dichiari il *numero massimo* di risorse di ogni tipo di cui può avere bisogno.
- Un algoritmo per evitare i deadlock esamina dinamicamente lo stato di allocazione delle risorse per accertarsi che la condizione di attesa circolare non possa mai verififcarsi.
- Lo stato di allocazione delle risorse è definito dal numero di risorse disponibili e assegnate e dal numero massimo di richieste dei processi.

Stati sicuri

- Un sistema è in uno stato sicuro soltanto se esiste una **sequenza di completamento sicura**.
  - Il sistema può stanziare le risorse per ogni processo della sequenza in un ordine preciso e continuare ad evitare un deadlock.
- La sequenza $<P_1, P_2, \ldots, P_n>$ è sicura se per ogni P_i, le richieste di risorse che P_i può attualmente fare possono essere soddifatte da:
  - risorse attualmente disponibili;
  - risorse detenute da tutti i processi P_j, con $j < i$.
- Se le risorse di cui il processo P_i ha bisogno non sono immediatamente disponibili, allora P_i può aspettare finchè tutti i P_j hanno terminato.
  - Quando hanno finito, P_i può ottenere tutte le risorse necessarie, completare il suo task, restituire le risorse assegnate e terminare.
  - Quando P_i termina, P_{i+1} può ottenere le risorse necessarie, e così via.

Punti chiave

- Se il sistema è in uno stato sicuro ⇒ no deadlock.
- Se il sistema è in uno stato non sicuro ⇒ possibilità di deadlock.
- Per evitare il deadlock occorre assicurarsi che il sistema non entri mai in uno stato non sicuro.
- Si possono definire algoritmi per il deadlock avoidance mediante i quali si verifica che il sistema permanga sempre in uno stato sicuro.



Spazi di stato sicuro, non sicuro e di deadlock

Algoritmo del banchiere

Teorema di Habermann

- Ipotesi di base:
  - Istanze multiple delle risorse.
  - Ogni processo deve dichiarare il numero massimo di istanze per ogni tipo di risorsa di cui può avere bisogno.
  - La richiesta massima di risorse non può eccedere la disponibilità totale del sistema.
  - Quando un processo ottiene tutte le sue risorse deve restituirle in un periodo di tempo finito.
- Quando un processo richiede delle risorse, mediante l'algoritmo si determina se l'assegnazione lascia il sistema in uno stato sicuro.
  - Se si, le risorse vengono allocate.
  - Se no, il processo deve attendere per la de-allocazione di risorse da parte di altri processi.

Strutture dati

Supponiamo: n = numero dei processi, e m = numero di tipi di risorse.

- **Available**: vettore di lunghezza m. Se Available [j] = k, ci sono k istanze di risorse di tipo R_j disponibili.
- **Max**: matrice n x m . Se Max [i,j] = k, allora il processo P_i può chiedere al più k istanze di risorse di tipo R_j.
- **Allocation**: matrice n x m. Se Allocation[i,j] = k allora al processo P_i sono attualmente assegnate k istanze di risorsa del tipo R_j.
- **Need**: matrice n x m. Se $Need$[i,j] = k, allora il processo P_i può avere bisogno di altre k istanze di risorse del tipo R_j per completare il suo compito.

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

Algoritmo di sicurezza

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

   Work = Available

   Finish [i] = false for *i* = 1, 2, …, *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. Let:

   a) *Work = Work + Allocation [i,·]*

   b) *Finish [i] = true*

   go to step 2.

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

- **Può richiedere m x n^2 iterazioni per stabilire se uno stato è sicuro.**

Algoritmo di resource request

Let **Request$_i$** be the request vector for process P_i. If Request$_i$ [j] = k then process P_i wants k instances of resource type R_j.

When P_i made a request for resources, the following actions are taken:

1. If ***Request$_i$ ≤ Need [i,·]*** go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If ***Request$_i$ ≤ 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:

$$\textbf{\textit{Available = Available - Request}}_i$$

$$\textbf{\textit{Allocation [i,·] = Allocation [i,·] + Request}}_i$$

$$\textbf{\textit{Need [i,·] = Need [i,·] - Request}}_i$$

- If safe ⇒ the resources are allocated to P_i
- If unsafe ⇒ P_i wait (the old **resource-allocation state** is restored)

Esempio (1/3)

- Si consideri un sistema con 5 processi da P_1 a P_5 e 3 tipi di risorse:
  - *A* – 10 istanze,
  - *B* – 5 istanze,
  - *C* – 7 istanze.
- Supponiamo che all'istante T_0 ci sia la seguente situazione del sistema:

	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	

Esempio (2/3)

- Il contenuto della matrice *Need* è definito come:

  Need [,] = *Max [,]* – *Allocation [,]* , ossia:

	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

- Il sistema è in uno stato sicuro poichè la sequenza < P_4, P_2, P_5, P_3, P_1> soddisfa i criteri di sicurezza.

