

# **SISTEMI OPERATIVI**

Gestione del Processore  
Processi

## **Lezione 1 – Processi**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

### **Sommario**

- Multi-tasking
- Concetto di processo
- Stato di evoluzione della computazione di un processo
- Stato di uso del processore da parte di un processo
- Diagramma degli stati del processo
- Supporti per la gestione dei processi

## Multi-Tasking (1)

### Problema

- Scarso sfruttamento del processore a causa dei tempi morti per l'attesa del completamento delle operazioni di ingresso/uscita

### Soluzione

- Esecuzione di altri programmi quando un programma ha effettuato la richiesta di una operazione di ingresso/uscita ed è in attesa di risposta dalla periferica

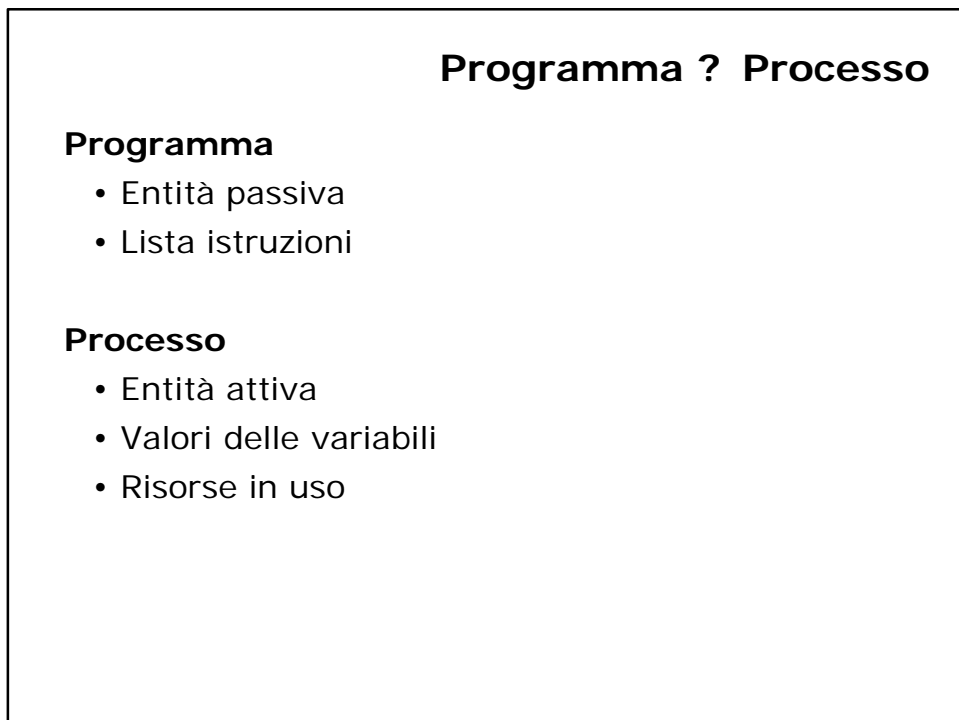
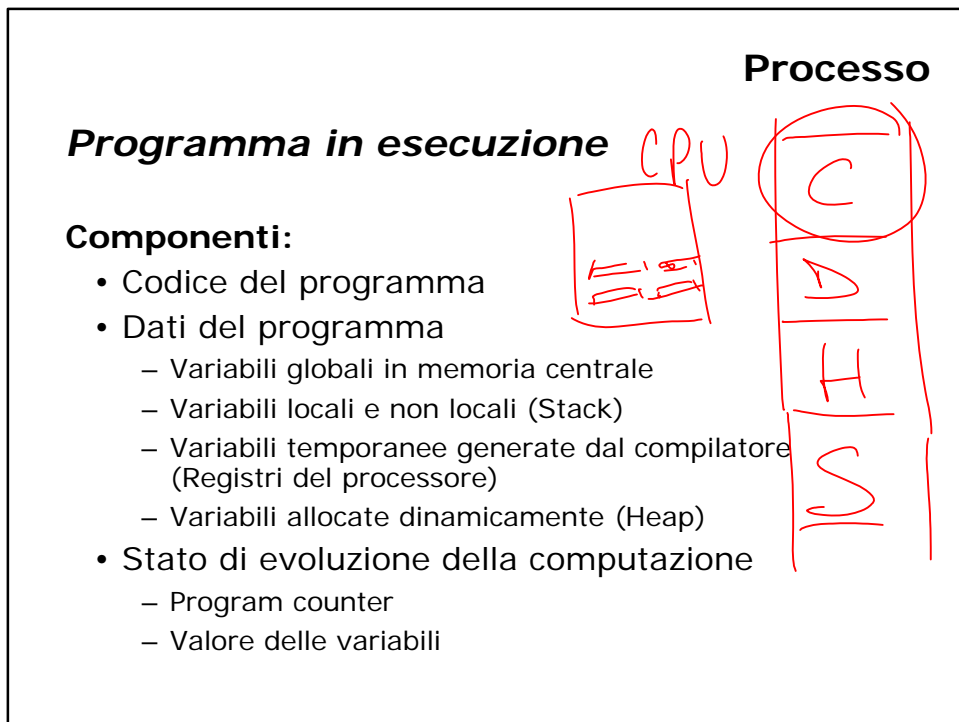
## Multi-Tasking (2)

### Realizzazione

- *Multiprogrammazione*  
per avere più programmi in memoria centrale da eseguire
- *Multi-tasking*  
per avere la turnazione dei programmi sul processore quando il programma in esecuzione è in attesa della risposta delle periferiche

#### Il processore

- gestisce più flussi di esecuzione indipendenti
- esegue più programmi  
apparentemente in parallelo



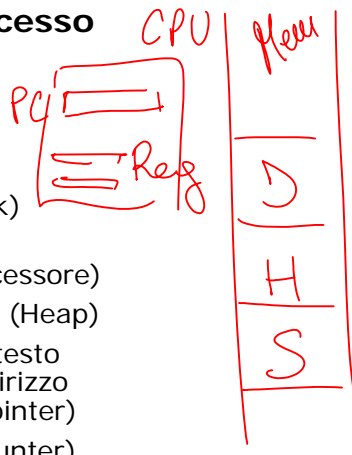
## Evoluzione della computazione di un processo

- Il processo è una funzione che trasforma informazioni eseguendo le istruzioni del programma partendo dai valori iniziali (eventualmente acquisiti durante l'esecuzione stessa attraverso le periferiche) e producendo i risultati finali (emessi attraverso le periferiche)
- Il processo è una macchina a stati finiti:
  - gli stati sono le informazioni su cui opera
  - le transizioni sono dovute alle istruzioni che modificano le informazioni

## Stato di evoluzione della computazione di un processo

**Insieme dei valori di tutte le informazioni da cui dipende l'evoluzione della computazione del processo**

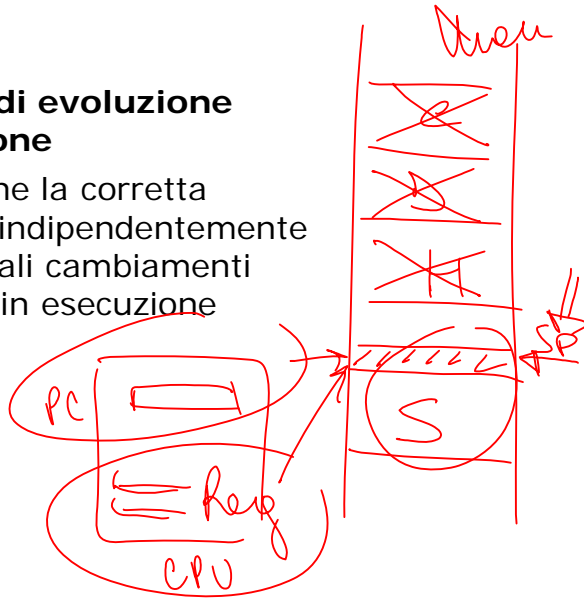
- Variabili globali del programma (area dati globali)
- Variabili locali e non locali delle procedure del programma (Stack)
- Variabili temporanee introdotte dal compilatore (registri del processore)
- Variabili allocate dinamicamente (Heap)
- Informazioni di gestione del contesto della chiamata di procedure (indirizzo di ritorno, base pointer, stack pointer)
- Istruzione corrente (Program counter)



## Cambiamento del processo in esecuzione

### Salvare lo stato di evoluzione della computazione

per garantirne la corretta esecuzione, indipendentemente dagli eventuali cambiamenti dei processi in esecuzione



## Uso del processore da parte di un processo

Durante la sua computazione, un processo può:

- usare il processore
  - la computazione evolve effettivamente
- attendere l'uso del processore, pur avendo tutte le altre risorse informative o fisiche necessarie
  - la computazione potrebbe evolvere, ma non lo fa poiché le istruzioni non possono essere eseguite
- attendere che una risorsa informativa o fisica diventi disponibile
  - la computazione non può evolvere poiché mancano alcune risorse oltre al processore

## Stato del processo

**Stato di uso del processore  
da parte di un processo**

brevemente *stato del processo*

**è la modalità di uso del processore  
da parte del processo**

### **Possibili stati:**

- |                         |              |
|-------------------------|--------------|
| • Nuovo                 | New          |
| • In esecuzione         | Running      |
| • In attesa             | Waiting      |
| • Pronto all'esecuzione | Ready-to-Run |
| • Terminato             | Terminated   |

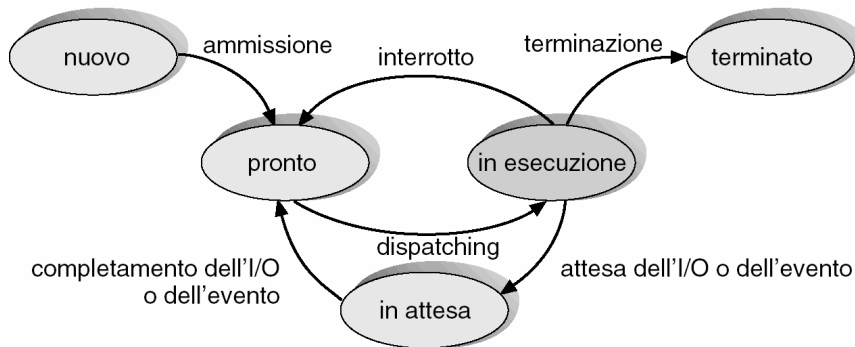
## Diagramma degli stati del processo (1)

**Rappresenta insieme degli stati del processo e  
le transizioni tra stati**

Grafo orientato

- Nodi: stati del processo
- Archi: transizioni tra gli stati del processo

## Diagramma degli stati del processo (2)



## Supporti per la gestione dei processi (1)

### Process Control Block (PCB)

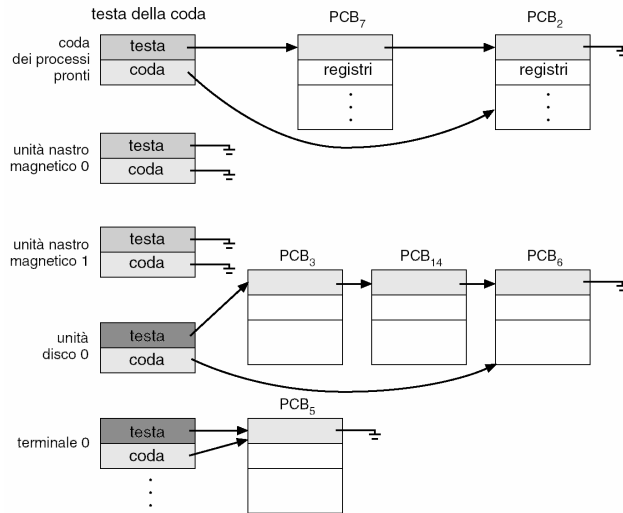
#### Blocco di Controllo del Processo

#### Informazioni sul processo a supporto della gestione del processore

- Identificatore del processo (Numero)
- Stato del processo
- Program counter
- Registri della CPU
- Informazioni per la schedulazione della CPU
- Informazioni per la gestione della memoria centrale (limiti di memoria)
- Informazioni sullo stato dell'I/O (ad esempio: file aperti)
- Informazioni per l'accounting

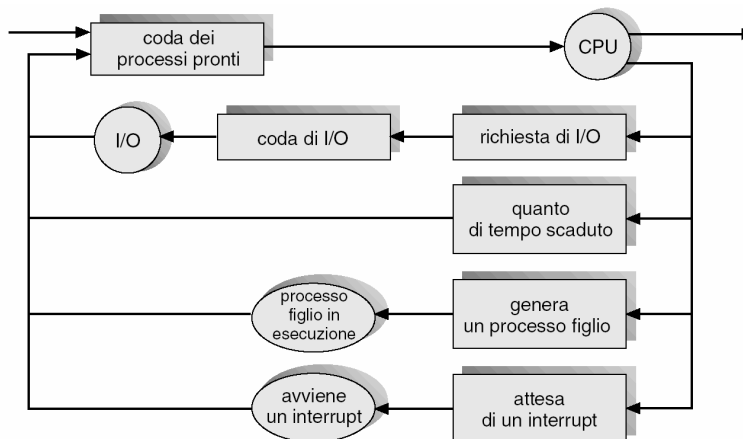
## Supporti per la gestione dei processi (2)

### Code dei processi nei vari stati



## Supporti per la gestione dei processi (3)

### Transizioni tra le code





## **In sintesi**

- Abbiamo visto:
  - Multi-tasking
  - Processo
  - Stato di evoluzione della computazione di un processo
  - Stato di uso del processore da parte di un processo
  - Diagramma degli stati del processo
  - Supporti per la gestione dei processi
- Ricordate che:
  - Lo stato del processo NON È lo stato di evoluzione della computazione del processo

# **SISTEMI OPERATIVI**

Gestione del Processore  
Processi

## **Lezione 2 – Creazione e terminazione dei processi**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

### **Sommario**

- Modelli computazionali
- Modalità e funzioni per
  - creazione e attivazione dei processi
  - terminazione dei processi

## **Processi come flussi di operazioni**

- Processo = flusso di esecuzione di computazione
- Flussi separati = processi separati
  - Come si generano?
  - Come un programma si trasforma in un insieme di processi?
  - Come interagiscono i processi?
- Flussi sincronizzati
  - processi evolvono sincronizzandosi
- Flussi indipendenti
  - processi evolvono autonomamente

## **Modellazione della computazione a processi**

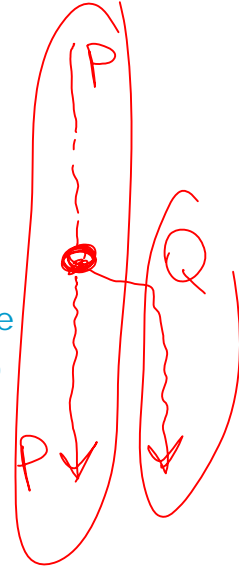
- Modelli di computazione:
  - Processo monolitico
  - Processi cooperanti
- Modelli di realizzazione del codice eseguibile:
  - Programma monolitico
  - Programmi separati
- Realizzazione dei modelli di computazione:
  - Programma monolitico è eseguito come processo monolitico
  - Programma monolitico genera processi cooperanti
  - Programmi separati sono eseguiti come processi cooperanti (ed eventualmente generano ulteriori processi cooperanti)

## Generazione di un processo

- Il processo in esecuzione chiama una funzione di sistema operativo che crea e attiva un nuovo processo

*fork*

- Processo generante → processo padre
- Processo generato → processo figlio

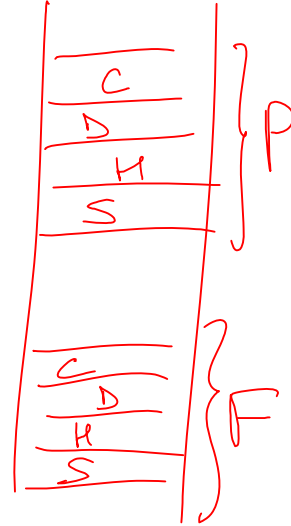


## Risorse dei processi

- Condivise col padre
- Parzialmente condivise col padre
- Indipendenti dal padre (ottenute dal sistema)
- Passaggio dei dati di inizializzazione

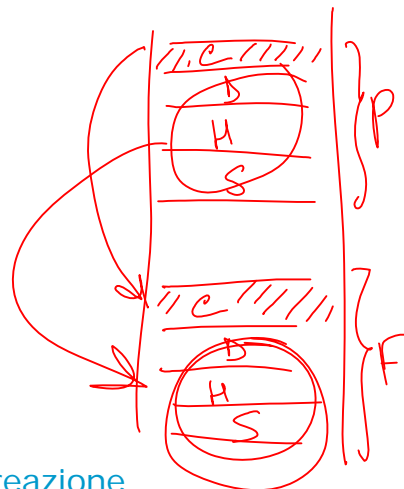
## Spazio di indirizzamento (1)

- Lo spazio di indirizzamento del processo figlio è sempre **distinto** da quello del processo padre



## Spazio di indirizzamento (2)

- Spazio del processo figlio è il duplicato dello spazio del processo padre



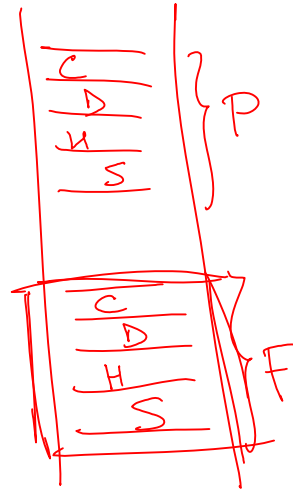
- stesso programma
- stessi dati all'atto della creazione

## Spazio di indirizzamento (3)

- Il processo figlio ha un nuovo contenuto del suo spazio di indirizzamento

*exec*

- nuovo programma



## Esempio

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    int pid;

    /* genera un altro processo */
    pid = fork();

    if (pid < 0) { /* si è verificato un errore */
        fprintf(stderr, "Fork fallita");
        exit(-1);
    }
    else if (pid == 0) { /* processo figlio */
        execlp("/bin/ls", "ls", NULL);
    }
}
```

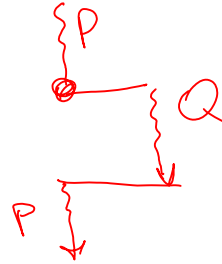
## Esecuzione dei processi

- Il padre continua l'esecuzione in modo concorrente ai figli

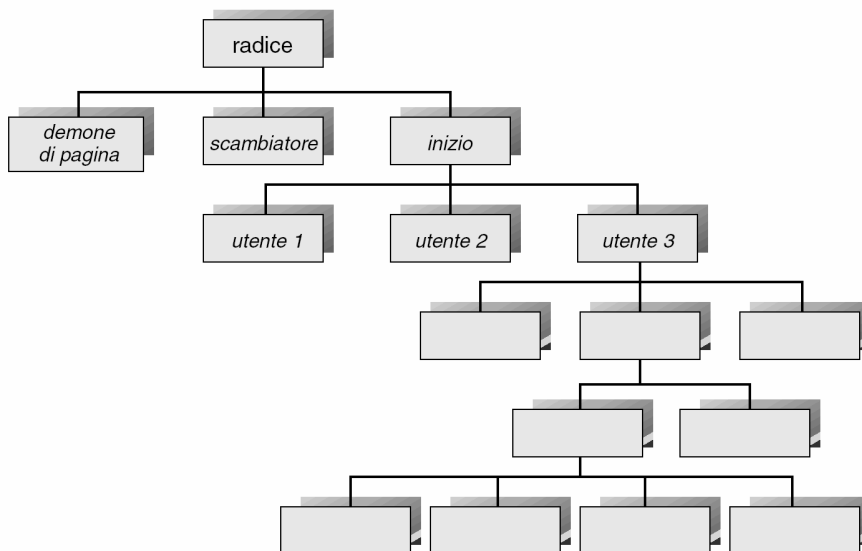


- Il padre attende finché alcuni o tutti i suoi figli sono terminati SD

Want



## Albero dei processi



## Terminazione di un processo (1)

### Terminazione normale dopo l'ultima istruzione

*exit*

- restituzione un valore di stato
- deallocazione delle risorse

