

## Processi – parte I

---

## Processi – parte I

---

- Definizioni
- Concetto di processo
- Processi cooperanti
- Comunicazione tra processi

## Definizioni

---

- **Algoritmo:** procedimento logico che in un numero finito di passi giunge alla soluzione di un problema.
- **Programma:** formalizzazione di un algoritmo attraverso un linguaggio di programmazione.
- **Processo:** è una sequenza di operazioni effettuate da un programma in esecuzione su un determinato insieme di dati di input.

## Definizioni

---

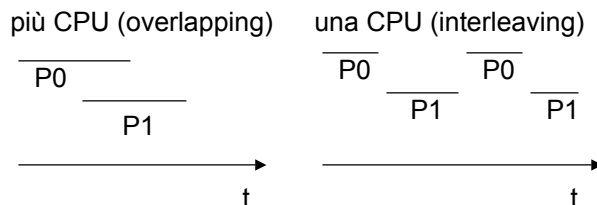
- Un SO esegue una varietà di programmi:
  - Sistemi Batch – job
  - Sistemi in time sharing – programmi utente o task

## Definizioni

- **Processo Sequenziale:** è un processo deterministico:
  - dato uno stesso input produce sempre lo stesso output, indipendentemente dal momento in cui viene eseguito, dalla velocità di esecuzione e da quanti altri processi siano attivi sul sistema

## Definizioni

- Due processi si dicono concorrenti se la loro esecuzione si sovrappone nel tempo.
- Due processi sono concorrenti se la prima operazione di uno inizia prima dell'ultima operazione dell'altro.

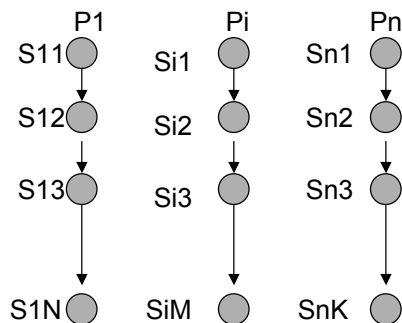


## Relazioni tra processi

- Processi concorrenti:
  - Indipendenti
  - Interagenti
- Interazione tra processi:
  - Competizione
  - Comunicazione

## Processi indipendenti

- Stato non condiviso: non hanno variabili globali in comune, l'intersezione tra i contesti è vuota.
- Esecuzione deterministica e riproducibile.



## Processi interagenti

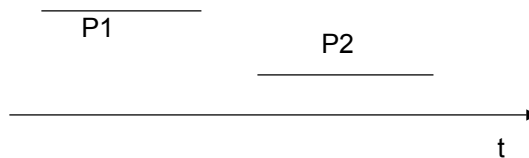
- Modificano variabili comuni e/o accedono a risorse comuni.
- Esecuzione non deterministica e non riproducibile:
  - Il risultato dipende dalla sequenza di esecuzione.
  - Il risultato non è sempre lo stesso per gli stessi dati di ingresso.

## Competizione e cooperazione

- Competizione:
  - Uso di risorse che non possono essere usate contemporaneamente
- Cooperazione:
  - Esecuzione di un'attività comune mediante scambio di informazioni

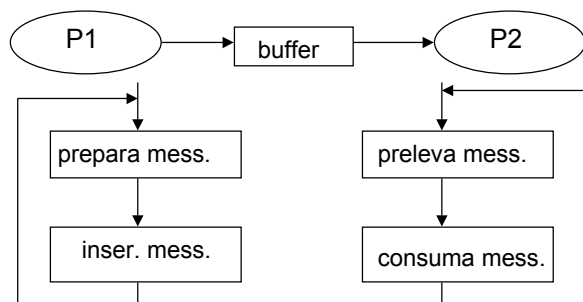
## Sezione critica

- Insieme di operazioni con le quali un processo modifica variabili (o usa risorse) comuni ad altri processi.
- Competizione:
  - Mutua esclusione tra P1 e P2.
  - Non è rilevante l'ordine con cui vengono eseguiti.



