

```
#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

# Tecniche di prevenzione

Stefano Quer

Dipartimento di Automatica e Informatica

Politecnico di Torino

## Prevenzione

### ❖ Le tecniche di **prevenzione**

- Cercano di **controllare** le modalità di richiesta delle risorse per **prevenire** il verificarsi di **almeno una** delle condizioni necessarie
- Cioè cercano di prevenire il verificarsi di almeno una delle condizioni seguenti
  - Mutua esclusione (mutual exclusion)
  - Possesso e attesa (hold and wait)
  - Impossibilità di prelazione (no preemption)
  - Attesa circolare (circular wait)

## Mutua esclusione

Uno stallo si verifica a causa della "mutua esclusione" quando un processo rimane indefinitivamente in attesa su una risorsa non condivisibile

Quindi uno stallo potrebbe essere evitato se

1. Non esistessero risorse non condivisibili
2. Non si potesse rimanere in attesa di una risorsa non condivisibile

Strategia 1

Proibire risorse non condivisibili  
Tale strategia è considerata generalmente molto restrittiva

Strategia 2

Proebire l'attesa su risorse non condivisibili  
Tale strategia è considerata complessa da realizzare

## Possesso e attesa

Un stallo si verifica a causa di un "possesso e attesa" quando un P possiede una o più risorse e ne chiede di ulteriori

Quindi una condizione di possesso e attesa potrebbe essere evitata imponendo che un P chieda una risorsa solo quando non ne possiede altre

### **Request All First (RAF)**

I P devono acquisire tutte le risorse necessarie prima di iniziare l'attività di elaborazione

- Scarso utilizzo delle risorse
- Risorse eventualmente assegnate molto prima di essere utilizzate

### **Release Before Request (RBR)**

Ai P è consentito richiedere risorse solamente se non ne hanno già acquisite in precedenza  
Prima di ogni nuova richiesta ogni P deve rilasciare le risorse già possedute

- Possibilità di starvation
- P che richiedono molte risorse molto utilizzate possono dover "ricominciare" molto spesso

## Impossibilità di prelazione

Un stallo si verifica a causa dell' "impossibilità di prelazione" quando una risorsa non può essere sottratta a un P

In generale è complesso sottrarre risorse a altri processi in esecuzione  
Però si potrebbe ottenere un effetto simile

Permettere la prelazione delle risorse possedute dal processo stesso

Se un processo, che mantiene alcune risorse, ne chiede un'altra che non può essere allocata immediatamente, è costretto a rilasciare tutte le risorse mantenute (preemption)  
Le risorse liberate sono aggiunte alla lista delle risorse che il processo attende di acquisire  
Il processo sarà svegliato solo quando potrà riacquisire tutte le sue vecchie risorse e quelle nuove che richiede

## Impossibilità di prelazione

Un stallo si verifica a causa dell' "impossibilità di prelazione" quando una risorsa non può essere sottratta a un P

Permettere la prelazione di risorse possedute da un altro processo purchè esso sia in fase di attesa

Se la risorsa richiesta non è disponibile si verifica quale processo la possiede  
Se il processo che la possiede è a sua volta in attesa si sottrae a tale processo la risorsa richiesta assegnandola al processo che ne ha fatto richiesta  
In caso contrario il processo viene messo in attesa e in futuro potranno essere prelazionate le sue risorse

Entrambe le strategie

- Sono applicabili a risorse su cui è semplice salvare e ripristinare lo stato (registri CPU, memoria centrale, etc.)
- Non sono applicabili quando lo stato della risorsa non è ricostruibile (file, stampanti, etc.)

## Attesa circolare

Un stallo si verifica a causa di un' "attesa circolare" quando un insieme di  $P$  è tale per cui ogni  $P$  dell'insieme attende una risorsa posseduta da un altro  $P$  dell'insieme

Per evitare tale condizione si potrebbe imporre un ordinamento totale tra tutte le classi di risorse

### **Hierarchical Resource Usage (HRU)**

- Impone una relazione di ordinamento totale tra i vari tipi di risorse, associando a ciascuno di essi un numero intero. Esempio: HD = 1, DVD = 5, stampanti = 12
- Forza ogni processo a richiedere le risorse con un ordine crescente di enumerazione

In generale la verifica HRU sia applicato può essere effettuata dal

- Programmatore
- Sistema operativo. Il tool "witness", disponibile in UNIX versione FreeBSD, controlla l'ordine dei lock effettuati dai processi

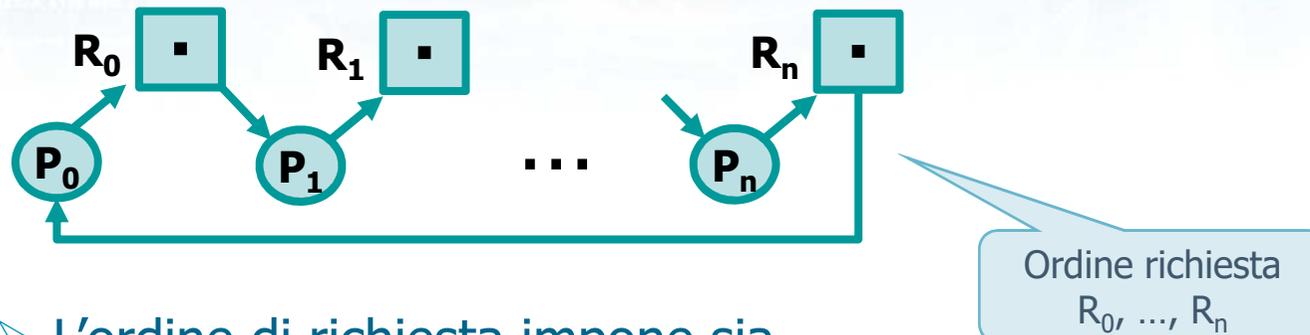
## Attesa circolare

- ❖ Sia  $F$  la funzione che impone un ordine univoco tra tutte le classi di risorse  $R_i$  del sistema
- ❖ Un processo  $P$ 
  - Abbia precedentemente richiesto una istanza della risorsa  $R_{old}$  e faccia richiesta di una istanza di  $R_{new}$
  - Se  $F(R_{new}) > F(R_{old})$ 
    - La risorsa viene concessa
  - Se  $F(R_{new}) \leq F(R_{old})$ 
    - Il processo deve rilasciare tutte le risorse di tipo  $R_i$  per cui  $F(R_{new}) \leq F(R_i)$  prima di ottenere  $R_{new}$

## Attesa circolare

- ❖ È possibile dimostrare che tale condizione è sufficiente per evitare l'attesa circolare
  - Ovvero, se le risorse si richiedono in un certo ordine è vero che non è possibile avere attesa circolare?
  - Procediamo per assurdo, supponendo ci sia attesa circolare, ovvero supponiamo che esista un insieme di processi che
    - Sono stati richiesti con ordine specificato, e.g., in ordine numerico crescente
    - Risultino in attesa circolare

## Attesa circolare



- L'ordine di richiesta impone sia
  - $F(R_k) < F(R_{k+1}), \forall k = 0 .. n - 1$
- Però in base all'attesa circolare si desume che
  - $F(R_0) < F(R_1) < \dots < F(R_n) < F(R_0)$
- Questo implica che
  - $F(R_0) < F(R_0)$
 il che è assurdo

Dato che  $P_i$  possiede  $R_i$  e ha richiesto  $R_{i-1}$   
 $R_i$  è stata richiesta prima di  $R_{i-1}$   
 Quindi  $F(R_i) > F(R_{i-1})$