La nozione di processo

Un sistema operativo è un programma complesso; per capirne la struttura e le attività si introduce la nozione di *process*o (o *agente*)

Processo:
attività controllata da un programma che si svolge su un processore e usa servizi e risorse del sistema

Nota: il sistema operativo NON è un processo; viene definito invece come un insieme di processi più il sw necessario a gestire i servizi

Il processo che *avanza:*
- utilizza in modo esclusivo un processore
- l'avanzamento non è continuo (interruzioni, sospensioni, prerilasci)

Possiamo rappresentare un processo mediante il concetto di *traccia di stati*, cioè una sequenza di rappresentazioni dello stato "dinamico"

Stato di un processo: rappresentato come una terna *P= (IM, IP, S)*

IM: immagine in memoria,
- informazione costituita da programma e dati
- modificata dal processo che avanza (solo i dati)

(distringuere il caricamento statico da quello dinamico)

IP: immagine nel processore
- informazioni sul processo mantenute nei registri del processore
- modificata dal processo che avanza

S : *stato di avanzamento* (o brevemente *stato).*
- assume i valori di *attesa,* di *pronto* e di *esecuzione.*

- Stato di *esecuzione*: il processo dispone del processore (o di un processore) e avanza nell'esecuzione del suo programma.
- Stati di *pronto* e di *attesa*: il processo non dispone del processore e non avanza.

Nota:
Estensione del concetto di processo: *processi esterni*
Rappresentano le attività svolte dai dispositivi periferici

Tabella dei processi (TdP)

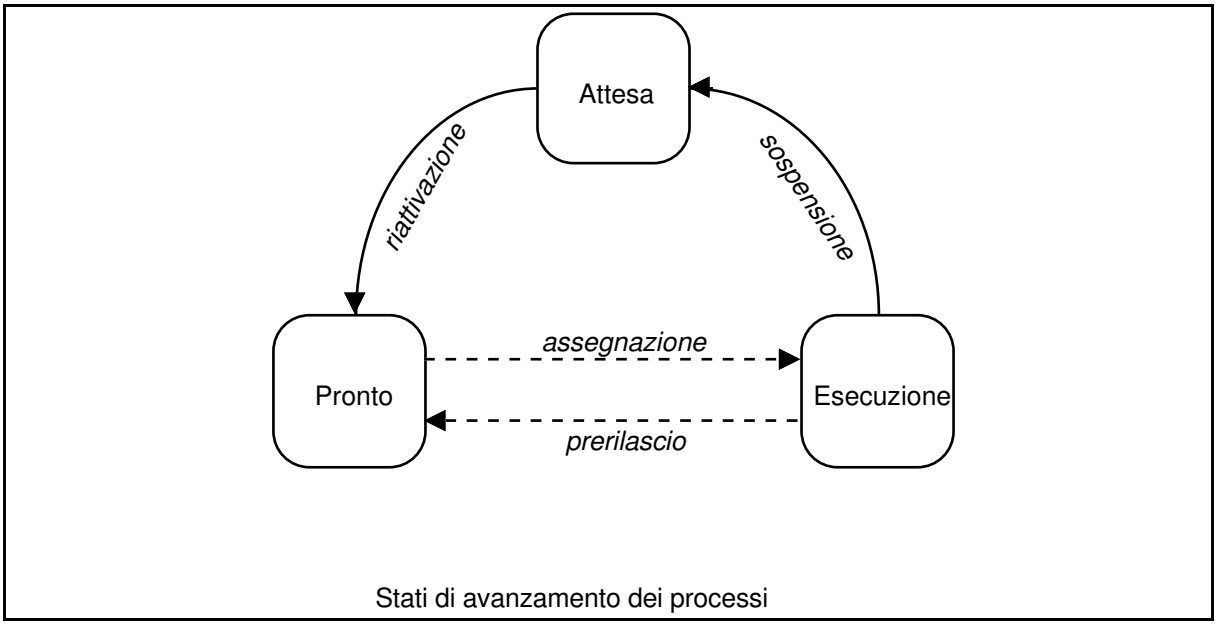
La tabella dei processi (TdP) rappresenta le diverse IP dei processi mediante campi che contengono alcuni valori;
l'insieme degli attributi permette di ricostruire lo stato del processo

Gestione processi	**Gestione memoria**	**Gestione file**
Registri	Punt. al segmento testo	Maschera UMASK
Registro CP	Punt. al segmento dati	Directory radice
PSW del programma	Punt. al segmento bss	Directory attiva
Registro punt. allo stack	Stato di uscita	Descrit. di file
Stato del processo	Stato dei segnali	Uid effettiva
Orario start processo	Id del processo	Gid effettiva
Tempo di CPU consumato	Genitore del processo	Parametri sys calls
Tempo di CPU del figlio	Gruppo del processo	
Ora del prossimo allarme	Uid reale	Vari bit flag
Puntatori coda messaggi	Uid effettiva	
Bit dei segnali in attesa	Gid reale	
Id del processo	Gid effettiva	
Vari bit flag	Bit map dei segnali	
	Vari bit flag	

Nota: durante l'esecuzione, il processo mantiene l'allocazione iniziale, anche se potrà variare la distribuzione fra dati, pila e parte inutilizzata; ad esempio, quando viene cambiata l'immagine con una `exec`, viene allocata una porzione di memoria la cui dimensione è specificata nell'intestazione del file che contiene il codice

Stati e transizioni di stato

Ogni processo durante la sua "vita" alterna gli stati di attesa, di pronto e di esecuzione passando attraverso *transizioni di stato*



Stati di avanzamento dei processi

• Transizione da esecuzione ad attesa: *sospensione*

- avviene per esplicita azione del processo in esecuzione;
- generalmente, il processo che si sospende si pone in attesa di un evento *E.*

Per esempio,un processo Pi si sospende in attesa di:

- interruzione da parte di un dispositivo,
- sincronizzazione con un altro processo,
- ricezione di un messaggio

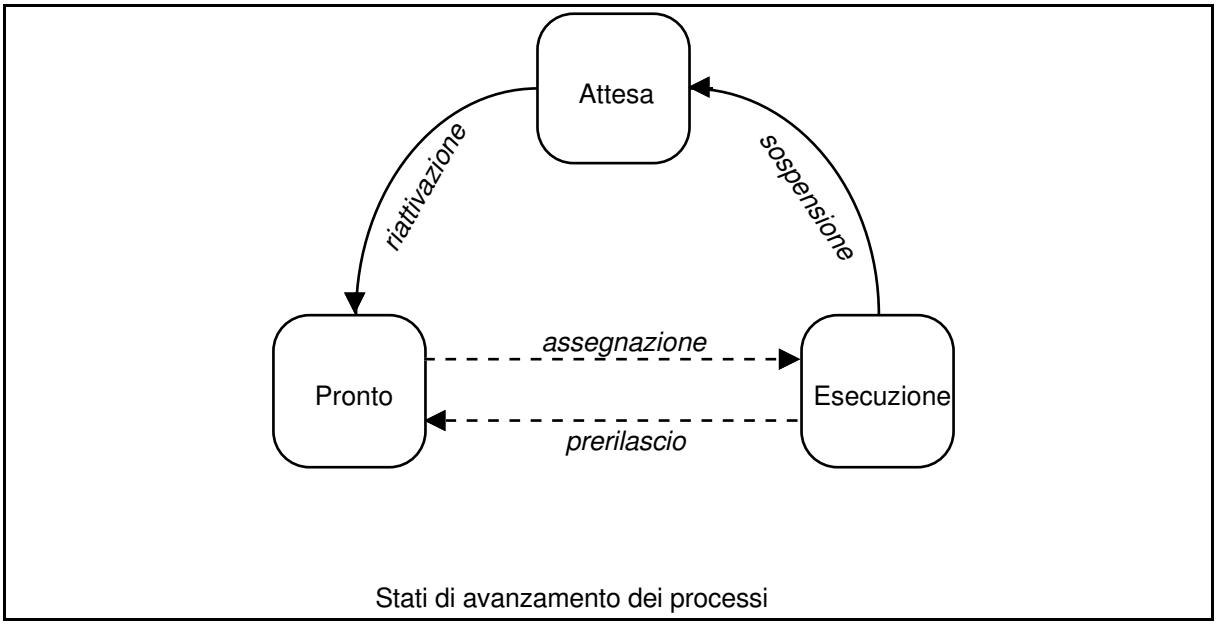
Il processo in attesa non è in competizione con i processi pronti; la riattivazione resta affidata alla cooperazione degli altri processi

