

```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAXPAROLA 30
#define MAXRIGA 80

int main(int argc, char *argv[])
{
    int freq[MAXPAROLA]; /* vettore di contatori
delle frequenze delle lunghezze delle parole */
    char riga[MAXRIGA];
    int i, inizio, lunghezza;
    FILE *f;

    for(i=0; i<MAXPAROLA; i++)
        freq[i]=0;

    if(argc != 2)
    {
        fprintf(stderr, "ERRORE: serve un parametro con il nome del file\n");
        exit(1);
    }
    f = fopen(argv[1], "r");
    if(f==NULL)
    {
        fprintf(stderr, "ERRORE: impossibile aprire il file %s\n", argv[1]);
        exit(1);
    }

    while( fgets( riga, MAXRIGA, f ) != NULL )
```

Stallo di processi

Definizione del problema e modellizzazione

Stefano Quer

Dipartimento di Automatica e Informatica

Politecnico di Torino

Stallo (deadlock)

- ❖ Condizione di **stallo (deadlock)**
 - Un P/T richiede una risorsa non disponibile, entra in uno stato di attesa e l'attesa non termina più
- ❖ Lo stallo consiste quindi in
 - Un insieme di P/T attendono tutti il verificarsi di un evento che può essere causato solo da un altro processo dello stesso insieme
- ❖ Deadlock **implica** starvation **non** il contrario
 - La starvation di un P/T implica che tale P/T attende indefinitamente ma gli altri P/T possono procedere in maniera usuale e non essere in deadlock
 - Tutti i P/T in deadlock sono in starvation

Condizioni per il verificarsi di un deadlock

Condizioni	Descrizione
Mutua esclusione (mutual exclusion)	Deve esserci almeno una risorsa non condivisibile . La prima richiesta è soddisfatta, le successive no.
Possesso e attesa (hold and wait)	Un processo mantiene almeno una risorsa e attende per acquisire almeno un'altra risorsa.
Impossibilità di prelazione (no preemption)	Non esiste il diritto di prelazione per almeno una risorsa. Almeno una risorsa non può essere sottratta ma solo rilasciata da chi la utilizza.
Attesa circolare (circular wait)	Esiste un insieme $\{P_1, \dots, P_n\}$ di P tale che P_1 attende una risorsa tenuta da P_2 , P_2 attende una risorsa tenuta da P_3, \dots, P_n attende una risorsa tenuta da P_1 .

Devono verificarsi tutte contemporaneamente per avere un deadlock

Condizioni necessarie ma non sufficienti
Sono distinte ma non indipendenti (e.g., 4→2)

Sommario dell'unità

❖ Modellizzazione del deadlock

❖ Strategie di gestione

➤ Dello "struzzo"

- Si ignora il problema supponendo la probabilità di un deadlock nel sistema sia bassissima
- Metodo utilizzato da molti sistemi operativi, Windows e Unix inclusi
- Tanto meno appropriato quanto più aumentano la concorrenza e la complessità dei sistemi

Sezione (corrente) 01

➤ A posteriori

- Rilevare
- Ripristinare

... e nel caso si sia verificato uno stallo ...

➤ A priori

- Prevenire
- Evitare

Sezione 02

Sezione 03

Termine simile ma tecnica diversa

Modellizzazione del deadlock

- ❖ Grafo di allocazione (o assegnazione) delle risorse
 - $G = (V, E)$
 - Permette di descrivere e analizzare deadlock
- ❖ L'insieme dei vertici V è suddiviso in
 - $P = \{P_1, P_2, \dots, P_n\}$
 - Insieme dei processi del sistema (n in tutto)
 - I processi sono indistinguibili e in numero indefinito
 - Ogni processo utilizza una risorsa facendone accesso mediante protocollo standard costituito da
 - Richiesta
 - Utilizzo
 - Rilascio

Modellizzazione del deadlock

➤ $R = \{R_1, R_2, \dots, R_m\}$

- Insieme delle risorse del sistema (m in tutto)
- Le risorse sono suddivise in classi (tipi)
- Ogni risorsa di tipo R_i ha W_i istanze
- Tutte le istanze di una classe sono **identiche**: una qualsiasi istanza soddisfa una richiesta per quel tipo di risorsa

