



UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

*Appunti di
Sistemi Operativi I*

Anno 2021

Valentino Bocchetti

Contents

1 Il sistema operativo	13
1.1 Altre definizioni	14
2 Sistemi mainframe	14
2.1 Configurazione della memoria per un sistema a lotti (sistema batch)	14
3 Sistemi multiprogrammati	14
4 Sistemi a partizione del tempo d'elaborazione	14
5 Sistemi da scrivania	15
6 Sistemi paralleli	15
7 Sistemi distribuiti	16
7.1 Ambiente di elaborazione (Client-Server o Peer-to-Peer)	16
8 Batterie di sistemi (sistemi cluster)	17
9 Sistemi d'elaborazione real-time	17
10 Sistemi palmari	17
11 Ambienti d'elaborazione	17
12 Architettura di un sistema di calcolo	18
12.1 Funzionamento	18
12.2 Gli interrupt	18
12.3 Gestione dell'interruzione	19

12.4 Diagramma temporale delle interruzioni per un singolo processo che emette dati	19
12.5 Struttura di I/O	19
12.6 Accesso alla memoria (DMA)	20
12.7 Struttura della memoria	21
12.8 Gerarchia delle memorie	22
12.9 Migrazione di un intero n da un disco a un registro	22
12.10 Duplice modo di funzionamento	22
12.11 Uso di una chiamata del sistema per l'esecuzione di una chiamata di I/O	24
12.12 Protezione della memoria	24
12.13 Uso di un registro di base e un registro di limite	25
12.14 Architettura di protezione degli indirizzi	25
12.15 Protezione HW	25
12.16 Protezione della CPU	26
12.17 Struttura di una rete locale	26
13 Gestione dei processi	27
13.1 Definizione	27
13.2 Gestione della memoria centrale	27
13.3 Gestione dei file	28
13.4 Gestione del sistema I/O	28
13.5 Gestione della memoria secondaria	28
13.6 Reti (sistemi distribuiti)	29
13.7 Sistemi di protezione	29
13.8 Interprete dei comandi	29

13.9	Servizi di un sistema operativo	29
13.10	Funzioni addizionali di un SO	30
13.11	Chiamate del sistema	30
13.12	Passaggio di parametri in forma di tabella	30
13.13	Tipi di chiamate del sistema	30
13.14	Struttura ed esecuzione dei programmi in MS-DOS	31
13.15	Struttura ed esecuzione dei programmi nei sistemi UNIX	32
13.16	Metodo stratificato	33
13.17	Macchine virtuali	33
13.17.1	Macchina virtuale Java	34
13.18	Struttura stratificata dell' OS/2	34
13.19	Modelli di comunicazione	35
13.20	Programmi di sistema	35
13.21	Microkernel (Orientamento a micronucleo)	35
13.22	Scopi della progettazione	36
13.23	Meccanismo e criteri	36
13.24	Realizzazione	36
13.25	Generazione di sistemi (SYSGEN)	36
14	I processi nel dettaglio	36
14.1	Stato di un processo	37
14.2	Blocco di controllo di un processo (PCB)	37
14.3	CPU e la commutazioni tra processi	38
14.4	Code di scheduling dei processi	39

14.5 Diagramma di accodamento per lo scheduling dei processi	40
14.6 Scheduler	41
14.7 Cambio di contesto	41
14.8 Creazione di un processo	41
14.9 Terminazione di un processo	42
14.10 Processi cooperanti e indipendenti	43
14.11 Processo produttore e consumatore	43
14.11.1 Memoria limitata - Soluzione con memoria condivisa	43
14.12 Comunicaione tra processi (IPC)	45
14.12.1 Mailbox	45
14.12.2 Code di messaggi (buffering)	45
14.12.3 Comunicazione nei sistemi Client-Server	46
15 I Thread	46
15.1 Thread Linux & Unix	47
15.2 Definizione	47
15.3 Parallelismo	47
15.3.1 Sistema monoprocesso time-sliced	47
15.4 Sistema preemptive e non preemptive	48
15.5 Motivazioni dei thread	48
15.6 Processi a singolo thread e multithread	49
15.7 Vantaggi e svantaggi dei thread	49
15.8 Thread a livello utente	49
15.9 Thread a livello del nucleo	49

15.10	Modelli di programmazione multithread	50
15.10.1	Modello da molti a uno	50
15.10.2	Modello da uno a uno	50
15.10.3	Modello da molti a molti	50
15.11	Pthreads	50
15.11.1	I thread nel linguaggio java	50
15.12	Questioni di programmazione multithread	50
16	Scheduling della CPU	51
16.1	Concetti Fondamentali	51
16.2	Scheduler della CPU	52
16.3	Il dispatcher	52
16.4	Criteri di scheduling	52
16.5	Sceduling FCFS (first-come, first-served)	52
16.6	Scheduling shortest-job-first	53
16.7	Scheduling per priorità	54
16.8	Scheduling a code multiple	54
16.9	Scheduling per sistemi multi CPU	55
16.10	Scheduling per sistemi Real time	55
16.11	Latenza di Dispatch nel dettaglio	56
16.12	Valutazione degli algoritmi	56
17	Sincronizzazione dei processi	56
17.1	Race condition	56
17.2	Soluzione al problema della sezione critica	57

17.3 Architetture di sincronizzazione	57
17.4 Semafori	57
17.5 Stallo e attesa indefinita	57
17.6 Tipi di semaforo	58
17.7 Problemi tipici di sincronizzazione	58
17.8 Regioni critiche	58
18 Stallo dei processi	58
18.1 Caratterizzazione delle situazioni di stallo	58
18.2 Grafo di assegnazione delle risorse	59
18.3 Metodi per la gestione delle situazioni di stallo	59
18.4 Stato sicuro	60
19 Gestione della memoria	60
19.1 Spazio di indirizzi di un processo	60
19.2 Indirizzo logico	62
19.2.1 Segment selector	62
19.2.2 Offset	63
19.2.3 Descrittore di segmento	63
19.2.4 Traduzione degli indirizzi logici in indirizzi fisici	64
19.2.5 Privilegi di esecuzione	65
19.2.6 Tabelle dei Descrittori di Segmento	65
19.3 Swapping	67
19.4 Sincronizzazione dei processi	67
19.5 Segmentazione	69

19.6 Segmentazione vs Paginazione	69
19.7 Page	69
19.7.1 Page Frame	71
19.7.2 Paging Unit	71
19.7.3 La tabella delle pagine	71
19.8 Tipi di Paginazione	72
19.9 TLB	72
19.10 Protezione della memoria	72
19.11 I file hash	72
19.11.1 Funzione di accesso h	72
19.11.2 Divisione	73
19.11.3 Risoluzione delle Collisioni	73
19.11.4 Tabella Hash per la runqueue	73
20 La memoria virtuale	73
20.1 Algoritmo FIFO (First-in First-out)	74
21 Interfaccia del File System	74
21.1 Concetto di file	74
21.2 Struttura dei file	75
21.3 Attributi dei file	75
21.4 Operazione sui file	75
21.5 Metodi d'accesso	76
21.6 Simulazione dell'accesso sequenziale a un file ad accesso diretto	77
21.7 Struttura di directory	77

21.8 Tipica organizzazione di un file system	77
21.9 Informazioni sui file nella directory	77
21.10 Operazioni eseguite su una directory	78
21.11 Organizzare (logicamente) una directory	78
21.12 Directory a singolo e doppio livello	78
21.13 Struttura di directory ad albero	79
21.14 Struttura di directory a grafo aciclico	80
21.15 Montaggio di un file system	81
21.16 Condivisione di file	81
21.17 Protezione dei file	81
21.17.1 Modalità d'accesso	81
22 Struttura del file system	82
22.1 Struttura del file	82
22.2 File system stratificato	82
22.3 File system virtuali (VFS)	82
22.3.1 Schema di un file system virtuale	83
22.4 Realizzazione delle directory	83
22.5 Funzioni di Hash	83
22.5.1 Collisione	83
22.5.2 Proprietà	84
22.5.3 Costruzione della funzione	85
22.5.4 Cifrari a blocchi	85
22.6 Metodi di assegnazione	85

22.6.1 Assegnazione contigua	85
22.6.2 Assegnazione concatenata	85
22.6.3 Assegnazione indicizzata	86
22.7 Gestione dello spazio libero	86
22.8 Efficienza e prestazioni	87
22.9 I/O senza una buffer cache unificata	87
22.10 Ripristino	87
22.11 NFS	87
22.12 Protocollo di montaggio	87
22.13 Protocollo NFS	88
22.14 Architettura NFS	88
22.15 Traduzione dei nomi di percorso	88
23 Sistemi di I/O	88
23.1 Architetture e dispositivi di I/O	88
23.2 Tipica struttura del bus di un PC	89
23.3 Indirizzi delle porte dei dispositivi di I/O nei PC (elenco parziale)	89
23.4 Interrogazione ciclica (polling)	90
23.5 Interrogazione (interrupt)	90
23.6 Cicli di I/O basato sulle interruzioni	90
23.7 Accesso diretto alla memoria (DMA)	91
23.8 Interfaccia di I/O per le applicazioni	91
23.9 Struttura relativa all' I/O di un nucleo	92
23.10 Dispositivi con trasferimento a blocchi o a caratteri	92

23.11	Dispositivi di rete	92
23.12	Orologi e temporizzatori	93
23.13	I/O bloccante e non bloccante	93
23.14	Sottosistema per l' I/O del nucleo	93
23.15	Gestione degli errori	93
23.16	Strutture dati del nucleo	94
23.17	Strutture dati del nucleo per la gestione dell' I/O in UNIX	94
23.18	Struttura di STREAMS	95
23.19	Prestazioni	95
23.20	Comunicazione fra calcolatori	96
23.21	Migliorare le prestazioni	96
23.22	Successione delle funzionalità dei servizi di I/O	97
24	Memoria Secondaria e terziaria	97
24.1	Struttura dei dischi	97
24.2	HDD vs SSD	97
24.3	Struttura HDD	98
24.4	Floppy-disk, CD-ROM, Dischi rimovibili, Dischi WORM, Nastri	98
24.5	Hard disk Z-CAV	98
24.6	Partizionamento di un Hard Disk	98
24.7	Scheduling dell'Hard Disk (tradizionale)	99
24.8	Metodologie di scheduling	99
24.9	Scheduling per scansione circolare (C-SCAN)	99
24.10	Scheduling C-LOCK	99

24.11	Gestione dell'unità a disco	99
24.12	DOS Partition	100
24.12.1	Boot Code	101
24.13	Gestione dell'area di swap	101
24.14	Configurazione RAID (Redundant Array of Independent/inexpensive Disk)	102
24.15	Connessione dei dischi	102
24.16	Compiti del sistema operativo	102

1 Il sistema operativo

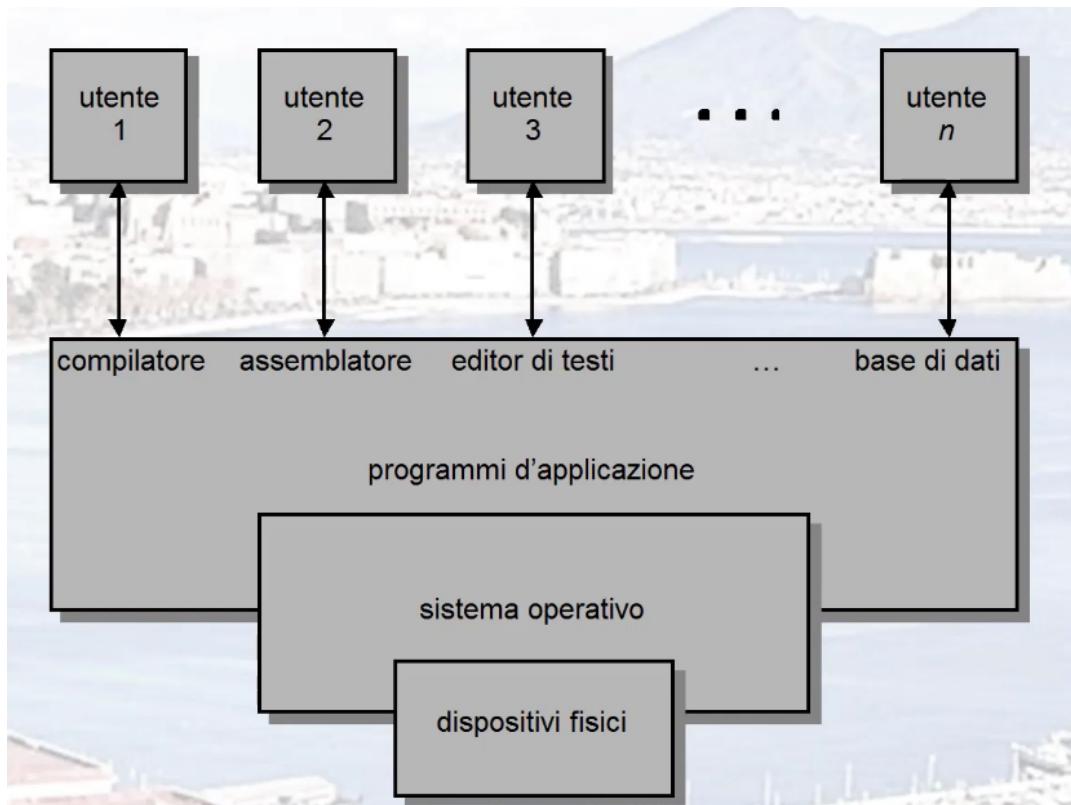
Definiamo un sistema operativo come un insieme di programmi che funge da intermediario tra un utente e gli elementi fisici di un calcolatore (hw)

Obiettivi del SO

- Eseguire programmi di applicazione e rendere più facile la risoluzione dei problemi che gli utenti devono affrontare
- Rendere il sistema conveniente da utilizzare
- Garantire un utilizzo efficiente dei dispositivi fisici

Componenti

- Dispositivi fisici (HW) → CPU, memoria, I/O;
- SO → controlla e coordina l'uso dei dispositivi da parte dei programmi d'applicazione per gli utenti;
- Programmi d'applicazione → compilatori, sistemi di BD, videogames, etc...;
- Utenti → persone, macchine, altri calcolatori



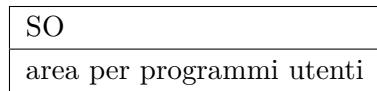
1.1 Altre definizioni

- Assegnatore di risorse → gestisce e assegna le risorse;
- Programma di controllo → controlla l'esecuzione dei programmi utenti e le operazioni dei dispositivi di I/O;
- Nucleo (**kernel**) → il solo programma che funziona sempre nel calcolatore

2 Sistemi mainframe

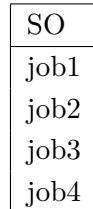
Sono sistemi di grande volume (*es* centro di calcolo) che hanno lo scopo di ridurre il tempo di elaborazione raggruppando insiemi lavori con requisiti simili

2.1 Configurazione della memoria per un sistema a lotti (sistema batch)



3 Sistemi multiprogrammati

Il SO tiene contemporaneamente nella memoria centrale diversi lavori e la CPU di conseguenza non rimane mai inattiva



Caratteristiche

- Tutti i job che entrano nel sistema sono mantenuti in un gruppo che consiste di tutti i processi d'elaborazione presenti nel disco che attendono il caricamento nella memoria centrale;
- Gestione della memoria;
- Scheduling della CPU;
- Allocazione delle risorse

4 Sistemi a partizione del tempo d'elaborazione

La CPU viene ripartita tra i vari lavori tenuti in memoria sul disco (assegnazione della CPU ad un lavoro solo se questo è presente in memoria)

La CPU esegue più lavori commutando le loro esecuzioni. Presente una comunicazione diretta tra utente e sistema → passaggio tra utenti (ogni utente) ha l'impressione di disporre dell'intero calcolatore. Ciascun utente dispone di almeno un proprio programma nella memoria

5 Sistemi da scrivania

- PC;
- Dispositivi I/O;
- Comodità e prontezza d'uso per l'utente

Possibilità di utilizzare diversi sistemi operativi (**GNU LINUX** o altri sistemi operativi se si decide di non voler utilizzare seriamente il PC)

6 Sistemi paralleli

Sono sistemi con più unità di elaborazione, con più CPU in stretta collaborazione

Vantaggi

- Maggiore produttività (throughput);
- Economia di scala;
- Incremento dell'affidabilità
 - degradazione controllata
 - tolleranza ai guasti

I sistemi paralleli si dividono in:

- Multielaborazione simmetrica (SMP)
 - Ciascuna unità di elaborazione esegue un'identica copia del SO
 - Si possono eseguire molti processi contemporaneamente senza causa un calo rilevante delle prestazioni
- Multielaborazione asimmetrica (AMP)
 - Ad ogni unità di elaborazione spetta un compito specifico
 - Presenza di una unità centrale che controlla il sistema e diversi slave che attendono istruzioni

7 Sistemi distribuiti

Basati sulle reti per cooperare nella soluzione dei problemi di calcolo (ognuno presenta una propria memoria locale)

Vantaggi

- Condivisione delle risorse;
- Maggiore velocità;
- Affidabilità;
- Comunicazione.

Richiedono un'infrastruttura di rete

- Locale (LAN → Local Area Networks);
- Geografica (WAN → Wide Area Networks);
- Metropolitana (MAN → Metropolitan Area Networks);
- Small (SAN).