• Transizione da stato di attesa a stato di pronto: *riattivazione*

- avviene per esplicita azione di un processo P_j in esecuzione (eventualmente processo esterno).
- Schema tipico: P_i sospeso in attesa di evento *E,* viene riattivato da P_j che segnala *E.*

Esempi di eventi: interruzione, sincronizzazione, arrivo di un messaggio

Stati e transizioni di stato



Stati di avanzamento dei processi

I processi non hanno alcun controllo sulle seguenti transizioni, che sono entrambe causate dalla funzione di gestione del processore (scheduler);

• Transizione da pronto a esecuzione: *assegnazione del processore*

• Transizione da esecuzione a pronto:
prerilascio (o revoca) del processore

Scheduler:
- controlla le velocità di avanzamento dei processi pronti
- utilizza il processore per fare avanzare i processi pronti (politica)

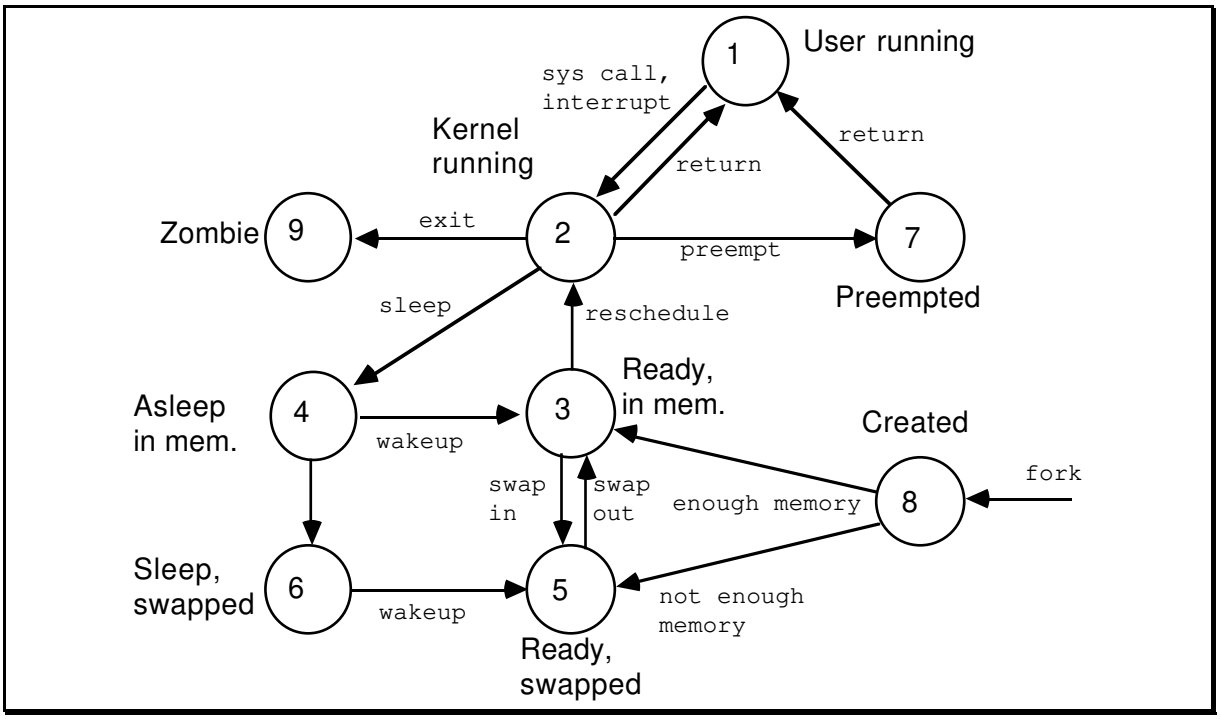
Ipotesi:
Uniprocessore: un processore ripartito in tempo tra n processi
Multiprocessore: # processi > # processori.

Dal punto di vista dei processi, nessuna differenza tra i due casi, salvo la diversa velocità di avanzamento.
Se fosse # processi= # processori, un processo potrebbe essere sempre attivo e le transizioni di stato non avrebbero motivo di esistere.

Il caso di Unix

In Unix i possibili stati dei processi sono molti di più

(Legenda:
in fonte helvetica sono mostrati gli stati dei processi,
in `courier` i msg che attivano le transizioni)



Macchine astratte concorrenti

La nozione di macchina astratta è utile per trascurare dettagli realizzativi nello studio dell'esecuzione dei programmi

Macchina astratta sequenziale:
- un solo program counter astratto (*thread*)
- un unico spazio di indirizzamento logico

Macchina astratta concorrente:
- più program counter astratti (multi-threaded)
- più spazi di indirizzamento logici

Macchina astratta concorrente con dati condivisi:
- gli spazi di indirizzamento hanno intersezione non vuota

Macchina astratta concorrente senza dati condivisi:
- gli spazi di indirizzamento hanno intersezione vuota

Un sistema operativo su monoprocessore viene tradizionalmente descritto con una macchina astratta concorrente con dati condivisi

Problemi:
- realizzazione della condivisione
- realizzazione di meccanismi di accesso in mutua esclusione
- soddisfacimento di proprietà di buon funzionamento

Condivisione dei dati

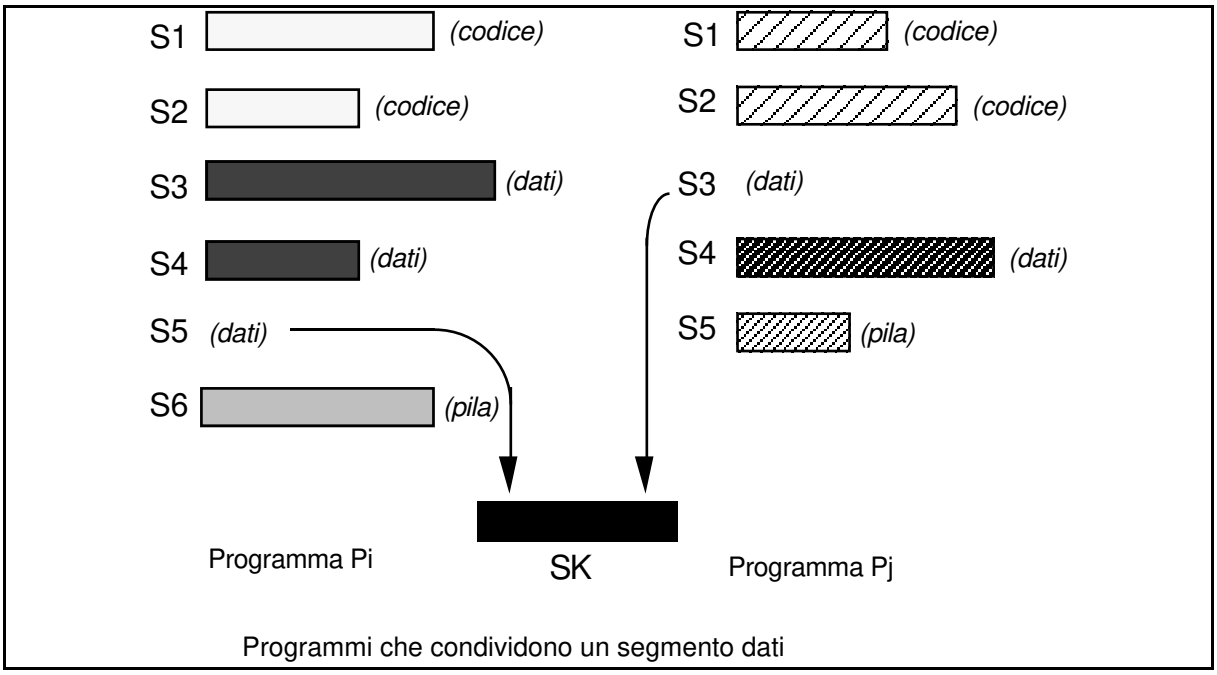
In una macchina astratta concorrente con dati condivisi:

- i programmi accedono a una stessa area contenente dati
- realizzazione semplice nei sistemi con memoria comune (regolata dai meccanismi di protezione della memoria).