```
}  
else if (pid == 0) { /* processo figlio */  
    execlp("/bin/ls", "ls", NULL);  
}  
else { /* processo padre */  
    /* il processo padre attenderà il completamento del figlio */  
    wait(NULL);  
    printf("Figlio terminato");  
    exit(0);  
}  
}
```

## Terminazione di un processo (2)

### Terminazione in caso di anomalia

*abort*

#### Cause possibili

- Eccessivo uso di una risorsa
- Compito non più necessario
- Terminazione a cascata



## **In sintesi**

- Caratteristiche e funzioni  
dell'attivazione e  
della terminazione  
dei processi

## **SISTEMI OPERATIVI**

Gestione del Processore  
Processi

### **Lezione 3 – Sospensione e riattivazione dei processi**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Come si realizza il multi-tasking
- Time sharing
- Come si realizza il time-sharing
- Sospensione di un processo
- Riattivazione di un processo

### **Classificazione dei processi rispetto all'uso delle risorse fisiche**

- processo I/O-bound (legato all'I/O)
  - molte operazioni di I/O
- processo CPU-bound (legato al processore)
  - molte operazioni aritmetico-logiche e in memoria centrale

### **Realizzazione del multi-tasking (1)**

- Obiettivo:
  - la turnazione dei processi sul processore per massimizzare lo sfruttamento processore
- Metodologia:
  - Sospensione del processo in esecuzione
  - Ordinamento dei processi in stato di pronto ([scheduling dei processi](#))
  - Selezione del processo in stato di pronto da mettere in esecuzione ([dispatching](#))
  - Riattivazione del processo selezionato

## Realizzazione del multi-tasking (2)

- Politiche:
  - Definizione delle opportunità di sospensione del processo in esecuzione
  - Definizione dell'ordinamento dei processi pronti (scheduling dei processi)
- Meccanismi:
  - Sospensione del processo in esecuzione con salvataggio del suo contesto di esecuzione
  - Dispatching del processo da mettere in esecuzione
  - Riattivazione di un processo con ripristino del suo contesto di esecuzione

## Politiche di sospensione dei processi nel multi-tasking

Il processo in esecuzione viene sospeso:

- dopo aver effettuato una richiesta di I/O
  - dopo aver creato un sottoprocesso attendendone la terminazione
  - quando rilascia volontariamente il processore
- Sospensione sincrona con l'evoluzione della computazione in procedure del sistema operativo

*implicit*

*explicit*

## Time Sharing (1)

### Multi-tasking a condivisione di tempo

Obiettivo:

gestire la turnazione dei processi sul processore  
in modo da creare l'illusione  
di *evoluzione contemporanea* agli utenti interattivi

Problema:

processi CPU-bound non rilasciano  
il processore abbastanza frequentemente  
da permettere tale illusione

Soluzione:

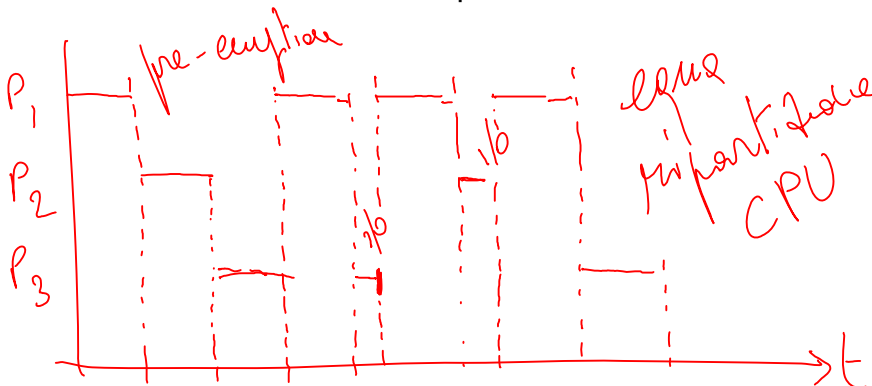
forzare il rilascio del processore (pre-emption)

## Time Sharing (2)

Quanto di tempo (**time slice**)



intervallo di tempo massimo  
di uso consecutivo del processore  
consentito a ciascun processo



## Time Sharing (3)

Real-time clock

dispositivo che scandisce il tempo  
generando periodicamente una interruzione

Problema:

periodo RTC  $p_{RTC}$  troppo breve  
→ sovraccarico di gestione dell'interruzione

Soluzione:

$? = k p_{RTC}$

## Politiche di sospensione dei processi nel time-sharing

Il processo in esecuzione viene sospeso:

1. dopo aver effettuato una richiesta di I/O ✓
2. dopo aver creato un sottoprocesso attendendone la terminazione ✓
3. quando rilascia volontariamente il processore ✓
4. quando scade il quanto di tempo

→ Rispetto all'evoluzione della computazione

- Sospensione sincrona: 1, 2, 3
- Sospensione asincrona: 4

→ Rispetto alla scrittura del programma

- Sospensione implicita: 1, 2, 4
- Sospensione esplicita: 3

## Sospensione del processo in esecuzione

- Attivazione della procedura di sospensione
  - Sincrona rispetto computazione, in stato supervisore (in procedure di I/O, creazione processi)
  - Sincrona rispetto computazione, in stato utente (in rilascio volontario)
  - Asincrona rispetto computazione (allo scadere del time slice nel time sharing)
- Salvataggio del contesto di esecuzione
  - Salvare tutti i registri del processore sullo stack
  - Salvare lo stack pointer nel Process Control Block

*Come per risposta  
e interrupt*

## Riattivazione del processo

- Ripristino del contesto di esecuzione
  - Ripristinare il valore del registro che punta alla base dello stack prendendolo dal Process Control Block del processo da riattivare
  - Ripristinare il valore dello stack pointer prendendolo dal Process Control Block del processo da riattivare
  - Ripristinare tutti i registri del processore prendendoli dallo stack

## Cambiamento del processo in esecuzione (1)

### Cambiamento di contesto

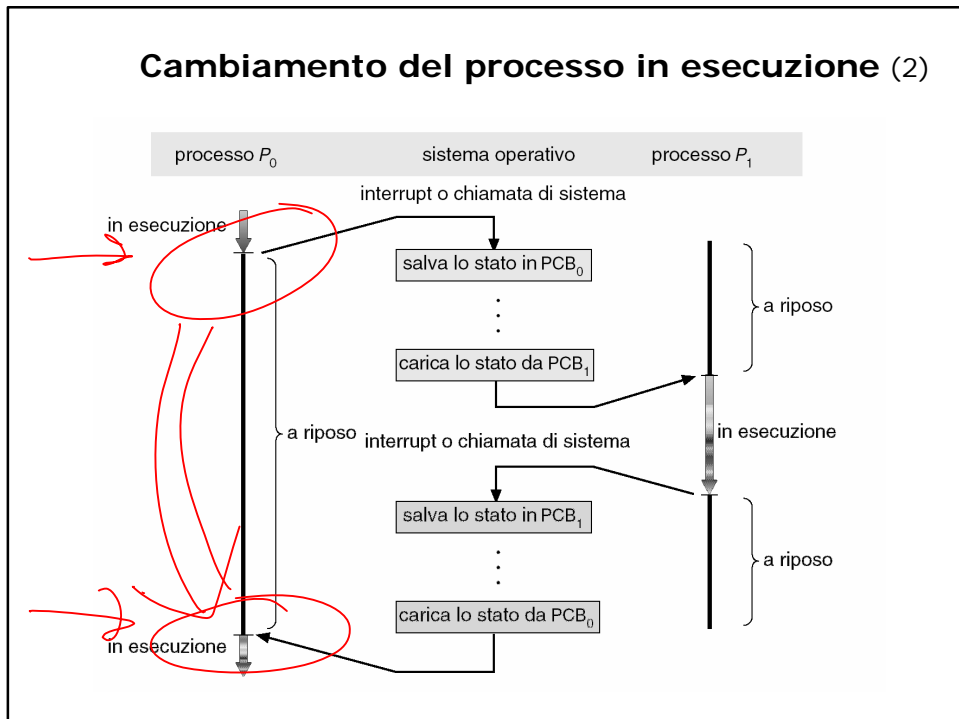
#### Context Switch

Sospensione del processo in esecuzione

+

Riattivazione del processo da mettere in esecuzione

## Cambiamento del processo in esecuzione (2)





## **Dispatching del processo per esecuzione**

- Prendere il primo processo in stato di pronto nella lista dei processi pronti generata dalla schedulazione dei processi

## **In sintesi**

- Realizzazione del multi-tasking
- Concetto di time-sharing
  - Time slice
  - Pre-rilascio (pre-emption)
- Realizzazione del time sharing
- Sospensione di un processo
- Riattivazione di un processo

# **SISTEMI OPERATIVI**

Gestione del Processore  
Thread

## **Lezione 1 – Thread**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

### **Sommario**

- Motivazioni
- Concetto di thread
- Benefici
- Supporti di gestione

## Motivazioni

### Attività tipiche di un'applicazione:

- Controllo del flusso di operazioni
- I/O
- Elaborazione

### Applicazioni ad alta disponibilità di servizio e basso tempo di risposta:

- Sistemi di wordprocessing interattivi avanzati
- Server web
- Sistemi informativi complessi

### Problemi:

- Esecuzione di più flussi di controllo nello stesso processo per attività simili
- Attesa in operazioni di I/O impedisce di servire richieste di attività simili
- Condivisione della memoria centrale tra vari processi per ridurre i tempi di scambio di informazioni

### Soluzione in ambiente tradizionale a processi:

- Processo server + vari processi client

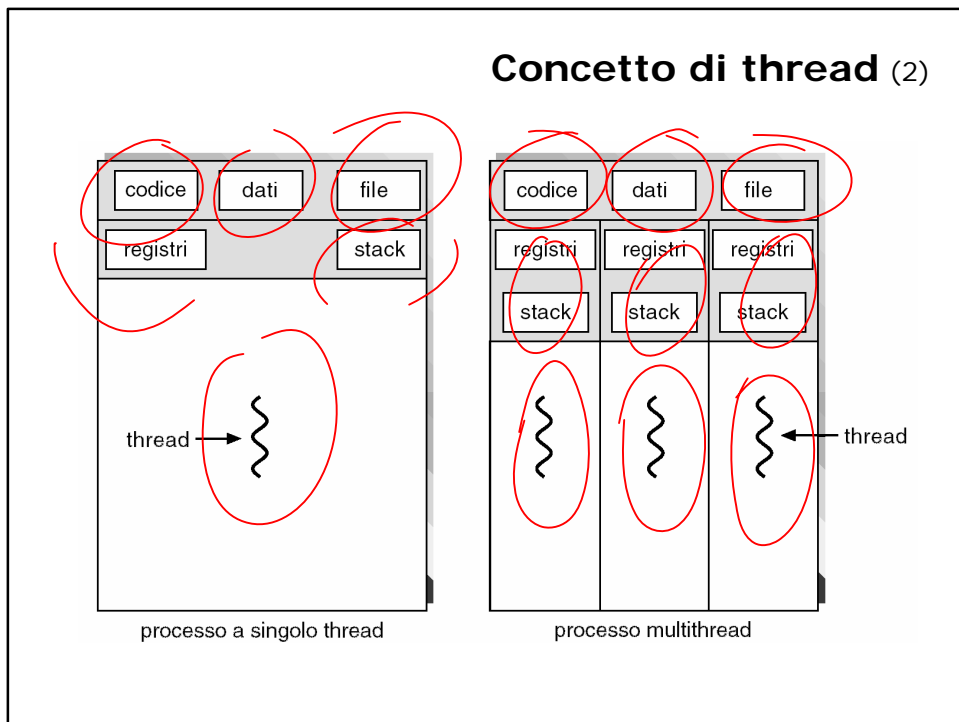
*non è efficiente!*

## Concetto di thread (1)

### Flusso di controllo dell'esecuzione di istruzioni di un programma

Un processo tradizionale (**processo pesante**)  
ha un solo thread

Un processo multi-thread  
ha più thread  
operanti contemporaneamente  
con parte delle informazioni in memoria  
centrale condivise



## Benefici

- Prontezza di risposta
- Condivisione di risorse
- Economia
- Utilizzo di architetture multiprocessore

## **Supporti di gestione**

### **Livello di gestione dei thread:**

- utente
- kernel

### **Libreria di thread:**

- spazio utente: chiamata a funzione locale
- spazio kernel: chiamata di sistema

## **In sintesi**

### **Il thread è**

- il flusso di controllo delle operazioni nel processo
- l'unità di base di utilizzo della CPU

**Condividono le risorse del processo che li ha generati**

**Consentono una maggiore efficienza del sistema**

**Possono essere supportati a livello di kernel o utente**

# **SISTEMI OPERATIVI**

Gestione del Processore  
Thread

## **Lezione 2 – Modelli multi-thread**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

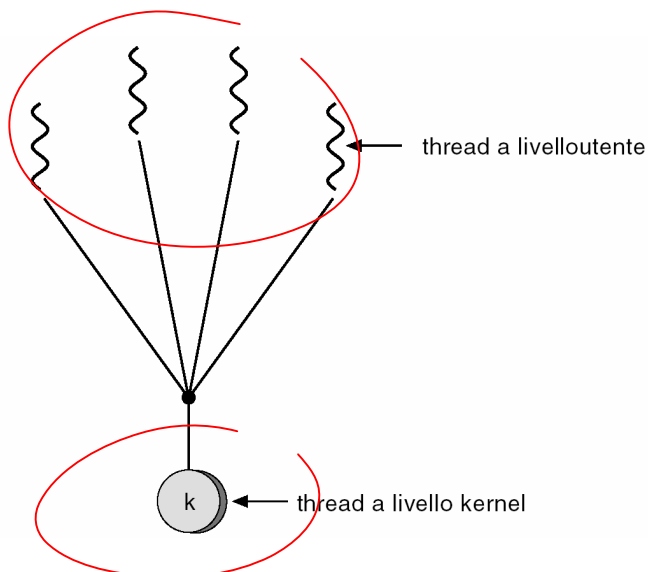
### **Sommario**

- Modelli di mappaggio di sistemi multi-thread utente nel kernel
  - Modello multi-thread multi-a-uno
  - Modello multi-thread uno-a-uno
  - Modello multi-thread multi-a-molti
  - Modello a due livelli
- Modelli di organizzazione della cooperazione tra thread
  - thread simmetrici
  - thread gerarchici
  - thread in pipeline

## Realizzazione di sistemi multi-thread

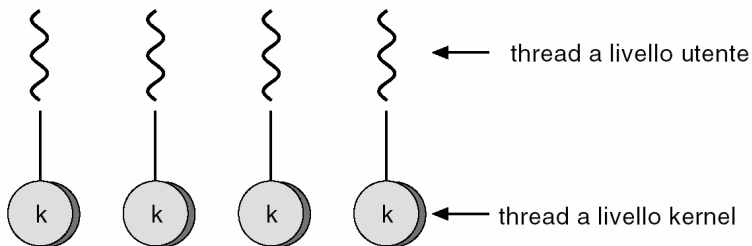
- Sistema operativo con supporto per soli processi
  - Simulazione di sistemi multi-thread a livello utente all'interno di un processo
- Sistema operativo con supporto per thread nel kernel
  - Esecuzione di sistemi multi-thread a livello utente, ciascuno mediante thread a livello di kernel
  - Esecuzione di sistemi multi-thread a livello utente, mediante simulazione di gruppi di thread utente ciascuno con un thread a livello di kernel

## Modello multi-a-uno



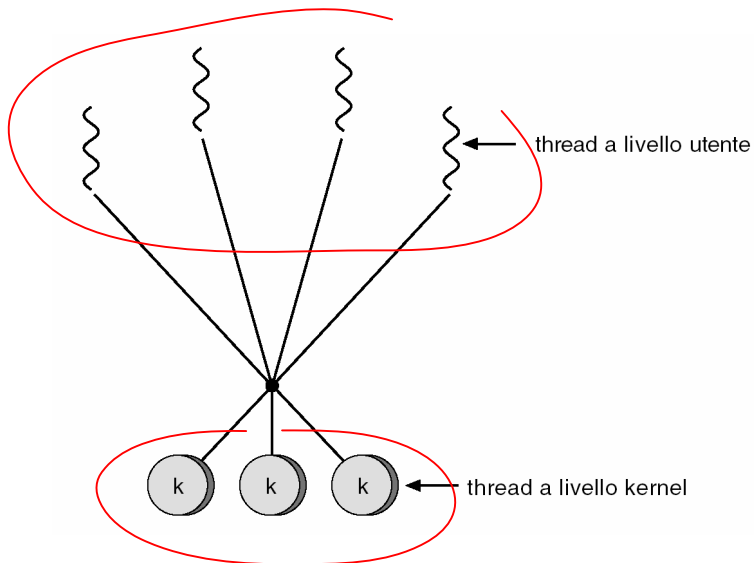
**Problema: serializzazione dei thread**

## Modello uno-a-uno

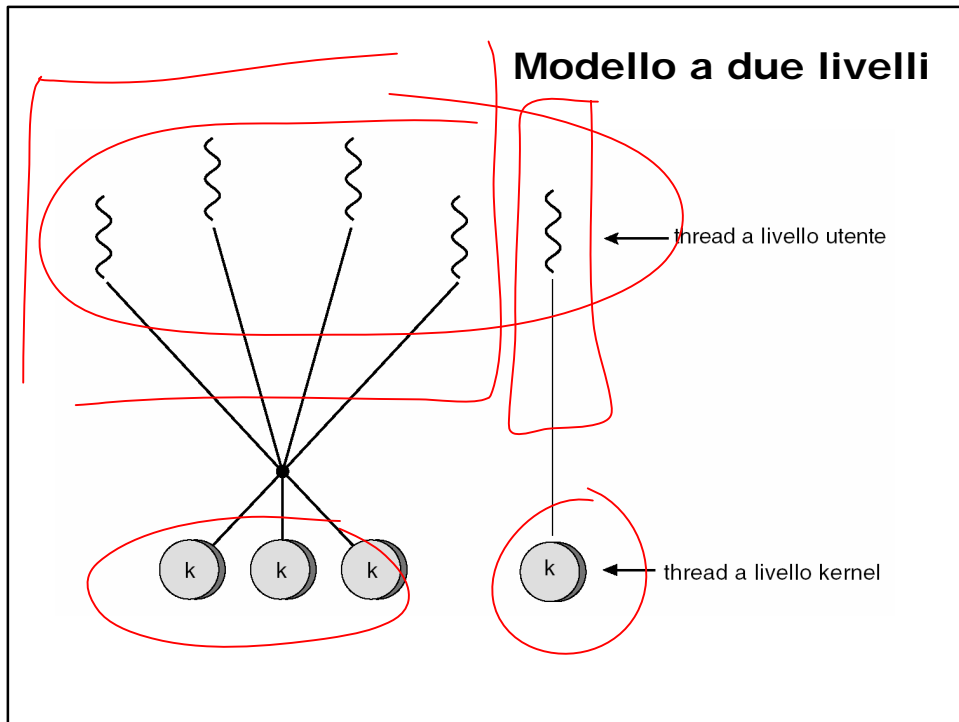


**Problema: efficienza e numero thread limitati**

## Modello multi-a-molti



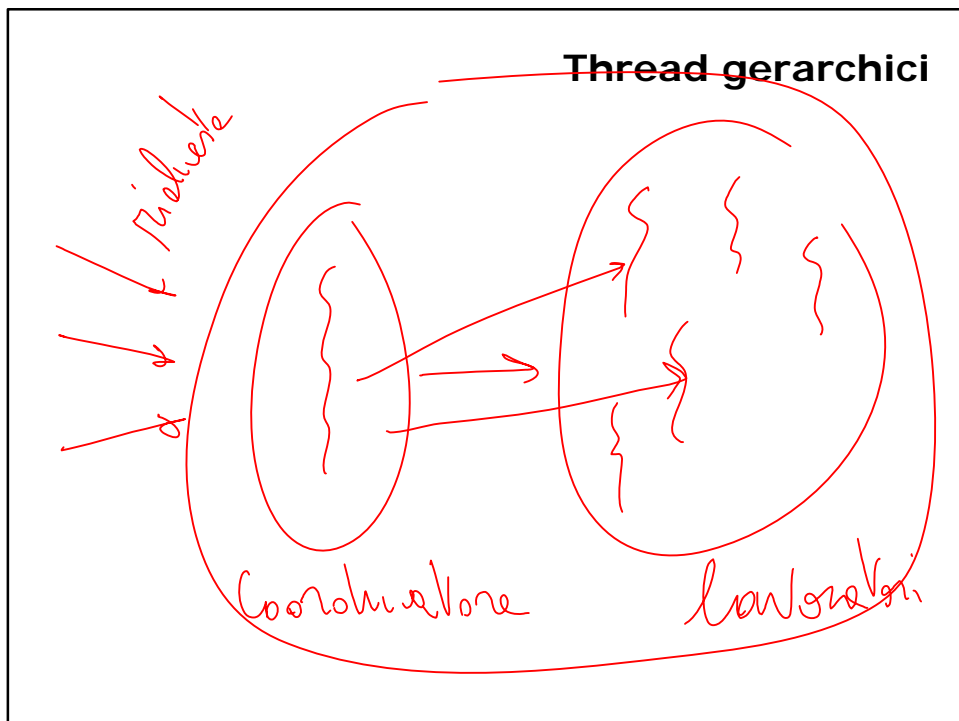
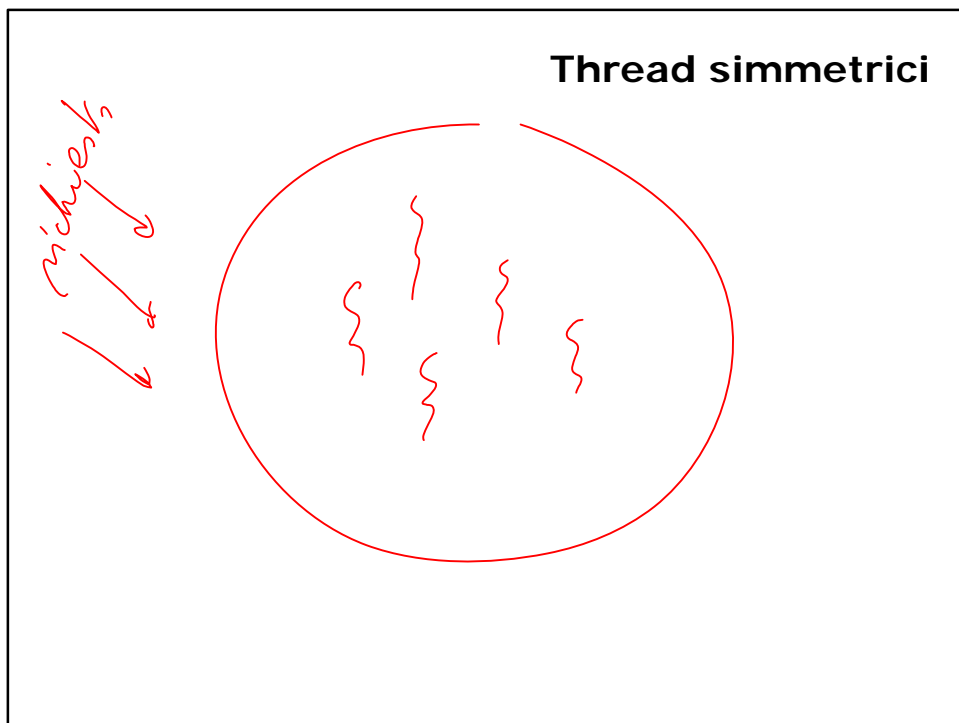


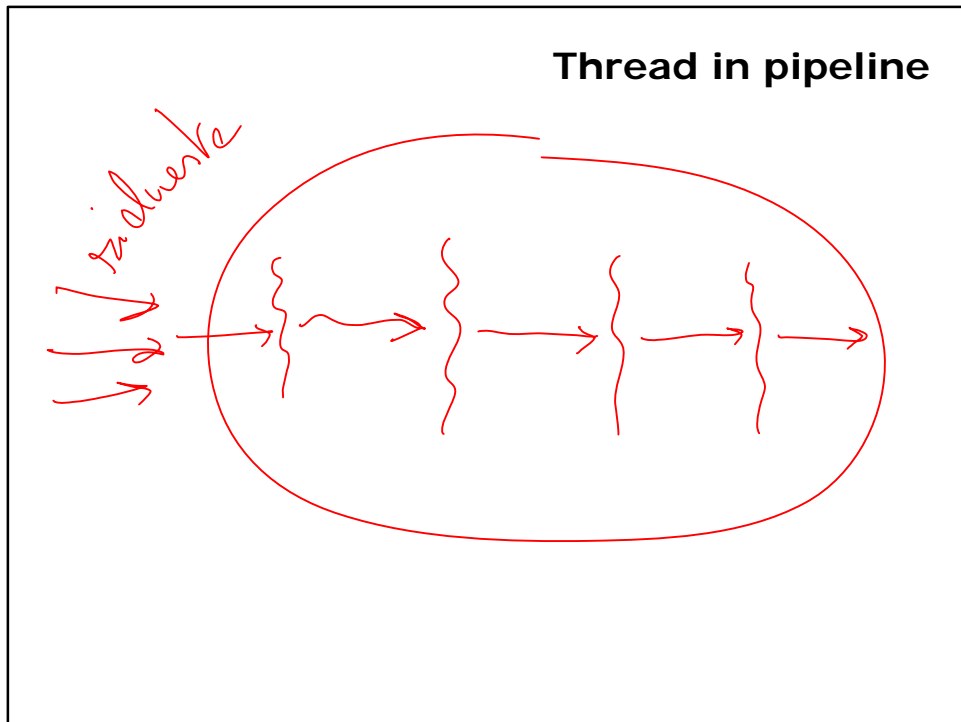


## Organizzazione della cooperazione

I thread possono cooperare organizzando le proprie interazioni secondo i modelli:

- thread simmetrici
- thread gerarchici
- thread in pipeline





### In sintesi

**La gestione dei thread a livello utente è realizzabile con thread a livello kernel con:**

- modello molti-a-uno
- modello uno-a-uno
- modello molti-a-molti
- modello a due livelli

**L'interazione tra thread è organizzabile con:**

- thread simmetrici
- thread gerarchici
- thread in pipeline

# **SISTEMI OPERATIVI**

Gestione del Processore  
Thread

## **Lezione 3 – Gestione dei thread**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

### **Sommario**

- Funzioni di gestione dei thread
  - Creazione
  - Esecuzione
  - Cancellazione (Terminazione)
  - Sincronizzazione e comunicazione
  - Processi leggeri

## Creazione di thread

### `fork()` all'interno di un thread

- duplicazione di tutti i thread del processo
- duplicazione del solo thread chiamante

## Esecuzione in thread

### `exec()` in un thread

- il programma chiamato rimpiazza l'intero processo

## **Cancellazione dei thread**

**Eliminazione del thread target prima che abbia completato le sue attività**

### **Modalità:**

- cancellazione asincrona
  - terminazione immediata
- cancellazione differita
  - periodicamente il thread target verifica se può terminare (punti di cancellazione)
  - terminazione ordinaria

## **Sincronizzazione e comunicazione**

**Procedure di sistema operano  
dal thread originante**

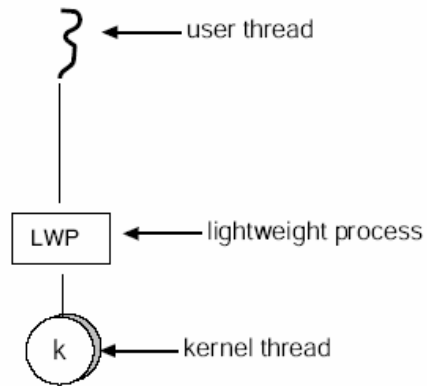
**verso i thread del processo destinatario:**

- tutti i thread del processo destinatario
- sottoinsieme di thread
- thread specifico

## Processi leggeri

LightWeight Process (LWP)

Processore virtuale  
per modello molti-a-molti  
e modello a due livelli



## In sintesi

### Funzioni di gestione dei thread:

- creazione
- esecuzione
- cancellazione
- sincronizzazione e comunicazioni
- processi leggeri

# **SISTEMI OPERATIVI**

Gestione del Processore  
Schedulazione del Processore

## **Lezione 1 – Schedulazione**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

### **Sommario**

- Obiettivo
- Livelli della schedulazione
  - a breve termine
  - a medio termine
  - a lungo termine
- Attivazione
  - non pre-emptive
  - pre-emptive



## Obiettivo

### Gestione della turnazione dei processi sul processore

*Definizione delle politiche  
di ordinamento dei processi  
per la turnazione sul processore*

## Livelli di schedulazione

- A **breve** termine
- A **medio** termine
- A **lungo** termine

## **Schedulazione a breve termine (1)**

### **short-term scheduler**

CPU scheduler

- Ordina i processi che sono
  - già presenti in memoria centrale
  - nello stato di pronto all'esecuzione