7.1 Ambiente di elaborazione (Client-Server o Peer-to-Peer)

Esistono due principali tipi di server:

- server per l'elaborazione;
- file server

Nei sistemi Peer-to-Peer invece ciascun pc può essere server o client

- Il carico è suddiviso su più server (si evitano i sovraccarichi);
- Necessita una politica di gestione
- Nuova entità (**load balancer**)

8 Batterie di sistemi (sistemi cluster)

Sono basate sull'uso congiunto di più unità di elaborazione riunite per svolgere attività d'elaborazione comunicano

Esistono 2 tipi di batterie:

- Asimmetriche
 - Un calcolatore resta nello stato di **attesa attiva** (hot standby) mentre l'altro esegue le applicazioni
- Simmetriche
 - Due o più calcolatori eseguono le applicazioni e allo stesso tempo si controllano reciprocamente

9 Sistemi d'elaborazione real-time

Esistono 2 tipi di sistemi d'elaborazione real-time:

- Sistemi d'elaborazione in tempo reale stretto (hard real-time) → richiedono un tempo molto breve di risposta
- Sistemi d'elaborazione in tempo reale debole (soft real-time) → richiedono tempi rapidi, ma non critici

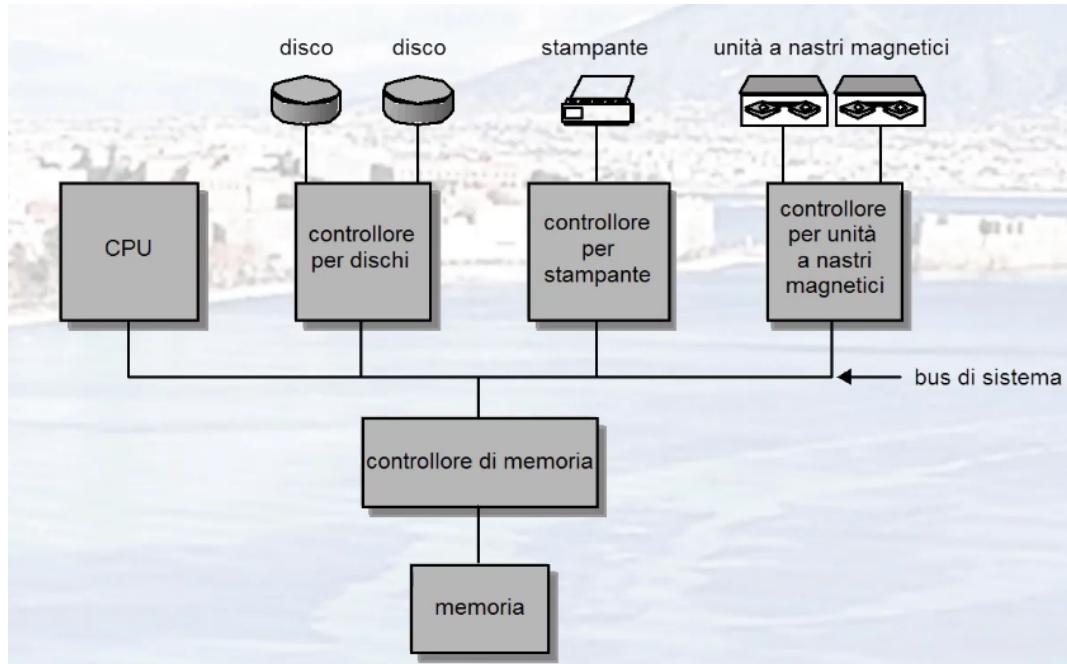
10 Sistemi palmari

- PDA;
- Smartphone

11 Ambienti d'elaborazione

- Elaborazione tradizionale;
- Elaborazione basata sul web;
- Dispositivi d'elaborazione integrati

12 Architettura di un sistema di calcolo



(Il *bus di sistema* permette la comunicazione delle componenti)

12.1 Funzionamento

- I dispositivi di I/O e la CPU possono operare in modo concorrente
- Ciascun controller gestisce un particolare dispositivo fisico
 - Presentano proprio buffer locale
- La CPU sposta i dati da/verso la memoria principale da/verso i buffer locali
- L'I/O avviene dal dispositivo al buffer locale del controller
- Il controller informa la CPU di aver terminato un'operazione attraverso un **interrupt**

12.2 Gli interrupt

Hanno lo scopo di causare il trasferimento del controllo all'appropriata procedura di servizio dell'evento a esso associato. Deve inoltre salvare l'indirizzo dell'istruzione interrotta

Un **segnale d'eccezione** (trap) può essere causato da un programma in esecuzione a seguito di un evento eccezionale o a seguito di una richiesta specifica effettuata da un programma utente

Un SO moderno è detto **interrupt driven**

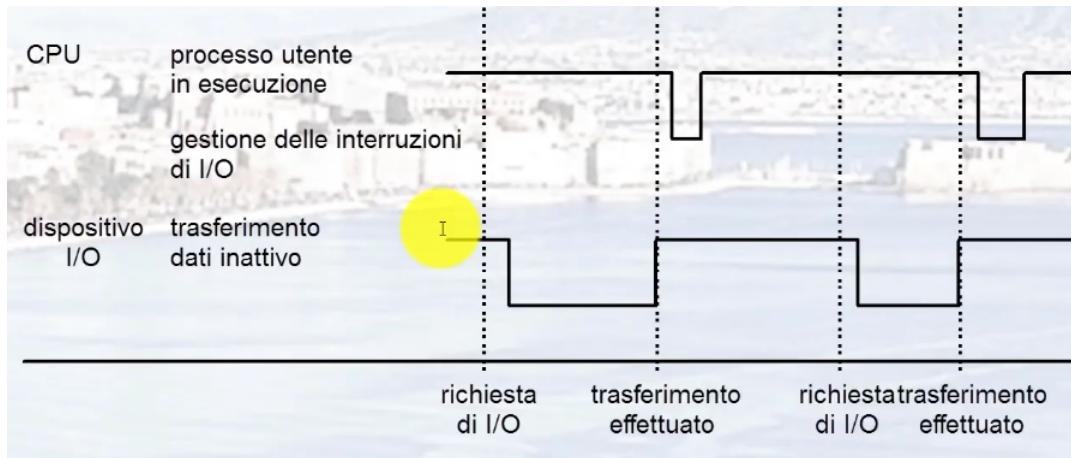
12.3 Gestione dell'interruzione

L'SO memorizza l'indirizzo di ritorno nello stack di sistema. Determina il tipo di interruzione verificatasi:

- polling (interrogazione ciclica, ma poco efficiente);
- vectored interrupt system;

Segmenti separati di codice determinano quale azione debba essere eseguita per ciascun tipo di interruzione

12.4 Diagramma temporale delle interruzioni per un singolo processo che emette dati



12.5 Struttura di I/O

Una volta iniziata l'operazione di I/O si restituisce il controllo al processo utente solo dopo il completamento dell'operazione di I/O.

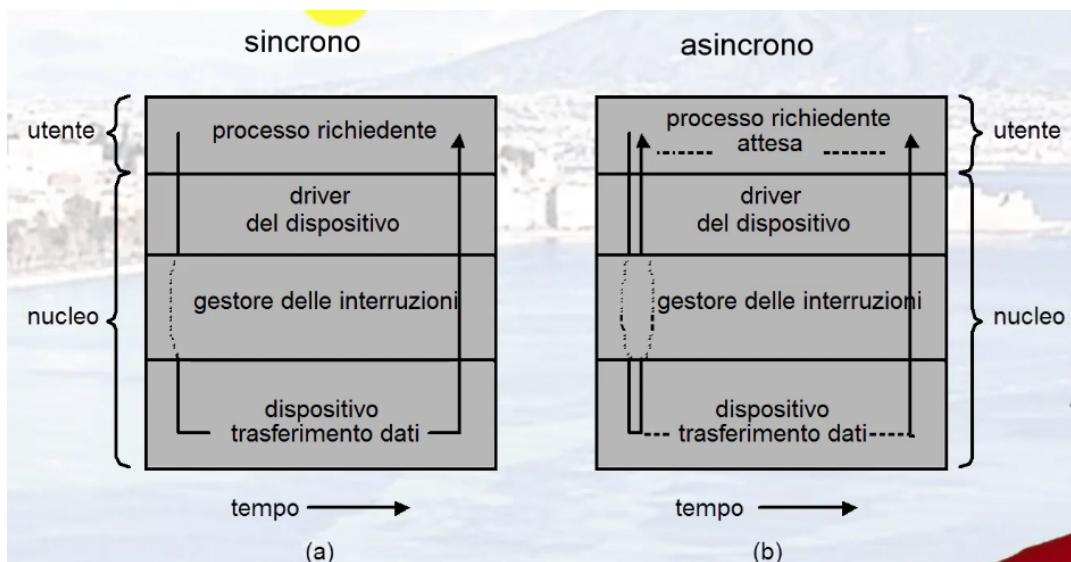
- L'istruzione **wait** sospende la CPU fino al successivo segnale di interruzione
 - Nei sistemi in cui non è prevista una istruzione del genere, si può generare un ciclo di attesa
- Si ha al più una richiesta pendente alla volta

Una volta iniziata l'operazione di I/O si restituisce immediatamente il controllo al processo utente senza attendere il completamento dell'operazione di I/O.

- **System call** → richiesta al SO di consentire al programma utente di attendere il completamento dell'operazione
- **Tabella di stato dei dispositivi** → contiene elementi per ciascun dispositivo di I/O che ne specificano il tipo, l'indirizzo e lo stato
 - Il SO individua il controller del dispositivo che ha emesso il segnale di interruzione, accede alla tabella dei dispositivi, risale al suo stato e modifica l'elemento della tabella indicando l'occorrenza dell'interruzione

Dispositivo	Stato
Dispositivo1	StatoDispositivo1

Due metodi di I/O

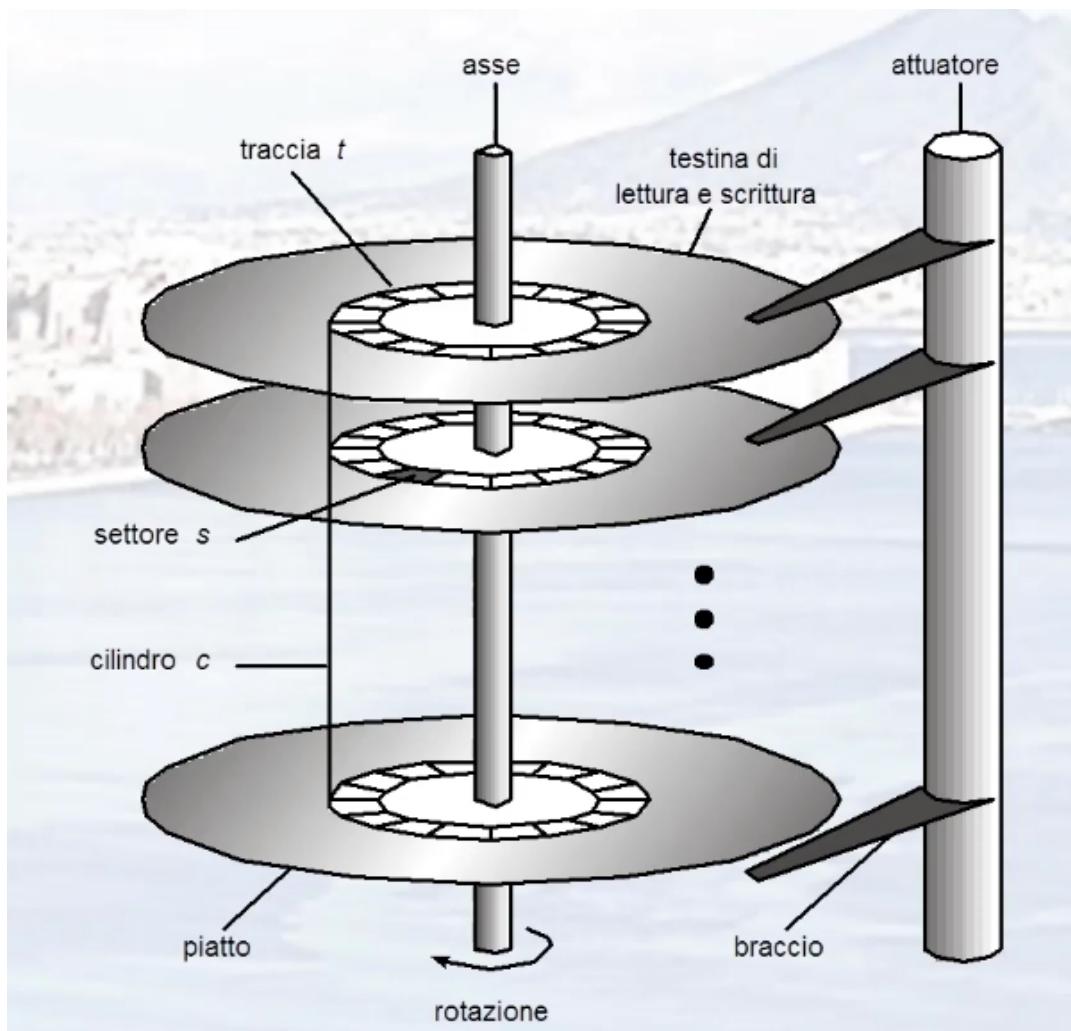


12.6 Accesso alla memoria (DMA)

- Diretto
 - Tecnica usata con dispositivi I/O veloci
 - Il controller trasferisce un intero blocco di dati dalla propria memoria di transito direttamente nella memoria centrale (o viceversa), senza l'intervento della CPU
 - Il trasferimento richiede una sola interruzione per ogni blocco di dati trasferito (rispetto a essere eseguito per ogni byte)

12.7 Struttura della memoria

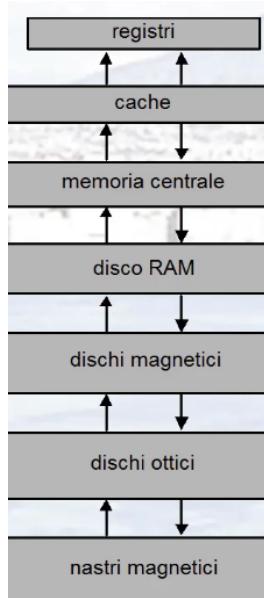
- Memoria centrale
 - Dispositivo di memoria direttamente accessibile dalla CPU
- Memoria secondaria (hard-disk)
 - Estensione della memoria centrale capace di conservare in modo permanente gradi quantità di informazioni
- Disco magnetico → Piatto rigido di metallo o vetro ricoperto di materiale magnetico
 - La superficie del disco è divisa logicamente in tracce circolari suddivise in settori
 - I controller dei dischi determinano l'interazione logica tra il dispositivo e il calcolatore



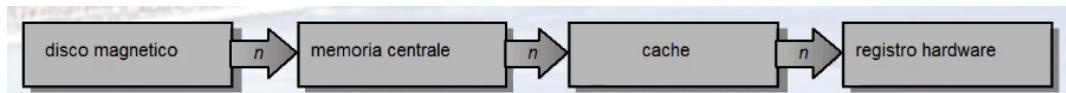
12.8 Gerarchia delle memorie

Si possono classificare le memorie in base alla velocità, costo o volatilità

- Cache → copia temporanea di informazioni in un'unità più veloce
 - La memoria centrale può essere considerata come una cache per la memoria secondaria



12.9 Migrazione di un intero n da un disco a un registro



12.10 Duplice modo di funzionamento

La condivisione delle risorse di sistema rende necessario che il SO garantisca che un programma malfunzionante non causi una scorretta esecuzione di altri programmi. Distinguiamo:

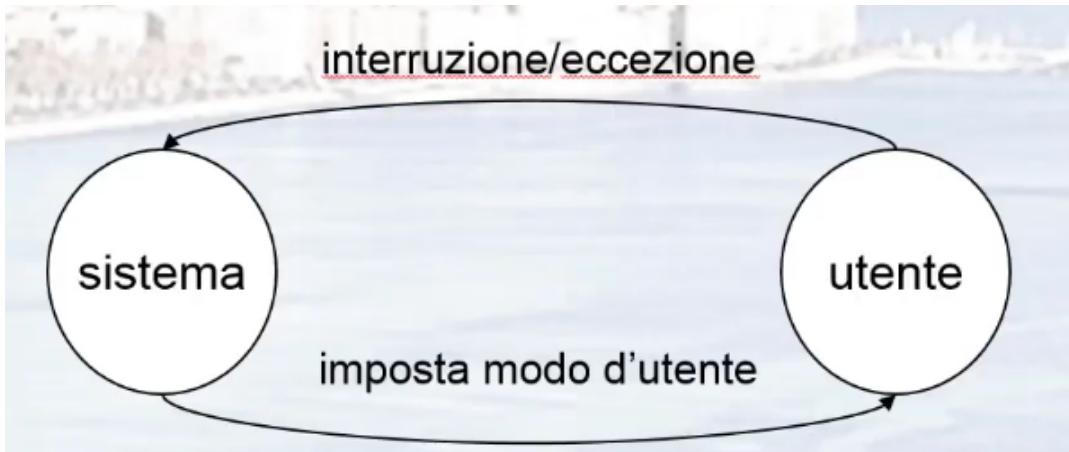
- **Modo di utente** (user mode) → l'istruzione corrente si esegue per conto di un utente;
- **Modo di sistema** (monitor mode) → l'istruzione corrente si esegue per conto del SO.



