

## **Lezione 3 – Tecniche per evitare il deadlock**

Sistemi Operativi I

Modulo 3 - Gestione del Processore

Unità didattica 6 - Deadlock

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

### **Sommario**

- Principio dell'evitare il deadlock (deadlock avoidance)
- Informazioni per evitare il deadlock
- Stato sicuro
- Algoritmo del grafo di allocazione delle risorse
- Algoritmo del banchiere

## Obiettivi

- Alto sfruttamento delle risorse
- Alta efficienza del sistema
- Semplicità di gestione

## Principio di evitare il deadlock

**Verificare *a priori***  
**se la sequenza di richieste e rilasci di risorse**  
**effettuate da un processo porta al deadlock**  
**tenendo conto delle sequenze dei processi**  
**già accettati nel sistema**

## Informazioni per evitare il deadlock

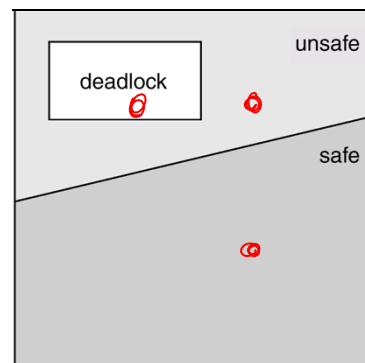
**Necessità di informazioni *a priori* sul comportamento dei processi:**

- numero massimo di risorse per ogni processo
- risorse assegnate
- risorse disponibili
- richieste e rilasci futuri di risorse

## Stato sicuro <sup>(1)</sup>

Uno stato si dice sicuro se il sistema può allocare le risorse richieste da ogni processo in un certo ordine garantendo che non si verifichi deadlock

- stato sicuro  
⇒ no deadlock
- stato non sicuro  
⇒ deadlock possibile



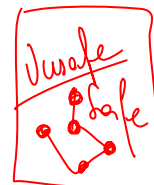
### Stato sicuro (2)

- Una sequenza di processi  $\langle P_1, P_2, P_3, \dots, P_n \rangle$  è una sequenza sicura per l'allocazione corrente se le richieste che ogni processo  $P_i$  può fare possono essere soddisfatte dalle risorse attualmente disponibili più tutte le risorse detenute dai processi  $P_j$  con  $j < i$
- Uno stato è sicuro se esiste una sequenza sicura

### Come evitare il deadlock?

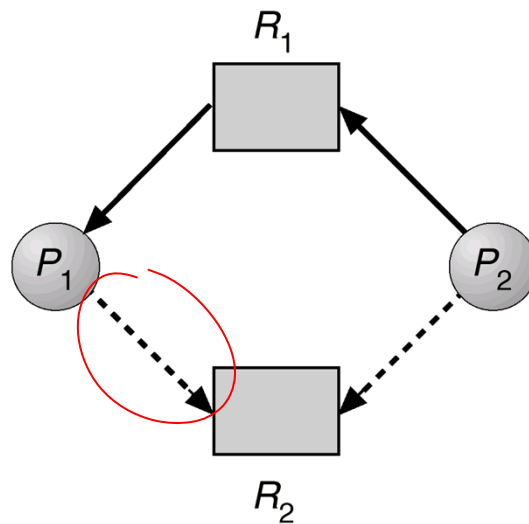
**Garantire che il sistema passi da uno stato sicuro ad un altro stato sicuro quando un processo chiede una nuova risorsa**

- Si parte da uno stato iniziale sicuro
- Una richiesta di risorsa viene soddisfatta se la risorsa è disponibile e se il sistema va in uno stato sicuro
- Se la risorsa non è disponibile, il processo deve attendere



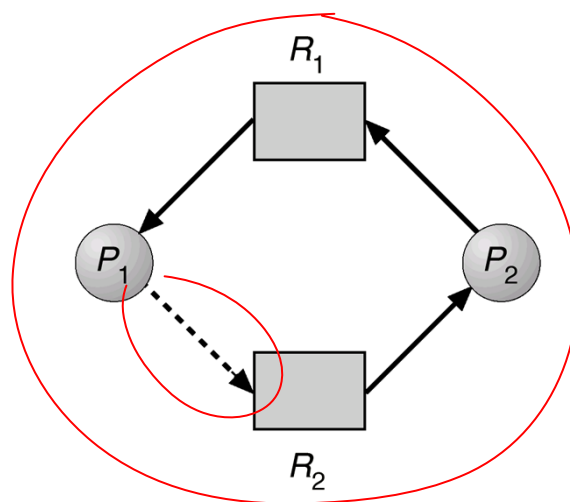
### Algoritmo del grafo di allocazione delle risorse (1)

Arco di prenotazione



### Algoritmo del grafo di allocazione delle risorse (2)

Stato non sicuro



### **Algoritmo del grafo di allocazione delle risorse (3)**

- ✓ Si costruisce il grafo di allocazione delle risorse, con gli archi di prenotazione
- Se si evidenziano cicli nel grafo, lo stato non è sicuro e quindi non si può accettare la richiesta di risorse dell'ultimo processo inserito

Vale solo per istanze singole delle risorse

### **Algoritmo del banchiere (1)**

- Gestisce istanze multiple delle risorse
- È meno efficiente dell'algoritmo del grafo di allocazione delle risorse
- Il numero massimo di istanze deve essere dichiarato a priori
- Un processo deve restituire in un tempo finito le risorse utilizzate

## Algoritmo del banchiere (2)

### Strutture dati

<b>m</b>	risorse
<b>n</b>	processi
<b>Available[1..m]</b>	risorse disponibili
<b>Max[1..n,1..m]</b>	massima richiesta di ogni processo
<b>Allocation[1..n,1..m]</b>	risorse attualmente assegnate
<b>Need[1..n,1..m]</b>	risorse da richiedere

## Algoritmo del banchiere (3)

### Algoritmo di verifica dello stato sicuro

Work[1..m]

Finish[1..n]

1. Work=Available; Finish[i]=false per  $i=0,1,\dots,n-1$

2. Si cerca  $i$  tale che:

– Finish[i]==false ✓

–  $Need_i \leq Work$

Se non esiste tale  $i$  vai al passo 4

3. Work=Work+Allocation[i]; Finish[i]=true

Vai al passo 2

4. Se, per ogni  $i$ , Finish[i]==true, allora lo stato è sicuro

## Algoritmo del banchiere (4)

### Algoritmo di richiesta delle risorse

Request[i]    richiesta del processo  $P_i$

1. Se  $\text{Request}[i] \leq \text{Need}[i]$ , vai al passo 2  
Altrimenti, solleva errore: processo ha ecceduto numero massimo di richieste
2. Se  $\text{Request}[i] \leq \text{Available}$ , vai al passo 3  
Altrimenti,  $P_i$  deve attendere: risorse non disponibili
3. Si ipotizzi di stanziare le risorse richieste:  
 $\text{Available} = \text{Available} - \text{Request}[i]$   
 $\text{Allocation}[i] = \text{Allocation}[i] + \text{Request}[i]$   
 $\text{Need}[i] = \text{Need}[i] - \text{Request}[i]$   
Se lo stato risultante è sicuro, al processo  $P_i$  vengono confermate le risorse assegnate  
Altrimenti,  $P_i$  deve aspettare per le richieste  $\text{Request}[i]$  e viene ristabilito il vecchio stato di allocazione delle risorse

## In sintesi

- Principio di evitare il deadlock
- Stato sicuro
- Algoritmo del grafo di allocazione delle risorse
- Algoritmo del banchiere