❖ L'insieme degli archi E è suddiviso in

➤ Archi di richiesta

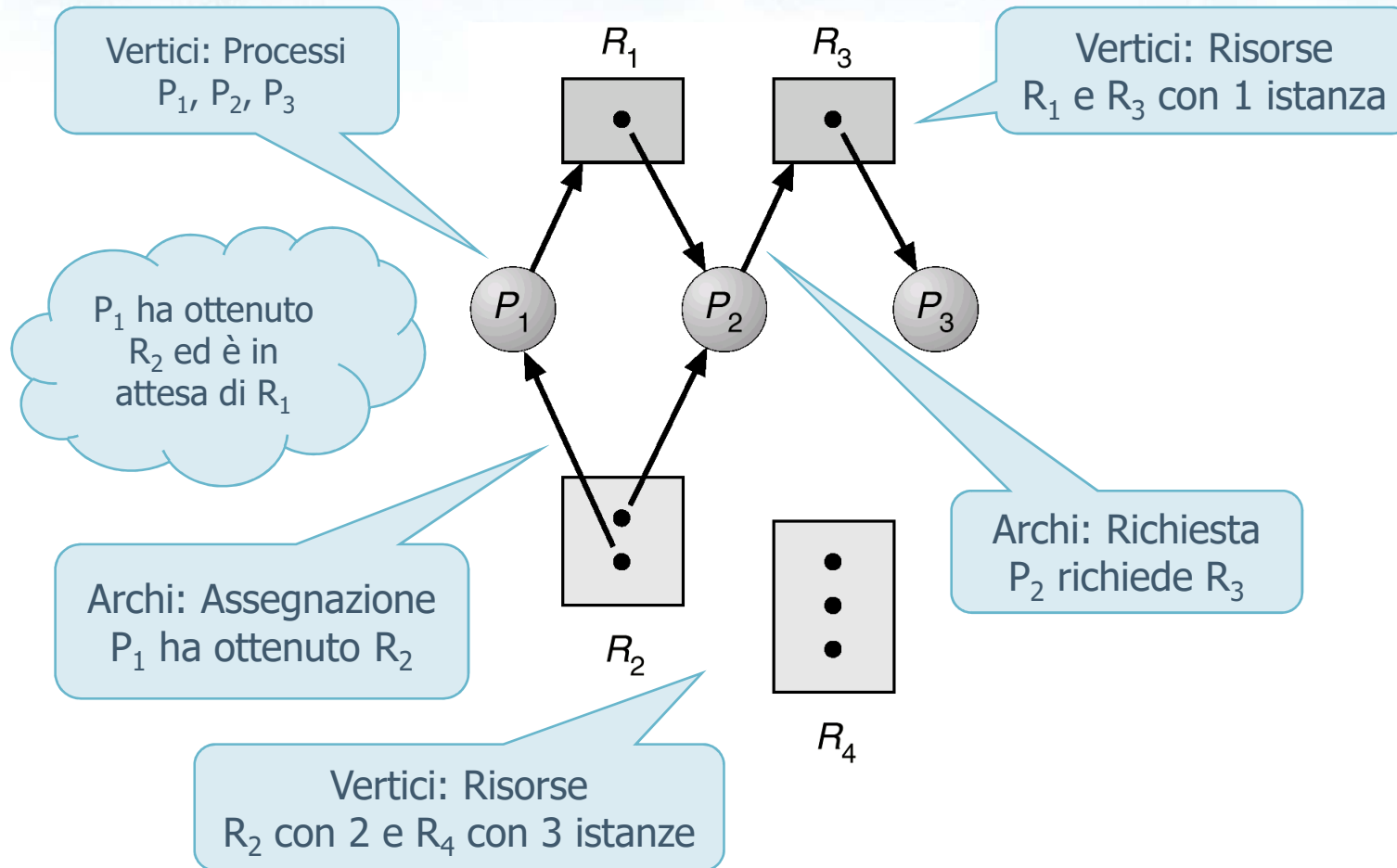
- $P_i \rightarrow R_j$, i.e., da processo a risorsa