Il processo posto in prima posizione dall'ordinamento è quello che il dispatcher metterà in esecuzione quando avverrà il cambiamento di contesto successivo

## **Schedulazione a breve termine (2)**

- Eseguito frequentemente per garantire una turnazione rapida dei processi
  - Tipicamente almeno una volta ogni 100 millisecondi
- Deve essere veloce per minimizzare il sovraccarico di gestione
- Algoritmi usualmente semplici

## **Schedulazione a lungo termine (1)**

### **long-term scheduler**

job scheduler

- Ordina i processi attivati nel sistema identificando il gruppo di processi che devono essere
  - caricati in memoria centrale per l'esecuzione
  - posti nello stato di pronto all'esecuzione
- Costruisce il gruppo mescolando processi CPU-bound e I/O-bound in modo da massimizzare lo sfruttamento atteso del processore

## **Schedulazione a lungo termine (2)**

- Può essere assente o minimale
- Algoritmi usualmente complessi
- Deve essere eseguito poco frequentemente per non sovraccaricare il sistema
  - Tipicamente una volta ogni qualche minuto

## **Schedulazione a medio termine (1)**

### **medium-term scheduler**

#### Problemi:

- prestazioni del sistema non ottimali a causa dell'alta concorrenza per l'uso del processore
- sfruttamento del processore non ottimale a causa di una distribuzione sbilanciata tra processi CPU-bound e I/O-bound
- memoria centrale esaurita a causa delle dimensioni dei processi caricati e della creazione dinamica di variabili
- scarsa memoria centrale disponibile a ciascun processo con conseguente sovraccarico di gestione della memoria centrale

## **Schedulazione a medio termine (2)**

#### Obiettivi:

- ridurre la concorrenza tra i processi
- ottimizzare la distribuzione dei processi tra CPU-bound e I/O-bound
- dare maggior memoria centrale ai processi caricati in essa

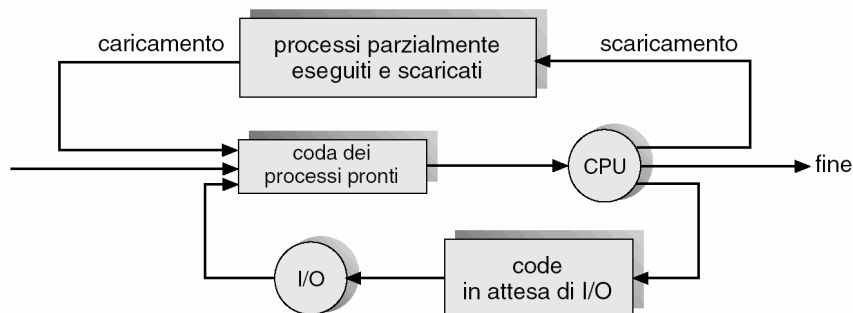
#### Soluzione:

- modificare dinamicamente il gruppo di processi caricati in memoria centrale e posti nello stato di pronto all'esecuzione dallo schedulatore a lungo termine al fine di adattare il gruppo di processi caricato in memoria centrale all'effettivo carico di lavoro

### Schedulatore a medio termine (3)

- Ordina i processi che erano stati selezionati dallo schedulatore a lungo termine per il caricamento in memoria centrale e l'ammissione allo stato di pronto all'esecuzione
  - ponendo realmente in memoria centrale solo alcuni processi pronti all'esecuzione tra questi in modo da garantire un bilanciamento ottimale effettivo tra CPU-bound e I/O-bound e un sovraccarico minimo per la gestione della memoria centrale
  - lasciando in un'area di memoria di massa temporanea gli altri processi del gruppo selezionato dallo schedulatore a lungo termine

### Schedulatore a medio termine (4)



→ Rimuove processi dalla memoria centrale: **swapping out**

→ Reintroduce in memoria centrale i processi: **swapping in**

## Attivazione (1)

- Schedulazione senza rilascio anticipato (pre-rilascio, rilascio forzato)  
**non pre-emptive scheduling**
  - Il processo in esecuzione viene cambiato:
    - dopo aver richiesto una operazione di I/O
    - dopo aver creato un processo e ne attende la terminazione
    - quando rilascia volontariamente il processore
    - quando termina
  - Sincrona con l'evoluzione della computazione

## Attivazione (2)

- Schedulazione con rilascio anticipato  
**pre-emptive scheduling**
  - Sistemi time-sharing
  - Il processo in esecuzione viene cambiato:
    - quando scade il quanto di tempo concesso per l'esecuzione
  - Asincrona con l'evoluzione della computazione

## **In sintesi**

La schedulazione del processore  
realizza la turnazione dei processi sul processore  
in modo da

- massimizzarne lo sfruttamento
- creare l'illusione di evoluzione  
contemporanea dei processi  
in sistemi time-sharing

## **SISTEMI OPERATIVI**

Gestione del Processore  
Schedulazione del Processore

### **Lezione 2 – Criteri di valutazione**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Criteri di schedulazione
- Obiettivi di ottimizzazione
- Metodi di valutazione



## **Criteri di schedulazione**

- Utilizzo del processore
- Capacità o frequenza di completamento (**throughput**)
- Tempo di completamento (**turnaround time**)
- Tempo d'attesa
- Tempo di risposta

## **Obiettivi di ottimizzazione nella schedulazione**

- Massimizzare l'utilizzo del processore
- Massimizzare il throughput
- Minimizzare il tempo di completamento
- Minimizzare il tempo d'attesa
- Minimizzare il tempo di risposta
- Minimizzare la varianza dei parametri caratteristici per garantire predicibilità del comportamento del sistema

## **Metodi di valutazione**

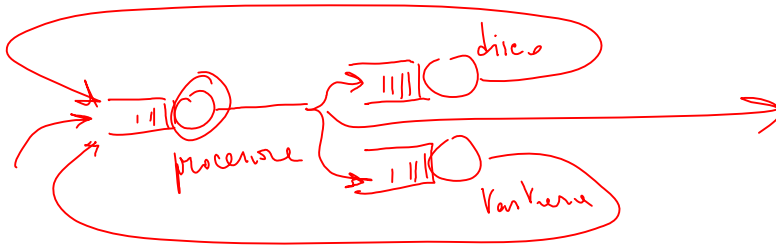
- Valutazione analitica
  - modellazione deterministica
- Valutazione statistica
  - modelli a reti di code
  - simulazione
- Implementazione

## **Modellazione deterministica**

- Modellazione delle prestazioni mediante una formula analitica deterministica
- Caratteristiche:
  - semplice
  - veloce
  - precisa
- Valuta un particolare carico di lavoro predeterminato
  - richiede dati esatti
  - i risultati non si possono generalizzare

## Modelli a reti di code (1)

- Il sistema è descritto come un insieme di servizi
- Ogni servizio ha un insieme di processi in attesa di ottenere il servizio
- Durante la sua vita, un processo richiede una sequenza di servizi
- Il sistema è rappresentato da una rete di code
- Ogni coda rappresenta un servizio
- Transizioni tra code rappresentano flussi di richieste



## Modelli a reti di code (2)

### Analisi delle reti di code

- Specificare la topologia del grafo della rete
- Specificare le caratteristiche di ogni servizio:
  - Frequenza d'arrivo delle richieste
  - Tempo di servizio
- Analisi della rete basata sulla Teoria delle Code
- Risultato dell'analisi statistica:
  - utilizzo della risorsa
  - lunghezza media della coda
  - tempo medio d'attesa

**Alcune semplificazioni sono necessarie!**

## **Simulazione**

- Realizzazione software del modello del sistema hw/sw, incluso schedulatore e processi applicativi
- Identificazione di insiemi di dati significativi per i processi
- Esecuzione dei processi nel simulatore
- Misura delle caratteristiche della schedulazione

## **Implementazione**

- Realizzazione effettiva del sistema hardware e software, con ingressi reali
- Misura delle caratteristiche della schedulazione nel sistema reale
- Soluzione molto onerosa
- Richiede la cooperazione degli utenti
- Scelta dell'algoritmo di scheduling in base alle esigenze e alle caratteristiche reali
- Raccolta automatica delle caratteristiche del carico di lavoro
- Adattabilità dinamica della schedulazione

## **In sintesi**

- Gli algoritmi di schedulazione hanno effetti differenti sul comportamento del sistema
- Diversi indicatori sono per valutare la bontà dell'algoritmo di schedulazione
- Le tecniche di valutazione degli algoritmi comprendono:
  - metodi analitici
  - metodi statistici
  - implementazione

## **SISTEMI OPERATIVI**

Gestione del Processore  
Schedulazione del Processore

### **Lezione 3 – Politiche di schedulazione**

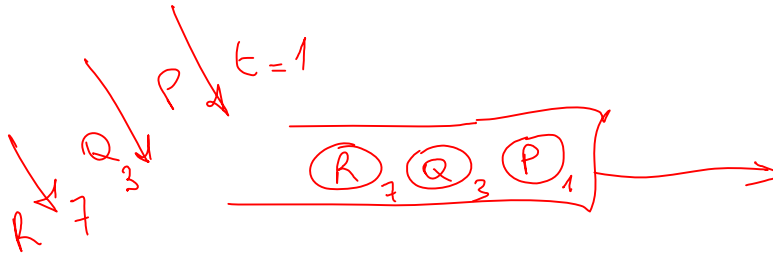
**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Politiche e algoritmi di schedulazione
  - First Come, First Served
  - Shortest Job First
  - Priorità
  - Round Robin
  - Coda a più livelli
  - Coda a più livelli con retroazione
- Schedulazione in sistemi multiprocessore

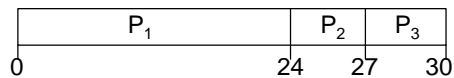
**First-Come, First-Served: FCFS (1)****Primo Arrivato, Primo Servito**

Non pre-emptive

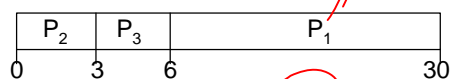
**First-Come, First-Served: FCFS (2)**

- Processi CPU-bound possono monopolizzare processore
- Tempo di attesa può essere alto

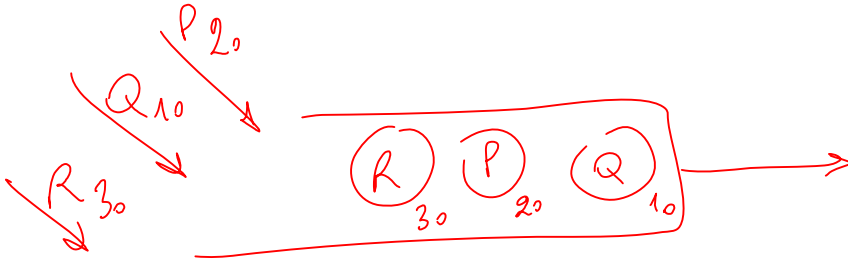
<u>Processo</u>	$P_1$	$P_2$	$P_3$
<u>Tempo di elaborazione</u>	24	3	3

Ordine di arrivo:  $P_1, P_2, P_3$ 

Tempo di attesa medio: 17

Ordine di arrivo:  $P_2, P_3, P_1$ 

Tempo di attesa medio: 3

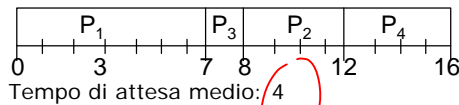
**Shortest-Job-First: SJF (1)****Processo Più Breve Prima**

- **Pre-emptive:** processo che diventa pronto interrompe processo in esecuzione richiedendo schedulazione
- **Non pre-emptive:** processo che diventa pronto non interrompe processo in esecuzione richiedendo schedulazione

**Shortest-Job-First: SJF (2)**

Processo	$P_1$	$P_2$	$P_3$	$P_4$
Tempo di arrivo	0	2	4	5
Tempo di elaborazione	7	4	1	4

- Non pre-emptive



- Pre-emptive

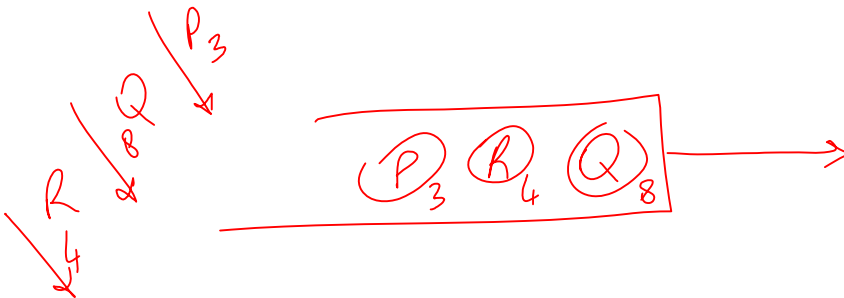




## Shortest-Job-First: SJF (3)

- Garantisce il tempo minimo di attesa, se si conosce il tempo di processore richiesto dai processi
- Predizione del tempo di processore richiesto da un processo:
  - simile ai tempi precedenti del processo
  - media esponenziale  $t_{n+1} = a t_n + (1-a)t_n$

## Priorità (1)



- Rappresentazione della priorità
  - Logica diretta: Indice di priorità alto → Priorità alta
  - Logica inversa: Indice di priorità basso → Priorità alta

## Priorità (2)

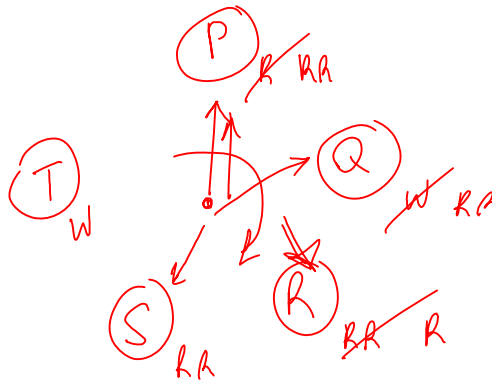
- **Pre-emptive:** processo che diventa pronto interrompe processo in esecuzione richiedendo schedulazione
- **Non pre-emptive:** processo che diventa pronto non interrompe processo in esecuzione richiedendo schedulazione

## Priorità (3)

- Problema:  
processi a bassa priorità potrebbero subire un blocco indefinito (*starvation*)
- Soluzione:  
progressivo invecchiamento della priorità (*aging*)  
con periodico ripristino al valore iniziale oppure ringiovanimento fino al valore iniziale

## Round-Robin RR (1)

### Rotazione



- Tipico dei sistemi time sharing
- Simile a FCFS con aggiunta di pre-emption

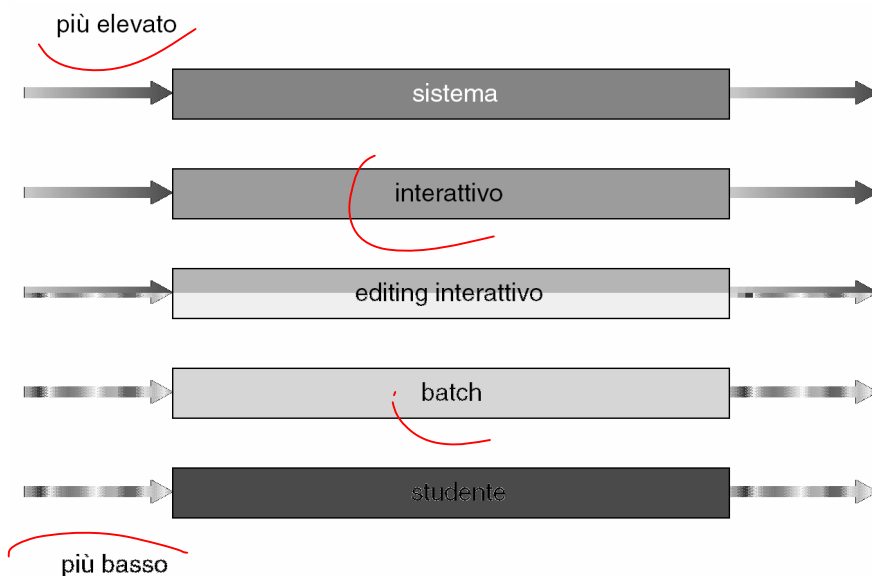
## Round-Robin RR (2)

- Distribuzione uniforme del tempo di elaborazione tra i processi pronti
- Velocità di esecuzione dei processi dipende dal numero di processi pronti
- Turnaround dipende dalla durata del quanto di tempo
- Comportamento dell'algoritmo RR dipende dalla durata del quanto di tempo:
  - molto lungo: RR  $\rightarrow$  FCFS
  - molto breve: RR  $\rightarrow$  condivisione del processore  
ognuno degli  $N$  processi sembra vedere un processore con  $1/N$  di capacità computazionale  
problema: sovraccarico di gestione dovuto ai frequenti cambiamenti di contesto
  - ideale empirico: 80% delle richieste di elaborazione deve essere completata in un quanto di tempo

## Coda a più livelli C+L (1)

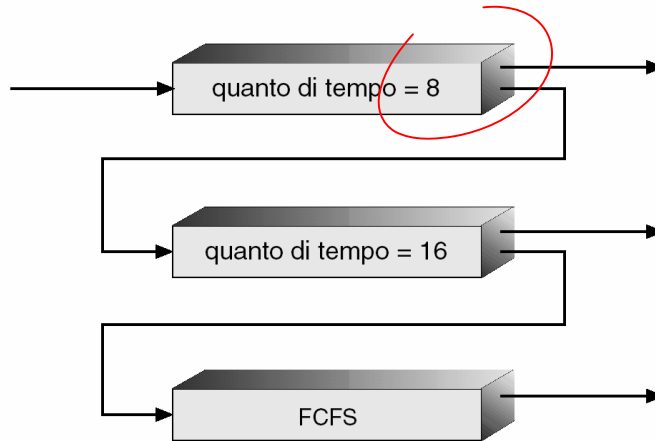
- I processi sono raggruppati per tipologie omogenee
- Ogni tipologia è assegnata in modo permanente a un livello della coda di schedulazione, rappresentato da una coda di attesa specifica
- Ogni coda di attesa ha un suo algoritmo di schedulazione
- L'insieme delle code di attesa viene schedulato da un algoritmo dedicato
  - Usualmente schedulazione pre-emptive a priorità fisse

## Coda a più livelli C+L (2)



## Coda a più livelli con retroazione C+LR (1)

- Coda a più livelli che permette ai processi di migrare da un livello a quello adiacente



## Coda a più livelli con retroazione C+LR (2)

- Code di attesa separate in funzione dell'uso dinamico del processore da parte dei processi
- Algoritmo di schedulazione specifico per ciascuna coda di attesa
- Politica di promozione
- Politica di degradazione
- Politica di allocazione

## **Schedulazione in sistemi multiprocessore (1)**

**La schedulazione deve considerare le caratteristiche dell'architettura del sistema di elaborazione:**

- processori