Esempio (3/3)

$Request_2$ = [1,0,2]

- $Request_2 \leq Need\ [2,\cdot\ ]$? $[1,0,2] \leq [1,2,2] \Rightarrow$ *true.*
- $Request_2 \leq Available$? $[1,0,2] \leq [3,3,2] \Rightarrow$ *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	

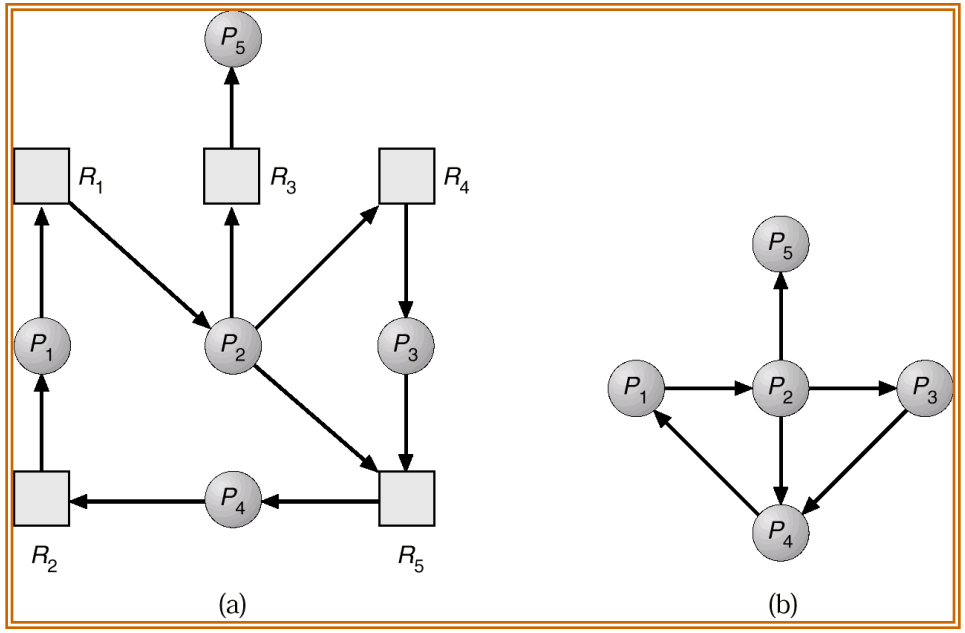
- Executing safety algorithm shows that sequence $<P_2, P_4, P_5, P_1, P_3>$ satisfies safety requirement.
- Can request for [3,3,0] by P_5 be granted? And request for [0,2,0] by P_1?

Rilevazione del deadlock

- Permettere al sistema di entrare in uno stato di deadlock.
- Algoritmo di rilevazione
- Schema di recupero

Risorse a singola istanza

- Mantiene un *grafo di attesa* (wait-for graph).
  - I nodi delle risorse sono rimossi.
  - $P_i \rightarrow P_j$ se P_i sta attendendo P_j.
- Il sistema deve invocare periodicamente un algoritmo che cerca un ciclo nel grafo.
- Un algoritmo per rilevare un ciclo in un grafo richiede n^2 operazioni, dove n è il numero di nodi del grafo.



Grafo di allocazione delle risorse

Grafo di attesa

Risorse ad istanze multiple

- Il wait-for graph non è applicabile nel caso di risorse ad istanze multiple.
- Un comune algoritmo per la rilevazione di un deadlock adopera strutture dati variabili nel tempo simili a quelle dell'algoritmo del banchiere:
  - **Available**: vettore di lunghezza *m,* indica il numero di risorse disponibili per ogni tipo.
  - **Allocation**: matrice *n* x *m,* definisce il numero di risorse di ciascun tipo attualmente assegnate a ogni processo.
  - **Request**: matrice *n* x *m*, indica la richiesta corrente di ogni processo. Se *Request* [*i, j*] = *k*, allora il processo P_i richiede altre *k* istanze della risorsa di tipo R_j.
- L'algoritmo di rilevazione controlla ogni possibile *sequenza di allocazione* per i processi da completare.

Algoritmo di rilevazione

1. Supponiamo che *Work* e *Finish* siano rispettivamente vettori di lunghezza *m* e *n*. Inizializziamo:
   (a) *Work = Available*
   (b) *Per i = 1,2, …, n, se Allocation [i,·] ≠ 0, allora Finish[i] = false; altrimenti, Finish[i] = true.*
2. Si cerca un indice *i* che soddisfi entrambe le condizioni seguenti :
   (a) *Finish*[*i*] = *false*
   (b) $Request_i \leq Work$

   Se una tale *i* non esiste, passare al punto 4.
3. *Work = Work + Allocation [i,·]*
   Finish[*i*] = *true*
   passare al punto 2.
4. Se *Finish*[*i*] = false per qualche *i*, $1 \leq i \leq n$, allora il sistema è in uno stato di deadlock e particolarmente il processo P_i è in stallo.