Intersezione non vuota delle rispettive immagini in memoria; deriva da una intersezione non vuota degli spazi logici.

Consentita in modo naturale dal modello di memoria con segmentazione (in cui l'immagine dei processi è segmentata)

Segmento condiviso: collegato a P_i e P_j e caricato in un'unica copia; occupa una partizione che appartiene agli spazi fisici di P_i e di P_j .



Programmi che condividono un segmento dati

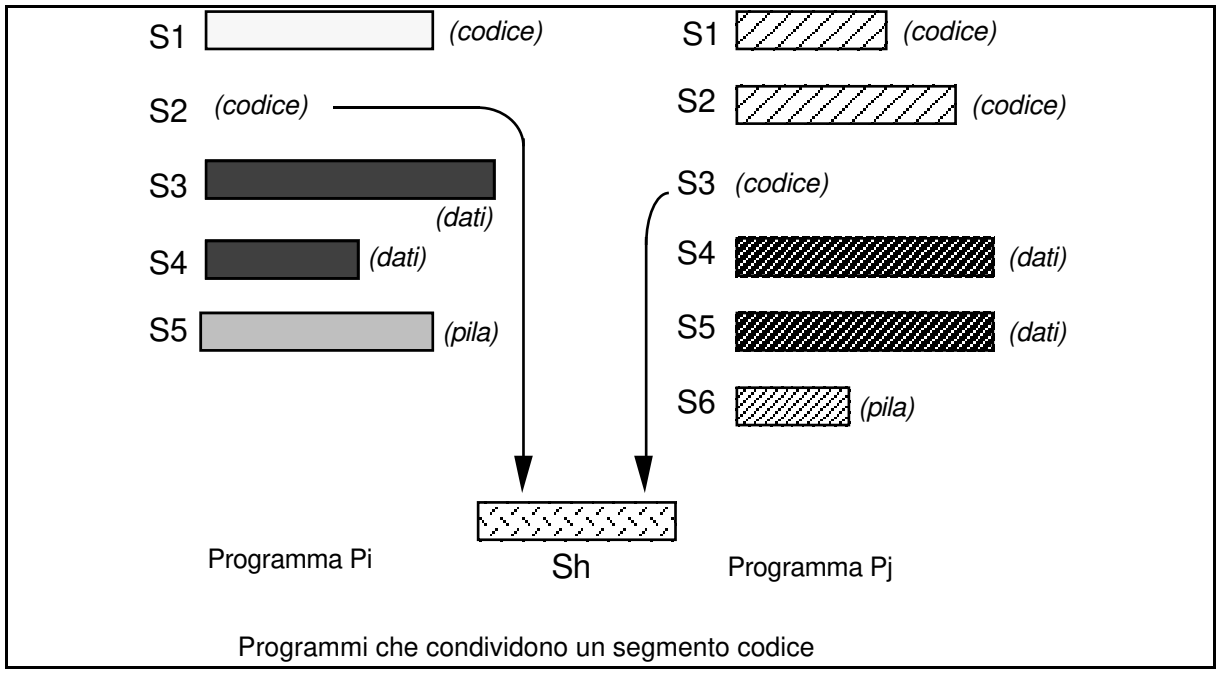
Condivisione del codice

Oltre ai dati, i processi possono condividere anche codice

In figura: (modello di caricamento con segmentazione)
- P_i e P_j condividono il segmento codice S_h , denominato S_2 nello spazio logico di P_i e S_3 nello spazio logico di P_j .

Segmento codice condiviso: tipicamente è una procedura comune:
- collegata con entrambi i programmi,
- caricata in un'unica copia.

La condivisione di codice:
- non è rilevante dal punto di vista del modello concorrente,
- non determina interazioni tra i processi.



Programmi che condividono un segmento codice

Condivisione dei meccanismi di nucleo

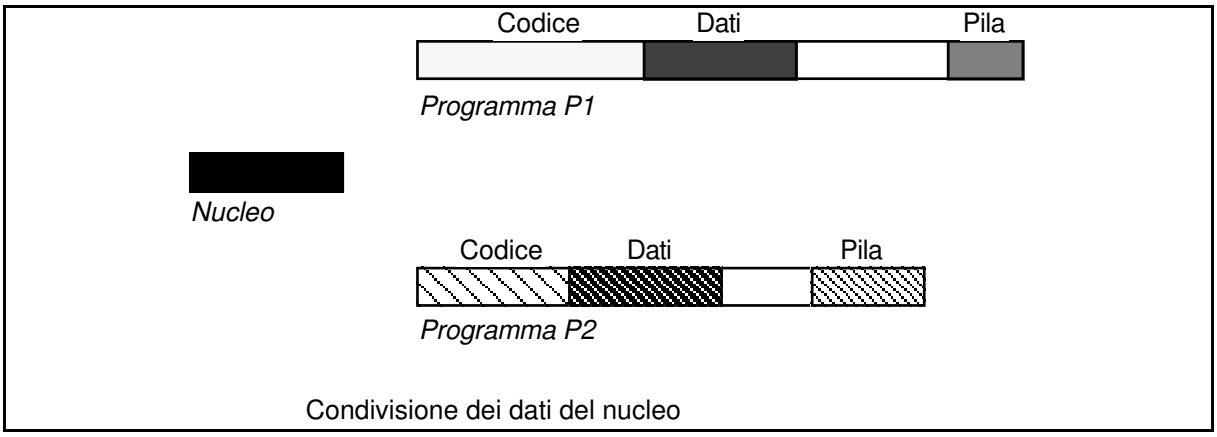
Forma "nascosta" di condivisione di dati: condivisione dei dati appartenenti ai *meccanismi* della macchina astratta concorrente, realizzati con un insieme di procedure *(nucleo* del S. O.) condivise da tutti i processi

Realizzazione con procedure e dati comuni, tipica dei sistemi con memoria comune.

Meccanismi per:
- mutua esclusione
- sincronizzazione
- comunicazione

Condivisione dei dati del nucleo:
- invisibile al livello di astrazione del modello concorrente,
(i processi utilizzano i meccanismi come *primitive)*
- rilevante ai fini della realizzazione dei meccanismi.

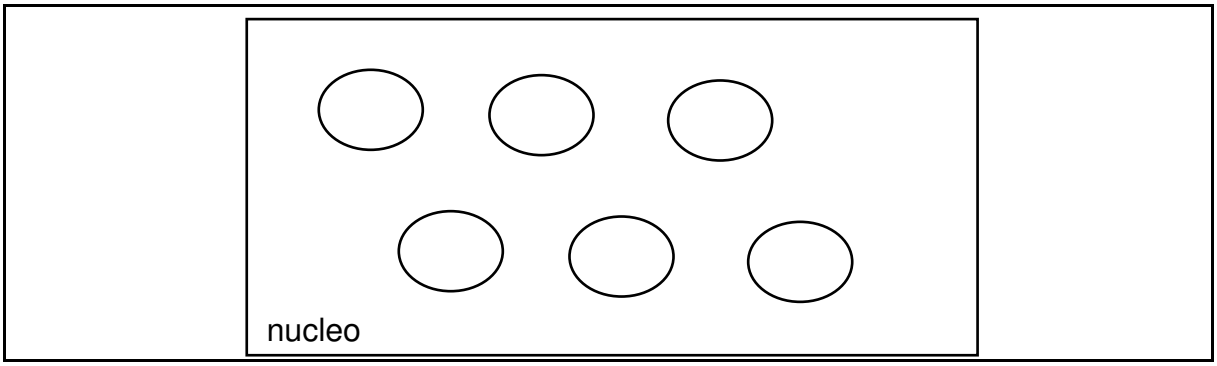
Se astraiamo dai servizi offerti da una macchina astratta concorrente, il *nucleo* del S. O. (procedure e dati) può essere concepito come una estensione dello spazio logico di ogni processo



Condivisione dei dati del nucleo

Architetture dei sistemi operativi

Un sistema operativo è un programma che in esecuzione può essere visto come composto da un *nucleo (kernel)* e da un insieme di *processi*



Il nucleo comprende:
- le strutture dati necessarie per descrivere i processi;
- i meccanismi per la generazione e la terminazione di processi
- i meccanismi di interazione;
Nei meccanismi è incorporata la funzione di gestione del processore.