  - omogenei ✓
  - eterogenei ✓
- memoria
  - solo condivisa
  - anche locale
- periferiche
  - accessibili da singolo processore
  - accessibili da tutti i processori

## **Schedulazione in sistemi multiprocessore (2)**

- Sistemi con processori omogenei, memoria solo condivisa e periferiche accessibili da tutti i processori:
  - Coda unica
  - Una coda per processore in memoria condivisa (suddivisione del carico - load sharing)
- Sistemi con processori omogenei, memoria anche locale e periferiche accessibili da tutti i processori:
  - Coda unica
  - Una coda per processore in memoria condivisa o in memoria locale (allocazione dei processi ai processori e load sharing)

### **Schedulazione in sistemi multiprocessore (3)**

- Sistemi con processori omogenei, memoria solo condivisa e periferiche accessibili da alcuni processori:
  - Una coda per processori omogenei in memoria condivisa, una coda per ogni processore che gestisce una specifica periferica
- Sistemi con processori omogenei, memoria anche locale e periferiche accessibili da alcuni processori:
  - Una coda per processori omogenei in memoria condivisa, una coda per ogni processore che gestisce una specifica periferica
  - Una coda per ognuno dei processori omogenei in memoria locale, una coda per ogni processore che gestisce una specifica periferica
- Sistemi con processori eterogenei:
  - Una coda per processore
  - Una coda per gruppi di processori omogenei

### **Schedulazione in sistemi multiprocessore (4)**

#### **Tipi di multiprocessamento:**

- **Multiprocessamento asimmetrico:**
  - Processore master esegue sistema operativo (schedulazione di tutti i processi per tutti i processori)
  - Processori slave eseguono solo processi applicativi
- **Multiprocessamento simmetrico:**
  - Ogni processore esegue il sistema operativo (schedulazione dei processi assegnati al processore) e i processi applicativi

## **In sintesi**

- Politiche e algoritmi di schedulazione
  - First Come, First Served
  - Shortest Job First
  - Priorità
  - Round Robin
  - Coda a più livelli
  - Coda a più livelli con retroazione
- Schedulazione in sistemi multiprocessore



## **SISTEMI OPERATIVI**

Gestione del Processore  
Schedulazione del Processore

### **Lezione 4 – Schedulazione per sistemi in tempo reale**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Sistemi in tempo reale
  - stretto
  - lasco
- Schedulazione con tempo di completamento garantito
- Schedulazione di processi periodici
- Schedulazione a frequenza monotona
- Schedulazione a scadenza più urgente

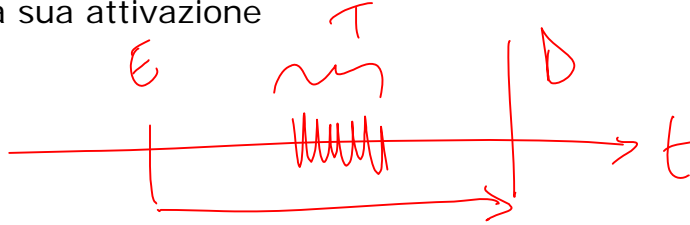
## Sistemi in tempo reale stretto

### Hard Real-Time System

Per la correttezza dell'applicazione

è obbligatorio

che un processo termini la sua computazione  
entro un tempo massimo garantito  
dalla sua attivazione



### Schedulazione in sistemi in tempo reale stretto

- Schedulazione con tempo massimo di completamento dei processi garantito
- Schedulazione di processi periodici
- Schedulazione a frequenza monotona
- Schedulazione a scadenza più urgente

## Tempo massimo di completamento garantito

Lo schedulatore può:

- accettare il processo garantendone il completamento entro il tempo massimo consentito
- rifiutare il processo

L'accettazione del processo è basata su:

- stima del tempo di completamento del processo
- prenotazione delle risorse necessarie al processo

Politica di schedulazione tradizionale

Problema: predicibilità del tempo di completamento

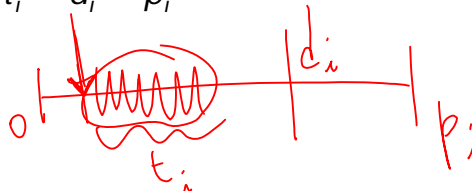
## Processi periodici (1)

### Processi eseguiti periodicamente

#### Caratteristiche del processo $P_i$ :

- tempo (fisso) di elaborazione  $t_i$
- scadenza  $d_i$
- periodo  $p_i$

$$0 = t_i = d_i = p_i$$



## Processi periodici (2)

### Politica di schedulazione:

- Round robin
- Priorità assegnata in base a scadenza  $d_i$  o frequenza  $1/p_i$

### Controllo dell'ammissione

- Verifica della possibilità di completamento entro la scadenza dichiarata, con la politica di schedulazione adottata

## Schedulazione a frequenza monotona

### Algoritmo per processi periodici con priorità e pre-emption

**Tempo di elaborazione omogeneo per ogni iterazione del processo  $P_i$**

### Priorità:

- statiche
- proporzionale alla frequenza  $1/p_i$

### Pre-emption:

- se un processo a più bassa priorità è in esecuzione e un processo a più alta priorità diventa pronto, il primo processo viene sospeso

## **Schedulazione a scadenza più urgente**

### **Earliest-Deadline First - EDF**

#### **Computazione dei processi:**

- Processi periodici e non-periodici
- Processi con tempo di elaborazione variabile

#### **Priorità:**

- inversamente proporzionale alla scadenza  $d_i$
- dinamica, in funzione dei processi che diventano pronti

## **Sistemi in tempo reale lasco <sup>(1)</sup>**

### **Soft Real-Time System**

#### **Solo i processi critici diventano prioritari**

#### **Schedulazione a priorità**

- Processi critici
- Processi non critici

## **Sistemi in tempo reale lasco (2)**

### **Priorità:**

- statica per processi critici
- eventualmente dinamica per processi non critici

### **Bassa latenza di dispatch:**

- Interrompibilità delle chiamate di sistema lunghe (pre-emption point)
- Kernel interrompibile

## **In sintesi**

**Nei sistemi in tempo reale le operazioni critiche devono essere eseguite entro un tempo massimo stabilito**

### **I sistemi in tempo reale possono essere:**

- hard real-time
- soft real-time

### **Algoritmi di schedulazione per sistemi in tempo reale:**

- Schedulazione con tempo di completamento garantito
- Schedulazione di processi periodici
- Schedulazione a frequenza monotona
- Schedulazione a scadenza più urgente

## **SISTEMI OPERATIVI**

Gestione del Processore  
Schedulazione del Processore

### **Lezione 5 – Schedulazione dei thread**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

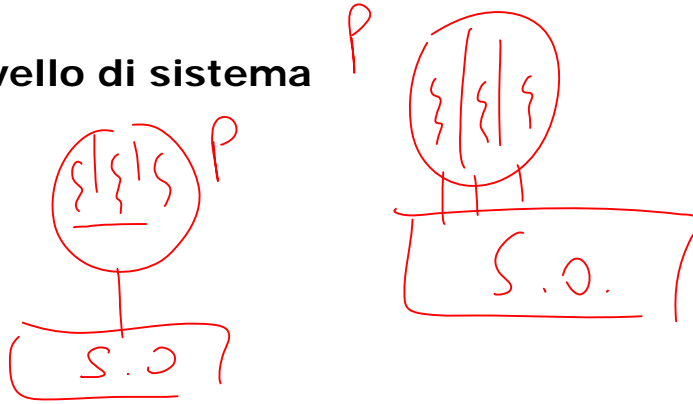
#### **Sommario**

- Livelli di schedulazione
  - Schedulazione nel processo
  - Schedulazione nel sistema operativo
- Caratteristiche

## Livelli di schedulazione

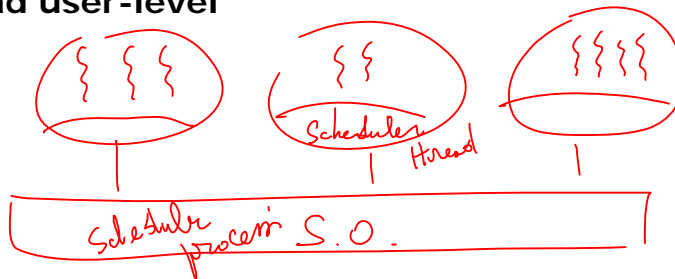
### Livello di processo

### Livello di sistema



## Schedulazione a livello di processo

### Thread user-level



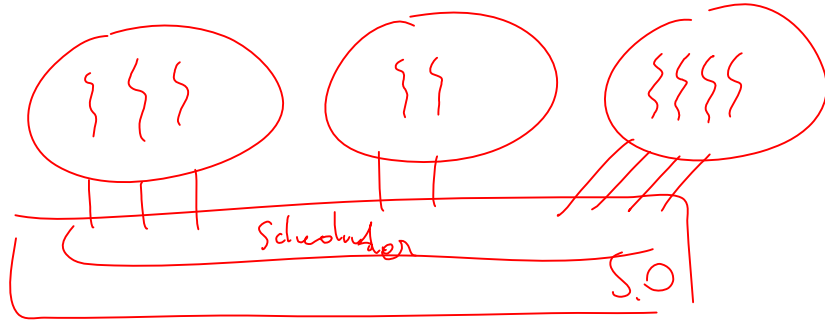
- Process-Contention Scope (PCS)
- gestita dallo schedulatore nella libreria dei thread
- schedulazione tipica:
  - a priorità fisse o modificabili dal programmatore, con pre-emption
  - FCFS
  - round robin



## Schedulazione a livello di sistema

**Thread kernel-level**

**Thread user-level mappati su thread kernel-level (eventualmente con LWP)**



- System-Contention Scope (SCS)
- gestita dallo schedulatore del sistema operativo

## In sintesi

- Livelli di schedulazione
  - nel processo
  - nel kernel

## **SISTEMI OPERATIVI**

Gestione del Processore  
Comunicazione tra Processi

### **Lezione 1 – Processi cooperanti**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Coordinamento
- Cooperazione
- Concetto di cooperazione tra processi o thread
- Vantaggi della cooperazione

## **Coordinamento**

- Sincronizzazione della computazione dei processi
  - per l'accesso a risorse condivise
  - per garantire una evoluzione congiunta diretta a raggiungere uno scopo applicativo comune

## **Cooperazione**

- Lavoro congiunto dei processi per il raggiungimento di scopi applicativi comuni con condivisione e scambio di informazioni

## Processi indipendenti

- Non hanno scopi comuni con altri processi
- Non influenzano e non sono influenzati da altri processi
- Non hanno informazioni condivise
  
- Competono per l'uso del processore ed, eventualmente, di periferiche condivise
  
- Coordinamento della computazione  
→ Sincronizzazione per l'uso di risorse condivise

## Processi cooperanti

- Hanno uno scopo applicativo comune
- Possono condividere informazioni
- Possono influenzare o essere influenzati da altri processi
  
- Scambio di informazioni  
→ Comunicazione
- Coordinamento della computazione  
→ Sincronizzazione

## **Vantaggi della cooperazione**

- Modularità
- Parallelizzazione
- Scalabilità
- Specializzazione
- Qualità del progetto e della realizzazione

## **Esempi**

### **Processi cooperanti**

- Produttore - Consumatore
- Client - Server
- Compilatore - Assemblatore - Loader

## **In sintesi**

- Abbiamo visto:
  - Concetto di coordinamento
  - Concetto di cooperazione
  - Cosa sono i processi cooperanti
  - Quali vantaggi offrono

## **SISTEMI OPERATIVI**

Gestione del Processore  
Comunicazione tra Processi

### **Lezione 2 – Comunicazione tra processi**

**Vincenzo Piuri**

---

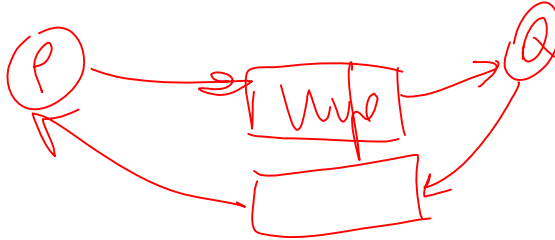
Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Concetto di comunicazione
- Necessità
- Entità coinvolte nella comunicazione
- Caratteristiche
- Implementazione

## Comunicazione

Meccanismi e politiche  
che permettono ai processi  
di scambiarsi informazioni  
per operare in modo cooperativo



## Necessità

- Trasferimento di informazioni da processo mittente a ricevente
- Condivisione di informazioni



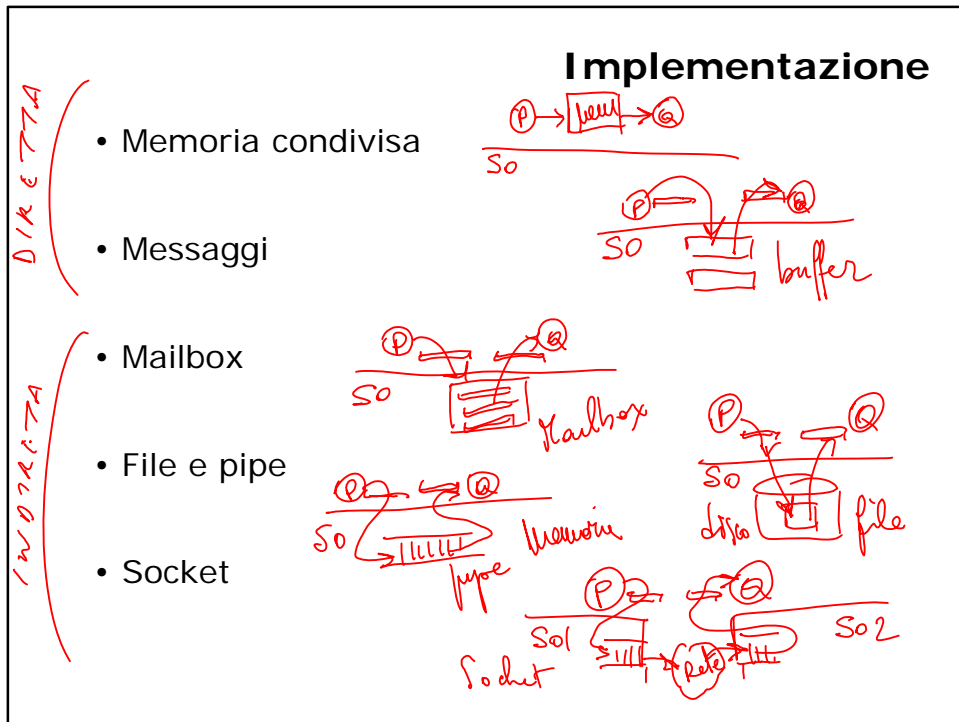
## Entità coinvolte nella comunicazione

- Processo produttore dell'informazione
- Processo utilizzatore dell'informazione
- Canale di comunicazione



## Caratteristiche

- Quantità di informazioni da trasmettere
- Velocità di esecuzione
- Scalabilità
- Semplicità di uso nelle applicazioni
- Omogeneità delle comunicazioni
- Integrazione nel linguaggio di programmazione
- Affidabilità
- Sicurezza
- Protezione



## In sintesi

- Abbiamo visto:
  - Cos'è la comunicazione
  - Quali caratteristiche deve avere
  - Come può essere implementata

## SISTEMI OPERATIVI

Gestione del Processore  
Comunicazione tra Processi

### Lezione 3 – Comunicazione con memoria condivisa

Vincenzo Piuri

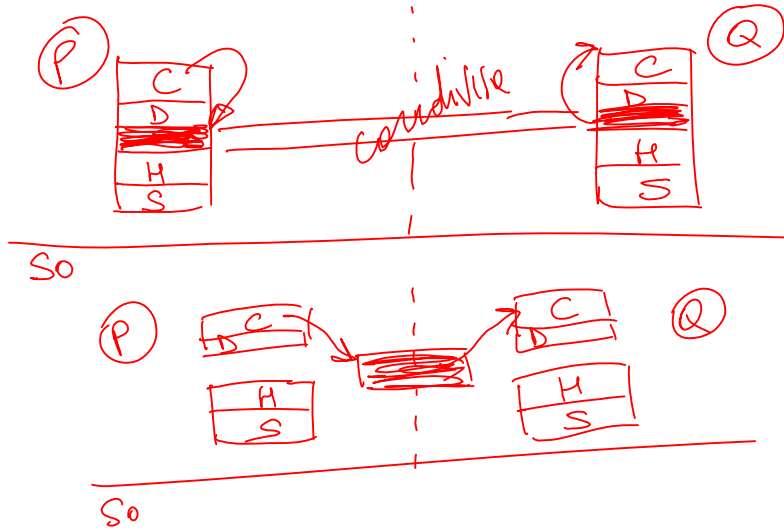
---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### Sommario

- Condivisione di parte dei dati  
in memoria centrale  
**variabili globali**
- Condivisione di un'area in memoria centrale  
per comunicazioni  
**buffer**
- Caratteristiche e problemi

## Condivisione di variabili globali

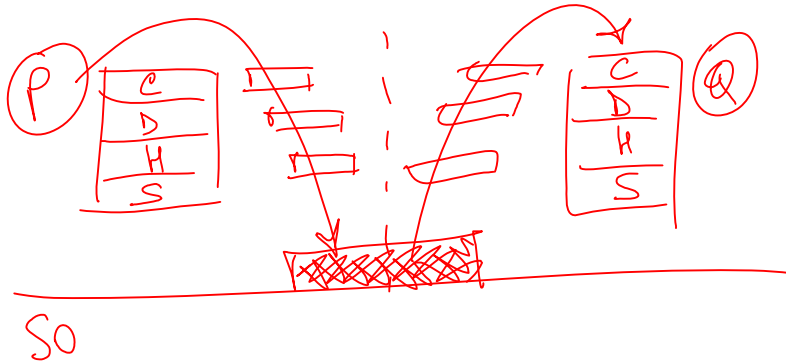


## Problemi della condivisione di variabili globali

- Identificazione dei processi comunicanti
  - Comunicazione diretta
- Consistenza delle operazioni
  - Lettura e scrittura sono incompatibili tra loro
- Sincronizzazione per l'accesso in mutua esclusione all'area dati comune ai due processi



### Condivisione di buffer (1)



### Condivisione di buffer (2)

- Problemi:
  - Identificazione dei processi comunicanti (comunicazione diretta)
  - Consistenza degli accessi
  - Sincronizzazione per accesso in mutua esclusione
- Realizzazioni:
  - Buffer con copiatura gestita dal sistema operativo
  - Buffer in memoria fisicamente condivisa

## Produttore – Consumatore (1)

- Struttura dati
  - Buffer di memoria condiviso con capacità:
    - illimitata
    - limitata
- Attività della comunicazione
  - Produttore genera informazioni elementari per Consumatore
  - Produttore memorizza ciascuna informazione elementare nel buffer di comunicazione condiviso
  - Consumatore acquisisce ciascuna informazione elementare prendendola dal buffer di comunicazione condiviso
  - Consumatore legge e usa ciascuna informazione elementare generata dal Produttore

## Produttore – Consumatore (2)