## Sezione critica

- Cooperazione:
  - È importante l'ordine con cui i processi vengono eseguiti
  - Esempio: produttore-consumatore



## Descrittore di processo

- Un processo è definito da:
  - contenuto dei registri macchina
  - in particolare program counter e stack pointer
  - stack
  - sezione dati
  - Info sulla memoria
  - Info di I/O
  - Info amministrative
  - ...

Identificatore
stato
PC (program counter)
PSW (Program Status Word)
Registri
limiti di memoria
lista dei file aperti
⋮
⋮

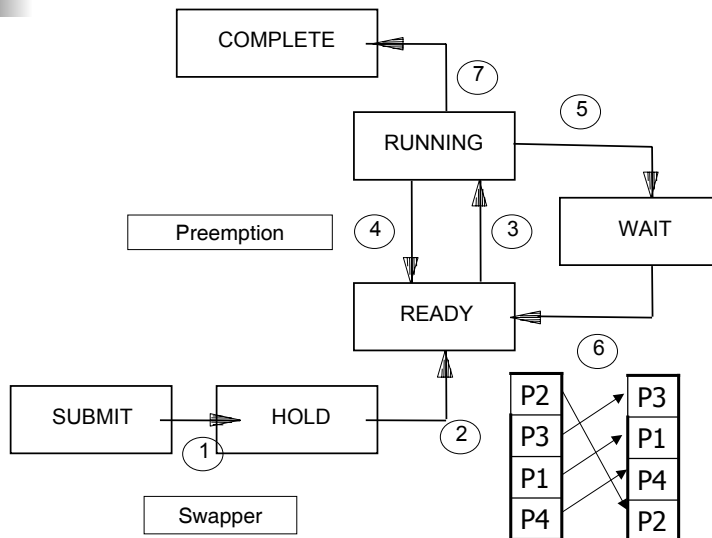
## Modello a processi

- Tutto il software in esecuzione è organizzato in un numero di **processi sequenziali**.
  - processi di S.O.
  - processi utente.

# Stati di un processo

- Durante la sua esecuzione un processo cambia di stato.
  - SUBMIT: il job viene suddiviso nei suoi processi componenti, che vengono creati.
  - HOLD: in attesa della disponibilità di risorse da parte del sistema.
  - READY: Logicamente pronto ad essere eseguito, in attesa della risorsa processore.
  - RUNNING: In esecuzione
  - WAITING: In attesa di un evento
  - COMPLETE: Il processo termina e rilascia le risorse utilizzate

# Stati di un processo

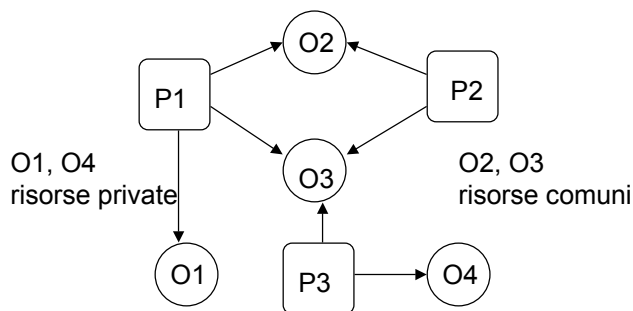


## Sincronizzazione

- Per garantire meccanismi di sincronizzazione sono state proposte diverse soluzioni:
  - Sleep e Wake-up
  - Semafori
  - Memoria condivisa
  - Scambio di messaggi
  - ...

## Modelli di interazione tra processi

- Modello ad ambiente globale



## Sleep e Wake-up

- SLEEP provoca la sospensione del processo che la esegue.
- WAKE UP risveglia un processo bloccato.
  - se il processo non è bloccato, il risveglio viene perduto.