I processi:
- avanzano (richiedono il processore) con legge regolata dalle transizioni di stato, dovute essenzialmente alle reciproche interazioni;
- richiedono e utilizzano alcune risorse (memoria, dispositivi, archivi, ecc.)
Le risorse sono amministrate dal sistema operativo secondo le sue politiche.

Possibili organizzazioni del sistema operativo (differiscono nella collocazione delle funzioni di gestione delle risorse)

1) modello **microkernel**
   gestione dei servizi e risorse svolta da processi; il nucleo (microkernel) si limita a realizzare i processi e i meccanismi di interazione

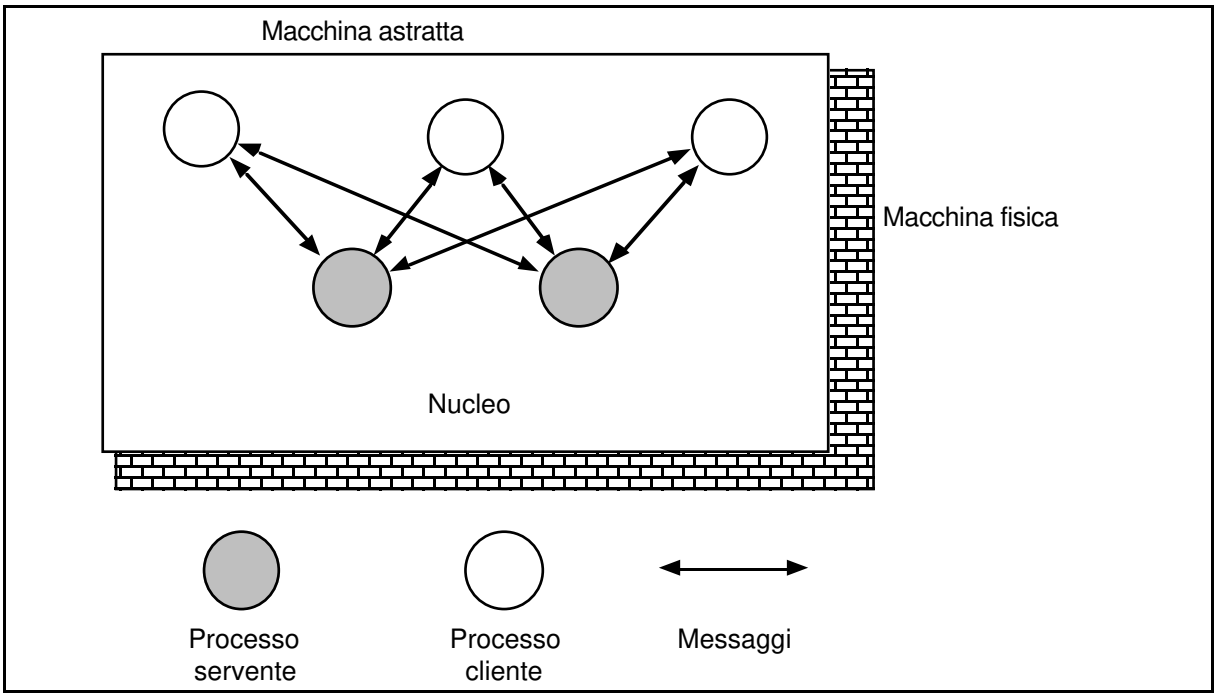
2) modello **monolitico**
   anche le funzioni di gestione delle risorse sono realizzate dal nucleo; l'intero sistema operativo tende ad essere identificato con il nucleo

Modello microkernel

Risorse gestita da processi serventi *(processi di controllo).*

I processi *clienti:*
- comunicano con i serventi per mezzo di messaggi;
- ottengono le risorse (caso delle risorse seriali) o le utilizzano (caso delle risorse condivise) esclusivamente attraverso i rispettivi processi di controllo

Il nucleo contiene essenzialmente i dati che descrivono i processi e le procedure che realizzano i meccanismi per la concorrenza.

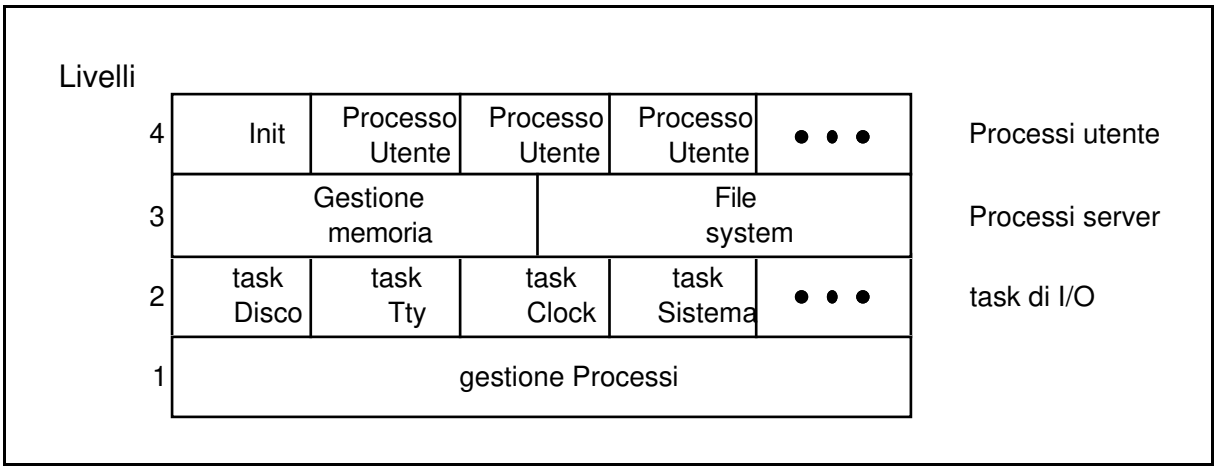


Nel modello cliente-servente:
- I processi non condividono dati (modello *privo di memoria comune:* questa affermazione non esclude la possibilità che i meccanismi del nucleo siano realizzati mediante dati condivisi);
- i dati che descrivono le risorsa appartengono ai processi di controllo; sono protetti dai meccanismi del processore e del nucleo che realizzano la separazione degli spazi fisici.

Caratteristiche del modello cliente-servente:
- separazione delle politiche dai meccanismi
- modularità;
- portabilità (anche da sistemi con memoria condivisa a sistemi distribuiti, o viceversa)

