c) Richiesta di risorse da parte del processo P_i

$Request_i$ = vettore di richieste correnti di P_i
$Request_i[j] = k$: P_i vuole k istanze della risorsa di tipo R_j

1. Se $Request_i \leq Need_i$ go to step 2.
   Altrimenti segnala l'errore che il processo ha richiesto piu' risorse del suo massimo
2. Se $Request_i \leq Available$ go to step 3.
   Altrimenti P_i deve attendere per risorse non disponibili
3. In pratica simuliamo l'allocazione delle risorse richieste da P_i modificando lo stato come segue:
   $Available := Available - Request_i$
   $Allocation_i := Allocation_i + Request_i$
   $Need_i := Need_i - Request_i$

- Se lo stato raggiunto e' safe $\Rightarrow$ le risorse vengono allocate a P_i
- Se unsafe $\Rightarrow$ P_i deve attendere, e il vecchio stato di allocazione viene ripristinato

Esempio

- 5 processi
  3 tipi di risorse: A (10 istanze), B (5 istanze), C (7 istanze)
- Fotografia del sistema al tempo T_0:

	Allocation	Max	Available
	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	

- Il contenuto della matrice *Need* e' definito come *Max – Allocation*

	Need
	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

- Il sistema e' in uno stato safe perche' esiste la sequenza safe $< P_1, P_3, P_4, P_2, P_0 >$

P_1 richiede (1,0,2)

- Request $\leq$ Available : $(1,0,2) \leq (3,3,2) \Rightarrow$ *true*.

	Allocation	Need	Available
	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	

- L'algoritmo di safety a questo punto mostra che $< P_1, P_3, P_4, P_0, P_2 >$ e' una sequenza safe

E se P_4 avesse richiesto (3,3,0) ?

E se P_0 avesse richiesto (0,2,0) ?

3. Rilevare – 4. Ripristinare

IDEA : Si permette al sistema di entrare in uno stato di deadlock

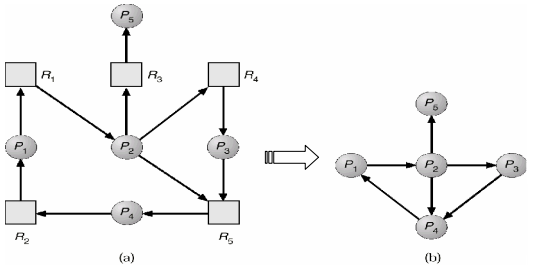
⇩

- Algoritmo di detection
- Schema di recovery

Risorse a istanza unica: semplificazione del grafo di allocazione risorse…

- Diventa un ***grafo di attesa***
  - i nodi sono solo processi
  - $P_i \rightarrow P_j$ se P_i ***sta attendendo*** qualche risorsa posseduta da P_j
- Una volta cancellate le risorse, si collassano gli archi
- Periodicamente si invoca un algoritmo che ***ricerca un ciclo nel grafo***

Dal grafo di allocazione al grafo di attesa



Grafo di allocazione risorse — Corrispondente grafo di attesa

Risorse a multiple istanze : un "banchiere on-line"

- *Available[m]*
  - *Available[j] = k*, ci sono *k* istanze di risorsa di tipo *Rj* disponibili
- *Allocation[n,m]*
  - *Allocation[i,j] = k*, il processo *Pi* possiede *k* istanze di *Rj*
- *Request[n,m]*
  - *Request [i,j] = k*, il processo *Pi* sta richiedendo *k* istanze di risorsa di tipo *Rj*

Algoritmo di detection…

IDEA DI BASE

Le risorse vengono ***allocate subito a processi sicuramente non coinvolti*** in un deadlock (Request$_i$ Work), supponendo che:

- non ne chiedano altre
- quelle che vengono allocate servano solo per completare il lavoro e quindi restituirle tutte

Se cosi' non e', si potra' verificare un deadlock che verra' rilevato nelle successive invocazioni dell'algoritmo

… step by step

1. *Work* e *Finish* come nel "banchiere"
   Work := *Available*
   For *i* = 1,2, …, *n*, if *Allocation*$_i \neq 0$, then
   Finish[i] := false ; else *Finish[i] := true*
2. Trova un processo *i* tale che
   Finish[i] = false
   Request$_i \leq$ *Work*
   Se tale processo non esiste *go to step 4*
3. *Work* := *Work* + *Allocation*$_i$
   Finish[i] := true
   go to step 2.
4. Se *Finish[i] = false*, per qualche *i*, $1 \leq i \leq n$, allora il sistema e' in stato di deadlock
   Inoltre, se *Finish[i] = false*, allora il processo P_i e' in deadlock

Si suppone che un processo termini dopo aver soddisfatto la richiesta corrente

Esempio

- 5 processi
  3 tipi di risorse: A (7 istanze), *B* (2 istanze), *C* (6 istanze)
- Fotografia del sistema al tempo T_0:

	Allocation	*Request*	*Available*
	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	

- La sequenza <P_0, P_2, P_3, P_1, P_4> risultera' in *Finish[i] = true* forall *i*

P_2 richiede una ulteriore istanza di tipo C

Qual e' lo stato del sistema ora?

	Request 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

Anche se si reclamassero le risorse possedute dal processo P_0 sarebbero comunque insufficienti per soddisfare le richieste degli altri processi

Esiste un deadlock tra i processi
P_1, P_2, P_3, P_4

Uso dell'algoritmo di detection

- ***Quando*** e ***quanto spesso invocare*** l'algoritmo dipende da:
  - ***Quanto spesso un deadlock*** puo' accadere?
  - ***Quanto pesante e' l'overhead*** della procedura di detection/recovery?
  - ***Quanti processi*** dovranno essere ***terminati per rompere il ciclo che determina il deadlock*** ?
    (e' una questione riguardante il recovery !)

Se l'algoritmo e' invocato in maniera arbitraria ci potrebbero essere troppi cicli nel grafo da non poter identificare un ***processo colpevole***

Schema di recovery 1: terminazione di processi

- Termina ***tutti i processi*** coinvolti nel deadlock
- Termina ***un processo alla volta*** finche' il ciclo di deadlock non si interrompe

Parametri da considerare per stabilire l'ordine di terminazione dei processi ***(selezione della vittima)***

- Priorita' del processo
- Quanto del processo e' stato eseguito, e quanto manca alla terminazione
- Risorse gia' utilizzate e risorse di cui ha bisogno per il completamento
- Processo interattivo o batch
- (Quanti processi devono essere terminati)

Schema di recovery 2: preemption delle risorse

- Di nuovo ***selezione di una vittima***: questa volta un processo e' vittima come conseguenza della scelta di una risorsa che gli viene tolta
- Il processo privato della risorsa deve tornare indietro nella computazione (***rollback*** fino ad uno ***stato safe*** oppure ***fino all'inizio*** ?!)
- ***Evitare starvation*** – un processo dovrebbe essere vittima solo un numero finito di volte (*anche nello schema di soluzione 1!*)
- Si puo' includere ***il numero di rollbacks nel fattore di costo*** calcolato nella selezione della vittima

Idea di un approccio combinato

- Combinare i 3 approcci di base
  - prevenire
  - evitare
  - rilevare e ripristinare

  permettendo ***l'uso dell'approccio ottimale per ogni tipo di risorsa nel sistema***
- Ordinare le classi di risorse e applicare la richiesta ordinata di risorse vista in precedenza
- All'interno poi di ogni classe di risorsa applicare l'approccio piu' appropriato