## Sleep-Wakeup: produttore e consumatore

```
#define N 100
int contatore=0;

void produttore (void)
{
    int messaggio;
    while (TRUE){
        produzione-messaggio (&messaggio);
        if (contatore==N) sleep ();
        inserimento-messaggio (messaggio);
        contatore=contatore+1;
        if (contatore==1) wakeup(consumatore);
    }
}

void consumatore (void)
{
    int messaggio;
    while(TRUE){
        if (contatore==0) sleep();
        prelievo-messaggio (&messaggio);
        contatore=contatore -1;
        if (contatore==N-1) wakeup (produttore)
        consuma-messaggio;
    }
}
```

## Semafori

- Variabile intera non negativa con valore iniziale  $\geq 0$ .
- Al semaforo è associata una lista di attesa dei processi  $Q_s$ .
- Sul semaforo  $s$  sono ammesse solo due operazioni primitive atomiche:
  - `wait(s)`
  - `signal(s)`

## Semafori

`wait(s)`

```
{  
    if(s==0)  
        <processo sospeso e  
        inserito in  $Q_s$ >  
    else s=s-1;  
}
```

`signal(s)`

```
{  
    if(processo in  $Q_s$ )  
        <modifica il suo  
        stato in pronto>  
    else s=s+1;  
}
```

## Semafori: produttore-consumatore

```
#define TRUE 1
#define N 100
struct semaforo pieno = 0;
struct semaforo vuoto = N;
struct semaforo s = 1;
```

```
void produttore (void)
{
    int mess;

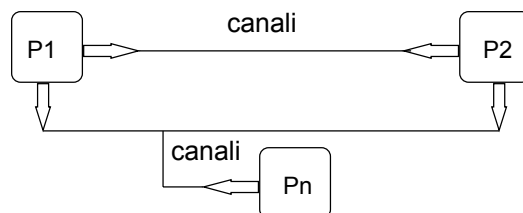
    while(TRUE)
    {
        produci_mess(&mess);
        wait(vuoto);
        wait(s);
        inserisci_mess(mess);
        signal(s);
        signal(pieno);
    }
}
```

```
void consumatore (void)
{
    int mess;

    while(TRUE)
    {
        wait(pieno);
        wait(s);
        preleva_mess(&mess);
        signal(s);
        signal(vuoto);
        consuma_mess(mess);
    }
}
```

## Modelli di interazione tra processi

### ■ Modello a scambi di messaggi



## Send e receive

- I processi comunicano e si sincronizzano, spedendo e ricevendo messaggi.
- Primitive send e receive:
  - **send(msg,P)**
  - **receive(msg,Q)**

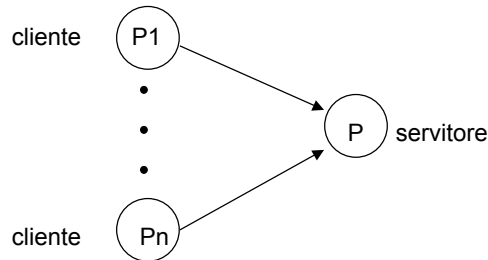
## Send e receive

- Schema a "pipeline"
  - send(msg,P2)
  - receive(msg,P1)



## Send e receive

- Schema diretto e asimmetrico:
  - `send(msg,P)` invia msg a P
  - `receive (msg,id)` riceve da tutti



## Send e receive

- Schema indiretto i processi comunicano tramite una mailbox A:
  - `send (msg,A)`
  - `receive (msg,A)`
- Il S.O. mette a disposizione chiamate di sistema per gestire le mailbox

## Comunicazione asincrona

- Il processo mittente continua l'esecuzione dopo che il messaggio è stato inviato.
- Non c'è garanzia che il messaggio sia stato ricevuto.
- Flessibile, ma manca di carenza espressiva.
  - massimo parallelismo
  - mittente e destinatario del messaggio non si conoscono.

## Comunicazione sincrona

- Il processo mittente si blocca fino a quando il messaggio non è stato ricevuto (rendez-vous semplice).
- Il processo rimane in attesa fino a quando il ricevente non ha terminato di svolgere l'azione richiesta.