```
import java.util.*;
public class BoundedBuffer implements Buffer
{
    private static final int BUFFER_SIZE = 5;
    private int count; // numero di elementi nel buffer
    private int in; // punta alla successiva posizione libera
    private int out; // punta alla successiva posizione piena
    private Object[] buffer;
    public BoundedBuffer() {
        // il buffer è inizialmente vuoto
        count = 0;
        in = 0;
        out = 0;
        buffer = new Object[BUFFER_SIZE];
    }
    //i produttori chiamano questo metodo
    public void insert(Object item) {
        ...
    }
    // i consumatori chiamano questo metodo
    public Object remove() {
        ...
    }
}
```

## Produttore – Consumatore (3)

```
public void insert(Object item) {  
    while (count == BUFFER_SIZE)  
        ; // non fare nulla - non ci sono buffer liberi  
    // aggiungi un elemento al buffer  
    ++count;  
    buffer[in] = item;  
    in = (in + 1) % BUFFER_SIZE;  
}
```

## Produttore – Consumatore (4)

```
public Object remove() {  
    Object item;  
    while (count == 0);  
    // non fare nulla - nulla da consumare  
    // rimuovi un elemento dal buffer  
    --count;  
    item = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
    return item;  
}
```



## **In sintesi**

- Tecniche di comunicazione tramite memoria condivisa
  - Condivisione variabili globali
  - Condivisione buffer di comunicazione
- Caratteristiche e problemi

## **SISTEMI OPERATIVI**

Gestione del Processore  
Comunicazione tra Processi

### **Lezione 4 – Comunicazione con scambio di messaggi**

**Vincenzo Piuri**

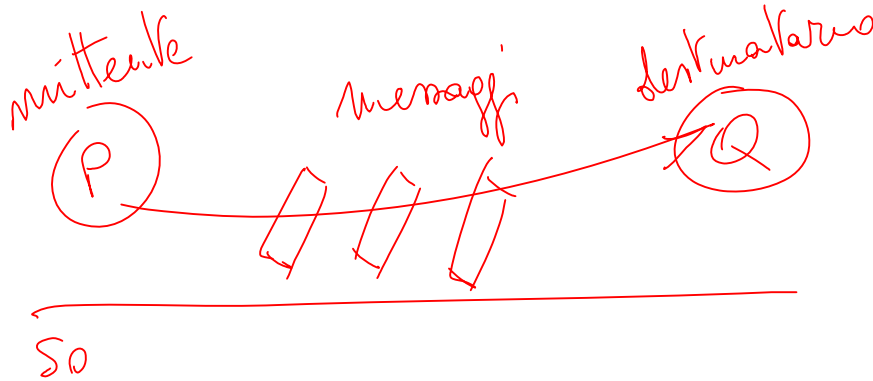
---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Modello della comunicazione a messaggi
- Caratteristiche dei messaggi
- Funzioni
- Sincronizzazione dei processi comunicanti
- Caratteristiche e problemi
- Implementazione

## Modello della comunicazione a messaggi



## Messaggi

### Contenuto

- Processo mittente
- Processo destinatario
- Informazioni da trasmettere
- Eventuali altre informazioni di gestione dello scambio messaggi

### Dimensione

- Fissa
- Variabile

## Buffer

### Assegnazione

- a ciascuna coppia di processi
- di uso generale

### Quantità di buffer assegnati

- illimitata
- limitata
- nulla

## Funzioni (1)


### Invio

 send(Q, messaggio)

- Deposita messaggio in un buffer libero
- Primitiva bloccante:  
blocca mittente fintanto che non ci sono  
buffer liberi per completare la comunicazione  
con il destinatario

## Funzioni (2)

### Ricezione

 `receive(P, messaggio)`

- Riceve messaggio da un buffer
- Primitiva bloccante:  
blocca destinatario fintanto che non c'è  
un messaggio ricevibile

## Funzioni (3)

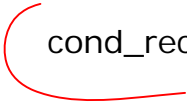
### Invio condizionale

 `cond_send(Q, messaggio): error_status`

- Deposita messaggio in un buffer libero
- Primitiva non bloccante:  
se non ci sono buffer liberi per effettuare  
la comunicazione con il destinatario,  
ritorna condizione di errore non bloccando  
il mittente e non depositando più il messaggio

## Funzioni (4)

### Ricezione condizionale

 `cond_receive(P, messaggio): error_status`

- Riceve messaggio da un buffer
- Primitiva non bloccante:  
se non ci sono messaggi ricevibili,  
ritorna condizione di errore non bloccando  
il destinatario e non ricevendo più il messaggio

## Sincronizzazione dei processi comunicanti

### Comunicazioni asincrone

- Bloccanti solo se l'operazione non può essere completata

### Comunicazioni sincrone

- La comunicazione avviene solo con la presenza contemporanea dei due processi
  - Il processo mittente aspetta sempre che il processo destinatario esegua la funzione di ricezione

## **Identificazione dei processi comunicanti**

### **Comunicazione simmetrica**

- Mittente e destinatario sono sempre univocamente identificati

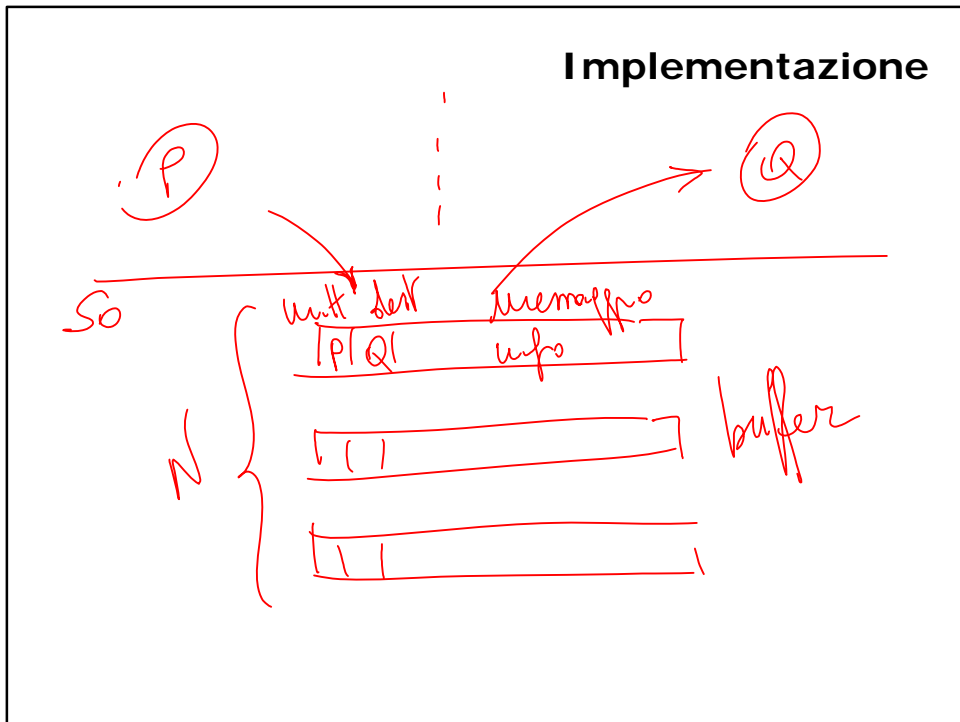
#### **Comunicazione diretta**

### **Comunicazione asimmetrica**

- Destinatario o mittente possono non essere identificati univocamente
- Ricezione di messaggi provenienti
  - da un processo di un gruppo specificato
  - da un processo qualunque
- Invio di messaggi
  - a un processo di un gruppo specificato
  - a un processo qualunque

## **Caratteristiche e problemi**

- Identificazione dei processi comunicanti
- Memoria non condivisa tra processi
- Sincronizzazione per l'accesso ai messaggi gestita implicitamente dal sistema operativo



## In sintesi

- Abbiamo visto:
  - comunicazione tramite messaggi
  - caratteristiche dei messaggi
  - funzioni di sistema operativo
  - caratteristiche e problemi
  - implementazione



## **SISTEMI OPERATIVI**

Gestione del Processore  
Comunicazione tra Processi

### **Lezione 5 – Comunicazione con mailbox**

**Vincenzo Piuri**

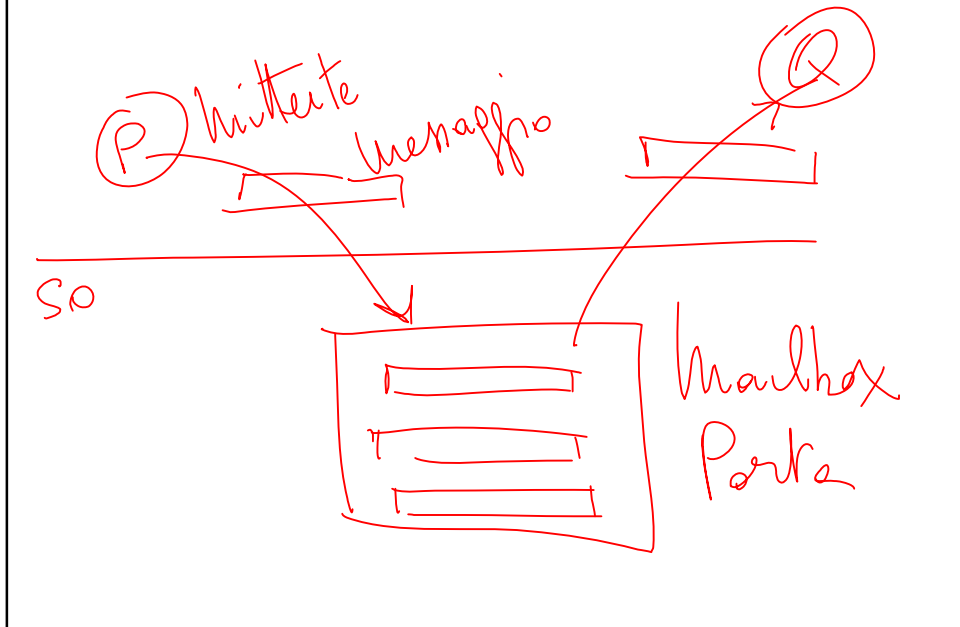
---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Modello della comunicazione a messaggi con mailbox
- Caratteristiche dei messaggi
- Funzioni
- Sincronizzazione dei processi comunicanti
- Caratteristiche e problemi
- Comunicazioni con molti possibili mittenti o riceventi

### Modello della comunicazione con mailbox



### Messaggi

#### Contenuto

- Processo mittente
- Mailbox destinataria
- Informazioni da trasmettere
- Eventuali altre informazioni a supporto della gestione dei messaggi nella mailbox

#### Dimensione

- Fissa
- Variabile

## Mailbox

### Capacità

- illimitata
  - un numero illimitato di messaggi può essere depositato
- limitata
  - un numero finito di messaggi può essere depositato
- nulla
  - nessun messaggio può essere depositato

## Funzioni <sup>(1)</sup>

### Creazione mailbox

create(M)

### Cancella mailbox

delete(M)

## Funzioni (2)

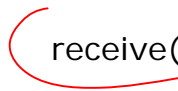
### Invio

 send(M, messaggio)

- Deposita messaggio nella mailbox
- Capacità
  - illimitata → non bloccante
  - limitata → bloccante se la mailbox è piena
  - nulla → bloccante se non c'è un processo in ricezione

## Funzioni (3)

### Ricezione

 receive(M, messaggio)

- Riceve messaggio dalla mailbox
- Bloccante se non c'è almeno un messaggio da ricevere

## **Funzioni (4)**

### **Invio condizionale**

`cond_send(M, messaggio): error_status`

- Deposita messaggio nella mailbox se la comunicazione può essere completata (in funzione della capacità della mailbox)
- Se l'invio bloccasse il mittente, ritorna condizione di errore non bloccando il mittente e non depositando più il messaggio

## **Funzioni (5)**

### **Ricezione condizionale**

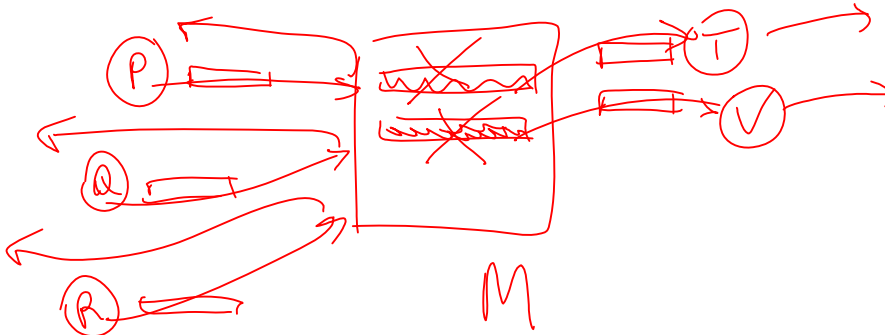
`cond_receive(M, messaggio): error_status`

- Riceve messaggio dalla mailbox se c'è almeno un messaggio
- Se non ci sono messaggi ricevibili, ritorna condizione di errore non bloccando il destinatario e non ricevendo più il messaggio

## Sincronizzazione

### Capacità della mailbox

- illimitata → comunicazione asincrona
- nulla → comunicazione sincrona
- limitata → comunicazione bufferizzata



## Caratteristiche e problemi

- Nessuna identificazione dei processi comunicanti  
**Comunicazione indiretta**
- Memoria non condivisa tra processi
- Sincronizzazione per l'accesso ai messaggi gestita implicitamente dal sistema operativo

## **Ordinamento delle code dei messaggi e dei processi in attesa**

Politiche di ordinamento delle code dei messaggi  
nella mailbox e dei processi in attesa

- First In, First Out
- Priorità
- Scadenza

## **Proprietà della mailbox**

### **Sistema operativo**

- Non è correlata a un processo

### **Processo**

- Solo il processo proprietario riceve da questa mailbox
- Altri processi possono solo inviare
- Se termina il processo proprietario scompare la mailbox

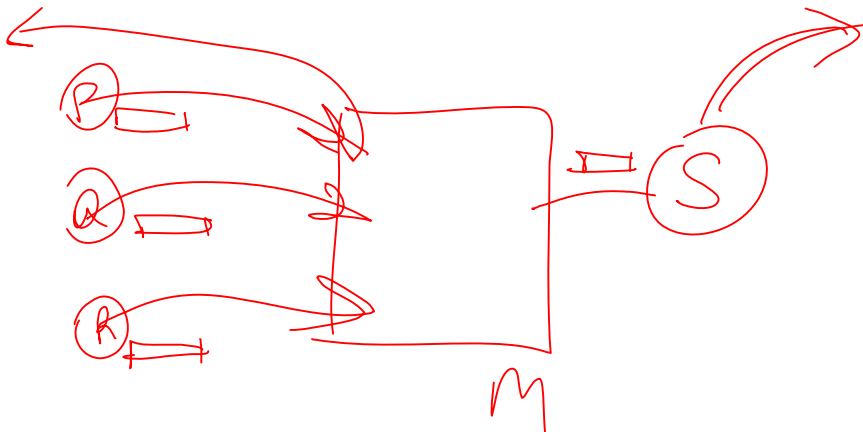
## Comunicazioni con molti possibili mittenti o riceventi

- Comunicazioni da molti mittenti a un ricevente
- Comunicazioni da un mittente a molti possibili riceventi
- Comunicazioni da molti mittenti a molti possibili riceventi

Ogni comunicazione coinvolge comunque sempre solo due processi (un mittente e un ricevente)

## Comunicazioni molti a uno

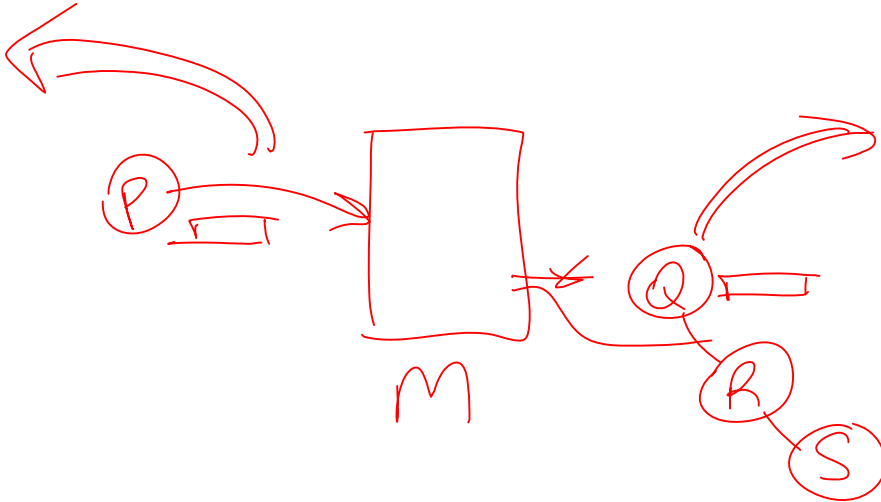
- Un processo di servizio della coda
- Più processi client richiedenti il servizio





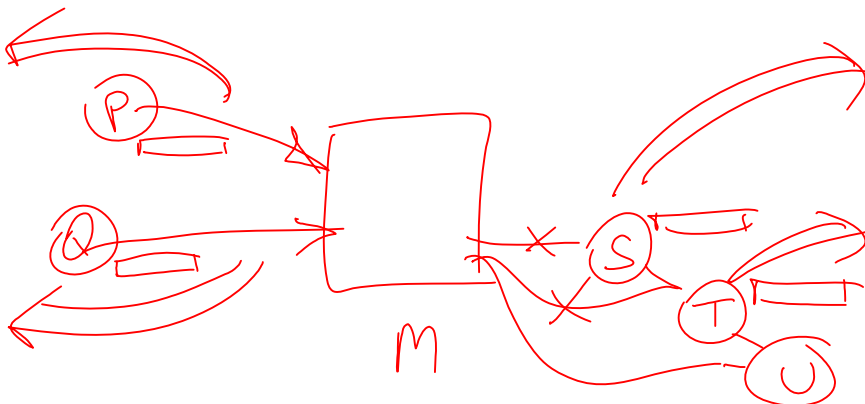
## Comunicazioni uno a molti

- Processi di servizio multipli



## Comunicazioni molti a molti

- Più processi di servizio della coda
- Più processi client richiedenti il servizio



## In sintesi

- Abbiamo visto:
  - comunicazione tramite messaggi scambiati a mailbox
  - caratteristiche dei messaggi
  - funzioni di sistema operativo
  - caratteristiche e problemi
  - comunicazioni con molti possibili mittenti o riceventi

## **SISTEMI OPERATIVI**

Gestione del Processore  
Comunicazione tra Processi

# **Lezione 6 – Comunicazione con file**

**Vincenzo Piuri**

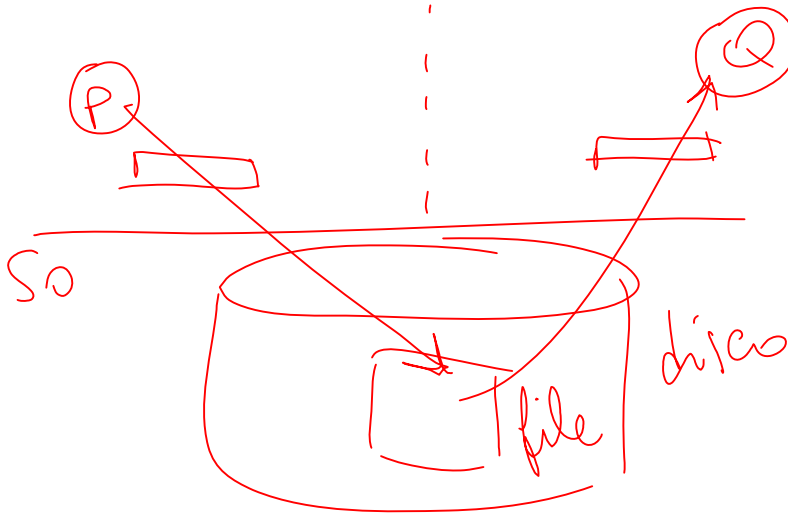
---

Università degli Studi di Milano - SSRI - CDL ONLINE

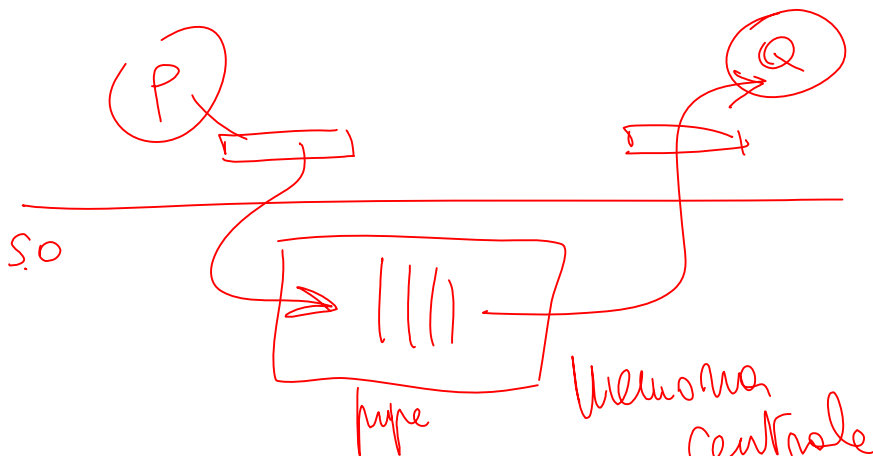
## **Sommario**

- Modello della comunicazione mediante file condivisi
- Modello della comunicazione mediante pipe
- Caratteristiche dei messaggi
- Funzioni
- Caratteristiche e problemi

## Comunicazione mediante file condivisi



## Comunicazione mediante pipe



- Comunicazione tra due processi
- File in memoria centrale con scrittura solo in aggiunta e lettura unica solo sequenziale

## **Caratteristiche dei messaggi**

### **Contenuto**

- Processo mittente
- Informazioni da trasmettere
- Eventuali altre informazioni a supporto della gestione dei messaggi

### **Dimensione**

- Fissa
- Variabile

## **Funzioni**

**Creazione**

**Cancellazione**

**Lettura**

**Scrittura**

## **Caratteristiche e problemi**

- Sincronizzazione dei processi in lettura e scrittura effettuata secondo politiche e meccanismi del file system
- Ordinamento dei messaggi
  - nel file: dipende da processo scrivente
  - nella pipe: FIFO
- Ordinamento dei processi in attesa
  - nel file: dipende dalla gestione del file system

## **In sintesi**

- Abbiamo visto:
  - Comunicazioni mediante file condivisi
  - Comunicazioni mediante pipe
  - Caratteristiche dei messaggi
  - Funzioni di sistema operativo
  - Caratteristiche e problemi

## **SISTEMI OPERATIVI**

Gestione del Processore  
Comunicazione tra Processi

### **Lezione 7 – Comunicazione con socket**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Modello della comunicazione mediante socket
- Caratteristiche dei messaggi
- Funzioni
- Caratteristiche e problemi

## Modello della comunicazione mediante socket



Generalizzazione in rete delle comunicazioni con pipe

## Architettura (1)

### Architettura della comunicazione

- Architettura client-server
- Client invia richieste a una porta specifica
- Server in ascolto su una porta specifica
- Server riceve le richieste dei client

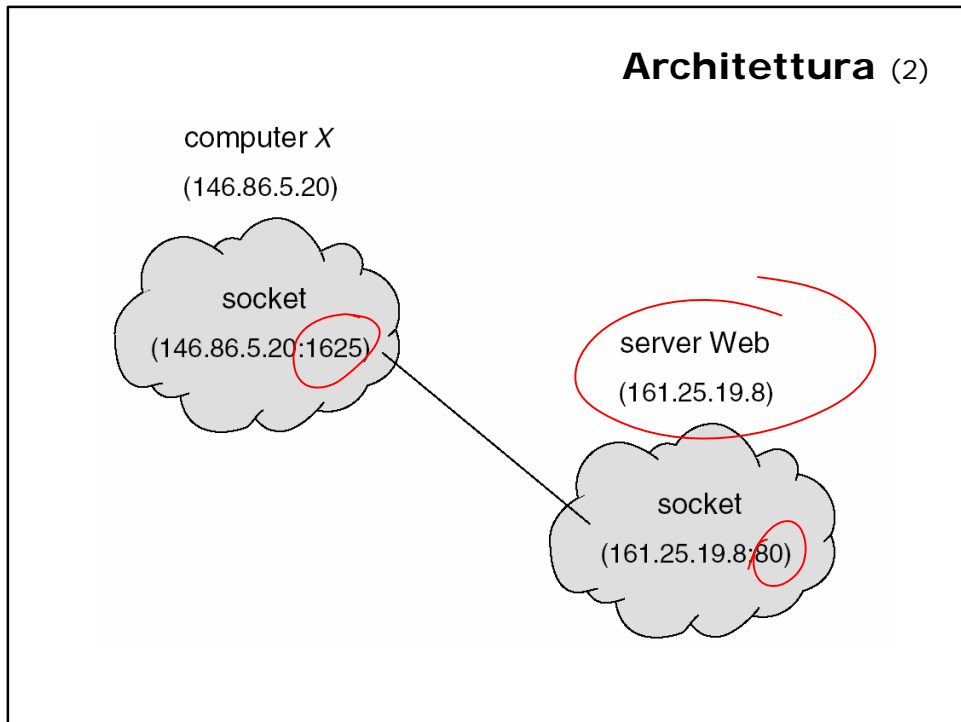
### Porta

- Indirizzo della macchina ospite (indirizzo IP)
- Numero di porta

Esempi:

- telnet: porta 23
- ftp: porta 21
- web o http: porta 80
- fino alla 1024: porte **note**





## Caratteristiche dei messaggi

### Contenuto

- Informazioni da trasmettere

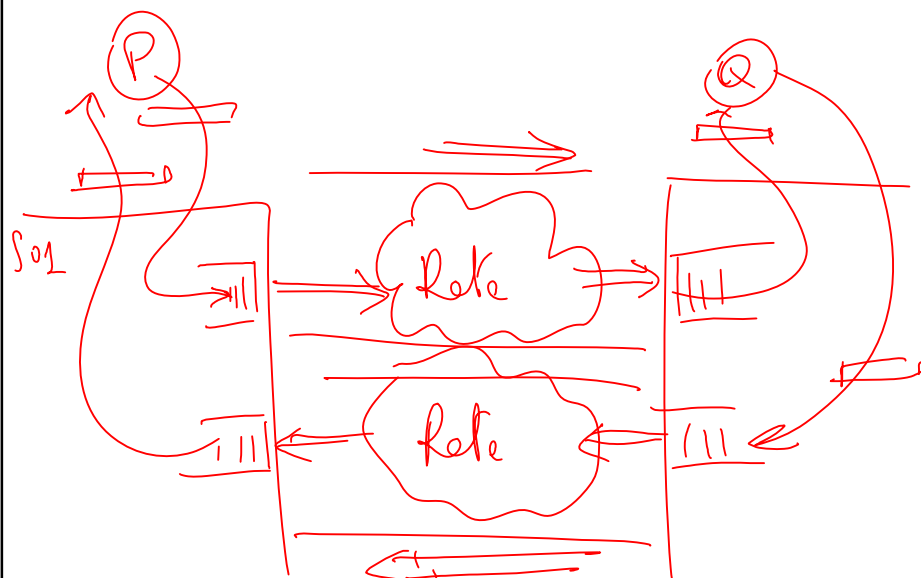
### Dimensione

- Fissa
- Variabile

## Funzioni

**Creazione**  
**Cancellazione**  
**Lettura**  
**Scrittura**

## Canale di comunicazione



## **Caratteristiche e problemi**

- Ordinamento dei messaggi: FIFO
- Ordinamento dei processi in attesa: FIFO
- Connessione:
  - con gestione della connessione
  - senza gestione della connessione
  - con multicast

## **In sintesi**

- Abbiamo visto:
  - Modello della comunicazione mediante socket
  - Cos'è un socket
  - Caratteristiche dei messaggi
  - Funzioni
  - Caratteristiche e problemi

## **SISTEMI OPERATIVI**

Gestione del Processore  
Sincronizzazione dei Processi

### **Lezione 1 – Processi concorrenti**

**Vincenzo Piuri**

---

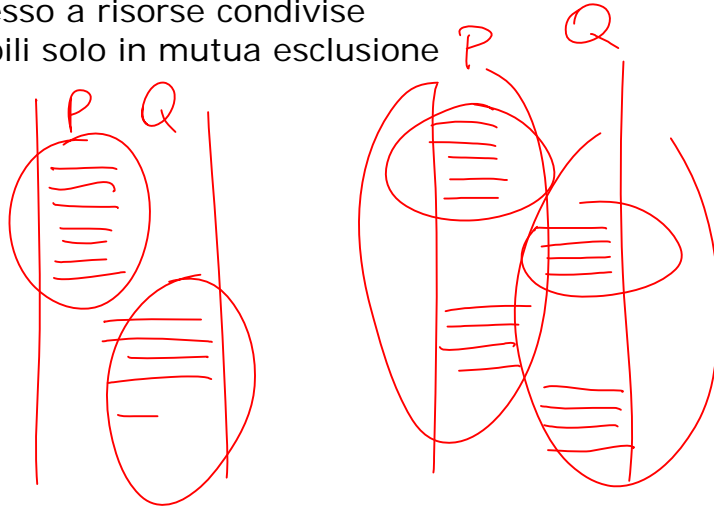
Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Concetto di concorrenza tra processi o thread
- Sezioni critiche

## Concorrenza

- Accesso a risorse condivise  
usabili solo in mutua esclusione



## Sincronizzazione per l'uso di risorse condivise

- Risorse fisiche (ad esempio: periferiche)
- Risorse informative (ad esempio: variabili in memoria centrale, file)

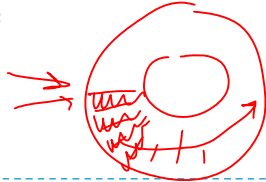
## Problema del produttore-consumatore

### Produttore