Un s.o. a microkernel: Minix

Minix è un sistema Unix-like per PC

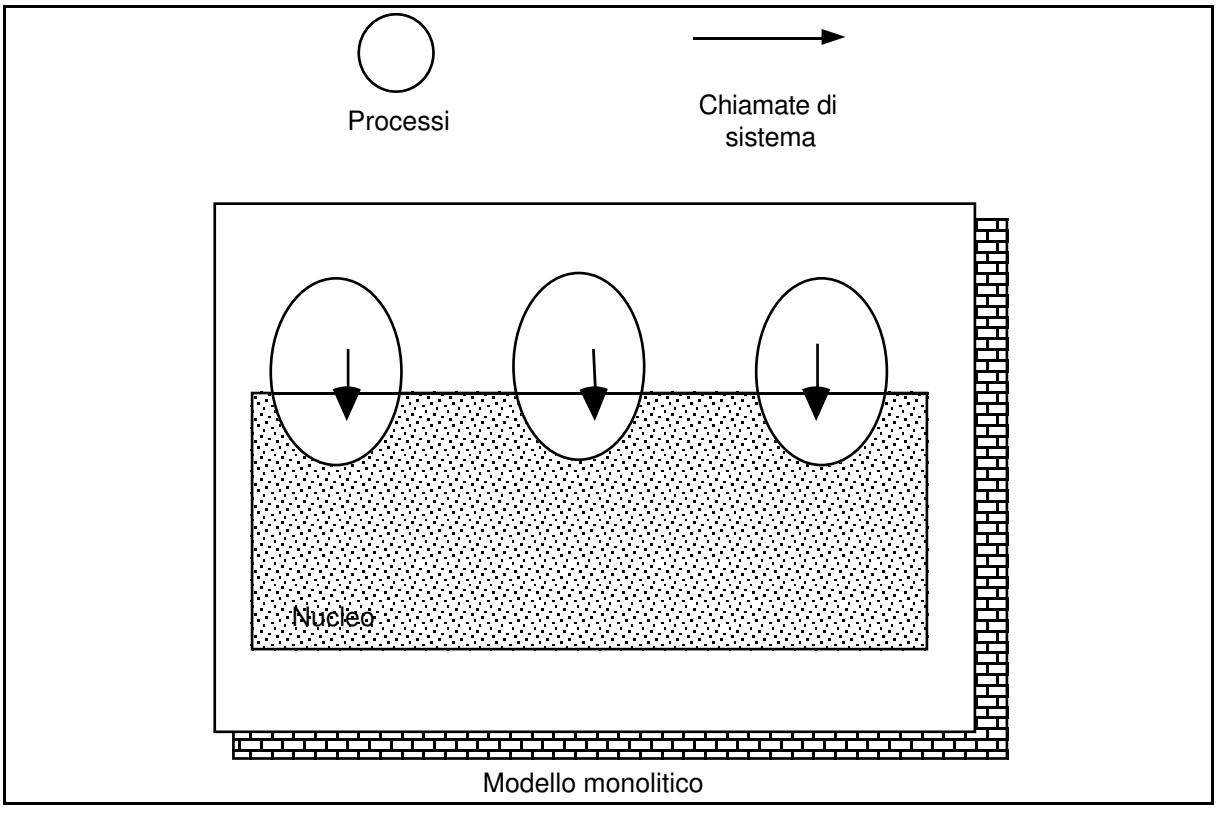


Modello monolitico

Il nucleo comprende:
- i dati che descrivono i processi e le procedure che realizzano i meccanismi *(procedure di servizio)*;
- i dati e le procedure per la gestione delle risorse *(chiamate di sistema:* realizzate come primitive)

Il nucleo ha una struttura gerarchica:
- primo livello: meccanismi del modello concorrente; interagisce con la macchina fisica;
- secondo livello: procedure corrispondenti alle chiamate di sistema; è la sola componente del nucleo visibile ai processi.

Modello monolitico: caratteristico del sistema UNIX.



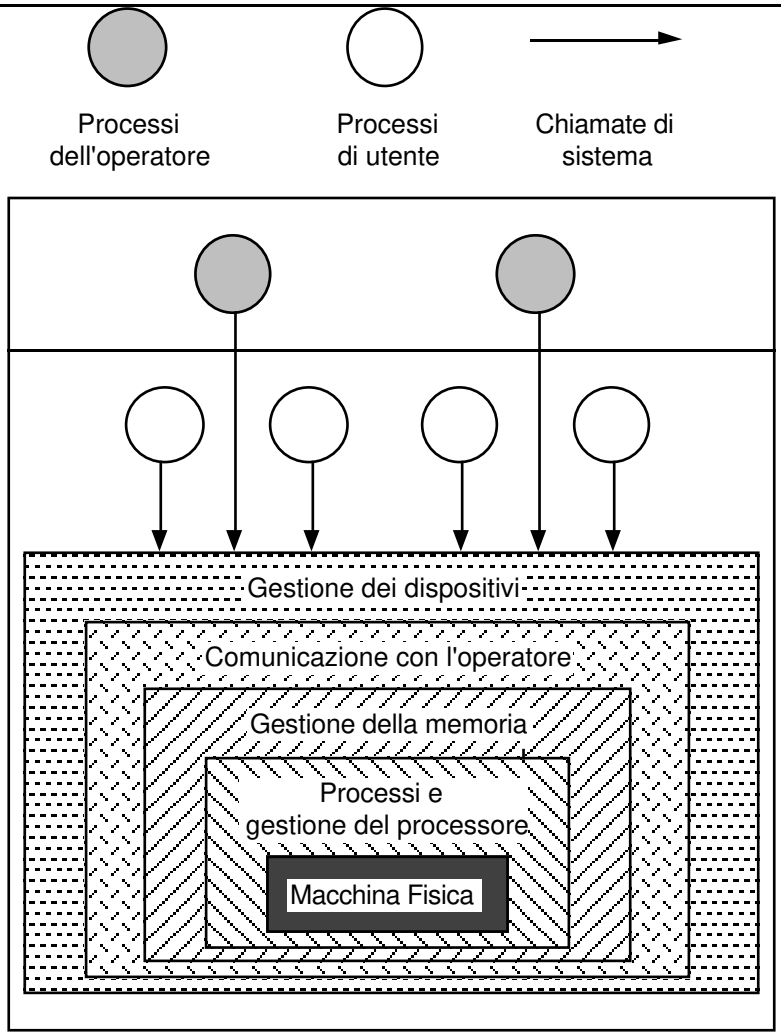
Modello monolitico

Modello monolitico

L'organizzazione gerarchica del nucleo può essere estesa al livello dei processi

- **il sistema operativo THE**, concepito per servizio batch, ha 6 livelli (i primi quattro corrispondono al nucleo)

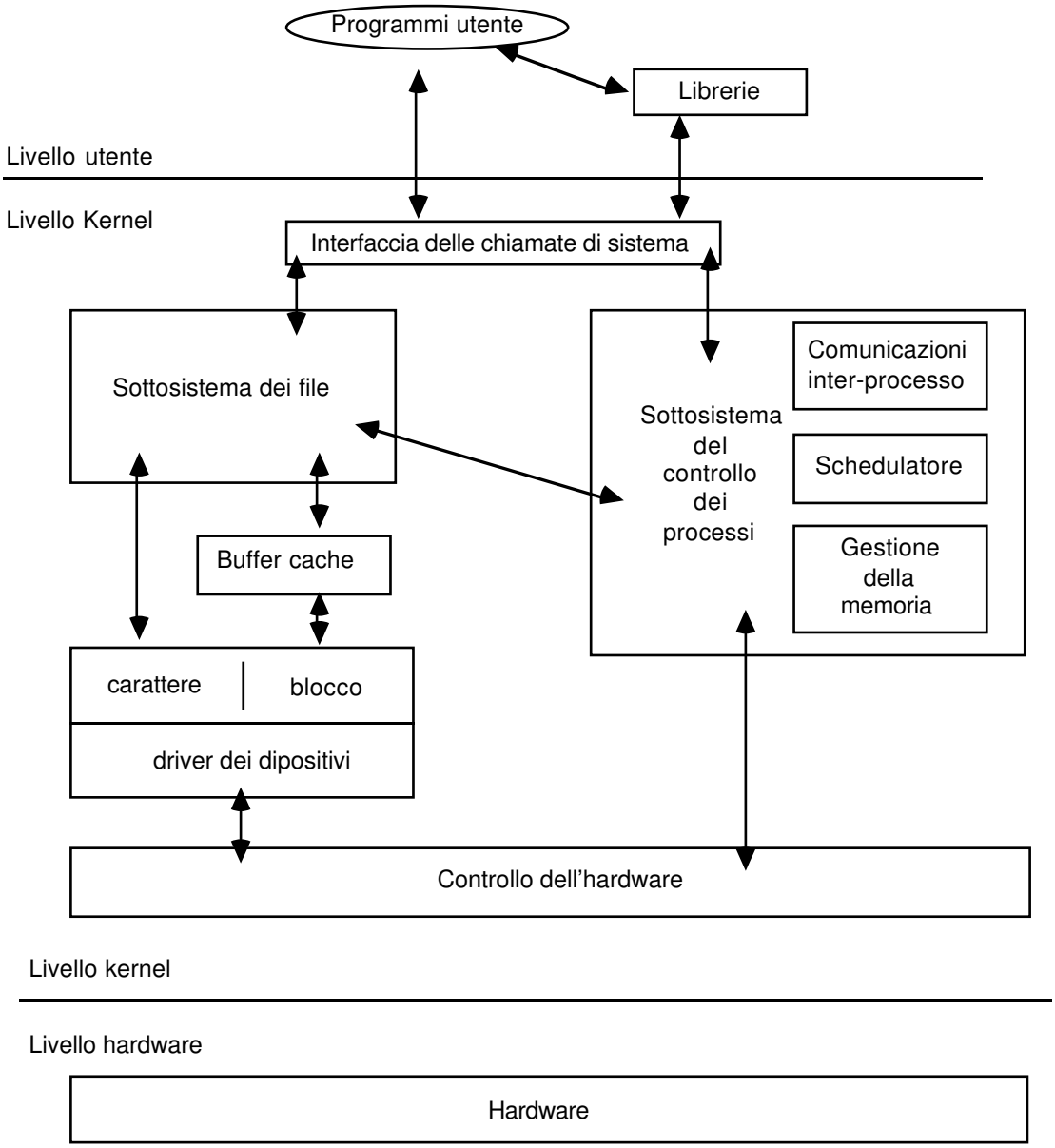
1) processi e gestione del processore (astrazione dei processi);
2) gestione della memoria (astrazione dello spazio logico);
3) comunicazione dei processi con l'operatore (astrazione di una console dedicata per ogni "interprete dei comandi");
4) gestione dei dispositivi (astrazione dei dispositivi virtuali: lettore di schede e stampante dedicati per ogni "interprete dei comandi");
5) processi "interpreti dei comandi"
6) processi dell'operatore