L'algoritmo richiede *m* x n^2 operazioni per rilevare se il sistema è in un deadlock.

Esempio

- Consideriamo un sistema con 5 processi da P_1 a P_5 e 3 tipi di risorse:
  - A (7 istanze),
  - *B* (2 istanze),
  - *C* (6 istanze).
- Supopniamo che all'istante T_0 si abbia il seguente stato di allocazione delle risorse:

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_1	0 1 0	0 0 0	0 0 0
P_2	2 0 0	2 0 2	
P_3	3 0 3	0 0 0	
P_4	2 1 1	1 0 0	
P_5	0 0 2	0 0 2	

- La sequenza <P_1, P_3, P_4, P_2, P_5> provoca *Finish*[*i*] = true per ogni *i*.

Esempio

- Supponiamo che P_3 faccia una richiesta supplementare per un'istanza di tipo C:

	Request		
	A	*B*	*C*
P_1	0	0	0
P_2	2	0	1
P_3	0	0	1
P_4	1	0	0
P_5	0	0	2

- Stato del sistema?
  - Anche se possiamo riprendere le risorse del processo P_1, il numero di risorse disponibili non è sufficiente per soddisfare le richieste degli altri processi.
  - C'è un deadlock costituito dai processi P_2, P_3, P_4, e P_5.

Uso dell'algoritmo di rilevazione

- Quando e con quale frequenza invocare l'algoritmo di rilevazione dipende da:
  - Con quale frequenza può accadere un deadlock?
  - Quanti processi saranno coinvolti nel deadlock quando questo si verificherà?
- Se il sistema è soggetto frequentemente a deadlock, l'algoritmo di rilevazione dovrebbe essere invocato spesso. Si ricordi che:
  - Le risorse assegnate a processi in deadlock saranno bloccate fino alla rimozione dello stallo.
  - Il numero di processi coinvolti in un deadlock può crescere nel tempo.
- I deadlock avvengono solo quando una richiesta di risorse non può essere soddisfatta immediatamente.
  - Al limite si potrebbe richiamare l'algoritmo ogni qual volta ci si trova in questa situazione identificando così anche il processo causa del deadlock.
  - Ne derivano overhead e ritardi sensibili.
- Se l'algoritmo di rilevazione è invocato in momenti casuali possono essere trovati molti cicli nel grafo di risorsa.
  - Potremmo non essere in grado di dire quale dei molti processi in deadlock ne è la causa.

Ripristino dal deadlock (1/2)

- Il ripristino dal deadlock può avvenire:
  - a carico dell'utente che viene informato dal sistema dell'esistenza di uno stallo o per via automatica;
  - abortendo uno o più processi o requisendo risorse detenute da processi in deadlock.
- Abort di processi:
  - Abortire tutti i processi in deadlock (i risultati parziali dell'elaborazione di tutti i processi coinvolti devono essere scartati).
  - Abortire un processo alla volta fino ad eliminare il ciclo di deadlock (è notevole l'overhead poiché dopo ogni abort si deve lanciare l'algoritmo di rilevazione).
  - In quale ordine devono essere terminati i processi?
    - Priorità del processo.
    - Per quanto tempo il processo ha elaborato e per quanto tempo ancora il processo proseguirà prima di completare l'operazione pianificata.
    - Quanti e quali tipi di risorse sono state adoperate dal processo.
    - Di quante altre risorse il processo necessita per completare l'elaborazione.
    - Quanti processi devono essere terminati.
    - Caratteristica del processo (interattivo o batch).

Ripristino dal deadlock (2/2)

- Rilascio anticipato delle risorse.
  - Selezione della vittima (minimizzazione del costo).
    - I fattori di costo includono:
      - Numero di risorse occupate.
      - Per quanto tempo il processo ha elaborato e per quanto tempo ancora il processo proseguirà prima di completare l'operazione pianificata.
  - Rollback.
    - Prelazionando una risorsa da un processo occorre riportarlo ad uno stato sicuro e far ripartire il processo da quello stato.
    - La soluzione più semplice è un rollback totale del processo.
  - Starvation.
    - In un sistema in cui la selezione della vittima è basata soprattutto sui fattori costo, alcuni processi potrebbero essere sempre scelti come vittime.
    - Occorre verificare che un processo possa essere selezionato solo un numero limitato di volte.
      - Il numero di rollback viene incluso nei fattori costo.