Il bit di modo indica quale modo sia attivo:

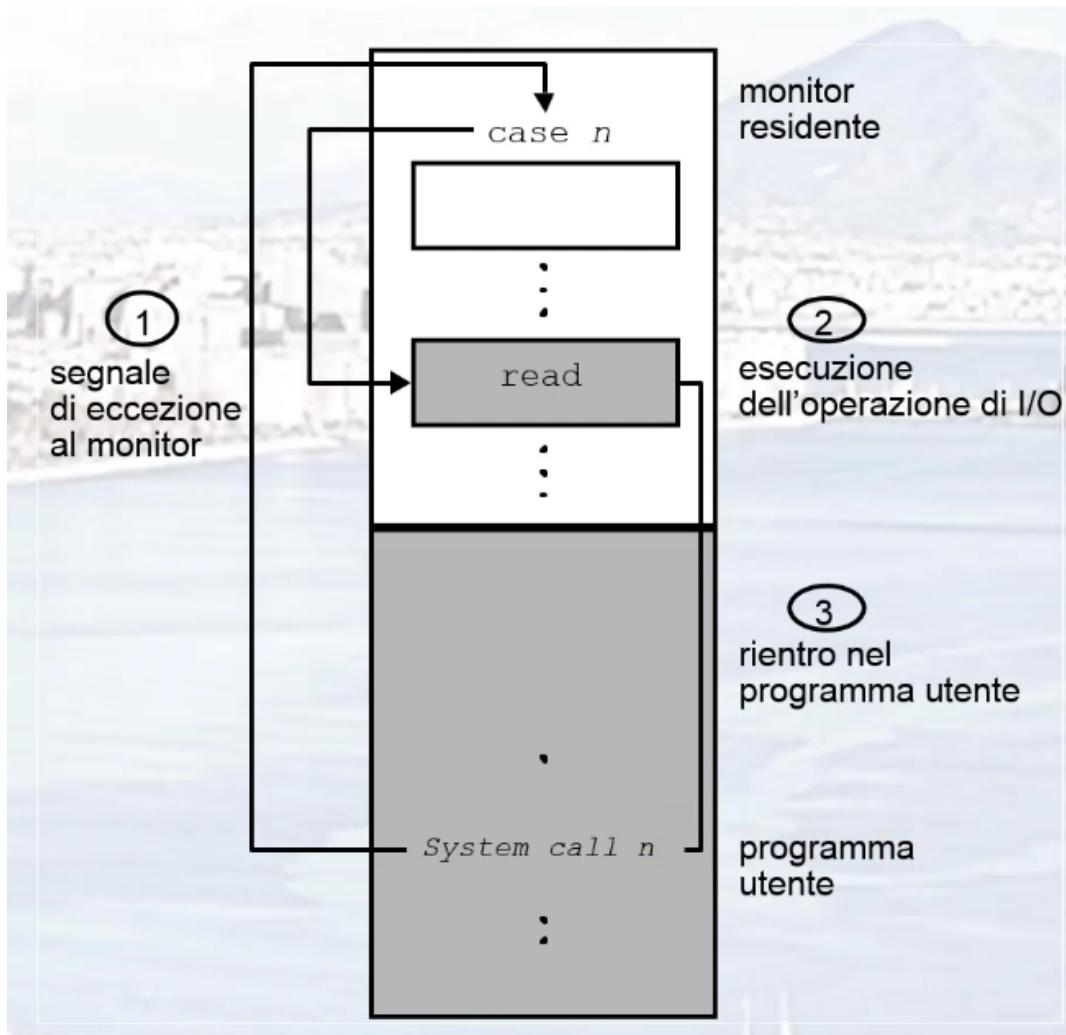
- 0 per quello di sistema;
- 1 per quello utente

Quindi ogni volta che si verifica un'interruzione o un'eccezione il bit di modo cambia il suo valore da 1 a 0



La CPU permette l'esecuzione di **istruzioni privilegiate** soltanto nel modo di sistema. Esempio di questo tipo di istruzioni sono quelle di I/O

12.11 Uso di una chiamata del sistema per l'esecuzione di una chiamata di I/O



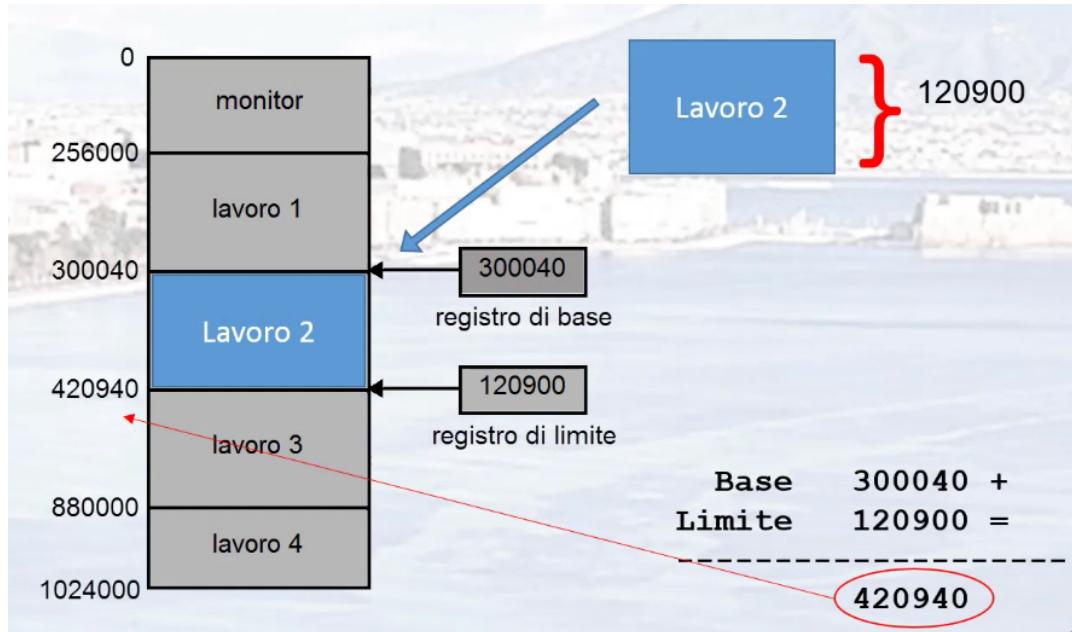
12.12 Protezione della memoria

Indispensabile per la protezione della memoria è conoscere almeno il vettore delle interruzioni e le relative procedure di servizio contenute nel codice del SO

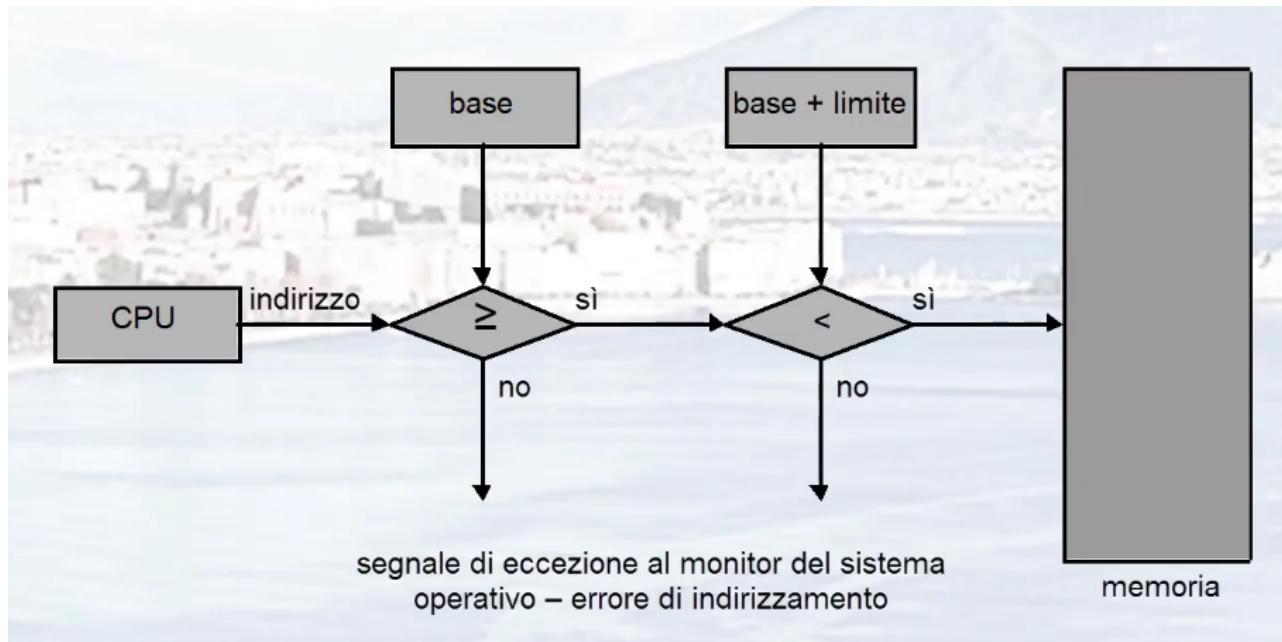
Questo tipo di protezione sfrutta l'utilizzo di 2 registri che contengono l'intervallo degli indirizzi validi a cui il programma può accedere:

- **Registro di base** → contiene il più basso indirizzo della memoria fisica al quale il programma dovrebbe accedere;
- **Registro di limite** → contiene la dimensione dell'intervallo

12.13 Uso di un registro di base e un registro di limite



12.14 Architettura di protezione degli indirizzi



12.15 Protezione HW

Il SO ha la possibilità di accedere sia alla memoria ad esso riservato, si a quella riservata agli utenti. Questo privilegio consente al SO di caricare i programmi utenti nelle relative aree di memoria

12.16 Protezione della CPU

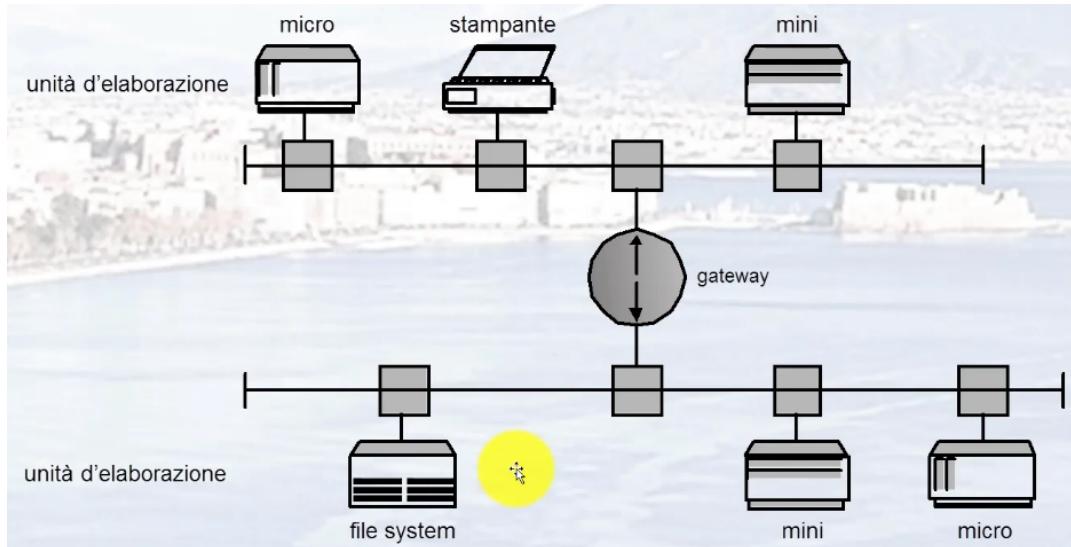
Temporizzatore (timer)

Invia un segnale di interruzione alla CPU a intervalli di tempo specificati per assicurare che il SO mantenga il controllo dell'elaborazione

- Il timer si decrementa a ogni impulso;
- Quando raggiunge il valore 0 si genera un segnale di interruzione.

I timer vengono spesso utilizzati per realizzare la partizione del tempo di elaborazione (*time sharing*), per il calcolo del calcolo dell'ora corrente

12.17 Struttura di una rete locale



13 Gestione dei processi

13.1 Definizione

Processo

Un processo è un programma in esecuzione (da non confondere con un programma, che invece è una lista di istruzioni che devono essere eseguite).

Per svolgere i propri compiti, un processo necessita di alcune risorse, tra cui:

- Tempo di CPU;
- memoria;
- file;
- dispositivi di I/O.

Il SO è responsabile delle seguenti attività connesse alla gestione dei processi:

- **creazione e cancellazione** dei processi utenti e di sistema;
- **sospensione e ripristino** dei processi;
- fornitura di meccanismi per
 - sincronizzazione dei processi
 - comunicazione tra processi

Un processo comprende:

- Program counter;
- Stack;
- Data section.

13.2 Gestione della memoria centrale

La **memoria centrale** è un vasto vettore di dimensioni che variano tra le centinaia di migliaia e le miliardi di parole, ciascuna delle quali è dotata del proprio indirizzo. È un magazzino di dati velocemente accessibile ed è condivisa dalla CPU e da alcuni dispositivi di I/O.

È di tipo volatile, perde quindi le informazioni in caso di guasto del sistema.

Il SO è responsabile delle seguenti attività connesse alla gestione della memoria centrale:

- tenere traccia di quali parti della memoria sono occupate e da cosa;
- decidere quali processi si debbano caricare nella memoria quando vi sia spazio disponibile;
- assegnare e revocare lo spazio di memoria secondo le necessità

13.3 Gestione dei file

Un file è una raccolta di informazioni correlate definite dal loro creatore. Comunemente rappresentano programmi (sorgente, oggetto) e dati.

Il SO è responsabile delle seguenti attività connesse alla gestione dei file:

- **creazione e cancellazione** di file;
- **creazione e cancellazione** di directory;
- fornitura delle funzioni fondamentali per la gestione di file e directory;
- associazione dei file ai dispositivi di memoria secondaria;
- creazione di copie di riserva dei file su dispositivi di memorizzazione non volatili.

13.4 Gestione del sistema I/O

È costituito dalle seguenti parti:

- **sistema buffer-caching**;
- interfaccia generale per i driver dei dispositivi

13.5 Gestione della memoria secondaria

Essendo la memoria centrale troppo piccola, il calcolatore deve disporre di una memoria a sostegno di questa

Il SO è responsabile delle seguenti attività connesse alla gestione dei dischi:

- gestione dello spazio libero;
- assegnazione dello spazio;
- scheduling del disco

13.6 Reti (sistemi distribuiti)

Un **sistema distribuito** è un insieme di unità di elaborazione che *NON* condividono la memoria,i dispositivi periferici o un clock. Le unità di elaborazione sono collegate da una **rete di comunicazione** che avviene mediante un protocollo (regole di comunicazione).

Un sistema di questo tipo offre all'utente l'accesso a varie risorse del sistema. L'accesso ad una risorsa condivisa permette di:

- accelerare il calcolo;
- aumentare la disponibilità dei dati;
- incrementare l'affidabilità.

13.7 Sistemi di protezione

È definita da ogni meccanismo che controlla l'accesso da parte di programmi, processi o utenti alle risorse di un sistema di calcolo. Questo meccanismo deve:

- distinguere tra uso autorizzato e non;
- specificare i controlli che devono essere attivati;
- fornire strumenti di miglioramento dell'affidabilità.

13.8 Interprete dei comandi

Molti comandi si impartiscono al SO attraverso istruzioni di controllo che riguardano:

- creazione e gestione dei processi;
- I/O;
- gestione della memoria centrale e secondaria;
- accesso al file-system;
- protezione;
- reti

13.9 Servizi di un sistema operativo

- Esecuzione di un programma;
- Operazione di I/O;
- Gestione del file system;
- Comunicazioni;
- Rilevamento di errori.

13.10 Funzioni addizionali di un SO

- Assegnazione delle risorse;
- Contabilizzazione dell'uso delle risorse;
- Protezione.

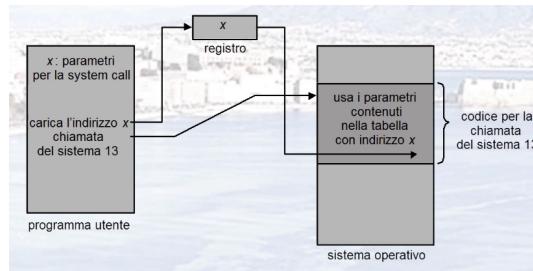
13.11 Chiamate del sistema

Costituiscono l'interfaccia tra un processo e il SO. Generalmente sono disponibili in forma di istruzioni in linguaggio assemblato, ma alcuni sistemi permettono che le chiamate siano invocate direttamente da un programma scritto in un linguaggio di alto livello

Per passare parametri al SO si utilizzano 3 metodi generali:

- Passare i parametri in **registri**;
- Memorizzare i parametri in un *blocco* o tabella di memoria e passare l'indirizzo del blocco in forma
 - di parametro
 - in un registro
- Collocare (*push*) i parametri in una pila da cui sono prelevati (*pop*) dal SO

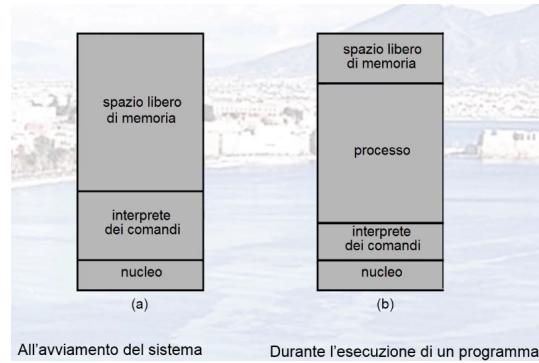
13.12 Passaggio di parametri in forma di tabella



13.13 Tipi di chiamate del sistema

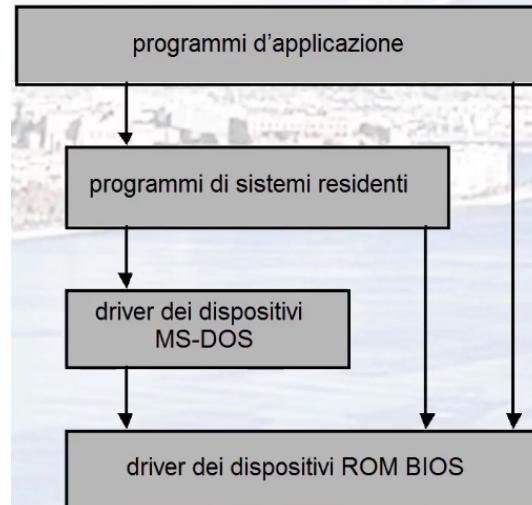
- Controllo dei processi;
- Gestione dei file;
- Gestione dei dispositivi;
- Gestione delle informazioni;
- Comunicazione.

13.14 Struttura ed esecuzione dei programmi in MS-DOS

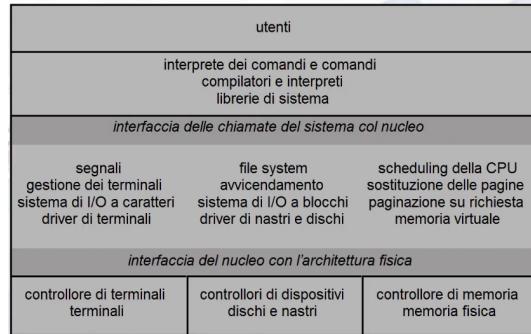


Progettato per fornire la massima funzionalità nel minimo spazio.

- Non suddiviso in moduli;
- Nonostante sia dotato di una struttura semplice, le sue interfacce e i livelli di funzionalità non sono ben separati

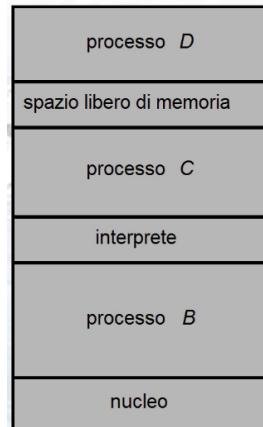


13.15 Struttura ed esecuzione dei programmi nei sistemi **UNIX**



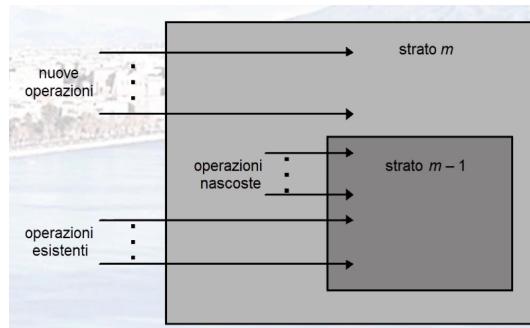
Questo tipo di sistema è formato da due parti:

- Programmi di sistema;
- Nucleo (Kernel)
 - È tutto ciò che si trova sotto l'interfaccia delle chiamate di sistema e sopra i dispositivi fisici
 - Fornisce
 - * file system
 - * scheduling della CPU
 - * gestione della memoria e altre funzioni del SO



13.16 Metodo stratificato

Metodo che suddivide il SO in un certo numero di strati (o livelli), ciascuno costruito sopra gli strati inferiori. Lo strato più basso (0) è quello fisico, il più alto invece è l'interfaccia utente.



