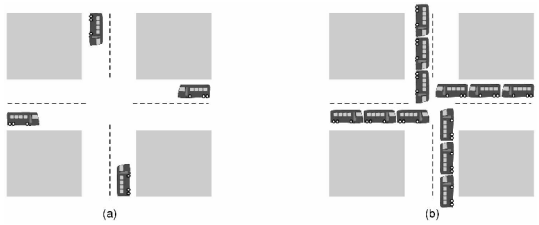
2. Evitare

Si richiede che il sistema abbia ***piu' informazioni*** disponibili ***a priori*** sulle richieste di risorse da parte dei processi

Sono necessari ***algoritmi piu' complessi*** per ***decidere se allocare*** o meno una risorsa

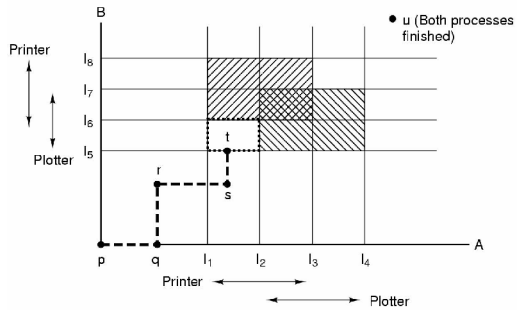
Si tratta di ***decisioni dinamiche*** piuttosto che regole a priori come nella prevenzione

Differenza tra situazione "pericolosa" (*unsafe*) e deadlock



situazione pericolosa ***deadlock***

… e pericoli tra due processi



Una soluzione semplice

- Ogni processo dichiara all'inizio il ***massimo numero di richieste*** di ogni tipo di risorsa di cui puo' aver bisogno
- ***Dinamicamente*** si esamina lo ***stato di allocazione risorse del sistema*** per assicurare che non ci sara' mai una condizione di attesa circolare

⇩

Lo ***stato*** e' definito da: numero di risorse ***disponibili***, numero di risorse ***allocate***, ***massima richiesta*** da parte di ogni processo

Definizione di stato *safe*: OS controlla la situazione…

- Il sistema e' in uno ***stato safe*** se ***esiste una sequenza di processi safe***
- La ***sequenza <P_1, P_2, …, P_n> si dice safe*** se per ogni P_i, le risorse che P_i ***puo' ancora richiedere*** possono essere allocate tra le risorse disponibili + le risorse possedute da tutti i P_j, con $j<i$

… e cioe':

• Se le risorse che *Pi* puo' ancora richiedere non sono immediatamente disponibili, allora *Pi* puo' attendere finche' tutti i *Pj* hanno finito

• Quando *Pj* ha finito, *Pi* puo' ottenere le risorse necessarie, eseguire, rilasciare le risorse allocate e terminare

• Quando *Pi* termina, *Pi+1* puo' ottenere le sue risorse, e cosi' via

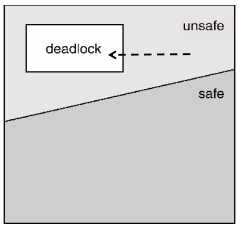
Considerazioni

Stato safe $\Rightarrow$ no deadlock
Stato unsafe $\Rightarrow$ possibilita' di deadlock

Evitare significa assicurare che il sistema ***non entri mai in uno stato unsafe***

E cioe' quando un processo richiede una risorsa non disponibile, ***il sistema deve decidere*** se ***l'immediata allocazione*** di quella risorsa lascia il sistema ***in uno stato safe***

… e graficamente



Stato unsafe : e' il comportamento dei processi che determina o meno il fatto che accada un deadlock

Stato safe : OS puo' tenere sotto controllo la situazione come dettto prima

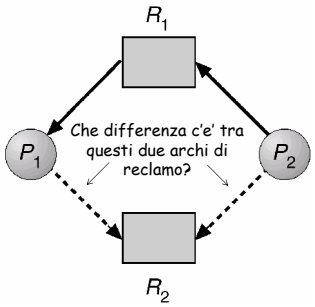
⇩

Quindi come stabilire se esiste una sequenza safe a valle dell'allocazione di una risorsa ?!

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

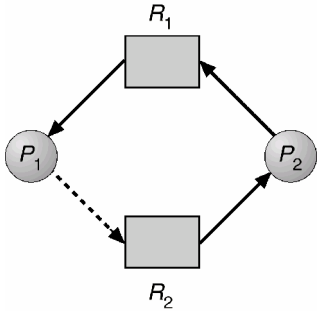
- Nuovo tipo di arco: ***Arco di reclamo*** $P_i \rightarrow R_j$ indica che il processo P_i ***puo' richiedere*** la risorsa R_j
- Un ***arco di reclamo*** diventa ***arco di richiesta*** quando il processo ***richiede*** effettivamente la risorsa
- Quando invece la ***risorsa*** viene ***rilasciata*** l'arco ***di assegnamento*** torna ad essere ***di reclamo***
- Le risorse devono essere ***reclamate a priori***

… ed un esempio



Quando il ***reclamo*** diventa ***richiesta***, prima di ***soddisfarla*** si verifica se genera un ciclo…

… come in questo caso: stato unsafe



Risorse a multiple istanze

Stessa idea di base del caso di singola istanza

- Ogni processo deve ***dichiarare a priori il massimo uso delle risorse*** (che naturalmente non deve essere superiore al numero totale di risorse del sistema!)
- Quando ***un processo richiede una risorsa*** potrebbe dover attendere anche se la risorsa fosse disponibile, in quanto ***OS deve stabilire se la sua immediata allocazione porta il sistema in uno stato unsafe***

Soluzione: Algoritmo del banchiere

a) Strutture dati principali

b) Verifica della safety

c) Richiesta di risorse

a) Strutture dati principali

n = numero di processi, *m* = numero di tipi di risorse

- *Available[m]* -
  - *Available[j] = k*, ci sono *k* istanze di risorsa di tipo R_j disponibili
- *Max[n,m]* -
  - *Max [i,j] = k*, il processo P_i puo' richiedere al piu' *k* istanze di risorsa di tipo R_j
- *Allocation[n,m]*
  - *Allocation[i,j] = k*, il processo P_i possiede *k* istanze di R_j.
- *Need[n,m]*
  - *Need[i,j] = k*, il processo P_i ***puo' avere bisogno*** ancora di *k* istanze di R_j per completare il suo compito

    Need [i,j] = Max[i,j] – Allocation [i,j]

b) Verifica della safety

1. *Work[m]* quantita' di risorse non allocate
   Finish[n] terminazione dei processi
   Work := *Available*
   Finish [i] = false forall *i* ..... *inizializzazioni*
2. Trova un processo *i* tale che:
   (a) *Finish [i] = false*
   (b) $Need_i \leq Work$ ..... *un passo della sequenza safe di processi*
   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] = true* forall *i*, allora il sistema e' in uno stato safe