➤ Archi di assegnazione

- $R_j \rightarrow P_i$, i.e., da risorsa a processo

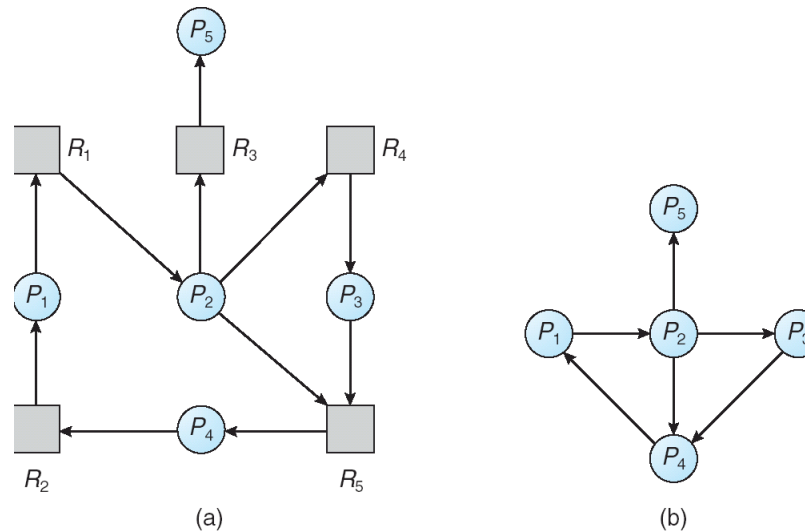
Se così non fosse
occorrerebbe
riformulare la
suddivisione in
classi

Modellizzazione del deadlock



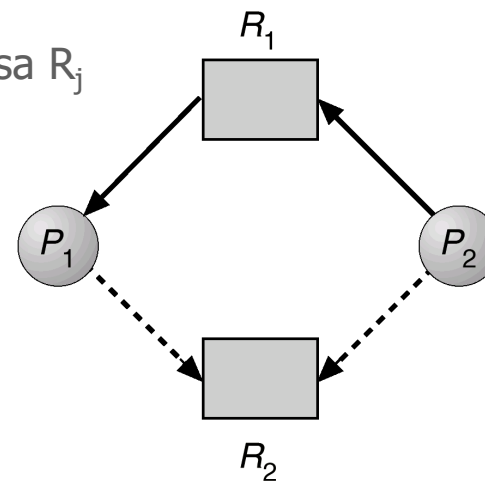
Modellizzazione del deadlock

- ❖ In alcuni casi è utile semplificare il grafo di allocazione in un **grafo di attesa**
 - Si eliminano i vertici di tipo risorsa
 - Si compongono gli archi tra i vertici rimanenti
- ❖ Utilizzo e considerazioni simili al grafo di assegnazione



Modellizzazione del deadlock

- ❖ In alcuni casi è utile estendere il grafo di allocazione in un **grafo di rivendicazione**
 - Si aggiungono al grafo di allocazione delle risorse degli archi di reclamo, di rivendicazione o di intenzione di richiesta (claim edge)
 - $P_i \rightarrow R_j$
 - Indica che il processo P_j richiederà (in futuro) la risorsa R_j
 - È rappresentato mediante linea tratteggiata



Tecniche di gestione a posteriori

- ❖ Si permette al sistema di entrare in uno stato di deadlock per poi intervenire
- ❖ L'algoritmo richiede quindi due passi
 - **Rilevazione** (della condizione di stallo)
 - Il sistema esegue un algoritmo di rilevazione dello stallo all'interno del sistema (**detection**)
 - **Ripristino** (del sistema)
 - Se esiste uno stallo si applica una strategia di ripristino (**recovery**)