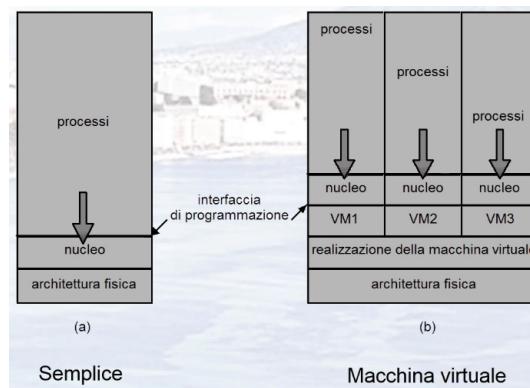
13.17 Macchine virtuali

Si sviluppa a partire logicamente dal metodo stratificato. Il SO crea l'illusione che un processo disponga della propria CPU con la propria memoria (virtuale)

Il calcolatore fisico condivide le risorse in modo da creare macchine virtuali.

La partizione del tempo d'uso della CPU si può usare sia per condividere la CPU sia per dare l'illusione che gli utenti dispongano di una propria CPU.

La gestione asincrona delle operazioni di I/O e dell'esecuzione di più processi, unita ad un file system, consente di creare lettori di schede e stampanti virtuali



Vantaggi e svantaggi delle macchine virtuali

Vantaggi:

- Protezione delle risorse del sistema (ogni vm è isolata dalle altre);
- Perfetto mezzo di ricerca e sviluppo dei SO.

Svantaggi:

- Assenza di una condivisione diretta delle risorse;
- Maggiore complessità della macchina comporta una maggiore difficoltà nel realizzare una vm (maggior lentezza nell'esecuzione).

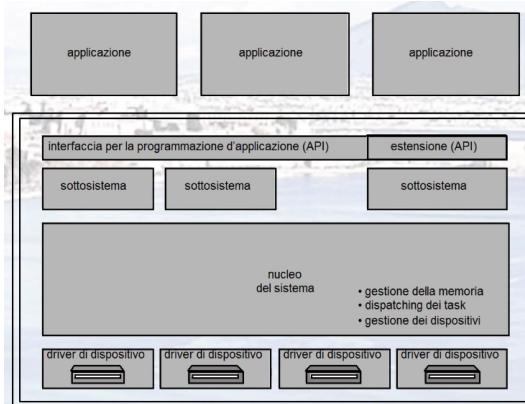
13.17.1 Macchina virtuale Java

I programmi Java sono **bytecode**, indipendente dall'architettura sottostante (gira tutto sulla JVM). Essa consiste in:

- Caricatore delle classi;
- Verificatore delle classi;
- Un interprete del linguaggio che esegue il bytecode.

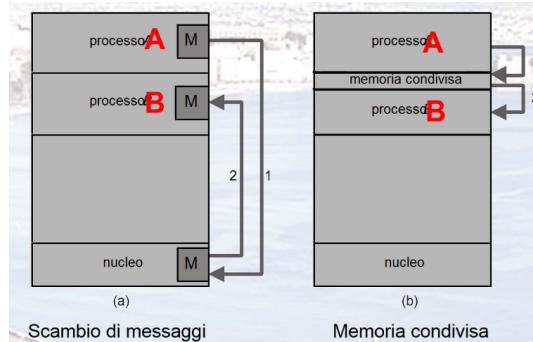
Il compilatore istantaneo Just-in-Time (JIT) ne migliora le prestazioni

13.18 Struttura stratificata dell' OS/2



13.19 Modelli di comunicazione

La comunicazione può avvenire o attraverso il modello a **scambio di messaggi** o mediante il modello a **memoria condivisa**



13.20 Programmi di sistema

Offrono un ambiente conveniente per lo sviluppo e l'esecuzione dei programmi. Possono essere classificati nelle seguenti categorie:

- Gestione e modifica dei file;
- Informazioni di stato;
- Ambienti di ausilio alla programmazione;
- Caricamento ed esecuzione dei programmi;
- Comunicazione;
- Programmi di applicazione;

13.21 Microkernel (Orientamento a micronucleo)

Lo scopo è progettare un SO rimuovendo dal nucleo tutti i componenti non essenziali, realizzandoli come programmi del livello d'utente e di sistema.

La comunicazione si realizza secondo il modello a scambio di messaggi. I vantaggi sono:

- Facilità di estensione del SO;
- Semplice da adattare alle diverse architettura;
- Affidabile
 - I servizi si eseguono in gran parte come processi utenti
- Sicuro

13.22 Scopi della progettazione

Lato utente → il SO deve essere:

- utile;
- Facile da imparare e usare;
- Sicuro ed efficiente.

Lato sistema → il SO deve essere:

- di facile progettazione, realizzazione e manutenzione;
- flessibile;
- senza errori ed efficiente.

13.23 Meccanismo e criteri

I meccanismi determinano come eseguire qualcosa, mentre i criteri invece stabiliscono cosa si debba fare

13.24 Realizzazione

Di solito si scrivevano con linguaggi di basso livello, adesso invece con quelli di alto livello (come c o c++) con il grande vantaggio di avere un porting maggiore

13.25 Generazione di sistemi (SYSGEN)

Il processo di generazione di sistemi configura o genera il sistema per ciascuna situazione specifica:

- **Booting** → avviamento di un calcolatore attraverso il caricamento del kernel;
- **Bootstrap program** → piccolo segmento di codice memorizzato in una ROM che individua il kernel, lo carica nella memoria e ne avvia l'esecuzione

14 I processi nel dettaglio

Un SO esegue differenti programmi:

- Un sistema a lotti (batch system) esegue diversi job;
- Un sistema a partizione del tempo (time shared system) esegue i programmi utenti o task.

14.1 Stato di un processo

Nel momento in cui un processo è in esecuzione è soggetto a cambiamenti di stato:

- **nuovo** → si crea il processo;
- **esecuzione** → le istruzioni vengono eseguite;
- **attesa** → il processo attende che si verifichi qualche evento;
- **pronto** → il processo attende di essere assegnato ad una unità di elaborazione;
- **terminato** → il processo ha completato l'esecuzione

Diagramma di transizione degli stati di un processo

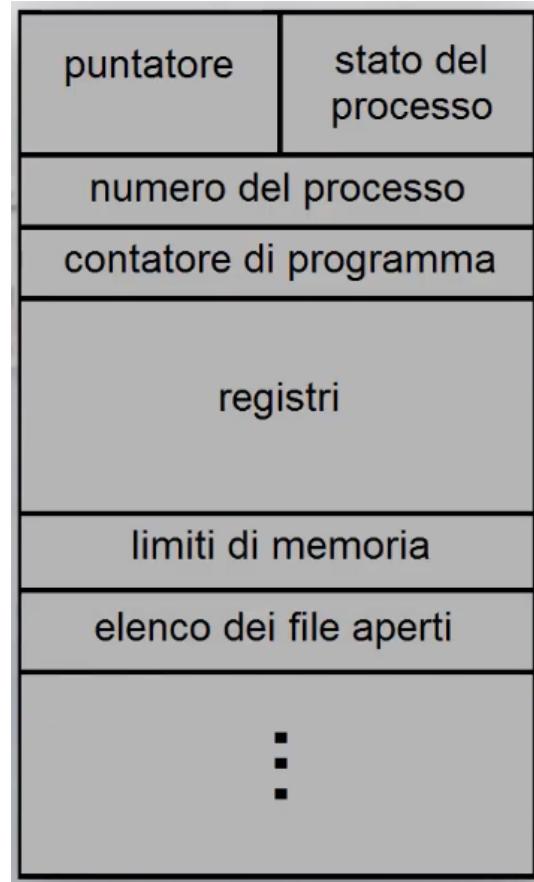


Latenza di dispatch → tempo impiegato dal dispatcher per sospendere un processo e avviare l'esecuzione di un nuovo processo

14.2 Blocco di controllo di un processo (PCB)

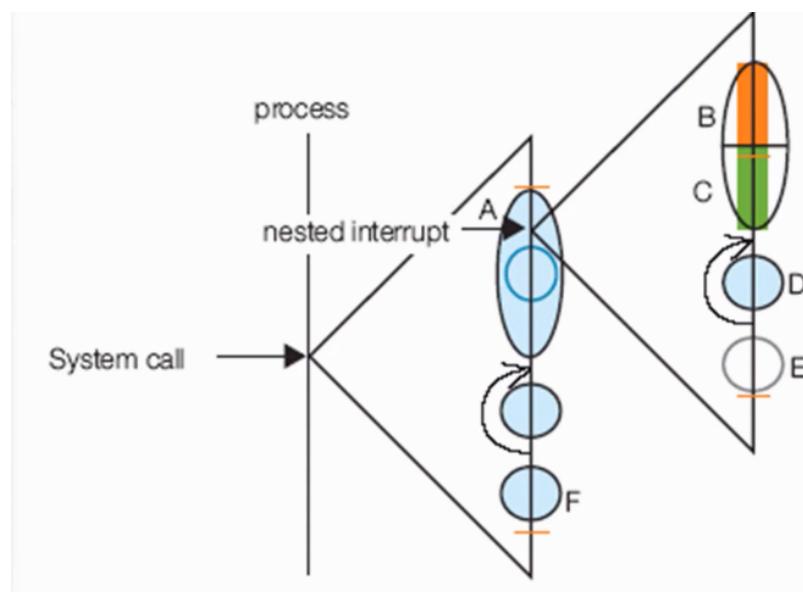
Nel momento in cui un processo nasce si attivano una serie di informazioni che vengono contenute in un blocco dati, il **PCB**. Questo blocco contiene:

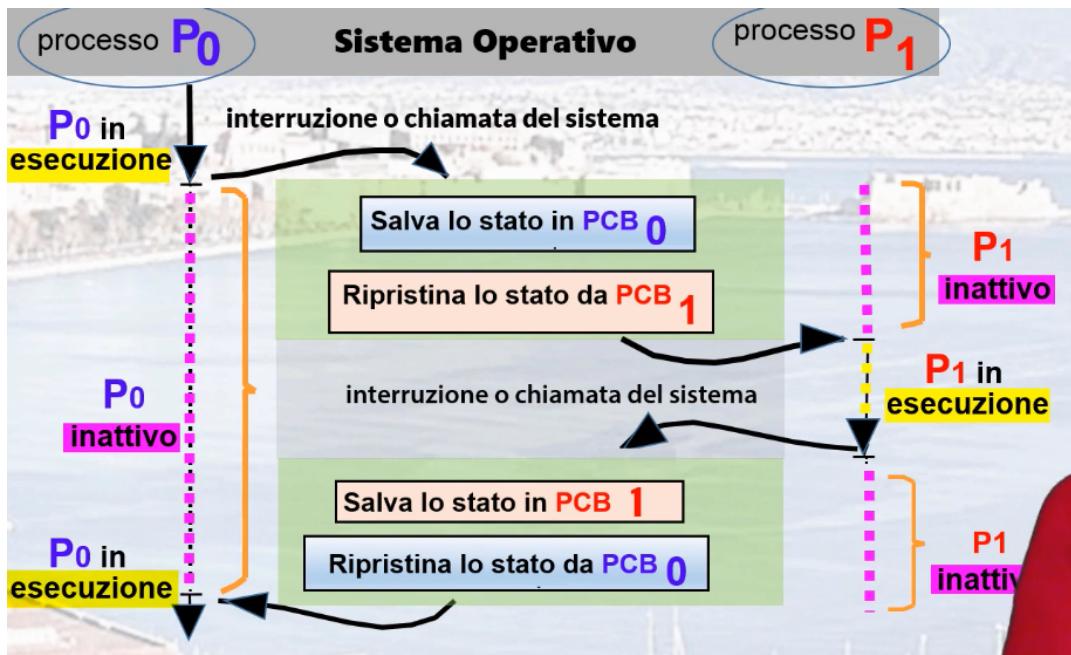
- stato del processo;
- Program counter;
- Registri di CPU;
- Informazioni sullo scheduling di CPU;
- Informazioni sulla gestione della memoria;
- Informazioni di contabilizzazione delle risorse;
- Informazioni sullo stato dell' I/O



14.3 CPU e la commutazioni tra processi

La CPU inoltre può essere commutata tra processi

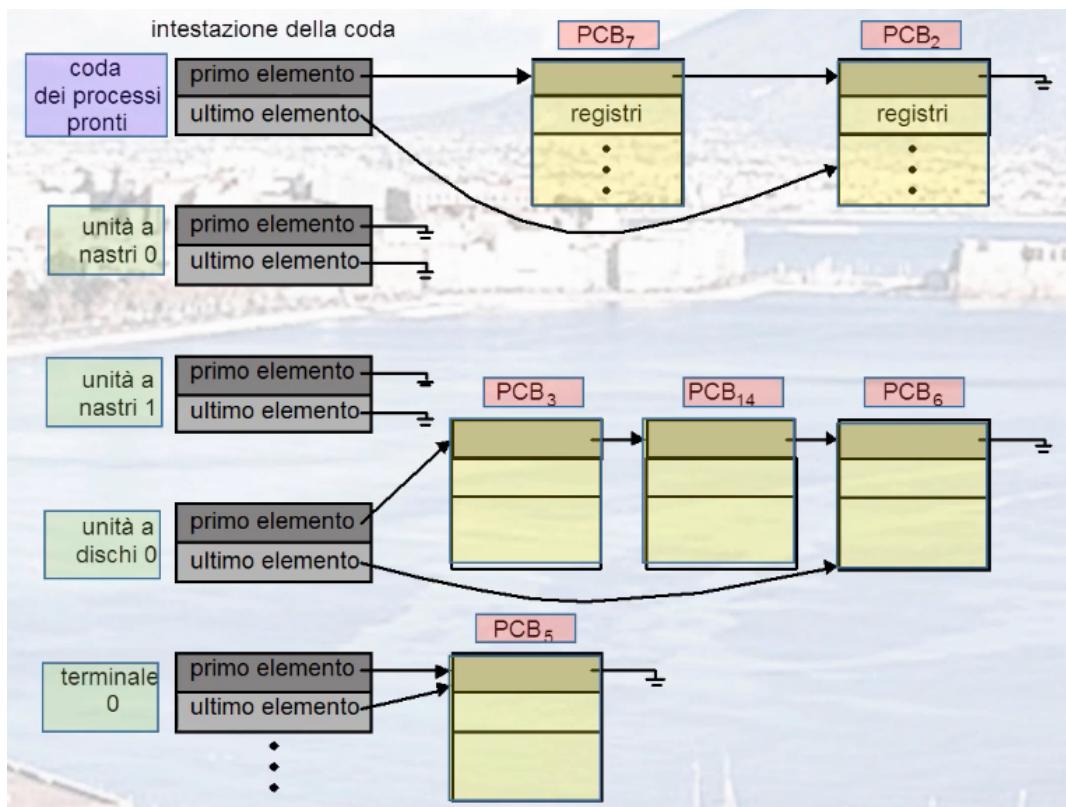




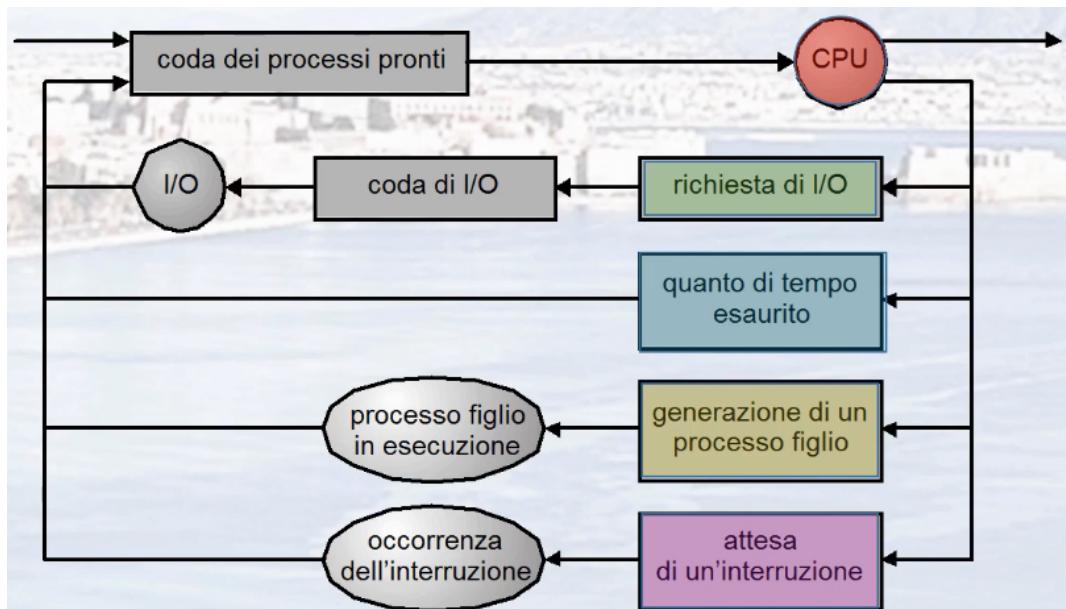
14.4 Code di scheduling dei processi

Distinguiamo:

- Coda dei processi (job queue) → insieme di tutti i processi di sistemi;
- Coda dei processi pronti (ready queue) → processi presenti nella memoria centrale, pronti e in attesa di essere eseguiti;
- Coda del dispositivo (device queue) → elenco dei processi che attendono la disponibilità di un particolare dispositivo di I/O



14.5 Diagramma di accodamento per lo scheduling dei processi



14.6 Scheduler

- Scheduler a lungo termine (job scheduler) → seleziona i processi che devono essere caricati nella coda dei processi pronti;
- Scheduler di CPU (short-term scheduler o CPU scheduler) → fa la selezione tra i lavori pronti per essere eseguiti e assegna la CPU ad uno di essi;
- Scheduler a medio termine → ha il compito di rimuovere i processi dalla memoria

I processi possono essere caratterizzati in:

- Processo con prevalenza di I/O (I/O bound process) → impiega la maggior parte del proprio tempo nell'esecuzione di operazioni di I/O;
- Processo con prevalenza d'elaborazione (CPU-bound process) → richiede poche operazioni di I/O e impiega la maggior parte del proprio tempo nelle elaborazioni

14.7 Cambio di contesto

È il passaggio della CPU a un nuovo processo, con conseguente registrazione dello stato del processo vecchio e il caricamento dello stato precedentemente registrato del nuovo processo. Il tempo impiegato a compiere questa operazione è puro sovraccarico (il sistema non compie alcun lavoro utile durante la commutazione)

14.8 Creazione di un processo

Ciascun processo può creare numerosi processi figli, che a loro volta possono fare lo stesso, creando così un albero di processi.