```
while (1) {
    while (count == BUFFER_SIZE)
        ; // non fare nulla
    // aggiungi un elemento nel
    buffer
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER_SIZE;
    counter++;
}
```

### Consumatore

```
while (1) {
    while (count == 0)
        ; // non fare nulla
    nextConsumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    counter--;
    // rimuovi un elemento nel buffer
}
```



L'esecuzione delle due procedure in modo concorrente può portare a risultati non corretti

## Corse critiche

### ++count

```
register1 = count
register1 = register1 + 1
count = register1
```

### --count

```
register2 = count
register2 = register2 - 1
count = register2
```

### Esecuzione concorrente

S0: producer esegue	register1 = count	{ register1 = 5 }
S1: producer esegue	register1 = register1 + 1	{ register1 = 6 }
S2: consumer esegue	register2 = count	{ register2 = 5 }
S3: consumer esegue	register2 = register2 - 1	{ register2 = 4 }
S4: producer esegue	count = register1	{ count = 6 }
S5: consumer esegue	count = register2	{ count = 4 }

## Sezione critica

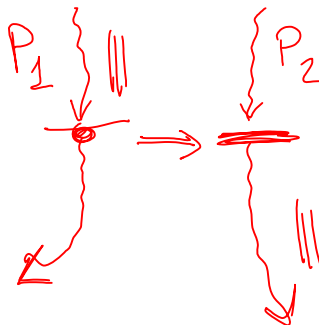
Porzione di codice che può generare errori se eseguita in modo concorrente.

Condizioni da soddisfare:

- **mutua esclusione**
- **progresso**
- **attesa limitata**

## Sincronizzazione di processi cooperanti

- Processi cooperanti  
lavorano insieme per uno scopo comune
- Sincronizzazione dell'evoluzione  
della computazione



## **In sintesi**

- **Concetto di concorrenza**
- **Sezioni critiche**
- **Uso delle tecniche di sincronizzazione anche per la cooperazione**



## SISTEMI OPERATIVI

Gestione del Processore  
Sincronizzazione dei Processi

### Lezione 2 – Variabili di lock

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### Sommario

##### **Sincronizzazione di processi concorrenti**

*Approcci a livello di istruzioni*

- Variabile di turno
- Algoritmi per la sincronizzazione mediante turno
- Variabile di lock
- Supporti hardware per le variabili di lock

## **Variabile di turno**

Variabile condivisa che definisce

*il turno di uso della risorsa*

cioè a quale processo spetta il diritto di uso in un certo istante.

## **Sincronizzazione di due processi concorrenti mediante variabile di turno**

Vediamo tre approcci per l'accesso  
in mutua esclusione alla rispettiva sezione critica  
mediante una variabile di turno

## Algoritmo 1

```

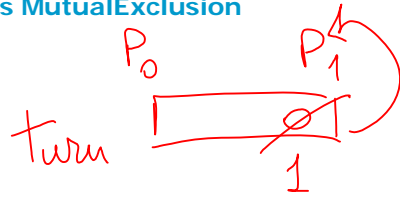
public class Algorithm_1 implements MutualExclusion
{
    private volatile int turn;

    public Algorithm 1() {
        turn = TURN 0;
    }

    public void enteringCriticalSection(int t) {
        while (turn != t)
            Thread.yield();
    }

    public void leavingCriticalSection(int t) {
        turn = 1 - t;
    }
}

```



Garantisce mutua esclusione  
 Impone stretta alternanza dei processi  
 Non garantisce progresso

## Algoritmo 2

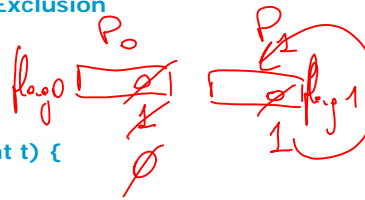
```

public class Algorithm_2 implements MutualExclusion
{
    private volatile boolean flag0, flag1;
    public Algorithm 2() {
        flag0 = false; flag1 = false;
    }

    public void enteringCriticalSection(int t) {
        if (t == 0) {
            flag0 = true;
            while(flag1 == true)
                Thread.yield();
        }
        else {
            flag1 = true;
            while (flag0 == true)
                Thread.yield();
        }
    }

    public void leavingCriticalSection(int t) {
        if (t == 0) flag0 = false; else flag1 = false;
    }
}

```



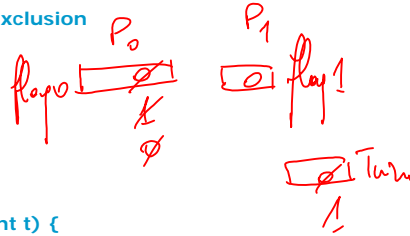
Non impone stretta alternanza dei processi  
 Non garantisce progresso  
 Possibile attesa infinita

## Algoritmo 3