Organizzazione del sistema operativo THE

Architettura di Unix

Unix segue il modello monolitico



Modello con macchine virtuali

Nei precedenti modelli, un'unica astrazione (processi) realizza sia la multiprogrammazione, sia la gestione delle risorse.
Il modello con macchine virtuali (esempio: **IBM VM/370**):
- usa due distinte astrazioni.
- è basato su un nucleo *Virtual machine monitor (VM),* definito sulla macchina fisica, che realizza le *macchine virtuali*.

Macchina virtuale:
- dispone di processore, memoria, memoria secondaria e dispositivi di I/O;
- le sue unità hanno proprietà identiche a quelle della macchina fisica;
- sono realizzate dal nucleo con la gestione del processore e delle altre risorse della macchina fisica.
- come i processi, avanzano quando dispongono del processore fisico, gestito dal *VM*

All'esecuzione di un'istruzione su una macchina virtuale può corrispondere:
- l'esecuzione dell'omologa istruzione sulla macchina fisica,
- una funzione del *VM* (ad esempio, nel caso delle istruzioni che scrivono nelle porte di I/O o interrogano i segnali di interruzione).

A differenza dei processi, le macchine virtuali:
- non interagiscono direttamente; non si sincronizzano e non comunicano;
- ci sono interazioni indirette, dovute alle istruzioni intercettate dal *VM*

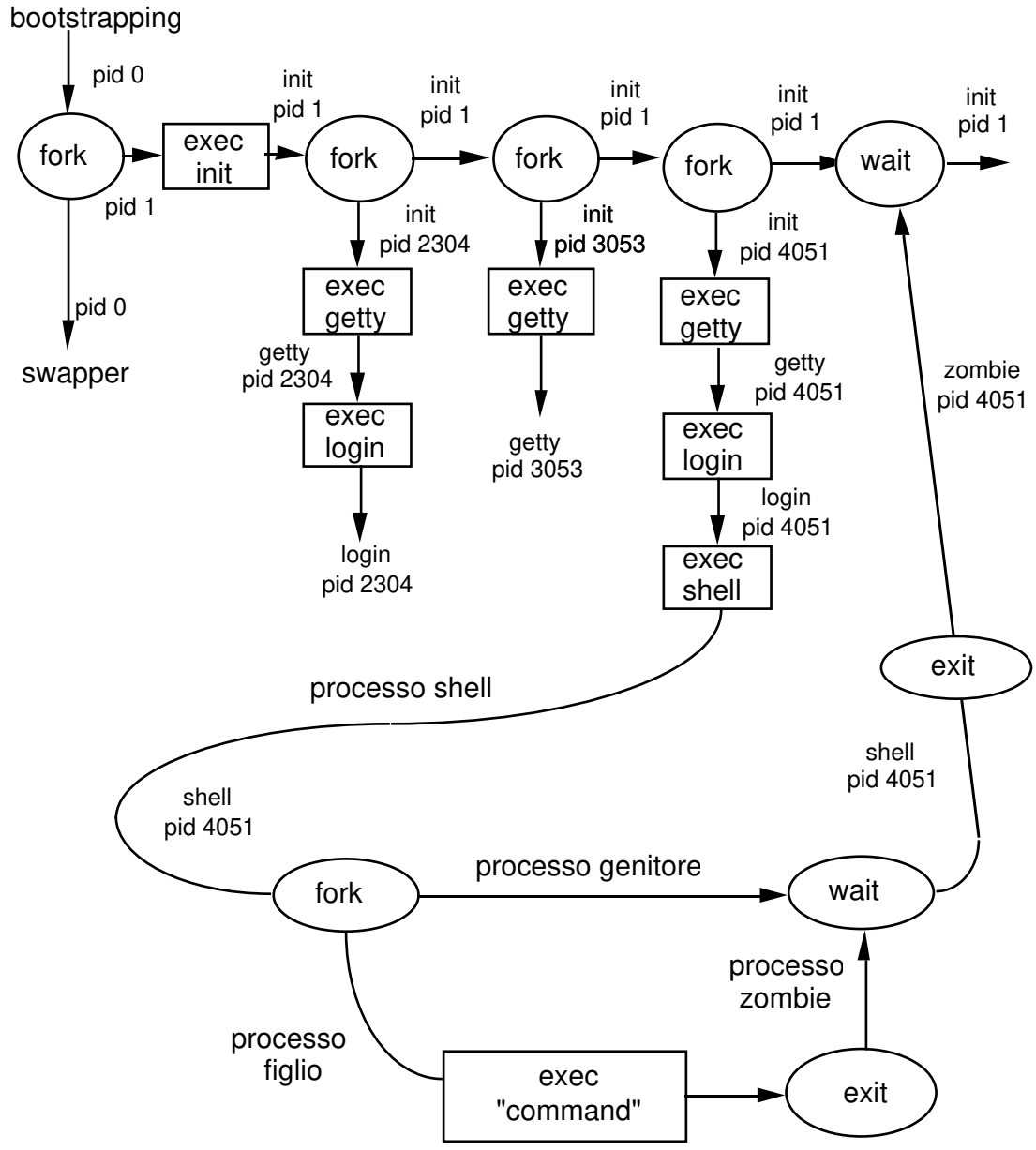
La macchina virtuale:
- replica le funzioni della macchina fisica;
- deve essere dotata a sua volta di un sistema operativo comprendente processi e funzioni di gestione delle risorse (in particolare, sistema di archiviazione)

Diverse macchine virtuali di uno stesso sistema possono avere sistemi operativi diversi (tipicamente *CMS* e *OS/360).*

- Quando un processo in esecuzione esegue una chiamata di sistema, questa viene diretta al sistema operativo della sua macchina virtuale
- Le procedure del nucleo attivate dalla chiamata di sistema possono conte-nere istruzioni che vengono intercettate dal *VM:* in tal caso attivano a loro volta le corrispondenti funzioni del nucleo che realizza le macchine virtuali

Il bootstrapping

La figura seguente riassume il bootstrapping di Unix



La funzione di schedulazione

Quando ci sono più processi, occorre che il sistema operativo scelga il processo da eseguire tra quelli pronti

In pratica l'insieme dei processi pronti corrisponde ad una agenda di compiti da eseguire; il proprietario dell'agenda (cioè il sistema operativo) può alterare a suo piacimento la sequenza di esecuzione

La scelta della sequenza esecutiva dei processi da parte del processore si chiama schedulazione

La funzione di schedulazione, applicando la propria *politica,* designa di volta in volta il processo (ovvero i processi) in esecuzione, selezionandolo (i) tra i processi pronti.

Il meccanismo di base della schedulazione si basa sulla possibilità di congelare e poi ripristinare i processi in esecuzione

Schedulazione a priorità

Semplice politica: scelta in base alla **priorità statica** dei processi.

Questa politica viene realizzata garantendo che in ogni momento il processo in esecuzione sia quello di max priorità tra tutti i processi pronti.

• Uniprocessore: il processo in esecuzione è il processo pronto di massima priorità
• Multiprocessore: gli *n* processi in esecuzione sono designati come quelli di massima priorità tra i processi pronti

La funzione di gestione del processore interviene:
- quando il processo in esecuzione si sospende: assegnazione del processore al processo pronto di massima priorità (eventualmente processo vuoto);
- quando viene riattivato un processo in attesa: eventuale prerilascio e riassegnazione del processore.