Per la condivisione delle risorse esistono 3 casi:

- Genitore e figlio possono condividere tutte le risorse;
- Il processo figlio può condividere un sottoinsieme di risorse del processo padre;
- Genitore e figlio non condividono alcuna risorsa

Per l'esecuzione invece:

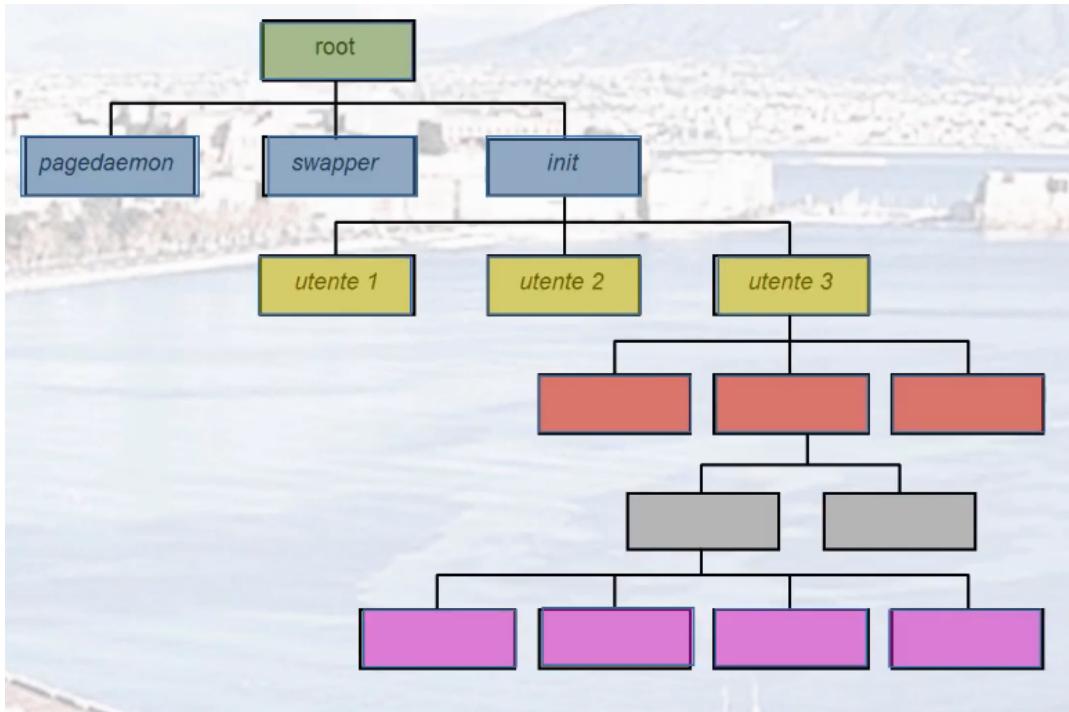
- Genitore e figlio possono essere eseguiti in modo concorrente;
- Il processo genitore attende che alcuni o tutti i suoi processi figli terminino

Bisogna considerare una serie di fattori:

- Lo spazio di indirizzi
 - Il processo figlio è un duplice del processo genitore;
 - Nel processo figlio si carica un programma;

Nel caso dei sistemi UNIX:

- la chiamata di sistema **fork** crea un nuovo processo;
- la chiamata di sistema **exec** dopo una fork sostituisce lo spazio di memoria del processso con un nuovo programma;



14.9 Terminazione di un processo

Un processo termina quando completa l'esecuzione della sua ultima istruzione e chiede al SO di essere cancellato usando la chiamata del sistema **exit**

- Il processo figlio può riportare alcuni dati al processo genitore attraverso la chiamata del sistema **wait**;
- Tutte le risorse del processo vengono liberate dal SO;

Un processo genitore può porre termine ad uno dei suoi processi figli (**abort**) in caso in cui:

- il processo figlio ha ecceduto nell'uso di alcune risorse;
- Il compito assegnato al processo figlio non è più richiesto;
- Il processo padre termina
 - il SO non consente al processo figlio di continuare l'esecuzione in tale circostanza;
 - Terminazione a cascata (cascading termination);

14.10 Processi cooperanti e indipendenti

Un processo si dice indipendente se non può influire su altri processi nel SO o subirne l'influsso.

Un processo si dice cooperante se può influire su altri processi nel SO o subirne l'influsso.

I vantaggi di questo tipo di processi sono:

- condivisione di informazioni;
- accellerazione del calcolo;
- Modularità;
- Convenienza

14.11 Processo produttore e consumatore

Un processo **produttore** produce informazioni che sono consumate da un processo **consumatore**. Distinguiamo in:

- memoria illimitata → non pone limiti alla dimensione del vettore;
- memoria limitata → presuppone l'esistenza di una dimensione fissata del vettore.

14.11.1 Memoria limitata - Soluzione con memoria condivisa

Supponiamo di usare una zona di memoria condivisa e usiamo un vettore circolare di grandezza **DIM_VETTORE** che fa uso di 2 indici per accedere al vettore.

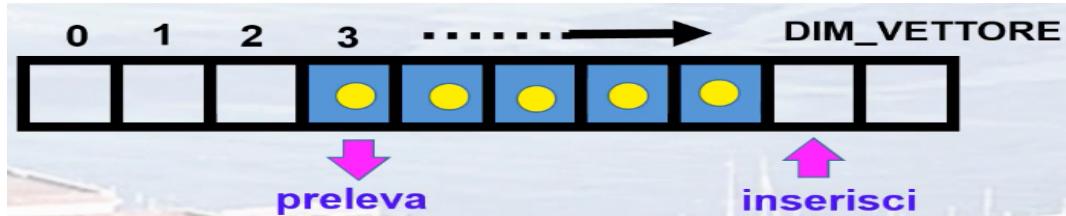
Con l'indice **inserisci** puntiamo alla prima casella disponibile del vettore.

Con l'indice **preleva** puntiamo alla prima casella occupata del vettore.

Al momento iniziale poniamo a zero sia inserisci che preleva.

Da tali definizioni si ricava che:

- Se *inserisci* == *preleva* allora il vettore è vuoto;
- Se $((inserisci + 1) \% \text{DIM_VETTORE}) == \text{preleva}$ allora il vettore è pieno;
- Max elementi nel vettore = $\text{DIM_VETTORE} - 1$



```
#define DIM_VETTORE 10
Typedef struct{
...
} elemento;
elemento[DIM_VETTORE];
int inserisci=0;
int preleva=0;
item appena_Prodotto; // Processo produttore
item da_Consumare; // Processo consumatore

// Per il processo produttore
while(1){ //loop all'infinito -> 1 sempre vero
    while (((inserisci + 1) \% DIM_VETTORE) == preleva)
// NON FA NULLA
    /* Nel momento in cui la condizione
    precendente è falsa esegue le due operazioni */
    vettore[inserisci] = appena_Prodotto;
    inserisci = (inserisci + 1) \% DIM_VETTORE;
}

// Per il processo consumatore
while(1){
    while (inserisci == preleva)
// NON FA NULLA
    /* Nel momento in cui la condizione
    precendente è falsa esegue le due operazioni */
    da_Consumare = vettore[preleva];
    preleva = (preleva + 1) \% DIM_VETTORE;
}
```

14.12 Comunicaione tra processi (IPC)

Si tratta di system call (non si ha l'obbligo di ricorrere a dati condivisi). I processi devono nominarsi reciprocamente in modo esplicito.

Un sistema di IPC fornisce 2 operazioni:

- **send** (messaggio) → dimensione fissa o variabile;
- **receive** (messaggio).

La realizzazione del canale di comunicazione può essere fisica (memoria condivisa, bus, reti) o logica (proprietà logiche). La comunicazione può essere di 2 tipi:

- sincrono (bloccante);
- asincrono (non bloccante);

I canali presentano le seguenti caratteristiche:

- I canali vengono stabiliti automaticamente;
- Un canale è associato esattamente ad una coppia di processi;
- Esiste esattamente 1 canale tra ciascuna coppia di processi;
- Il canale può essere unidirezionale o bidirezionale.

14.12.1 Mailbox

I messaggi vengono inviati a delle porte (le mailbox) che li ricevono:

- Ciascuna porta è identificata in modo univoco;
- Due processi possono comunicare solo se condividono una porta

14.12.2 Code di messaggi (buffering)

I messaggi scambiati tra processi comunicanti risiedono in code temporanee. Esistono 3 modi per realizzare queste code:

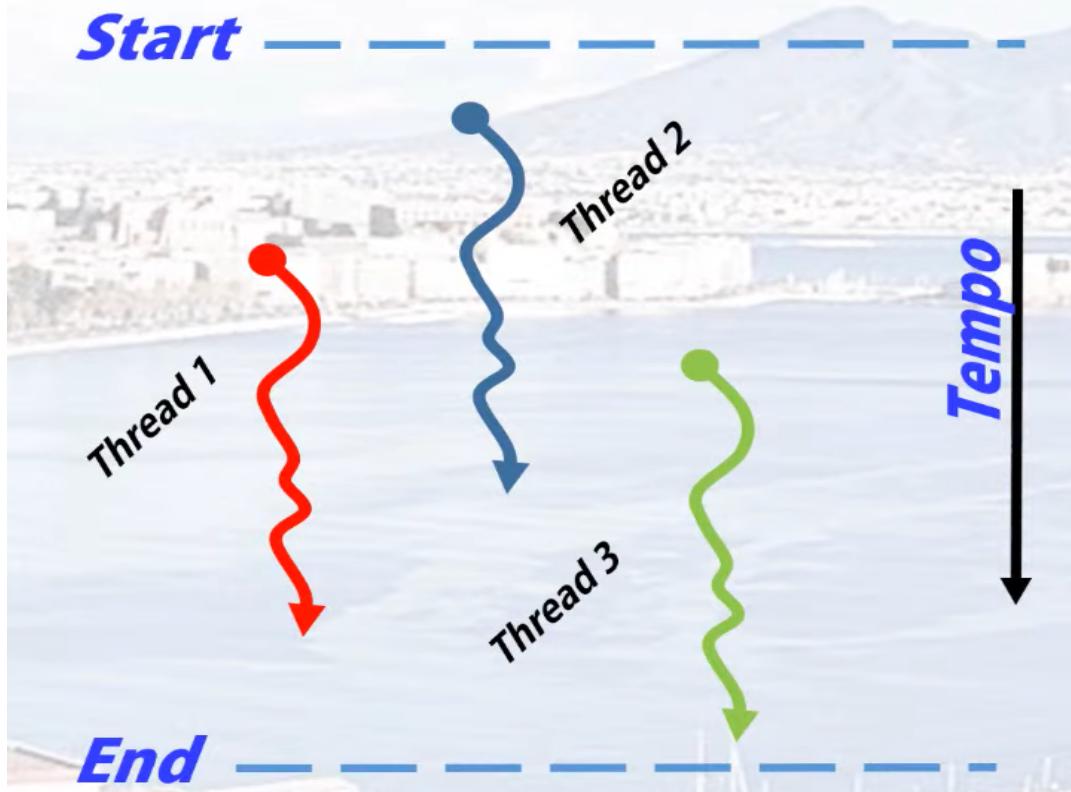
- **Capacità zero** (0 messaggi) → il trasmittente deve fermarsi finché il ricevente prende in consegna il messaggio;
- **Capacità limitata** → la coda ha lunghezza finita n (il trasmittente deve attendere nel caso il canale fosse pieno);
- **Capacità illimitata** → la coda ha lunghezza potenzialmente infinita.

14.12.3 Comunicazione nei sistemi Client-Server

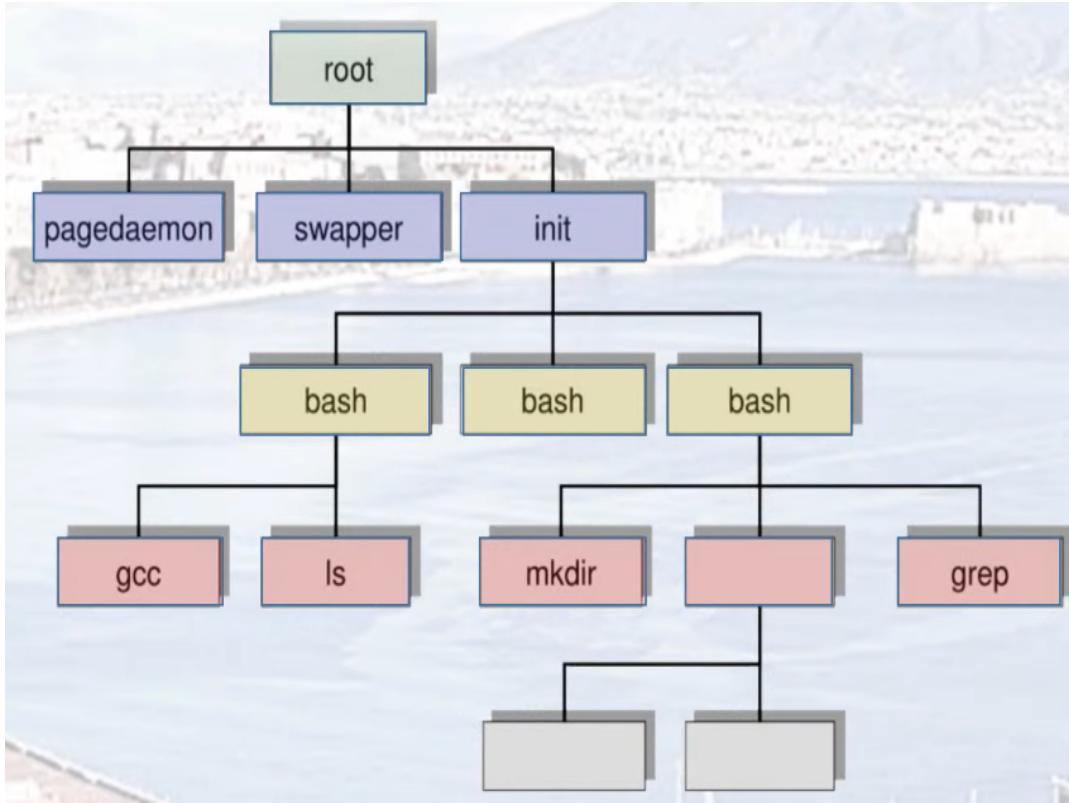
- Socket
 - estremità di un canale di comunicazione (indirizzo IP + porta)
- Chiamate di procedure remote (RPC)
 - astrae il meccanismo della chiamata di procedura (caso di sistemi collegati attraverso una rete)
- Invocazione di metodi remoti (RMI → Java)

15 I Thread

Entità di elaborazione presenti nel nostro sistema di elaborazione



15.1 Thread Linux & Unix



15.2 Definizione

Singolo flusso sequenziale di controllo all'interno di un processo. Un processo può contenere più thread.

Tutti i thread di un processo condividono lo stesso spazio di indirizzamento. Si definisce come **processo leggero** (ha un contesto semplice)

15.3 Parallelismo

Architettura in cui sono presenti più unità di elaborazione (CPU) sulle quali sono in esecuzione processi e thread. In ogni istante di tempo, posso avere più di un processo o più di un thread fisicamente in esecuzione.

15.3.1 Sistema monoprocesso time-sliced

È presente una singola unità di elaborazione. Il parallelismo di processi e thread viene simulato (concorrenza) allocando a ciascuno una frazione del tempo di CPU. Allo scadere di ogni unità di tempo, il SO opera un cambio di contesto (context switch). I thread sono processi leggeri perché il cambio di contesto è veloce

15.4 Sistema preemptive e non preemptive

Nel primo il cambio di contesto avviene quando il processo o il thread interrompe la propria esecuzione o volontariamente, o perchè in attesa di un evento (I/O).

Nel secondo invece allo scadere del time slice il processo o il thread viene forzatamente interrotto e viene operato il cambio di contesto.

La schedulazione dei processi e dei thread può avvenire secondo diversi algoritmi.

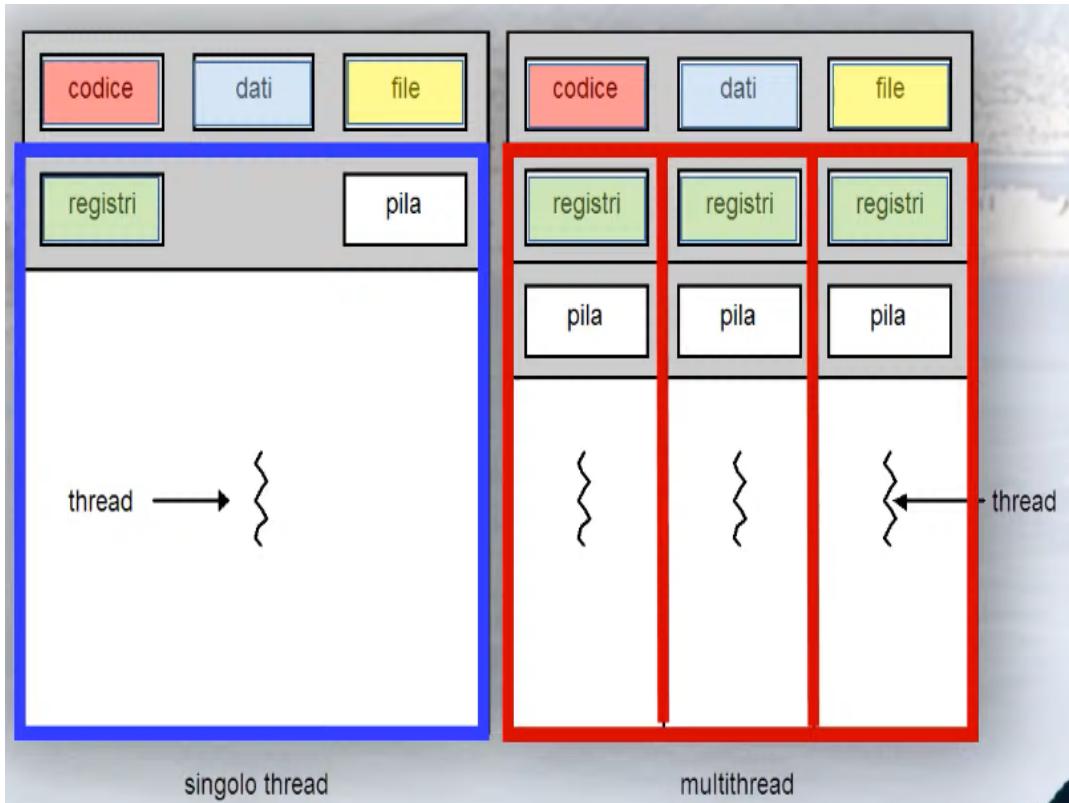
15.5 Motivazioni dei thread

È spesso necessario dividere il programma anche in "sotto-compiti" indipendenti

- Un programma può avere diverse funzioni concorrenti
 - operazioni ripetute nel tempo ad intervalli regolari;
 - esecuzione di compiti laboriosi senza bloccare la GUI del programma;
 - attesa di messaggi da un altro programma;
 - attesa di input da tastiera o da rete

L'alternativa al multithreading è il polling. Questo però oltre a essere scomodo da implementare, consuma molte risorse di CPU (è estremamente inefficiente)

15.6 Processi a singolo thread e multithread



15.7 Vantaggi e svantaggi dei thread

Vantaggi
Tempo di risposta
Condivisione delle risorse
Economia
Uso di più unità di elaborazione

15.8 Thread a livello utente

Sono gestiti come uno strato separato sopra il nucleo del SO, e realizzati tramite una libreria di funzioni per la creazione, lo scheduling, e la gestione, senza alcun intervento diretto del nucleo

15.9 Thread a livello del nucleo

Gestiti direttamente dal SO

15.10 Modelli di programmazione multithread

- Modello da molti a uno;
- Modello da uno a uno;
- Modello da molti a molti

15.10.1 Modello da molti a uno

Fa corrispondere molti thread a livello utente a un singolo thread al livello del nucleo

15.10.2 Modello da uno a uno

Fa corrispondere ciascun thread a livello utente con un thread al livello del nucleo.