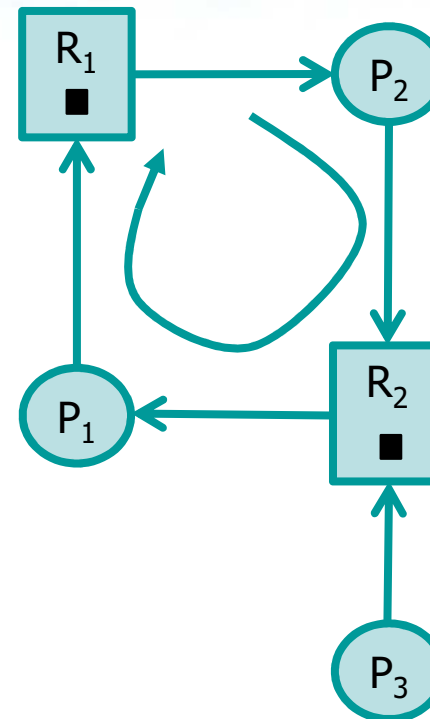
Rilevazione

- ❖ Dato un grafo di assegnazione è possibile verificare la presenza di uno stallo verificando la presenza di cicli
 - Se il grafo non contiene cicli allora non c'è deadlock
 - Se il grafo contiene uno o più cicli allora
 - C'è sicuramente deadlock, se esiste solo un'istanza per ciascun tipo di risorsa
 - C'è la possibilità di deadlock, se esiste più di un'istanza per tipo di risorsa
 - La presenza di cicli è condizione necessaria ma non sufficiente nel caso di istanze multiple

Nei casi di istanze multiple si veda l'“Algoritmo del Banchiere”

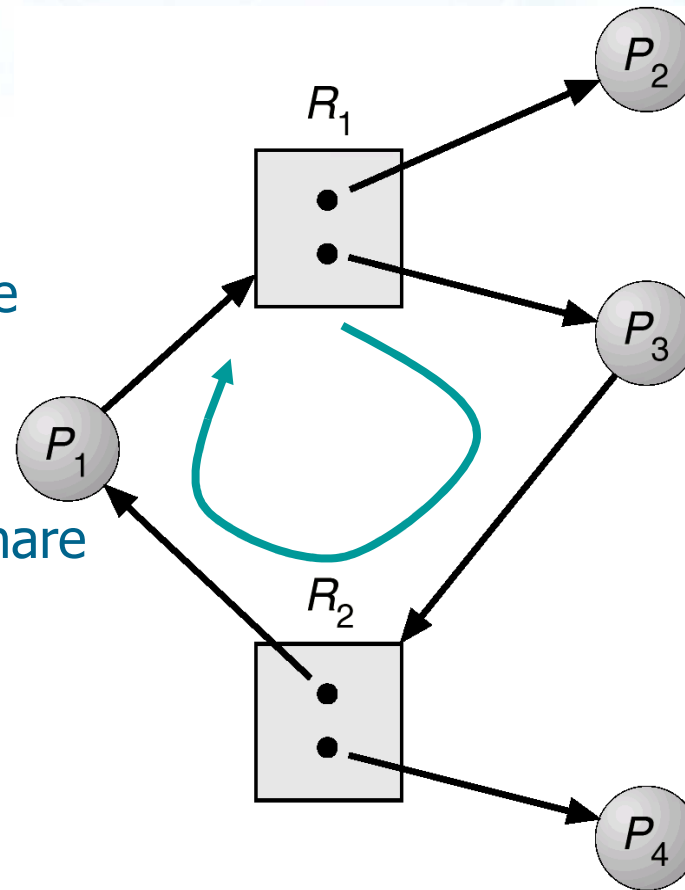
Esempio

- ❖ Processi
 - P_1, P_2, P_3
- ❖ Risorse
 - R_1 e R_2 con una istanza
- ❖ Esistenza di un ciclo
- ❖ Condizione di stallo
 - P_1 attende P_2
 - P_2 attende P_1