```

public class Algorithm_3 implements MutualExclusion
{
    private volatile boolean flag0;
    private volatile boolean flag1;
    private volatile int turn;
    public Algorithm_3() {
        flag0 = false;
        flag1 = false;
        turn = TURN_0;
    }
    public void enteringCriticalSection(int t) {
        int other = 1 - t;
        turn = other;
        if (t == 0) {
            flag0 = true;
            while(flag1 == true && turn == other)
                Thread.yield();
        }
        else {
            flag1 = true;
            while (flag0 == true && turn == other)
                Thread.yield();
        }
    }
    public void leavingCriticalSection(int t) {
        if (t == 0) flag0 = false; else flag1 = false;
    }
}

```



Garantisce mutua esclusione  
Garantisce progresso

## Variabile di lock

Variabile condivisa che definisce

*lo stato di uso di una risorsa*

cioè quando è in uso da parte di un processo

(cioè quando un processo è nella sua sezione critica).

**Lock = 0** → risorsa libera

**= 1** → in uso

## Uso ad interruzioni disabilitate

### Acquisizione della risorsa:

- disabilito le interruzioni ✓
- leggo la variabile di lock
- se la risorsa libera (lock=0), la marco in uso ponendo lock=1 e riabilito le interruzioni ✓
- se la risorsa è in uso (lock=1), riabilito le interruzioni e pongo il processo in attesa che la risorsa si liberi

### Rilascio della risorsa:

- pongo lock=0

## Hardware per la sincronizzazione

### Istruzione atomica

#### TEST-AND-SET

- legge la variabile di lock e la pone in un flag del processore
- pone lock = 1
- se il flag (= vecchio valore di lock) vale 0, la risorsa era libera, altrimenti era già occupata e il processo deve attendere

## In sintesi

- **Abbiamo visto:**

*il concetto e l'uso  
di variabili di turno e di lock  
per realizzare la mutua esclusione  
nell'accesso alle rispettive sezioni critiche  
di processi concorrenti per risorse condivise*

- **Ricordiamo che questi sono**

*approcci a livello di istruzione  
e quindi richiedono grande attenzione da parte del  
programmatore per un uso corretto*

## **SISTEMI OPERATIVI**

Gestione del Processore  
Sincronizzazione dei Processi

### **Lezione 3 – Semafori**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Concetto di semaforo:
  - semaforo binario
  - semaforo generalizzato
- Uso
- Realizzazione

## Obiettivo

- Innalzare il livello di astrazione portando la gestione della sincronizzazione in **funzioni del sistema operativo**
- Garantire la corretta gestione della sincronizzazione e dell'accesso alle variabili di supporto alla mutua esclusione
- Evitare usi errati delle operazioni di abilitazione e disabilitazione delle interruzioni o degli assegnamenti alle variabili di turno o lock

## Semaforo binario

Un semaforo binario  $S$  è una variabile binaria che rappresenta lo stato di uso della risorsa condivisa

**$S = 1$**  → risorsa libera  
 **$0$**  → in uso

Il semaforo  $S$  è manipolato dalle funzioni:

- **$acquire(S)$**  → acquisisce l'uso della risorsa
- **$release(S)$**  → rilascia la risorsa

*acquire* e *release* sono operazioni *atomiche* poiché sono procedure di sistema



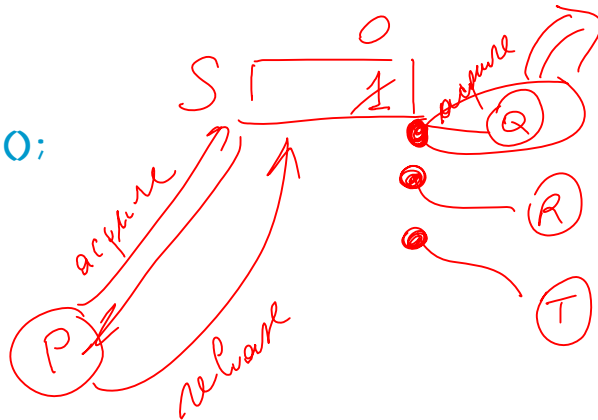
## Uso del semaforo binario

**Semaphore S;**

**acquire(S);**

**criticalSection();**

**release(S);**



## Implementazione del semaforo binario

### Attesa attiva in caso di risorsa non disponibile:

- strutture dati: variabile binaria S
- `acquire(S)` rimane in attesa attiva sulla variabile S fintanto che la risorsa non diventa disponibile

### Sospensione e rischedulazione in caso di risorsa non disponibile:

- strutture dati: variabile binaria S e coda dei processi in attesa di acquisire la risorsa
- `acquire(S)` sospende il processo in esecuzione in caso di risorsa non disponibile e lo inserisce nella coda di attesa del semaforo
- `release(S)` rilascia la risorsa e riattiva il primo processo della coda di attesa cedendogli la risorsa
- lo schedatore dei processi in attesa della risorsa definisce l'ordine di ottenimento della risorsa in base alla politica adottata per il semaforo

## Semaforo generalizzato

Un semaforo generalizzato  $S$  è una variabile intera che rappresenta lo stato di uso di un insieme di risorse omogenee condivise

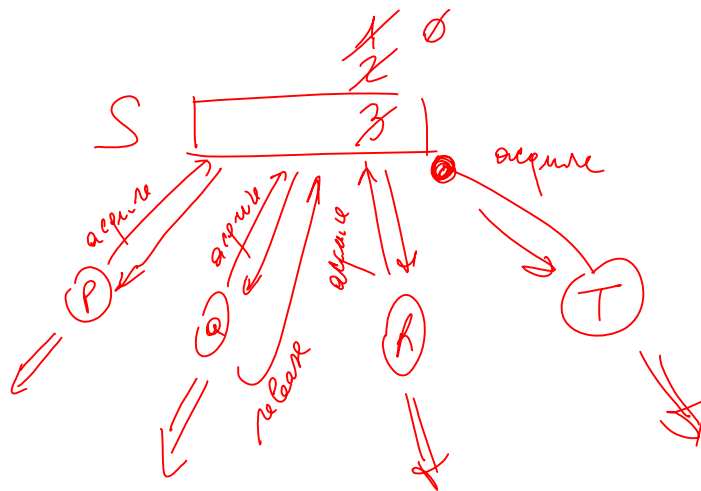
$S = n \rightarrow n$  risorse libere

$0 \rightarrow$  in uso

Il semaforo  $S$  è manipolato dalle funzioni:

- **acquire(S)**  $\rightarrow$  acquisisce l'uso di una risorsa
- **release(S)**  $\rightarrow$  rilascia la risorsa in uso

## Uso del semaforo generalizzato



## In sintesi

- **Abbiamo visto**

- *Semafori binari*
- *Semafori generalizzati*

- **Ricordiamo che questi sono**

*approcci a livello di funzioni  
del sistema operativo  
e quindi garantiscono un uso corretto delle risorse  
da parte del programmatore*

*ne diamo!*

## **SISTEMI OPERATIVI**

Gestione del Processore  
Sincronizzazione dei Processi

### **Lezione 4 – Monitor**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Concetto di monitor
- Realizzazione
- Uso

## Problemi legati all'uso dei semafori

### Errori di programmazione

- Violazioni della mutua esclusione
- Attese infinite

La responsabilità della correttezza  
è lasciata al programmatore

Il sistema operativo non ha potere di controllo  
e gestione

### Motivo?

Le primitive relative ai semafori sono chiamate  
di sistema operativo e come tali operano  
solo se chiamate in modo corretto

## Obiettivo del monitor

### Soluzione:

Innalzare il livello di astrazione per la gestione  
della sincronizzazione forzandone l'uso corretto

MONITOR: costrutto linguistico  
trasformato nelle corrette chiamate di sistema  
dal compilatore

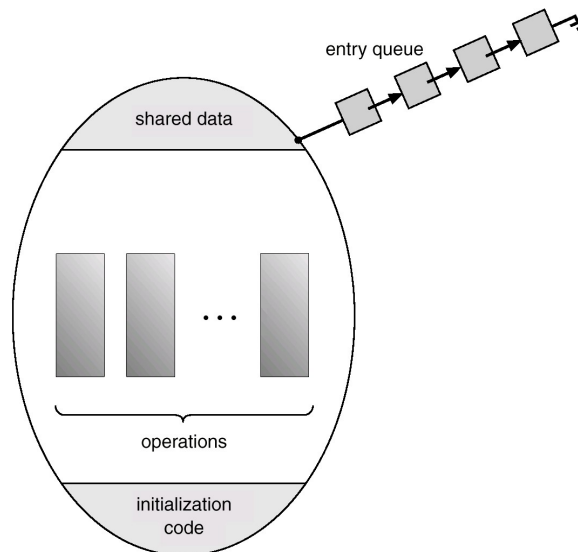
## Definizione di monitor

**Costrutto di sincronizzazione formulato  
a livello di linguaggio di programmazione**

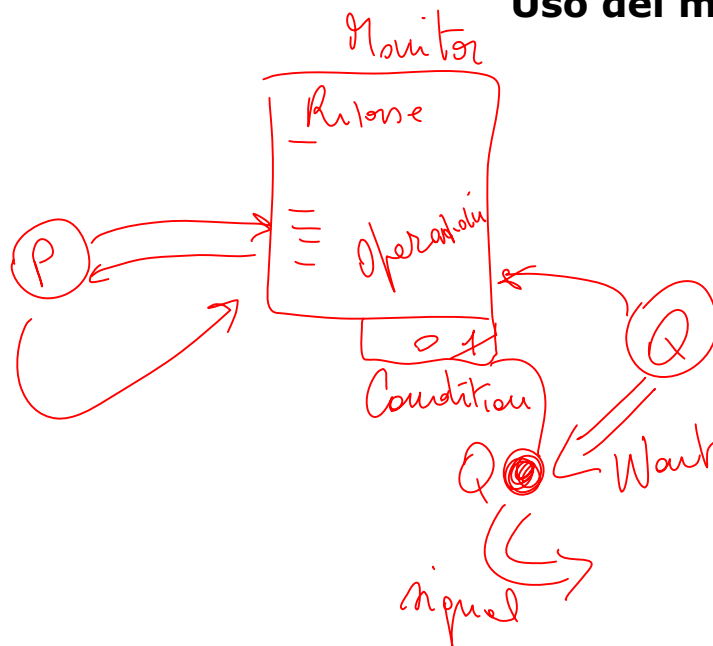
**Solo un processo alla volta può essere attivo  
in un monitor**