15.10.3 Modello da molti a molti

Fa corrispondere più thread a livello utente con un numero minore o uguale di thread al livello del nucleo

15.11 Pthreads

Lo standard POSIX (Portable Operating System Interface for UniX) → definisce l'API per la creazione e la sincronizzazione dei thread.

Non si tratta di una realizzazione ma di una definizione del comportamento dei thread.

I progettisti di SO possono realizzare le API così definite come meglio credono

15.11.1 I thread nel linguaggio java

Possono essere creati:

- creando una nuova classe derivata dalla classe Thread;
- sovrascrivendo il metodo run di quella classe.

In entrambi i casi i thread vengono gestiti dalla JVM

15.12 Questioni di programmazione multithread

- Chiamate del sistema *fork()* ed *exec()*
 - La semantica di queste chiamate cambia → si rischia di andare in contro a duplicazione di tutti i thread o del singolo thread chiamante;
- Cancellazione

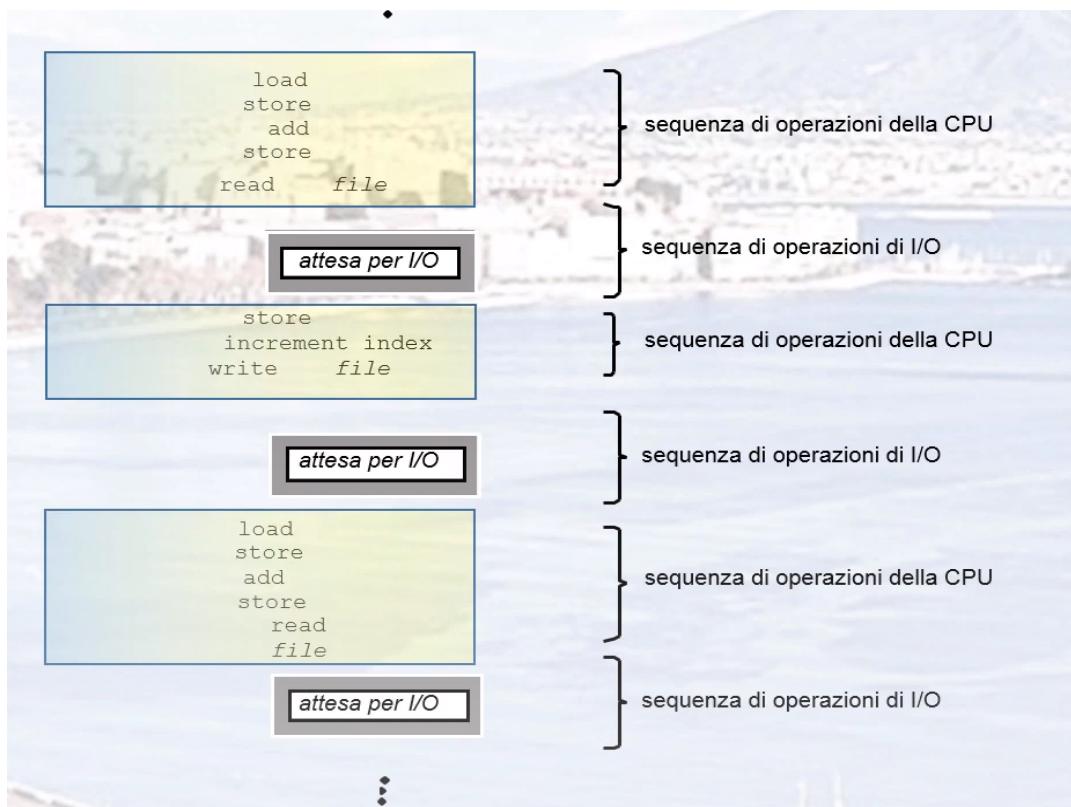
- Permette di terminare un thread prima che completi il suo compito;
- Gestione dei segnali;
- Gruppi di thread
 - Lo scopo è cercare di creare dei gruppi di thread il cui scopo è quello di snellire la gestione delle risorse;
- Dati specifici dei thread
 - In particolari circostanze, ogni thread può necessitare di una copia privata di certi dati;

16 Scheduling della CPU

16.1 Concetti Fondamentali

L'obiettivo della multiprogrammazione è avere sempre processi in esecuzione al fine di massimizzare l'utilizzo della CPU. L'esecuzione del processo consiste in un ciclo d'elaborazione e d'attesa del completamento delle operazioni di I/O. I processi si alternano tra questi 2 stati.

L'esecuzione del processo comincia con una sequenza di operazioni d'elaborazione svolte dalla CPU (CPU burst) seguita da una sequenza di operazioni di I/O (I/O burst), ricominciando poi questa sequenza.



16.2 Scheduler della CPU

Ogni volta che la CPU passa nello stato di inattività il SO scegli per l'esecuzione uno dei processi presenti nella coda dei pronti. Le decisioni riguardanti lo scheduling della CPU si possono prendere nelle seguenti circostanze:

- Un processo passa dallo stato di esecuzione a quello di attesa;
- Un processo passa dallo stato di esecuzione allo stato pronto;
- Un processo passa dallo stato di attesa allo stato pronto;
- Un processo termina.

Nel primo e nel quarto caso si dice che lo schema di scheduling è di tipo non-preemptive (senza diritto di prelazione). Negli altri casi invece lo schema di scheduling è di tipo preemptive

16.3 Il dispatcher

È il modulo che passa effettivamente il controllo della CPU ai processi scelti dallo scheduler a breve termine. Questa funzione riguarda:

- Il cambio di contesto;
- Il passaggio al modo d'utente;
- Il salto alla giusta posizione del programma d'utente per riavviare l'esecuzione

Latenza di dispatch → tempo impiegato dal dispatcher per sospendere un processo e avviare l'esecuzione di un nuovo processo.

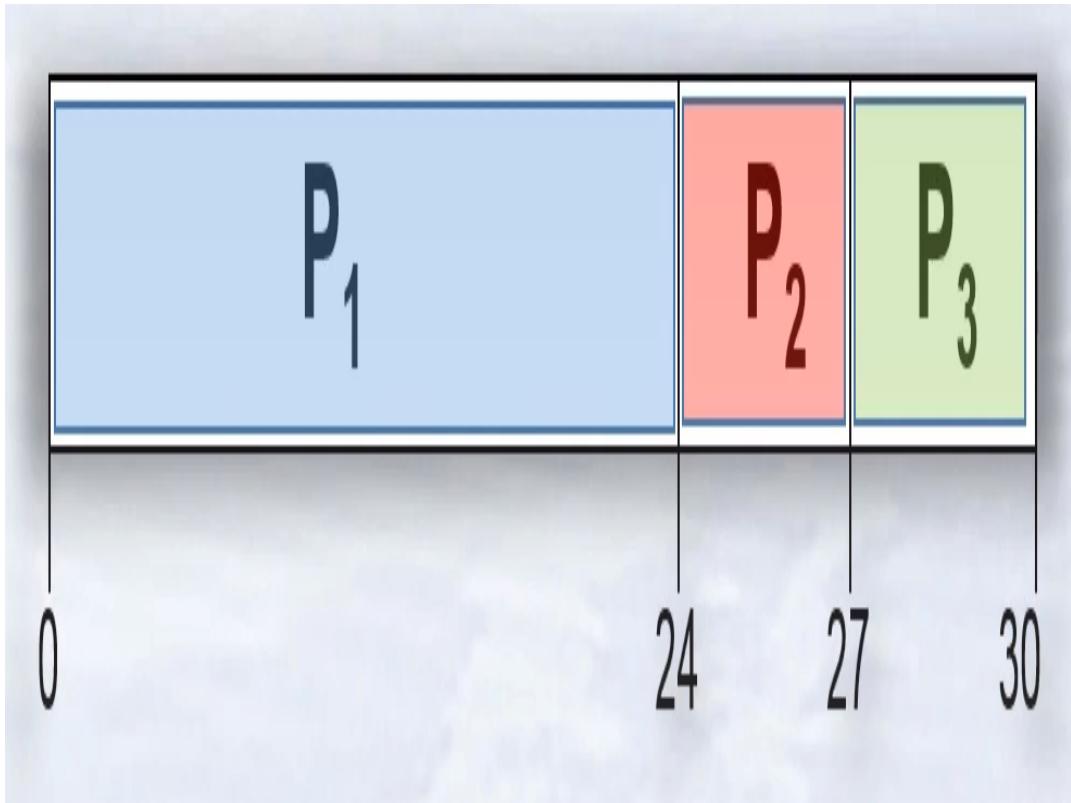
16.4 Criteri di scheduling

- MAX Utilizzo della CPU;
- MAX Produttività;
- Min Tempo di completamento;
- Min Tempo di attesa;
- Min Tempo di risposta.

16.5 Scheduling FCFS (first-come, first-served)

Si supponda che i processi (P1, P2, P3) arrivino al tempo 0 ma nell'ordine crescente (numerico);

Lo schema di Gantt risultante è:



Tempo d'attesa:

- P1 → 0;
- P2 → 24;
- P3 → 27.

Tempo di attesa medio:

$$\frac{0+24+27}{3} = 17$$

Effetto convoglio → tutti i processi attendono che un lungo processi liberi la CPU. (Sarebbe stato meglio nel caso in cui si fosse iniziato con il processo che richiedeva meno tempo, in modo da avere un tempo d'attesa medio minore)

16.6 Scheduling shortest-job-first

Associa a ogni processo la lunghezza della successiva sequenza di operazioni della CPU. Quando è disponibile, si assegna la CPU al processo che ha la più breve lunghezza della successiva sequenza di operazioni della CPU.

Due schemi:

- Senza prelazione → permette al processo correntemente in esecuzione di portare a termine la propria sequenza di operazioni della CPU
- Con prelazione → Se arriva un nuovo processo con sequenza di CPU inferiore a quella del tempo restante per eseguire il processo in corso, lo sostituisce al processo attualmente in esecuzione.

L'algoritmo di scheduling SJF è ottimale nel senso che rende minimo il tempo di attesa medio per un dato insieme di processi.

Nel caso di SJF il tempo d'attesa:

$$T_{\text{attesa}} = T_{\text{fine}} - T_{\text{arrivo}} - T_{\text{durata}}$$

16.7 Scheduling per priorità

Si assegna a ciascun processo un numero (intero) di priorità. Si assegna la CPU al processo con priorità più elevata (intero più piccolo).

L'algoritmo SJF è un caso particolare del più generale algoritmo di scheduling per priorità, dove questa è l'inverso della lunghezza (prevista) dalla successiva sequenza di operazioni della CPU (a una maggiore lunghezza corrisponde una minore priorità, e viceversa).

Problema → starvation (attesa indefinita per i processi a bassa priorità).

Soluzione → introduzione dell'*aging* (aumento graduale della priorità dei processi che attendono nel sistema da parecchio tempo).

Il tempo di completamento medio può migliorare se la maggior parte dei processi termina la successiva sequenza di operazioni della CPU in un solo quanto di tempo

16.8 Scheduling a code multiple

Una distinzione diffusa è quella che si fa tra i processi che si eseguono:

- In primo piano (*foreground*) o Interattivi;
- In *background* o a lotti (*batch*).

Questi 2 tipi di processi possono avere diverse necessità di scheduling:

- In primo piano → RR;
- In sottofondo → FCFS;

È necessario avere uno scheduling tra le code.

Scheduling con priorità fissa e prelazione (possibilità di starvation).

Possibilità di impostare i quanti di tempo.

16.9 Scheduling per sistemi multi CPU

Lo scheduling della CPU diventa più complesso in questo tipo di sistemi.

Sistemi omogenei → sistemi nei quali le unità d'elaborazione sono, in relazione alle loro funzioni identiche.

Condivisione del carico.

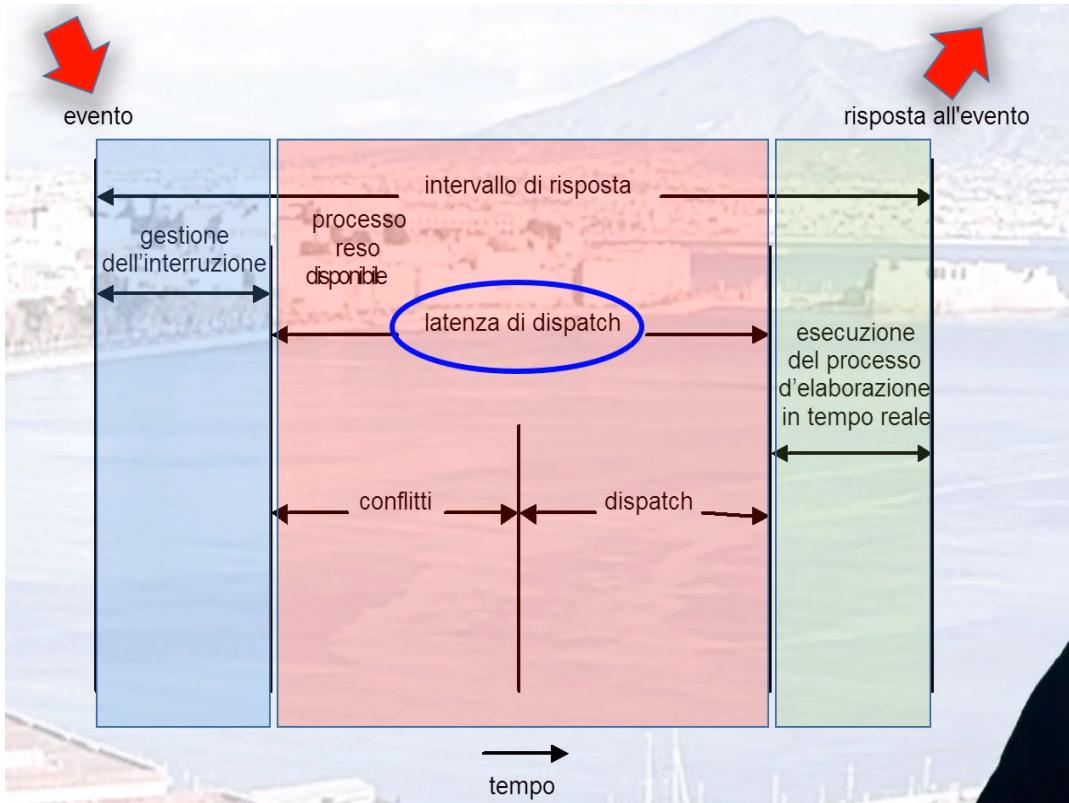
Multielaborazione asimmetrica → tutte le attività del sistema sono gestite da un'unica unità d'elaborazione, che quindi assume il ruolo di unità centrale (master server); le altre unità d'elaborazione eseguono soltanto il codice utente.

16.10 Scheduling per sistemi Real time

Sistemi hard Real time → sistemi capaci di garantire il completamento delle funzioni critiche entro un tempo definito (oltre sarebbe inutile).

Sistemi soft Real time → sistemi nei quali i processi critici hanno una priorità maggiore di quelli ordinari

16.11 Latenza di Dispatch nel dettaglio



16.12 Valutazione degli algoritmi

Modello deterministico → tipo di valutazione analitica che considera un carico di lavoro predeterminato e definisce le prestazioni di ciascun algoritmo per quel carico di lavoro.

Reti di code → se sono noti le distribuzioni degli arrivi e dei servizi si possono calcolare l'utilizzo, la lunghezza media delle code, il tempo medio di attesa, etc...

Simulazioni e realizzazione (tape trace).

17 Sincronizzazione dei processi

Fondamentale per la gestione dei dati, che devono mantenere coerenza nel tempo (vista la possibilità di accesso da parte di più processi allo stesso dato)

17.1 Race condition

Situazione in cui più processi accedono e modificano gli stessi dati in modo concorrente, e i risultati dipendono dall'ordine degli accessi.

Per prevenire questa situazione, i processi concorrenti devono essere sincronizzati.

17.2 Soluzione al problema della sezione critica

1. **Mutua esclusione** → se un processo P è in esecuzione nella sua sezione critica, nessun altro può esserlo nella propria;
2. **Progresso** → se nessun processo è in esecuzione nella sua sezione critica, solo i processi che desiderano entrare nella propria sezione critica possono partecipare alla decisione su chi sarà il prossimo ad entrarci (tal decisione non può essere ritardata indefiniteamente);
3. **Attesa Limitata** → se un processo ha già chiesto l'ingresso nella sua sezione critica esiste un limite al numero di volte che si consente ad altri processi di entrare nelle rispettive sezioni critiche prima che si accordi la richiesta del primo processo
 - (a) Si suppone che ogni processo sia eseguito a una velocità diverso da 0;
 - (b) Non si può fare alcuna ipotesi sulla velocità relativa degli n processi;

17.3 Architetture di sincronizzazione

Controlla e modifica il contenuto di una parola di memoria in modo atomico.

```
boolean TestAndSet(boolean &obiettivo){  
    boolean valore = obiettivo;  
    obiettivo = true;  
    return valore;  
}
```

L'istruzione *swap* agisce sul contenuto di due parole di memoria (eseguita come la precedente in modo atomico)

17.4 Semafori

Strumento di sincronizzazione che non richiede attesa attiva.

Un semaforo S è una variabile intera. A questa variabile si può accedere solo tramite 2 operazioni atomiche predefinite:

- **wait(S)** → while $S \leq 0$ do no-op; $S -$
- **signal(S)**; → $S +$

17.5 Stallo e attesa indefinita

Stallo (deadlock) → situazione in cui 2 o più processi attendono indefiniteamente un evento che può essere causato solo da uno dei processi in attesa.

Siano quindi S e Q 2 semafori inizializzati a 1 → starvation → situazione d'attesa indefinita nella coda di un semaforo.

17.6 Tipi di semaforo

Semaforo contatore → il suo valore intero può variare in un dominio logicamente non limitato;

Semaforo binario → il suo valore intero può essere soltato 0 o 1 (semplice da realizzare).

17.7 Problemi tipici di sincronizzazione

- Problema dei produttori e dei consumatori con memoria limitata;
- Problema dei lettori e degli scrittori;
- Problema dei 5 filosofi.