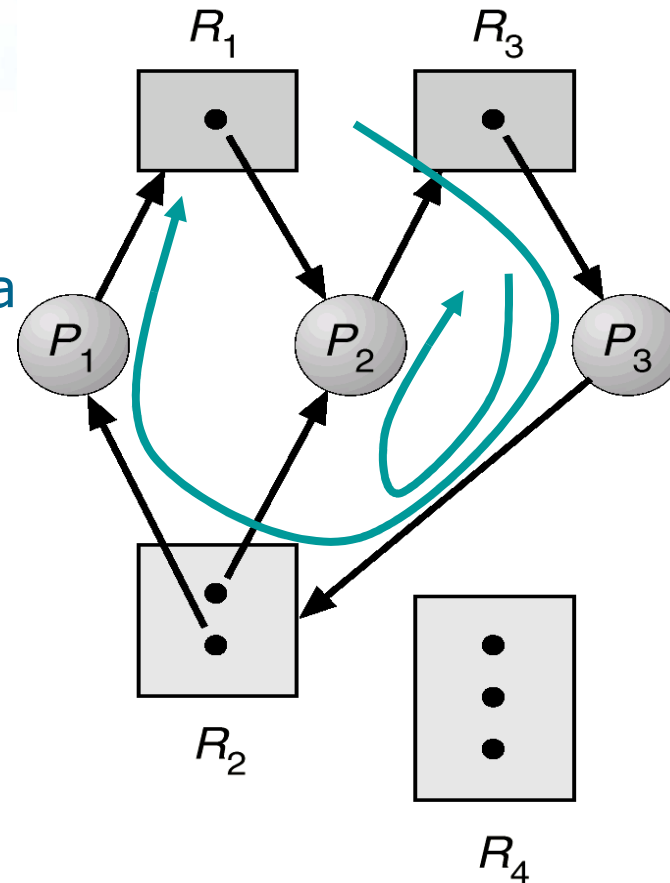
Esempio

- ❖ Processi
 - P_1, P_2, P_3, P_4
- ❖ Risorse
 - R_1 e R_2 con due istanze
- ❖ Esistenza di un ciclo
- ❖ Non esiste stallo
 - P_2 e P_4 possono terminare
 - P_1 può acquisire R_1 e terminare
 - P_3 può acquisire R_2 e terminare



Esempio

- ❖ Processi
 - P_1, P_2, P_3
- ❖ Risorse
 - R_1 e R_3 con una istanza
 - R_2 con due istanza
 - R_4 con tre istanze
- ❖ Esistenza di due cicli
- ❖ Condizione di stallo
 - P_1 attende R_1
 - P_2 attende R_3
 - P_3 attende R_2



Complessità

- ❖ Ogni fase di rilevazione ha un costo
 - **Determinazione dei ciclo nel grafo**
 - La presenza di cicli può essere verificata mediante visita in profondità
 - Un grafo è aciclico se un visita in profondità non incontra archi etichettati "backward" verso vertici grigi
 - Se si raggiunge un vertice grigio, ovvero si attraversa un arco backward, si ha un ciclo
 - Il costo temporale di tale operazione è pari a
 - $\Theta(|V|+|E|)$ per rappresentazioni con lista di adiacenza
 - $\Theta(|V|^2)$ per rappresentazioni con matrice di adiacenza

Complessità

- ❖ Quando spesso si effettuano le rilevazioni?
 - Ogni volta che un processo fa una richiesta non soddisfatta immediatamente
 - A intervalli di tempo fissi, e.g., ogni 30 minuti
 - A intervalli di tempo variabili, e.g., quando l'utilizzo della CPU scende sotto una certa soglia

Ripristino

- ❖ Per ripristinare un corretto funzionamento sono possibili diverse strategie
 - Terminare tutti i processi in stallo
 - Terminare un processo alla volta tra quelli in stallo
 - Si seleziona un processo, si termina, si ri-controlla la condizione di stallo, eventualmente iterando
 - Prelazionare le risorse a un processo alla volta tra quelli in stallo
 - Si seleziona un processo, si prelazionano le sue risorse, se ne fa il roll-back, si ri-controlla la condizione di stallo, eventualmente iterando

Ripristino

Strategia	Descrizione
Terminare tutti i processi in stallo	<ul style="list-style-type: none">• Complessità: semplice causare inconsistenze sulle basi dati• Costo: molto più alto di quanto potrebbe essere strettamente necessario
Terminare un processo alla volta tra quelli in stallo	<ul style="list-style-type: none">• Complessità: alta in quanto occorre selezionare l'ordine delle vittime con criteri oggettivi (priorità, tempo di esecuzione effettuato e da effettuare, numero risorse possedute, etc.)• Costo: elevato, i.e., dopo ogni terminazione occorre riverificare la condizione di stallo
Prelazionare le risorse a un processo alla volta	<ul style="list-style-type: none">• Complessità: occorre effettuare il rollback, i.e., fare ritornare il processo vittima a uno stato sicuro• Costo: la selezione di una vittima deve minimizzare il costo della prelazione

Conclusioni

- ❖ Rilevazione e ripristino sono operazioni
 - Complesse logicamente
 - Onerose temporalmente
- ❖ In ogni caso se un processo richiede molte risorse è possibile causare starvation
 - Lo stesso processo viene ripetutamente scelto come vittima e incorre in rollback ripetuti
 - È opportuno inserire il numero di terminazioni tra i parametri di scelta della vittima, facendone aumentare la priorità o simili