```
monitor monitor-name  
{  
    public entry p1(...) {  
        ...  
    }  
    public entry p2(...) {  
        ...  
    }  
}
```

## Realizzazione del monitor



## Uso del monitor



## In sintesi

- Abbiamo visto il concetto e l'uso del Monitor
- Ricordiamo che questo è un approccio a livello di linguaggio di programmazione e quindi garantisce un uso corretto delle risorse da parte del programmatore

per i programmi che lo usano!

## **SISTEMI OPERATIVI**

Gestione del Processore  
Sincronizzazione dei Processi

### **Lezione 5 – Problemi della starvation e del deadlock**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

Problemi tipici della sincronizzazione tra processi:

- Starvation
- Deadlock



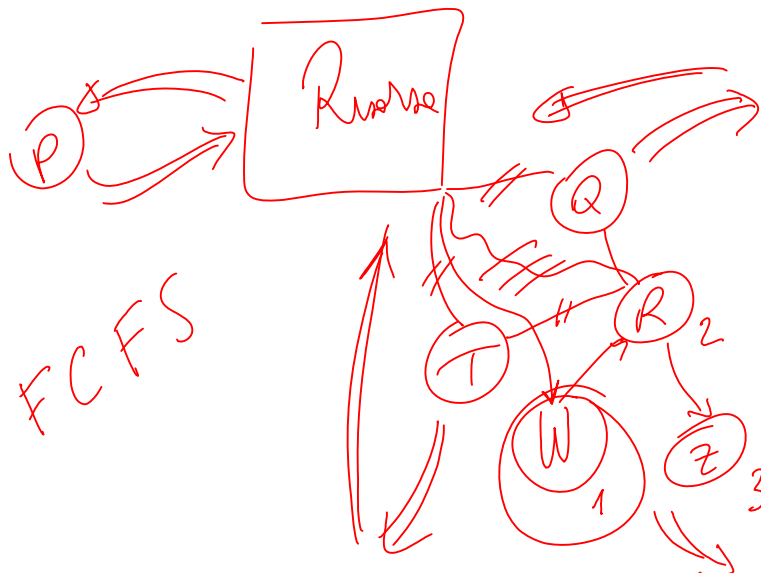
## Starvation – Blocco indefinito (1)

**Un processo in attesa di usare una risorsa rimane bloccato indefinitamente in attesa poiché altri processi ottengono sempre prima tale risorsa**

### Causa:

Uso di una politica di schedulazione della coda di attesa che non garantisce a tutti i processi di ottenere in un tempo finito la risorsa

## Starvation – Blocco indefinito (2)



## **Starvation – Blocco indefinito (3)**

### **Soluzione:**

Scelta accurata dell'algoritmo di schedulazione della coda dei processi in attesa della risorsa

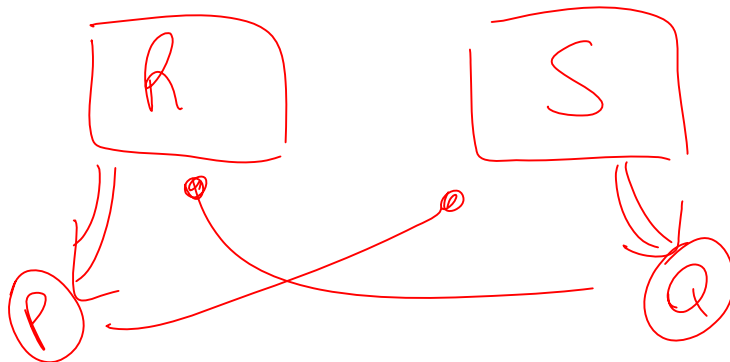
## **Deadlock – Stallo (1)**

**In un gruppo di due o più processi, ciascun processo aspetta una risorsa che é detenuta in modo mutuamente esclusivo da uno altro processo del gruppo**

### **Causa:**

Attesa circolare senza rilascio

## Deadlock – Stallo (2)



## Deadlock – Stallo (3)

Soluzione:

Impedire,  
Prevenire,  
Risolvere o  
Ignorare

le situazioni di attesa in stallo

## **In sintesi**

- Abbiamo visto due problemi tipici della sincronizzazione di processi
  - Starvation
  - Deadlock

## **SISTEMI OPERATIVI**

Gestione del Processore  
Sincronizzazione dei Processi

### **Lezione 6 – Transazioni atomiche**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Concetto di transazione atomica
- Transazioni atomiche individuali
  - Logging
  - Check pointing
- Transazioni atomiche concorrenti
  - Serializzazione
  - Protocolli basati su locking
  - Protocolli basati su timestamp

## Definizione di transazione

**Un insieme di istruzioni  
che eseguono un'unica funzione logica**

- Esempio:

read  
read  
manipolazione dei dati  
read  
manipolazione dei dati  
write  
manipolazione dei dati  
read  
manipolazione dei dati  
write  
read  
manipolazione dei dati  
read  
manipolazione dei dati  
read  
manipolazione dei dati  
write  
commit o abort

*atomicità*

## Atomicità della transazione

L'effetto della transazione sulle informazioni memorizzate deve essere permanente  
solo se **tutte** le operazioni  
sono state completate correttamente  
senza interferenze da parte di altri processi

La sequenza di operazioni di una transazione  
deve essere **atomica**  
come un'unica operazione indivisibile

Terminazione:

corretta → commit      effetti permanenti  
errata → abort      nessun effetto (roll back)

## Tipologie di archivi

### Archivio volatile

le informazioni non sopravvivono allo spegnimento del sistema

- memoria cache
- memoria centrale

### Archivio non volatile

le informazioni sopravvivono allo spegnimento del sistema

- dischi magnetici e ottici
- nastri magnetici

### Archivio stabile

le informazioni non vengono mai perse

- replicazione in molti archivi non volatili

## Transazioni atomiche individuali

Gestione basata su

- Logging
  - Write-ahead logging
- Check pointing

## Write-ahead logging

### **Log (registro) delle transazioni:**

registra in un archivio stabile le transazioni e il loro stato di esecuzione

- nome della transazione
- nome dell'oggetto dei dati
- vecchio valore dei dati
- nuovo valore dei dati

### **Meccanismo di write-ahead logging**

- Inizio transazione →  $\langle T_i \text{ starts} \rangle$
- Fine transazione →  $\langle T_i \text{ commits} \rangle$

## Recupero basato sul log (1)

### **undo( $T_i$ )**

- riporta i dati modificati dalla transazione  $T_i$  ai vecchi valori

### **redo( $T_i$ )**

- assegna ai dati modificati dalla transazione  $T_i$  il nuovo valore

Funzioni idempotenti



## Ripristino basato sul log (2)

- Abort della transazione
  - **undo**( $T_i$ )
- Fallimento del sistema di elaborazione
  - Per ogni transazione del log,
    - se il log contiene  $\langle T_i \text{ starts} \rangle$  ma non  $\langle T_i \text{ commits} \rangle$ , esegue **undo**( $T_i$ )
    - se il log contiene sia  $\langle T_i \text{ starts} \rangle$  sia  $\langle T_i \text{ commits} \rangle$ , esegue **redo**( $T_i$ )

## Check Pointing

### Problema del logging:

Tempo lungo di ripristino per lunghi log

### Soluzione:

Check pointing (punti di verifica)

Periodicamente si eseguono:

- scrittura su archivio stabile dei record del log memorizzati su archivio volatile
- scrittura dei dati modificati su archivio stabile
- scrittura del record  $\langle \text{checkpoint} \rangle$  su archivio stabile

## Ripristino basato su check pointing

- Fallimento del sistema di elaborazione
  - Per ogni transazione del log **a partire dal check point più recente**,
    - se il log contiene  $\langle T_i \text{ starts} \rangle$  ma non  $\langle T_i \text{ commits} \rangle$ , esegue **undo**( $T_i$ )
    - se il log contiene sia  $\langle T_i \text{ starts} \rangle$  sia  $\langle T_i \text{ commits} \rangle$ , esegue **redo**( $T_i$ )

## Transazioni atomiche concorrenti

Esecuzione concorrente di transazioni atomiche

- Esecuzione delle transazioni  
in modo seriale  
in un ordine arbitrario

**serializzabilità**

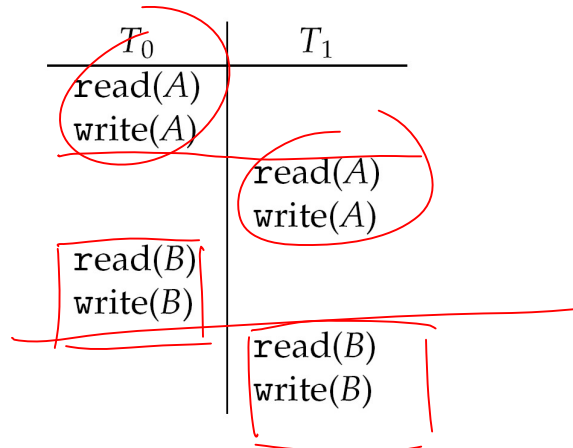
## Tecniche per la serializzabilità

- A livello di transazione
  - Transazioni eseguite in sezioni critiche
  - Condivisione di un semaforo *mutex* comune tra le transazioni
  
- A livello di operazioni nelle transazioni
  - Algoritmi di controllo della concorrenza delle operazioni
    - schedulazione concorrente seriale
    - schedulazione concorrente serializzabile
      - » protocollo di lock
      - » protocolli basati su timestamp

## Schedulazione concorrente seriale

$T_0$	$T_1$
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

## Schedulazione concorrente serializzabile



## Lock

**Lock (blocco) è una variabile associata a un dato che definisce l'accessibilità al dato stesso**

Lock = libero → accesso consentito  
= in uso → transazione sospesa  
in attesa di blocco libero

### Tipi di lock

- Lock condiviso
- Lock esclusivo

### **Protocollo di lock di base**

- $T_i$  esegue lock su dato  $Q$
- Se lock disponibile,  $T_i$  accede al dato
- Se lock non disponibile,
  - Se il lock richiesto è esclusivo,  $T_i$  attende finché il dato viene rilasciato
  - Se il lock richiesto è condiviso,
    - $T_i$  accede al dato se esso è correntemente bloccato con lock condiviso
    - $T_i$  attende se il dato è correntemente bloccato con lock esclusivo

Serializzabilità non garantita

### **Protocollo di lock a due fasi**

#### **Fase di crescita** (growing phase)

- una transazione può ottenere dei lock, ma non li può rilasciare

#### **Fase di contrazione** (shrinking phase)

- una transazione può rilasciare i lock, ma non ne può ottenere di nuovi

Assicura la serializzabilità  
Non previene gli stalli

## **Serializzazione nei protocolli di lock**

L'ordine di serializzazione  
di ogni coppia di transazioni in conflitto  
è determinato in esecuzione  
dal primo lock che richiedono  
e che implica incompatibilità

## **Timestamp**

**Timestamp (marca di tempo)  $TS(T_i)$**   
**è un attributo**  
**che rappresenta quando la transazione  $T_i$**   
**è entrata nel sistema**

**Timestamp è univocamente associato**  
**alle transazioni dal sistema**

### **Generazione del timestamp**

- Clock di sistema
- Contatore

### **Protocollo basato su timestamp (1)**

#### **Tipi di timestamp**

- W-timestamp(Q)
- R-timestamp(Q)

**Ogni operazione read o write in conflitto  
è eseguita nell'ordine della marca di tempo**

### **Protocollo basato su timestamp (2)**

#### **read(Q)**

- Se  $TS(T_i) < W\text{-timestamp}(Q)$ ,  
la lettura è negata e  $T_i$  esegue roll-back
- Se  $TS(T_i) \geq W\text{-timestamp}(Q)$ ,  
la lettura è eseguita e  
 $R\text{-timestamp}(Q) = \max\{R\text{-timestamp}(Q), TS(T_i)\}$

#### **write(Q)**

- Se  $TS(T_i) < R\text{-timestamp}(Q)$ ,  
la scrittura è negata e  $T_i$  esegue roll-back
- Se  $TS(T_i) < W\text{-timestamp}(Q)$ ,  
la scrittura è negata e  $T_i$  esegue roll-back
- Altrimenti, la scrittura è eseguita

### **Serializzazione nei protocolli di timestamp**

L'ordine di serializzazione  
di ogni coppia di transazioni in conflitto  
è determinato dal timestamp  
associato a ciascuna transazione alla sua attivazione

### **In sintesi**

- Concetto di transazione atomica
- Transazioni atomiche individuali
  - Definizione
  - Gestione
    - Logging
    - Check pointing
- Transazioni atomiche concorrenti
  - Serializzazione
  - Gestione
    - Locking
    - Timestamp



## **SISTEMI OPERATIVI**

Gestione del Processore  
Gestione del Deadlock

### **Lezione 1 – Caratterizzazione del deadlock**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Definizione di deadlock (stallo)
- Condizioni per l'occorrenza del deadlock
- Identificazione del deadlock:  
grafo di allocazione delle risorse
- Metodi di gestione del deadlock

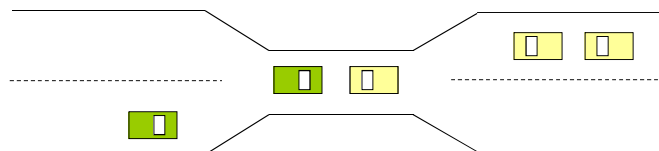
## Uso di risorse condivise

- Uso corretto e consistente delle risorse condivise
- Risorse condivise usabili
  - in modo non esclusivo
  - solo in modo mutuamente esclusivo
- Sincronizzazione per l'accesso all'uso di risorse condivise usabili solo in modo mutuamente esclusivo:
  - Richiesta di uso della risorsa
  - Uso della risorsa
  - Rilascio della risorsa

## Problema del deadlock

I processi in attesa possono permanere indefinitamente in tale stato se le risorse richieste sono in possesso di altri processi a loro volta in attesa

### • Esempio



### Condizioni per il verificarsi del deadlock

Si ha deadlock se si verificano simultaneamente le seguenti condizioni:

1. Mutua esclusione (**mutual exclusion**)
2. Possesso e attesa (**hold & wait**)
3. No rilascio anticipato (**no pre-emption**)
4. Attesa circolare (**circular wait**)

### Grafo di allocazione delle risorse <sup>(1)</sup>

**Grafo** di allocazione delle risorse  $G(V,E)$ :

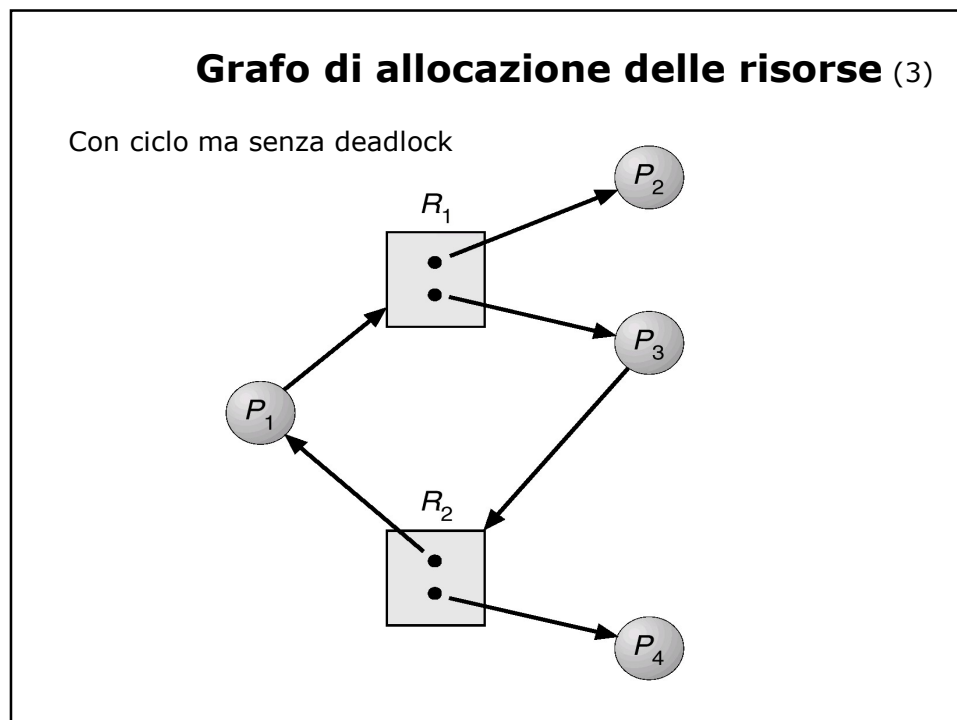
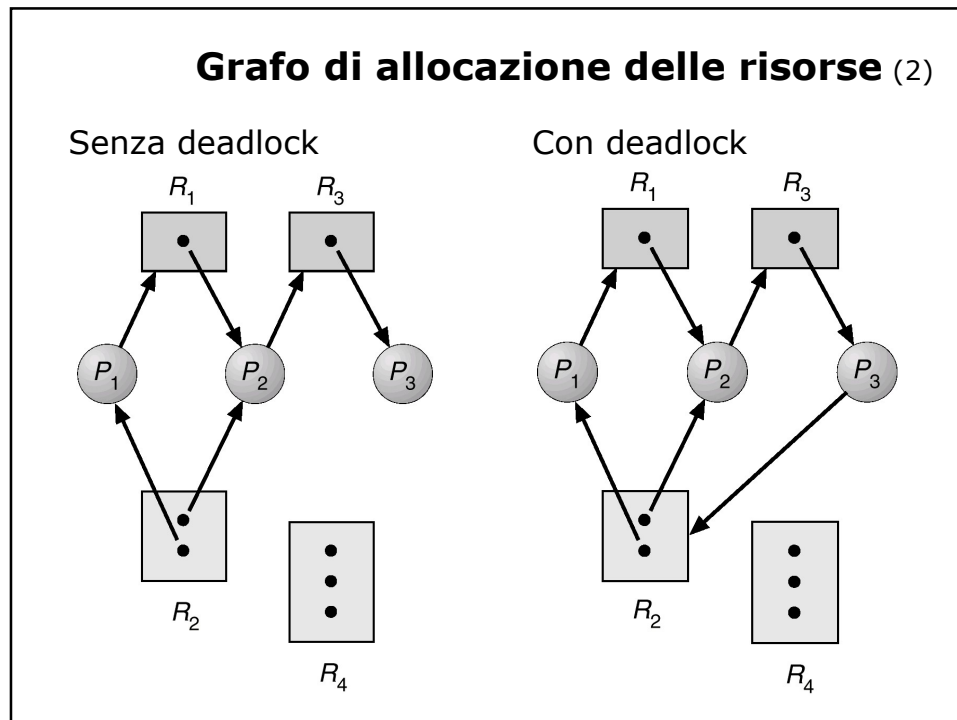
- Insieme di nodi  $V$
- Insieme di archi  $E$

**Nodi:**

- processi del sistema  $P = \{P_1, P_2, \dots, P_n\}$
- risorse del sistema  $R = \{R_1, R_2, \dots, R_m\}$   
eventualmente con più istanze identiche

**Archi:**

- arco di richiesta:  
da processo a risorsa  $P_i \rightarrow R_j$
- arco di assegnazione:  
da risorsa a processo  $R_j \rightarrow P_i$



### **Metodi di gestione dei deadlock**

- **Ignorare** il deadlock
- **Prevenzione** del deadlock  
(deadlock prevention)
- **Evitare** il deadlock  
(deadlock avoidance)
- **Rilevazione e recupero** del deadlock  
(deadlock detection & recovery)

### **In sintesi**

- Definizione di deadlock (stallo)
- Condizioni per l'occorrenza del deadlock
- Grafo di allocazione delle risorse e identificazione del deadlock
- Metodi di gestione del deadlock

## **SISTEMI OPERATIVI**

Gestione del Processore  
Gestione del Deadlock

### **Lezione 2 – Tecniche di prevenzione del deadlock**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Principio della prevenzione
- Tecniche per la condizione di mutua esclusione
- Tecniche per la condizione di possesso ed attesa
- Tecniche per la condizione di nessun rilascio anticipato
- Tecniche per la condizione di attesa circolare

## Principio della prevenzione

**Prevenire il deadlock  
impedendo che le quattro condizioni  
per cui si verifica siano tutte soddisfatte**

### **Obiettivo:**

Far sì che almeno una delle seguenti condizioni non sia soddisfatta:

- Mutua esclusione
- Possesso ed attesa
- Nessun rilascio anticipato
- Attesa circolare

## Mutua esclusione

La condizione

- deve essere assolutamente soddisfatta per le risorse non usabili in modo condiviso
- non è necessaria per le risorse usabili in modo condiviso

La condizione può essere invalidata rimuovendola per le risorse **intrinsecamente** condivisibili

La condizione non può mai essere invalidata per le risorse **intrinsecamente** non condivisibili

## **Possesso ed attesa (1)**

La condizione può essere invalidata garantendo che ogni volta che un processo chiede risorse, non possenga già qualche altra risorsa

## **Possesso ed attesa (2)**

### **Tecniche:**

- Un processo chiede e ottiene tutte le risorse prima di iniziare l'esecuzione.
- Un processo che possiede alcune risorse e vuole chiederne altre deve
  - rilasciare tutte le risorse che possiede
  - chiedere tutte quelle che servono, incluse eventualmente anche alcune di quelle che già possedeva



## **Possesso ed attesa (3)**

### **Problemi:**

- Scarso utilizzo delle risorse
- Possibile starvation

## **Nessun rilascio anticipato (1)**

La condizione può essere invalidata mediante rilascio anticipato (pre-emption) per risorse il cui stato di uso all'atto del rilascio anticipato è ripristinabile

## **Nessun rilascio anticipato (2)**

### **Tecniche:**

- Se un processo detiene alcune risorse e ne chiede altre che non possono essere assegnate immediatamente:
  - tutte le risorse possedute sono rilasciate anticipatamente
  - le risorse rilasciate anticipatamente sono aggiunte alla lista delle risorse per cui il processo sta aspettando
  - il processo sarà fatto ripartire soltanto quando potrà ottenere le vecchie e le nuove risorse

## **Nessun rilascio anticipato (3)**

### **Tecniche:**

- Se un processo detiene alcune risorse e ne chiede altre:
  - se tutte le risorse richieste sono disponibili, vengono assegnate
  - se alcune delle risorse richieste non sono disponibili,
    - se sono assegnate ad un processo che sta aspettando ulteriori risorse, le risorse richieste e detenute dal processo in attesa vengono
      - » rilasciate anticipatamente e assegnate al processo richiedente
      - » inserite tra quelle per cui il processo è in attesa
  - se alcune risorse richieste non sono disponibili e non sono possedute da processi in attesa di altre risorse, il processo richiedente deve
    - attendere che si liberino
    - ripartire quando ottiene tutte le risorse necessarie

### **Attesa circolare (1)**

La condizione può essere invalidata impedendo che si creino attese circolari

### **Attesa circolare (2)**

#### **Tecniche:**

- Un ordinamento globale univoco viene imposto su tutti i tipi di risorsa  $R_i$
- Se un processo chiede  $k$  istanze della risorsa  $R_j$  e detiene solo risorse  $R_i$  con  $i < j$ ,
  - se le  $k$  istanze della risorsa  $R_j$  sono disponibili vengono assegnate
  - altrimenti il processo deve attendere

Un processo non potrà mai chiedere istanze della risorsa  $R_j$  se detiene risorse  $R_i$  con  $i > j$

## Attesa circolare (3)

### Tecniche:

- Un ordinamento globale univoco viene imposto su tutti i tipi di risorsa  $R_i$
- Se un processo chiede  $k$  istanze della risorsa  $R_j$  e detiene solo risorse  $R_i$  con  $i < j$ ,
  - se le  $k$  istanze della risorsa  $R_j$  sono disponibili vengono assegnate
  - altrimenti il processo deve attendere
- Se un processo chiede  $k$  istanze della risorsa  $R_j$  e detiene risorse  $R_i$  con  $i = j$ , il processo deve
  - rilasciare tutte le istanze delle risorse  $R_i$
  - chiedere tutte le istanze della risorsa  $R_j$  (quelle detenute precedentemente e le nuove  $k$ )
  - chiedere le istanze delle risorse  $R_i$  ( $i > j$ ) che deteneva precedentemente

## In sintesi

- Principio della prevenzione del deadlock
- Tecniche di prevenzione del deadlock
  - Eliminazione della condizione di mutua esclusione
  - Eliminazione della condizione di possesso e attesa
  - Eliminazione della condizione di nessun rilascio anticipato
  - Eliminazione della condizione di attesa circolare

## **SISTEMI OPERATIVI**

Gestione del Processore  
Gestione del Deadlock

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

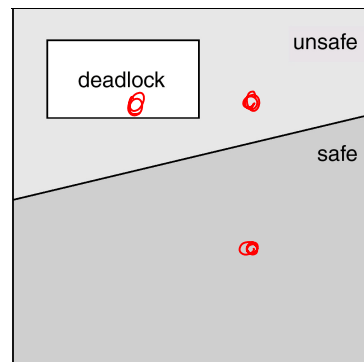
### Necessità di informazioni **a priori** sul comportamento dei processi:

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

## Stato sicuro (1)

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**

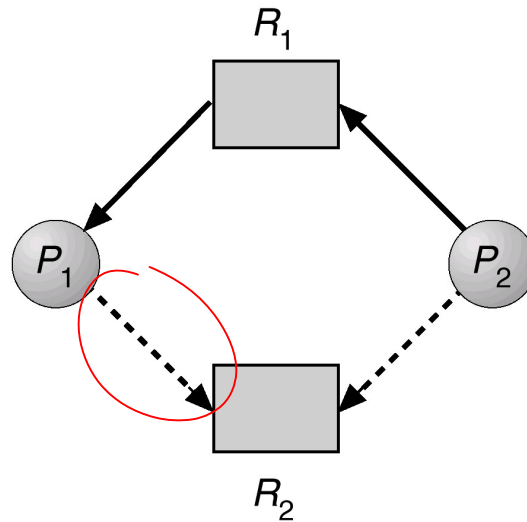
- Si parte da uno stato iniziale sicuro
- Una richiesta di risorsa viene soddisfatta se la risorsa è disponibile e se il sistema va in uno stato sicuro
- Se la risorsa non è disponibile, il processo deve attendere





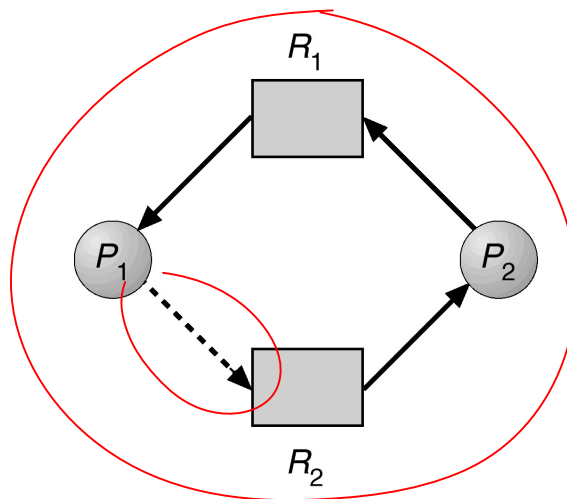
### Algoritmo del grafo di allocazione delle risorse (1)

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  $i=1,..,n$

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

## **SISTEMI OPERATIVI**

Gestione del Processore  
Gestione del Deadlock

### **Lezione 4 – Tecniche di rilevazione e ripristino del deadlock**

**Vincenzo Piuri**

---

Università degli Studi di Milano - SSRI - CDL ONLINE

#### **Sommario**

- Principio di rilevazione e ripristino
- Algoritmo di rilevamento con istanze delle risorse:
  - Solo singole istanze: Grafo di attesa
  - Istanze multiple: Algoritmo completo
- Tecniche di ripristino

## Principio di rilevazione e ripristino

Senza algoritmi di prevenzione o per evitare il deadlock, tale situazione può verificarsi

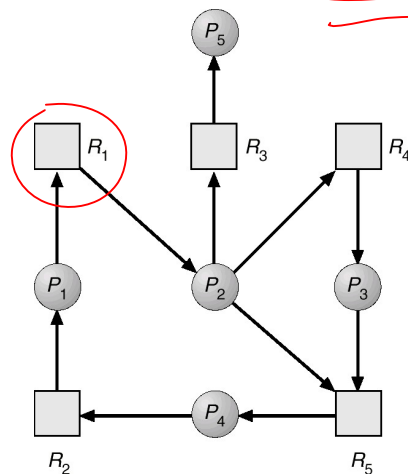
Il sistema deve essere in grado di

- rilevare la presenza di situazioni di deadlock **dopo** che sono avvenute
- ripristinare una situazione di corretto funzionamento eliminando il deadlock

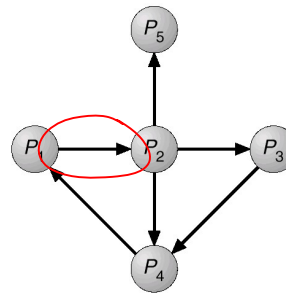
Sistemi con istanze singole o multiple delle risorse

### Rilevazione per sistemi con solo istanze singole delle risorse (1)

Grafo di allocazione delle risorse



Grafo di attesa (wait-for)



### **Rilevazione per sistemi con solo istanze singole delle risorse (2)**

- Analisi del grafo di attesa
- Se il grafo di attesa contiene cicli, si ha deadlock
- I processi in deadlock sono quelli coinvolti in ciascun ciclo presente nel grafo

### **Rilevazione per sistemi con istanze multiple delle risorse (1)**

#### **Strutture dati**

<b>m</b>	risorse
<b>n</b>	processi
<b>Available[1..m]</b>	risorse disponibili
<b>Allocation[1..n,1..m]</b>	risorse attualmente assegnate
<b>Request[1..n,1..m]</b>	risorse della richiesta corrente

## Rilevazione per sistemi con istanze multiple delle risorse (2)

### Algoritmo di rilevazione del deadlock

Work[1..m]

Finish[1..n]

1. Work=Available  
Per  $i=1, \dots, n$ ,  
    se Allocation[i]  $\neq 0$ , allora Finish[i]=false  
    altrimenti Finish[i]=true
2. Si cerca  $i$  tale che:
  - Finish[i]==false
  - Request[i]  $\leq$  WorkSe non esiste tale  $i$  vai al passo 4
3. Work=Work+Allocation[i]; Finish[i]=true  
Vai al passo 2
4. Se Finish[i]==false per qualche  $i$ , con  $1 \leq i < n$ ,  
allora si ha deadlock  
Se Finish[i]==false, allora il processo  $P_i$  è in deadlock

## Applicazione della rilevazione

### Quando invocare l'algoritmo di rilevazione?

- Ogni volta che una richiesta di allocazione non può essere soddisfatta immediatamente
  - rilevazione immediata
  - pochi risorse e processi bloccati
  - considerevole sovraccarico computazionale
- A intervalli di tempo prestabiliti
  - rilevazione più complessa
  - molte risorse e processi possono essere bloccati
  - minor sovraccarico computazionale



## Ripristino del deadlock (1)

### Terminare processi in deadlock

- Abortire tutti i processi in deadlock
  - Troppi processi terminati
  - Spreco di risorse computazionali
  - Costo elevato
- Abortire un processo alla volta fino a eliminare il deadlock
  - Pochi processi terminati
  - Algoritmo di rilevazione invocato più volte

## Ripristino del deadlock (2)

### Terminare processi in deadlock

- Ordine di eliminazione dei processi
  - priorità del processo
  - tempo di elaborazione del processo
  - risorse utilizzate
  - risorse richieste per terminare l'elaborazione
  - numero dei processi da terminare
  - processo interattivo o batch

## **Ripristino del deadlock (3)**

### **Rilascio anticipato delle risorse**

- Selezione della vittima (processo o risorsa)
  - la vittima è quella a costo minimo
- Rollback
  - all'ultimo stato sicuro
  - totale
- Starvation
  - non selezionare sempre la stessa vittima

### **In sintesi**

- Principio di rilevazione e ripristino
- Tecniche di rilevazione
- Tecniche di ripristino