17.8 Regioni critiche

Costrutto di sincronizzazione ad alto livello.

Una variabile condivisa v di tipo T è dichiarata come:

v : shared T

A questa variabile si può accedere solo dall'interno di un'instruzione:

region v when B do S

(dove B è una espressione booleana).

Nel momento in cui si esegue l'istruzione S nessun processo può accedere alla variabile.

18 Stallo dei processi

Un insieme di processi bloccati, in cui ciascun processo detiene una risorsa e attende di accederne ad un'altra (questa in possesso ad un'altro processo).

Questa problematica può essere risolta con il metodo del rollback (prelazione di risorse e ristabilimento di uno stato sicuro).

18.1 Caratterizzazione delle situazioni di stallo

Si può avere una situazione di stallo solo se si verificano contemporaneamente le seguenti 4 condizioni:

- Mutua esclusione;
- Possesso e attesa;
- Impossibilità di prelazione;
- Attesa circolare.

18.2 Grafo di assegnazione delle risorse

Un insieme di vertici V e di archi E .

Tipi di vertici

- $P = \{ P_1, P_2, \dots, P_n \}$ = tutti i processi
- $R = \{ R_1, R_2, \dots, R_n \}$ = tutti i tipi di risorsa

Tipi di Archi

- Arco di richiesta \rightarrow arco orientato $P_i \rightarrow R_j$
- Arco di assegnazione \rightarrow arco orientato $R_j \rightarrow P_i$

Osservazioni:

- Se il grafo non contiene cicli \rightarrow non si verificano situazioni di stallo;
- Se il grafo contiene un ciclo
 - Se c'è solo un'istanza per tipo di risorsa, allora si verifica una situazione di stallo;
 - Se vi sono più istanza per tipo di risorsa, allora c'è la possibilità che si verifichi una situazione di stallo.

18.3 Metodi per la gestione delle situazioni di stallo

- Assicurare che il sistema non entri mai in stallo;
- Consentire al sistema di entrare in stallo, individuarlo e quindi eseguire il ripristino;
- Ignorare del tutto il problema.

Un metodo alternativo per evitare le situazioni di stallo consiste nel richiedere ulteriori informazioni sui modi di richiesta delle risorse.

Il modello più utile e semplice richiede che ciascun processo dichiari il numero massimo delle risorse di ciascun tipo di cui necessita.

L'algoritmo per evitare lo stallo esamina dinamicamente lo stato di assegnazione delle risorse per garantire che non possa esistere una condizione di attesa circolare.

Lo stato di assegnazione delle risorse è definito dal numero di risorse disponibili e assegnate, e dalle richieste massime dei processi.

18.4 Stato sicuro

Uno stato si dice sicuro se il sistema è in grado di assegnare risorse a ciascun processo (fino al suo massimo) in un certo ordine e impedire il verificarsi di uno stallo.

Un sistema si trova in uno stato sicuro solo se esiste una sequenza sicura.

19 Gestione della memoria

19.1 Spazio di indirizzi di un processo

Lo spazio di indirizzi virtuali che ogni processo può usare è diverso dall'insieme degli indirizzi fisici che usa ad ogni istante

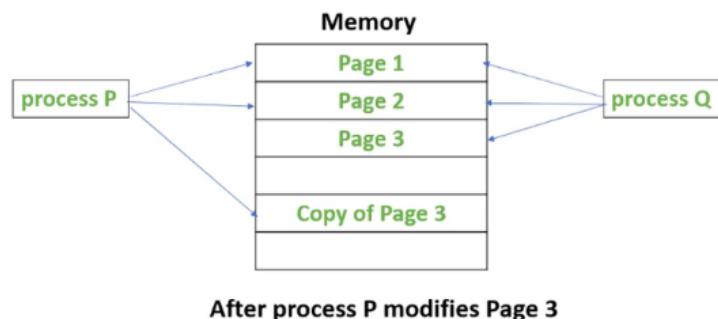
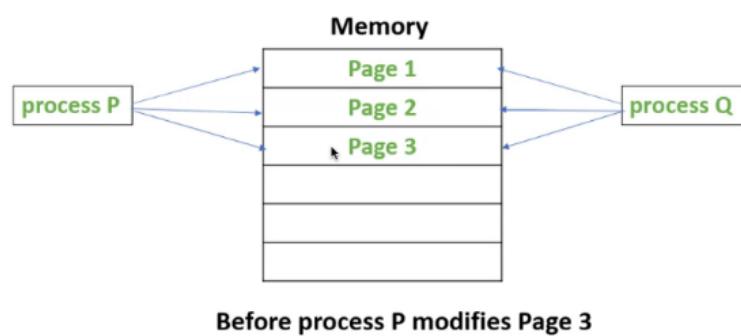
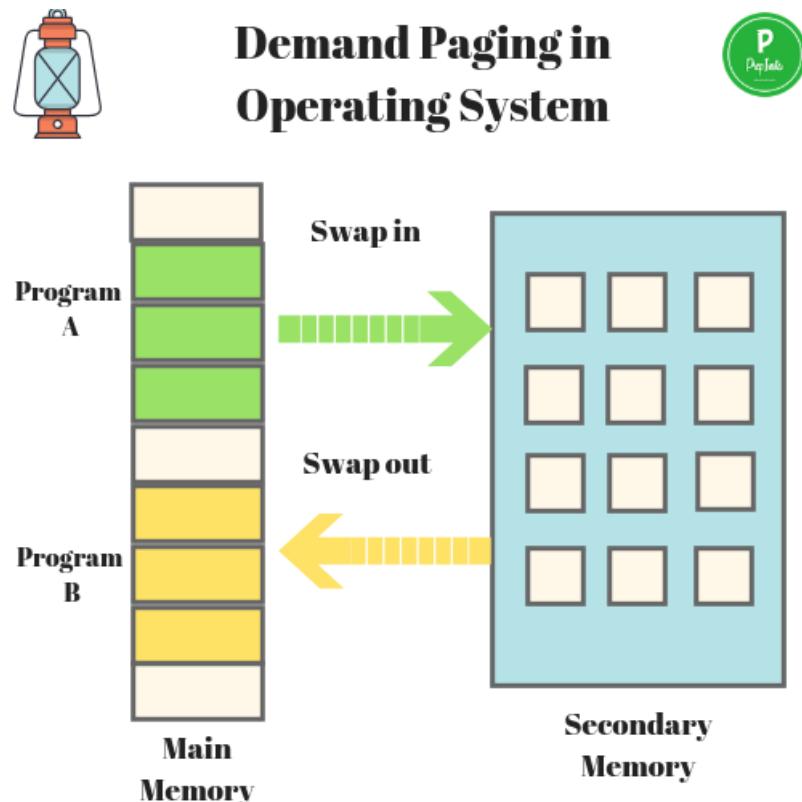
Il kernel rappresenta lo spazio di indirizzi come lista di descrittori di regioni di memoria

Alla creazione di un processo il kernel gli associa uno spazio di indirizzi che include regioni di memoria per:

- codice eseguibile;
- i dati inizializzati (e non) del programma;
- lo stack del programma;
- il codice eseguibile e i dati delle librerie condivise;
- lo heap (memoria dinamicamente richiesta dal processo);
- mapping in memoria di file usati dal programma

I moderni SO Unix usano strategie efficienti di allocazione di pagine basate sul [Demand Paging](#) e [Copy-on-Write](#)

Demand Paging e Copy-on-Write a confronto



Con la `fork` padre e figlio non lavorano sullo stesso spazio di indirizzamento, ma il figlio lavora su una copia dei dati del padre (a differenza dei `thread` che condividono la stessa memoria)

Ogni processo dispone di un proprio spazio di indirizzamento logico $[0, \text{max}]$ che viene allocata nella memoria fisica



19.2 Indirizzo logico

Un numero binario a 48 bit formato da un selettori di segmento (16 bit) e un offset (32 bit)

19.2.1 Segment selector

Identifica il segmento a cui appartiene l'indirizzo di memoria. Diviso in

- TI (1 bit) → indica in quale tabella è immagazzinato il descrittore di segmento
 - GDT → tabella dei descrittori globali;
 - LDT → tabella dei descrittori locali;
- indice (13 bit) → posizione descrittore in tabella (GDT o LDT);
- RPL (2 bit) → flag che indica il CPL (current privilege level) che aveva la CPU quando il segmento viene caricato;
- i 6 registri di segmento (cs, ss, ds, es, fs, gs) contengono i selettori di segmento
 - i rimanenti bit non sono usati (cs che ne usa 2 per il CPL)
 - * cs → code segment;
 - * ss → stack segment;
 - * ds → data segment;

- * es → extra segment;
- * fs, gs → inseriti per operazioni future;



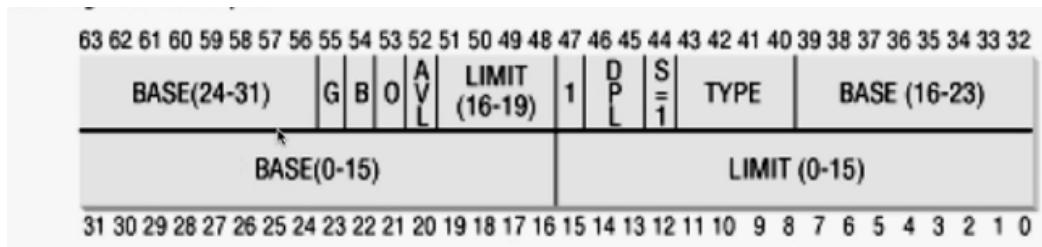
19.2.2 Offset

Indica la posizione dell'indirizzo rispetto all'inizio del segmento

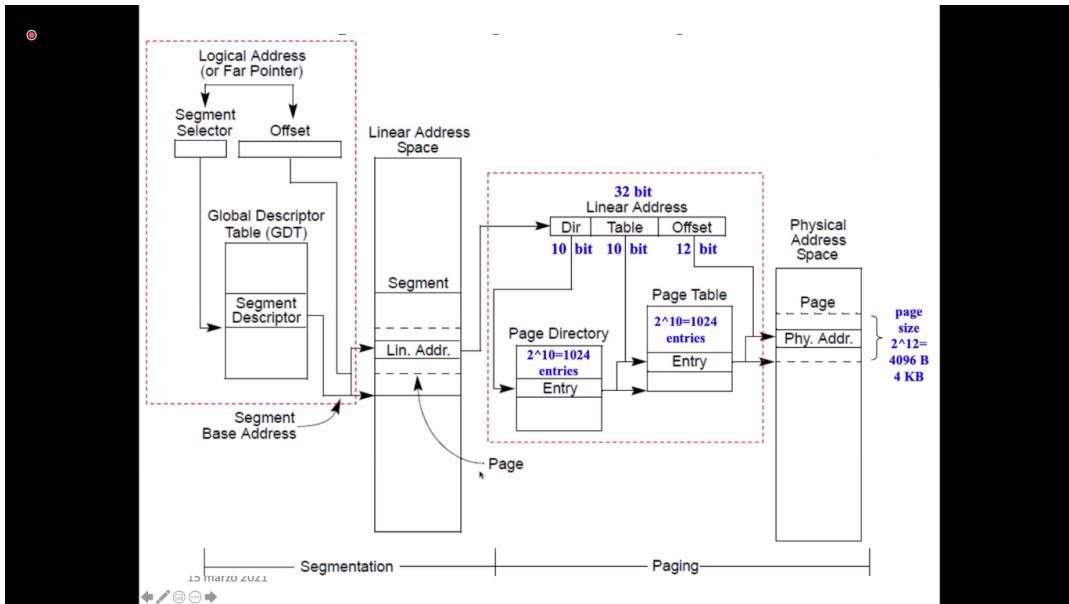
19.2.3 Descrittore di segmento

Dato di 64 bit che descrive le caratteristiche del segmento:

- BASE → indirizzo lineare base del primo byte del segmento;
- LIMIT → dimensione del segmento;
- TYPE → tipo di segmento;
- DPL (Descriptor Privile Level) → livello minimo di privilegi per accedere al segmento;
- bit 47° → 1 se presente in memoria, 0 altrimenti



19.2.4 Traduzione degli indirizzi logici in indirizzi fisici



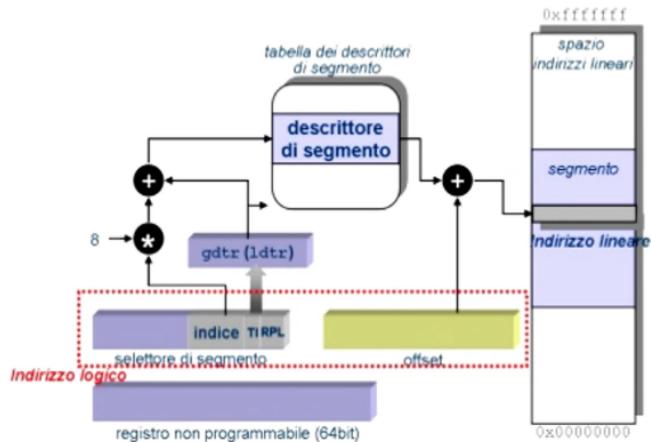
Ogni registro ha un registro non programmabile (con lo scopo di velocizzare la traduzione)

In esso è caricato il descrittore del segmento quando, per la prima volta, è caricato il selettore di segmento nel corrispondente registro segmento

Se il contenuto di un registro segmento cambia (flush), cioè un nuovo selettore è caricato, la segmentazione passa per la GTD (o LDT). Nel caso contrario (segmentazione diretta) non passa per la GTD (o LDT)

- ➊ il valore **TI** determina se il descrittore si trova in GDT o LDT
- ➋ l'**indirizzo base** della GDT (LDT) è nel registro **gdtr** (**ldtr**)
- ➌ l'**indice** $\times 8 + \text{ind. base}$ = posiz. descr. in GDT (o LDT). **Copia descr.** in r.n.p.
- ➍ dal descrittore si ricava l'**indirizzo (lineare)** **base del segmento**
- ➎ **ind. base segmento + offset = indirizzo lineare tradotto**

Figura: segmentazione: accesso in GDT

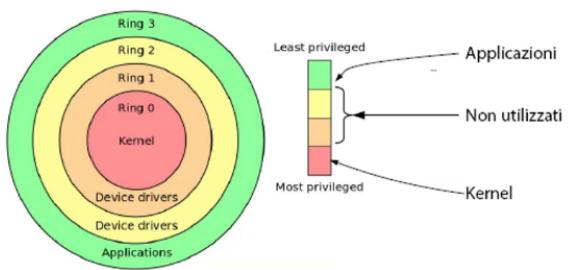


19.2.5 Privilegi di esecuzione

Meccanismo di difesa da istruzioni o accessi alla memoria causati da malfunzionamenti HW o comportamenti malevoli

La CPU è dotata di 4 modi di utilizzo disposti ad anello

- Ring 3:** applicazioni utente.
- Ring 2:** device driver.
- Ring 1:** device driver, hypervisor.
- Ring 0:** device driver e basso livello SO (kernel).

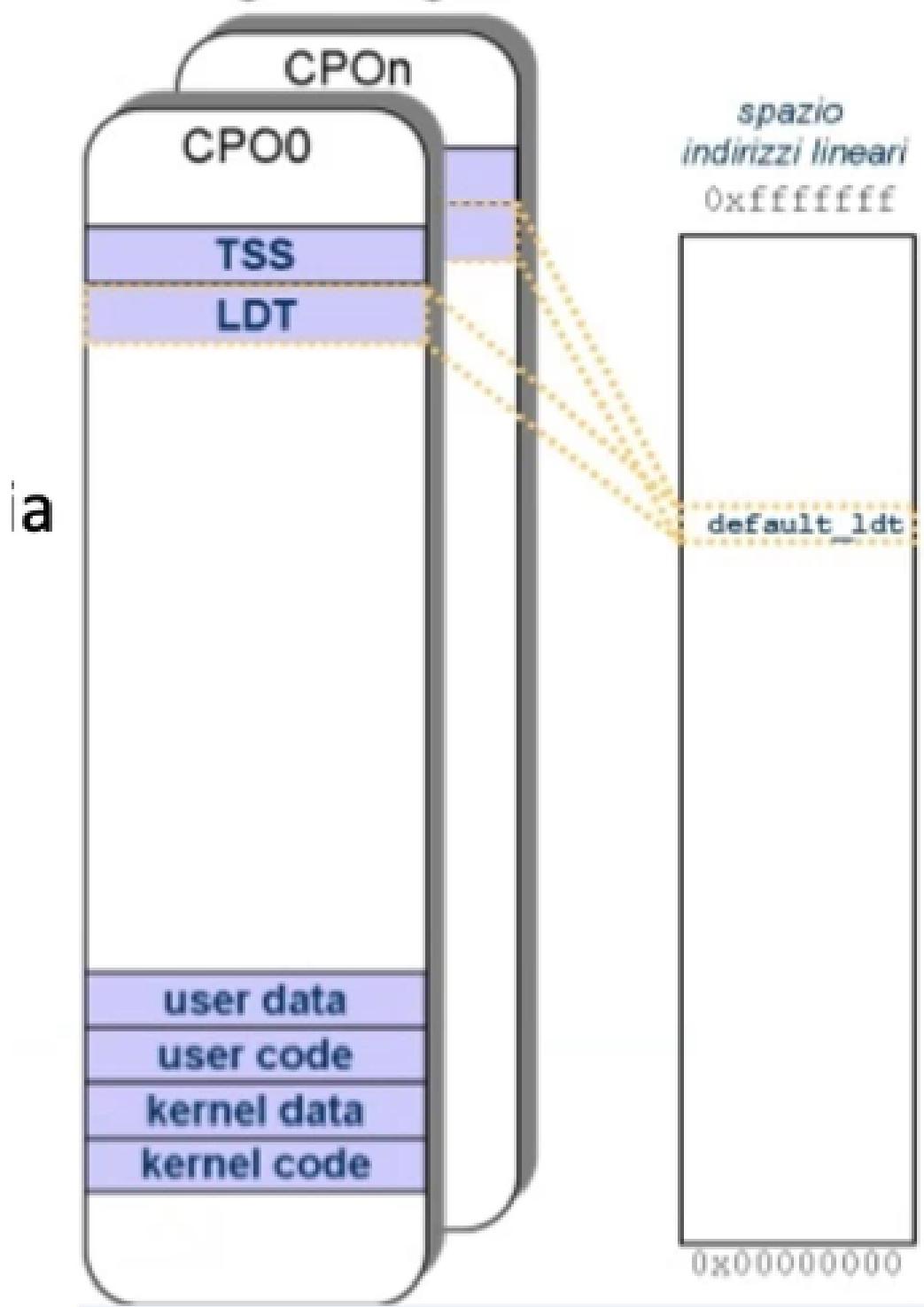


19.2.6 Tabelle dei Descrittori di Segmento

Vengono immagazzinati in GDT o LDT. Queste tabelle sono immagazzinate in memoria RAM.