Realizzazione:
- *CodaPronti* ordinata in base alla priorità dei processi (a pari priorità, in base all'ordine di inserimento nella coda).
- Assegnazione del processore causata dalla sospensione del processo in esecuzione:

```
InEsecuzione:= EstrazioneDaCodaPronti;
ripristino(InEsecuzione).
```

- Prerilascio e riassegnazione causati dalla riattivazione di un processo:

```
if primo(CodaPronti)↑.priorità > InEsecuzione↑.priorità
  then begin
    salvataggio(InEsecuzione);
    InserzioneInCodaPronti(InEsecuzione);
    InEsecuzione:= EstrazioneDaCodaPronti;
    ripristino(InEsecuzione)
  end
```

Schedulazione a tempo

Se tutti i processi hanno uguale priorità la politica a priorità si riduce a un turno di assegnazione *(RoundRobin).*

Di solito si pone un limite alla permanenza in esecuzione *(quanto di tempo).*

Alla scadenza del quanto di tempo:

```
salvataggio(InEsecuzione);
InserzioneInCodaPronti(InEsecuzione);
InEsecuzione:= EstrazioneDaCodaPronti;
ripristino(InEsecuzione).
```

La scadenza del quanto di tempo viene segnalata da una interruzione del dispositivo orologio

La schedulazione dei processi

In un sistema di elaborazione il numero di processi da eseguire è maggiore del numero di processori disponibili

Il compito dello *scheduler* è di esaminare lo stato dei processi e delle CPU e di ridefinirne l'allocazione sulle CPU in funzione di una certa politica di scheduling

Di solito si impiegano strategie diverse per diversi tipi di processi. Ad esempio, si distingue di solito tra processi batch e interattivi, tra processi di sistema e utente, o tra processi di gestione dell'I/O (IO bound) e processi che gestiscono attività puramente computazionali (CPU bound)

Usando un meccanismo generale di priorità e manipolando opportuamente i timer di sistema, possiamo progettare un'organizzazione generale per gestire una varietà di politiche di schedulazione. Possiamo definire uno scheduler universale, scegliendo opportunamente tre parametri fondamentali:

- una modalità di decisione
- una funzione di priorità
- una regola di arbitraggio

In alcuni momenti specificati dalla modalità di decisione, lo scheduler universale valuta la funzione di priorità per tutti i processi attivi; tale funzione restituisce per ciascun processo un valore che è la priorità corrente.

I processi a più alta priorità ottengono le CPU disponibili. La regola di arbitraggio si applica se esistono più processi di pari priorità.

Possiamo definire diverse politiche di schedulazione in base a diverse modalità di decisione, funzioni di priorità e regole di arbitraggio.

La modalità di decisione

La modalità di decisione specifica i momenti in cui occorre valutare e comparare le priorità dei processi pronti, allo scopo di scegliere quelli da eseguire

Ci sono due modalità principali: senza prerilascio e con prerilascio

Nel primo caso un processo, una volta iniziato, resta in esecuzione finché termina oppure si blocca autonomamente perché richiede qualche servizio al sistema operativo.

Gli algoritmi senza prerilascio sono semplici, ma non accettabili in sistemi in tempo reale o a condivisione di tempo. In questi tipi di sistemi si usa quasi sempre invece una modalità con prerilascio, che permette di fermare un processo in esecuzione se si presenta un processo di priorità maggiore.

La decisione di prerilascio si effettua o quando arriva un nuovo processo, o quando viene risvegliato un processo per effetto di una interruzione, o periodicamente ogni volta che un processo è restato in esecuzione per q secondi (in quest'ultimo caso q è il quanto di tempo).

Un'altra possibile strategia consiste nell'effettuare un prerilascio quando la priorità di un processo pronto aumenta fino a superare quella del processo in esecuzione. Questo capita nei sistemi dove la priorità è dinamica.

Le politiche con prerilascio sono più costose di quelle senza prerilascio a causa del tempo di gestione dei processi, della maggior complessità dello scheduler, e dello swapping delle immagini tra memoria secondaria e principale.

Una via di mezzo tra queste due classi di politiche è costituita dalla classe delle politiche selettive, dove a ciascun processo è associata una coppia di valori (a,p) dove a indica se il processo può essere prerilasciato, mentre b indica se il processo può provocare il prerilascio. L'attributo a è un attributo del processo, mentre p è una priorità. Questo schema può essere generalizzato ad una matrice in cui le case (i,j) determinano se un processo i può prerilasciare un processo j

La funzione di priorità e la regola di arbitraggio

La **funzione di priorità** è una funzione arbitraria calcolata su parametri di sistema e attributi di processo; alcuni dei parametri che possono determinare la priorità sono i seguenti:

- requisiti di memoria: è importante nei sistemi batch e nei sistemi interattivi;

- tempo di servizio ottenuto: di solito è il parametro più importante;

- tempo reale nel sistema: è il tempo di servizio ottenuto più il tempo di attesa di servizio;

- tempo totale necessario: parametro ipoteticamente noto in anticipo. Di solito si assegna maggior priorità ai processi che richiedono meno tempo, riducendo il tempo medio di attesa.

- priorità esterne: permettono di differenziare varie classi di utenti e di programmi di sistema. Tipicamente si assegna maggior priorità ai processi in tempo reale, poi agli interattivi, e da ultimo ai processi batch.

- carico del sistema: sotto carico pesante il sistema tenta di mantenere alta la priorità dei processi importanti, o alternativamente può tentare di migliorare il tempo medio di servizio per tutti i processi aumentando il quanto di tempo.

- tempestività della risposta: l'urgenza di effettuare un certo compito può variare col tempo. Possono essere imposte delle scadenze temporali per garantire il completamento di computazioni critiche dal punto di vista temporale.

La regola di arbitraggio serve per risolvere i conflitti tra processi pronti di pari priorità. Se tali conflitti sono poco probabili, la scelta può essere casuale.

In molti casi si preferisce una regola cronologica (FIFO).

Nei sistemi basati su quanto di tempo si usa di norma la regola ciclica round-robin.

Algoritmi di schedulazione

Qualsiasi algoritmo di scheduling può essere definito scegliendo una modalità di decisione, una funzione di priorità e una regola di arbitraggio.

In molti casi la funzione di priorità è della forma P(a,r,t) dove a è il tempo di servizio ottenuto, r è il tempo reale passato nel sistema e t è il tempo di servizio necessario richiesto dal processo.

Nei sistemi interattivi t è non calcolabile e quindi i parametri si riducono a due soli: a e r.

Vediamo una sinossi di alcuni dei principali algoritmi di scheduling:

Politica	**Funzione priorità**	**modalità di decisione**	**regola di arbitraggio**
FIFO	-r	senza prerilascio	casuale
LIFO	r	senza prerilascio	casuale
SJN	-t	senza prerilascio	casuale/cronologica
SRT	a-t	con prerilascio (all'arrivo)	casuale/cronologica
RR	costante	con prerilascio (quanto di tempo)	ciclica
MLF	dinamica su più livelli	con prerilascio (quanto di tempo)	ciclica/cronologica
PD	$r-f^{-1}(a)$	con prerilascio (quanto di tempo)	casuale

Algoritmi di schedulazione

First-in/First out (FIFO) Questa strategia alloca i processi in ordine di arrivo, quindi la priorità è funzione solo di r.

Last-in/First out (LIFO) Questa strategia alloca i processi in ordine inverso di arrivo, ovvero la priorità dei processi decresce nel tempo. Non è usata comunemente.

Shortest Job Next (SJN) Si ipotizza che il tempo di servizio t sia noto in anticipo; si applica ai soli processi batch.

Shortest remaining time (SRT) È la versione con prerilascio di SJN: assegna la priorità più alta ai processi che hanno bisogno di minor tempo per terminare.

Round Robin (RR). Viene imposto un quanto di tempo ai processi, che hanno di solito tutti la stessa priorità. La schedulazione avviene allo scadere del quanto di tempo. Il quanto di tempo q può essere costante o variabile in funzione del carico del sistema. Se chiamiamo Q il tempo totale per far girare una volta tutti gli n processi pronti, e lo teniamo costante mentre n varia, allora possiamo avere q=Q/n.

Priorità Statica (PS): i processi vengono staticamente associati ad un valore di priorità.

Multilevel Feedback (MLF) Si basa su n diversi livelli di priorità. Ad ogni livello Π corrisponde un quanto di tempo T_Π, che è il massimo periodo di tempo di processore concesso ai processi del livello Π, ed inoltre è un multiplo del quanto di tempo atomico q; e se un processo supera T_Π, la sua priorità diventa Π-1.

Uno schema possibile è di porre $T_\Pi = 2^{n-\Pi} \times T_n$, dove n è il numero di livelli di priorità e T_n è il quanto concesso al livello massimo. In tal caso un processo del livello n ottiene T_n, $2T_n$ al livello n-1, $4T_n$ al livello n-2, e così via. La regola di arbitraggio può essere FIFO o ciclica.

Esempio:

Cinque processi A, B, C, D, E richiedono più o meno nello stesso momento di essere eseguiti. La stima iniziale di tempo di CPU richiesto è rispettivamente 10, 6, 2, 4, 8 minuti.
Determinare il tempo medio di servizio TMS nei seguenti casi:

a) FIFO, nell'ordine A B C D E
b) SJN
c) RR1, con turno nell'ordine A B C D E e quanto di tempo 1
d) RR2, con turno nell'ordine A B C D E e quanto di tempo 2
e) PS, assumendo rispettivamente i valori 3, 5, 2, 1, 4

TMS(FIFO): (10+16+18+22+30)/5 = 96/5

TMS(SJN): (2+6+12+20+30)/5 = 70/5

TMS(RR1): (8+17+23+28+30)/5 = 106/5

TMS(RR2): (6+14+20+26+28)/5 = 94/5

TMS(PS): (6+14+24+26+30)/5 = 100/5

Schedulazione policy-driven (PD)

Una schedulazione PD utilizza una funzione f(r) chiamata funzione della politica che prescrive una correlazione tra tempo reale r che un processo ha passato nel sistema e la quantità desiderabile di servizio che il processo dovrebbe aver ricevuto in tale periodo di tempo.

I servizi si possono misurare con una funzione s(r) che calcola le unità di tempo durante le quali una certa risorsa è usata dal processo.

Ad esempio, una funzione s(r) che tiene conto della risorsa processore *CPU* e della risorsa processore d'ingresso/uscita *iop* con pesi w_{cpu} e w_{iop} rispettivamente, è la seguente:

$s(r) = w_{cpu} * unità_{cpu}(r) + w_{iop} * unità_{iop}(r)$

dove $unità_{cpu}$ e $unità_{iop}$ sono rispettivamente i tempi di servizio ottenuti sui processori centrale e di I/O.

La priorità di un processo dipende da quanto l'esecuzione del processo stesso si adegua alla funzione politica prescritta (ovvero sulla distanza tra la funzione politica f e la funzione dei servizi ottenuti s).

Supponiamo per semplicità di considerare solo la CPU: allora s(r) = a.
Sia $f(r_t)$ una politica che descrive la quantità di tempo di CPU che un processo dovrebbe aver ottenuto al tempo r_t.
Sia $f^{-1}(a_t)$ l'inversa di f, che restituisce il momento r_t in cui il processo dovrebbe aver ottenuto la quantità di servizio prevista dalla politica.

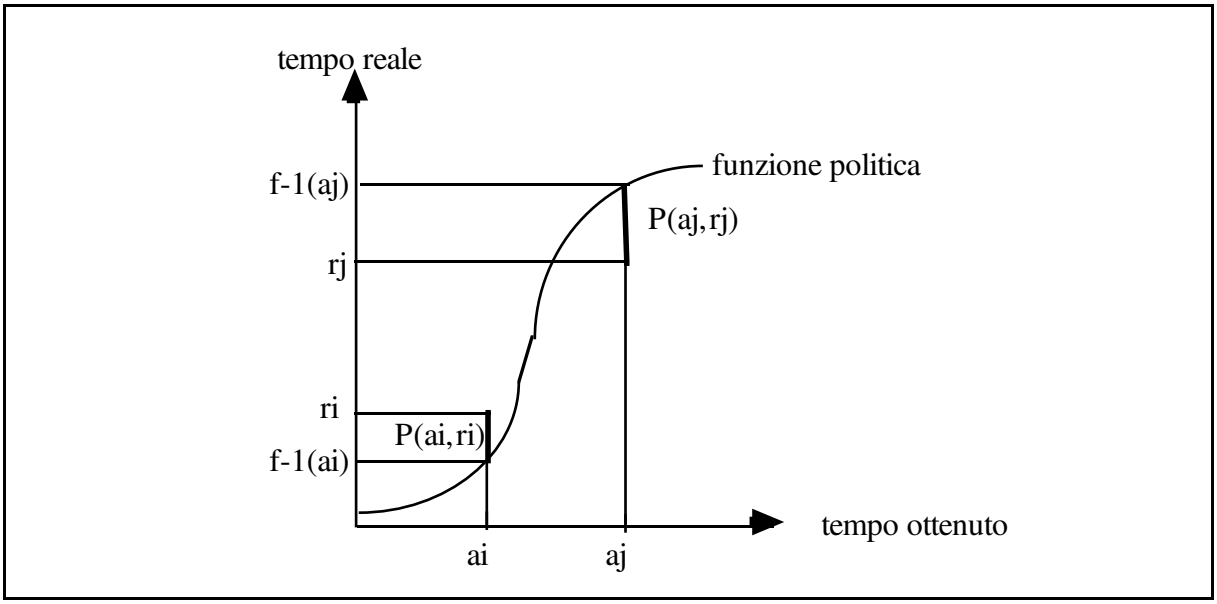
Idealmente a_t dovrebbe essere pari a $f(r_t)$ per ogni t. Sfortunatamente, può capitare che il carico del sistema superi la capacità di calcolo effettiva e quindi certi processi vengano ritardati, oppure un processo può superare il tempo prescritto dalla politica e quindi dovrebbe poi ottenerne di meno.

Schedulazione policy-driven (PD)

Le deviazioni modificano automaticamente la priorità, definita come differenza tra il tempo davvero ottenuto e quello idealmente prescritto.

$P(a_t,r_t) = r_t - f^{-1}(a_t)$

Da questa definizione segue che la priorità di un processo aumenta sempre fintantoche non riceve alcun servizio; conseguentemente, dovrebbe restare in esecuzione finché la sua priorità non viene azzerata.



Nella figura si vede il comportamento della priorità, che può essere positiva o negativa.

Confronto tra i metodi di scheduling

Gli algoritmi FIFO, LIFO, SJN e SRT sono stati sviluppati ed applicati principalmente per sistemi di processi batch.

I primi due sono i più semplici da implementare mediante liste di processi. SJN e SRT richiedono una lista ordinata, il che richiede tempo per il riordinamento tutte le volte che arriva un processo. SRT può richiedere il prerilascio, il che aumenta il tempo di calcolo da dedicare alla schedulazione.

Il criterio principale per paragonare i quattro algoritmi è il tempo medio di computazione calcolato su tutti i processi. Non è difficile costruire simulazioni in cui si vede che SJN e SRT sono superiori a FIFO.

Nei sistemi interattivi a tempo condiviso il parametro più importante è il tempo di risposta, che deve essere accettabile per gli utenti.

Ciò richiede un algoritmo con prerilascio a tempi ben definiti: si userà un algoritmo RR,MLF o PD. Il più semplice da implementare è RR, ed è infatti il più diffuso.

In tal caso il parametro di progetto più importante è il quanto di tempo q: se q è grande si riduce il tempo perso nelle schedulazioni, ma il tempo di risposta verrà rallentato. Se q tende all'infinito il processo non viene mai prerilasciato e tale algoritmi diventa FIFO. Se q tende a 0 il tempo perso nelle schedulazioni cresce insopportabilmente: bisognerà cercare un compromesso accettabile.

La discriminazione tra diverse classi di processi, specie tra CPU bound e IO bound, suggerisce algoritmi di tipo MLF.

La classe PD è la più generale perché permette la definizione di una politica di priorità arbitraria. Viene molto usata nei sistemi in tempo reale. Ad esempio, una funzione che rimane costante per tutta la vita del processo garantisce attenzione immediata dalla CPU e previene i sovraccarichi quando sono richiesti periodi più lunghi di servizio; tale funzione si usa con processi real time brevi.

Una funzione che aumenta fortemente nei primi millisecondi e poi rallenta va bene per i processi interattivi. Un processo batch potrebbe avere una funzione con valore iniziale basso ma con incremento costante.