I registri gdtr e ldtr contengono rispettivamente gli indirizzi fisici base della GDT e LDT

Figura: segmenti in GDT



19.3 Swapping

Meccanismo HW e SW che consente di estendere la memoria centrale utilizzando spazi aggiuntivi su supporti fisici. Lo swapping è basato sulle pagine (swapped-out)

19.4 Sincronizzazione dei processi

Uno dei compiti che spetta al SO è la sincronizzazione dei processi. Più processi su una macchina possono essere:

- In **competizione**
 - se si cerca di usare la stessa risorsa contemporaneamente;
- In **cooperazione**
 - quando uno ha bisogno dell'altro per evolvere;

La sincronizzazione avviene grazie a variabili condivise dette **semafori** o a scambio di messaggi tra processi

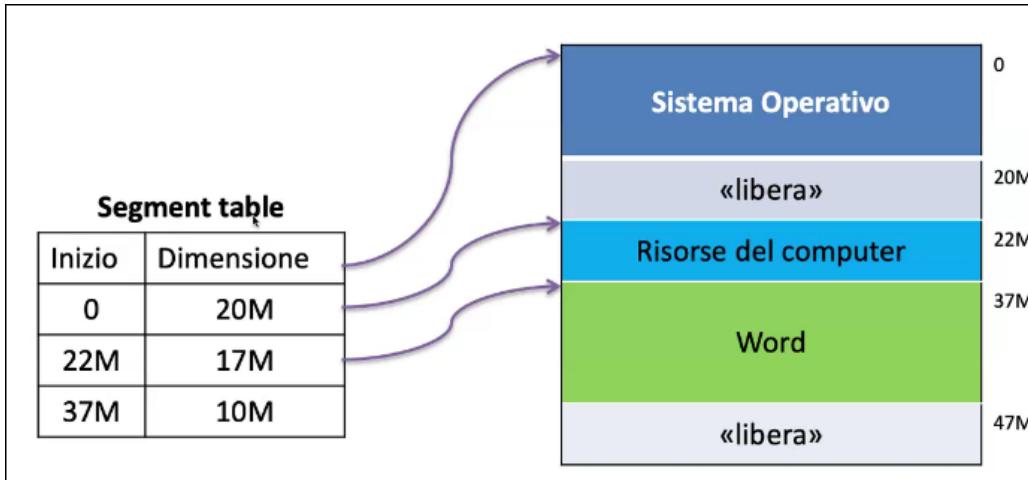
Ogni processo per essere eseguito deve essere caricato in memoria. Nei sistemi multitasking permette di caricare più processi contemporaneamente.

Il gestore della memoria deve:

- Tenere traccia di quali parti della memoria sono usate e da chi;
- Decidere quali processi caricare quando diventa disponibile spazio in memoria;
- Allocare e deallocare memoria.

Per ogni blocco di memoria il SO mantiene un **descrittore** con le informazioni essenziali. La memoria fisica è suddivisa in un determinato numero di aeree (segmenti)

Ogni entry nella **segment table** contiene l'indirizzo di base di un segmento e la sua lunghezza



Lo svantaggio principale è la frammentazione (la memoria non viene sfruttata completamente). Per ovviare a ciò esiste un processo di compattamento → traslazione dei programmi.

Politiche di gestione dei frammenti

- **First-fit** → alloca il primo frammento libero sufficiente;
- **Best-fit** → alloca il più piccolo frammento libero sufficiente;
 - ricerca sull'intera lista e produce frammenti piccoli
- **Worst-fit** → alloca il più grande frammento
 - ricerca sull'intera lista e produce frammenti grandi

Nei moderni SO si utilizza la **paginazione dei segmenti**, ossia un segmento viene realizzato tramite un insieme di pagine

La suddivisione della memoria centrale in blocchi di dimensioni fisse (pagine) tipicamente 2KB o 4KB.

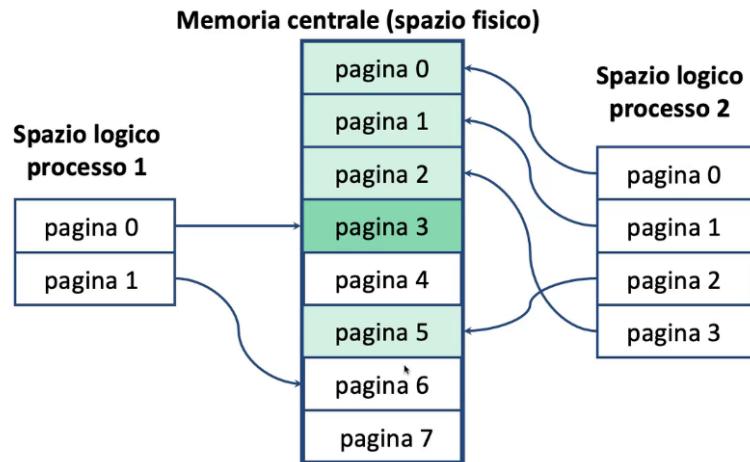
L'immagine di un processo viene suddivisa in pagine caricate all'interno della memoria principale in modo indipendente, anche in posizioni non contigue.

Il SO gestisce una **Page Table** che memorizza per ogni pagina (spazio logico) del processo e la sua posizione in memoria centrale (spazio fisico)

Frammentazione interna ed esterna

- **Frammentazione interna** → Con i blocchi di dimensioni fisse, all'interno di ogni blocco ci sarà uno spazio non utilizzato

- **Frammentazione esterna** → con i blocchi di dimensioni variabili, dopo un certo numero di allocazione e deallocazione di processi, si formeranno un certo numero di porzioni di memoria libere (buchi) di dimensioni insufficienti a contenere un processo



Se durante l'esecuzione del processo, la CPU fa riferimento ad un indirizzo che appartiene ad una pagina non caricata in memoria centrale (page fault), il SO provvede a mettere il processo in stato d'attesa e a far partire il caricamento della pagina richiesta (swap), dopodichè il processo può riprendere la sua esecuzione.

Nella paginazione esiste ancora il problema della frammentazione, che è però limitato al fatto che l'ultima pagina di ogni processo è solo parzialmente occupata. Mediamente si può considerare uno spreco pari a circa 1/2 pagina per ogni processo

19.5 Segmentazione

Schema di gestione della memoria che implementa una visione del programma per "blocchi funzionali"

19.6 Segmentazione vs Paginazione

Entrambi sono ridondanti → entrambi **partizionano/assegnano** la memoria fisica tra processi.

- La segmentazione assegna intervalli di indirizzi lineari a diversi ai processi;
- La paginazione mappa un intervallo di indirizzi lineari su intervalli di indirizzi fisici diversi

19.7 Page

Un blocco di dati che può essere scritto su un frame pagina.

Questo termine si riferisce sia alla pagina di indirizzo che alla pagina di dati memorizzati in RAM in corrispondenza.

Una frame pagina memorizza esattamente il blocco di dati (o istruzioni) contenuti in una pagina (di indirizzi lineari).

La paginazione mappa pagine di indirizzi lineari su frame pagine; una pagina è in RAM o in disco

19.7.1 Page Frame

Area fisica (intervallo di locazioni contigue) di memoria RAM di dimensione fissata

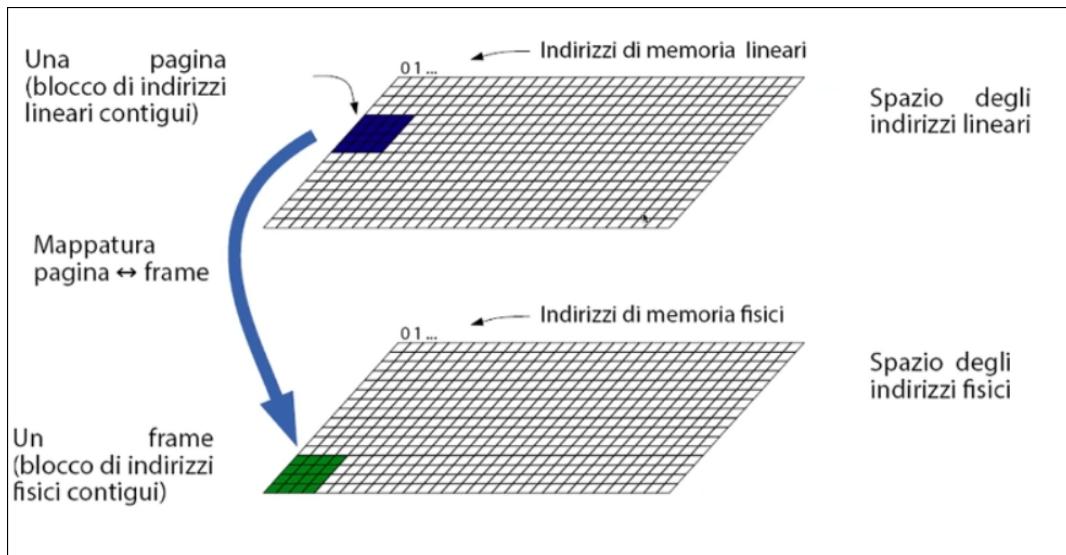
19.7.2 Paging Unit

Circuitistica della MMU che converte indirizzi lineari in fisici. Nella stessa pagina, indirizzi lineari contigui sono paginati su indirizzi fisici contigui.

L'ultimo indirizzo di una pagina e quello della pagina successiva possono essere mappati su indirizzi fisici non contigui

I permessi di accesso e i privilegi associati ad una pagina valgono per tutti gli indirizzi in essa contenuta.

La paginazione verifica anche che la richiesta di accesso abbia privilegi e modalità compatibili con quelli associati alla pagina di appartenenza



19.7.3 La tabella delle pagine

Ha una dimensione alta (viene mantenuta in RAM).

Nei processori intel l'indirizzo fisico iniziale della tabella è salvato nel registro di controllo CR3.

Come detto essendo gigantesca la sua dimensione si fa uso di uno **schema di paginazione gerarchico** → ciascun elemento della tabella delle pagine punta all'inizio di una tabella più piccola (che puntano ai frame fisici)

19.8 Tipi di Paginazione

- A 2 livelli (sistemi a 32 bit);
- A 3 livelli (in quanto il precedente non è più sufficiente);
- A 4 livelli (usati dai sistemi Linux), con 4 tipi di tabelle
 - Page Global Dir (PGD);
 - Page Upper Dir (PUD);
 - Page Middle Dir (PMD);
 - Page Table (PT).

19.9 TLB

Cache HW che velocizza la traduzione di indirizzi lineari in fisici

La prima volta che un indirizzo lineare è tradotto, l'indirizzo fisico corrispondente è registrato in un elemento della TLB. Per usi successivi dello stesso indirizzo lineare si userà l'indirizzo fisico presente in TLB.

Ogni CPU ha la propria TLB → sincronizzazione non necessaria:

- Processi che eseguono su diverse CPU possono usare stessi indirizzi logici corrispondenti a indirizzi fisici diversi;
- TLB funziona da cache per gli elementi della Page Table;
- Se cambia l'indirizzo fisico di una frame pagina (swapping), l'elemento della TLB corrispondente è invalidato;
- Se cambia il valore del registro CR3 (PD) tutta la TLB viene invalidata

19.10 Protezione della memoria

Assicurato (all'intero di ambienti paginati) dai bit di protezione associati a ogni blocco di memoria.

19.11 I file hash

Si servono di una funzione di hash o funzione di randomizzazione, tale che applicata al valore di uno dei campi restituisce l'indirizzo del blocco in cui è memorizzato il record. Consentono operazioni di ricerca molto efficienti.

Per file interni, l'hash è implementato con una tabella di hash.

19.11.1 Funzione di accesso h

Definita come:

$$h: K \rightarrow \{0, 1, 2, \dots, m-1\}$$

Dove:

- $K \rightarrow$ insieme dei valori distinti che il campo chiave può assumere;
- $m \rightarrow$ dimensione del vettore in cui si intende memorizzare gli elementi della tabellla

19.11.2 Divisione

Definita come:

$$h(k) = k \bmod |T|$$

Presenta una bassa complessità, ma può generare fenomeni di agglomerazione

19.11.3 Risoluzione delle Collisioni

Esistono diverse strategie per la risoluzione delle collisioni:

- Concatenamento \rightarrow alla i-esima posizione della tabella è associata la lista degli elementi tali che $h(k)=1$;
- Indirizzamento aperto \rightarrow tutti gli elementi sono contenuti nella tabella.

La scelta della strategia influisce sull'efficienza delle operazioni di inserimento, ricerca e cancellazione.

19.11.4 Tabella Hash per la runqueue

Il Kernel mantiene quattro tabelle hash, che gli permettono di collegare il PID di un processo al suo [process descriptor](#)

Le collisioni sono gestite mediante liste concatenate

20 La memoria virtuale

Sistema SW/HW in grado di simulare uno spazio di memoria centrale maggiore di quello fisicamente presente

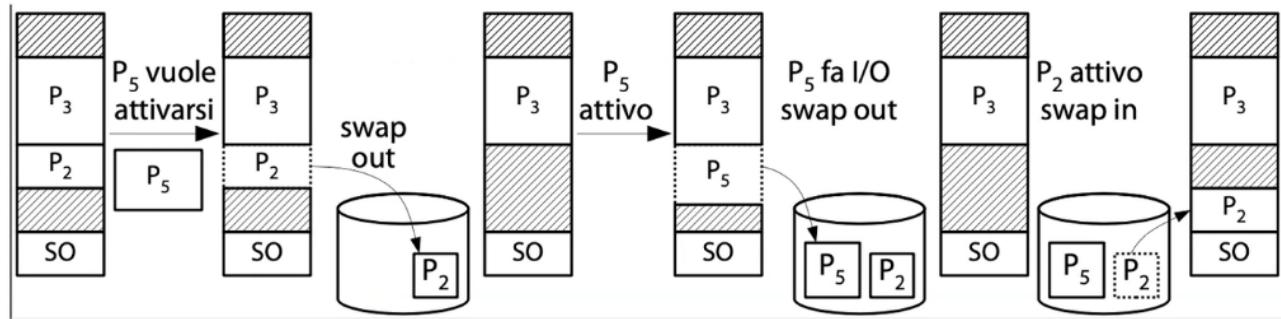
Rappresenta in modo astratto e logico la memoria cetrale di un calcolatore

Memory management unit

Gestisce le richieste di memoria delle applicazione traducendole in istruzioni da impartire all'unità HW di gestione della memoria

Vantaggi:

- Esecuzione concorrente dei processi;
- Esecuzione di applicazioni con richieste di memoria maggiori della memoria fisica;
- Condivisione da parte dei processi di una singola immagine di memoria di una libreria o del programma;
- Programmi rilocati



20.1 Algoritmo FIFO (First-in First-out)

Quando lo scheduler assegna ad una CPU, lascia il process descriptor nella sua posizione corrente all'intero della lista della run-queue (Realtime FIFO scheduler).

Il sistema non interromperà un processo in esecuzione con FIFO ad eccezione dei seguenti casi:

- Un altro processo FIFO con priorità maggiore diventa pronto;
- Il processo è bloccato da una I/O;
- Il processo rilascia volontariamente il processore.

Quando un processo viene interrotto, questo viene inserito nella coda associata alla sua priorità. Nel caso in cui più di un processo ha priorità alta, viene scelto il processo che ha atteso di più.

21 Interfaccia del File System

21.1 Concetto di file

Un file è un insieme di informazioni, correlate e registrate nella memoria secondaria, cui è stato assegnato un nome.

Tipi:

- Dati
 - Numerici;

- Alfabetici;
- Alfanumerici;
- Binari;
- Programmi

21.2 Struttura dei file

Nessuna → sequenza di parole, byte.

Strutture Semplici:

- Righe;
- Lunghezza fissa;
- Lunghezza variabile

Strutture complesse:

- Documento formattato;
- Relocatable load file

21.3 Attributi dei file

Nome	Unica informazione human-readable
Tipo	Informazione necessaria ai sistemi che gestiscono tipi di file diversi
Locazione	Puntatore al dispositivo e alla locazione del file in tale dispositivo
Dimensione	Dimensione corrente del file
Protezione	Permessi sul file
Ora, data e identificazione dell'utente	Dati utili ai fini della protezione e del controllo di utilizzo

Le informazioni sui file sono conservate nella struttura di directory, che risiede a sua volta nella memoria secondaria.

21.4 Operazione sui file

- Creazione;

- Scrittura;
- Lettura;
- Riposizionamento;
- Cancellazione;
- Troncamento;
- **Open** (F_i) → ricerca nella directory del disco l'elemento F_i e ne sposta il contenuto in memoria;
- **Close** (F_i) → rimuove il contenuto dell'elemento F_i dalla memoria alla directory del disco.

21.5 Metodi d'accesso

Accesso sequenziale

```
read next
write next
reset
no read aftr last write (rewrite)
```

Accesso diretto

```
read n
write n
position to n
    read next
    write next
rewrite n
```

Dove $n \rightarrow$ numero relativo del blocco

21.6 Simulazione dell'accesso sequenziale a un file ad accesso diretto

Accesso sequenziale	Realizzazione nel caso di accesso diretto
reset	cp = 0;
read next	read cp; cp = cp+1 ;
write next	write cp; cp = cp+1 ;

21.7 Struttura di directory

Un insieme di nodi contenenti informazioni su tutti i file.

Sia la directory sia i file risiedono sul disco. Le copie di backup di queste 2 strutture vengono in genere archiviate su nastro.

21.8 Tipica organizzazione di un file system

```
/dev/disk/by-uuid/b98a6637-c574-4a25-9023-6d8d1bac0cde / ext4 defaults 0 1 /dev/disk/by-uuid/BF57-4007  
/boot vfat defaults 0 2
```

21.9 Informazioni sui file nella directory

- Nome;
- Tipo;
- Indirizzo;
- Dimensione corrente;
- Dimensione massima;
- Data dell'ultimo accesso;
- Data dell'ultimo aggiornamento;

- ID del proprietario;
- Informazioni di protezione.

21.10 Operazioni eseguite su una directory

- Ricerca di un file;
- Creazione di un file;
- Cancellazione di un file;
- Elencazione di una directory;
- Ridenominazione di un file;
- Navigazione del file system.

21.11 Organizzare (logicamente) una directory

- Efficienza → individuare un file;
- Scelta del nome;
- Raggruppamento → in base alle caratteristiche del file.

21.12 Directory a singolo e doppio livello

Directory a singolo livello → una singola directory per tutti gli utenti.

Directory a doppio livello → directory separate per ciascun utente (directory padre + sottodirectory (dei singoli utenti)).

21.13 Struttura di directory ad albero

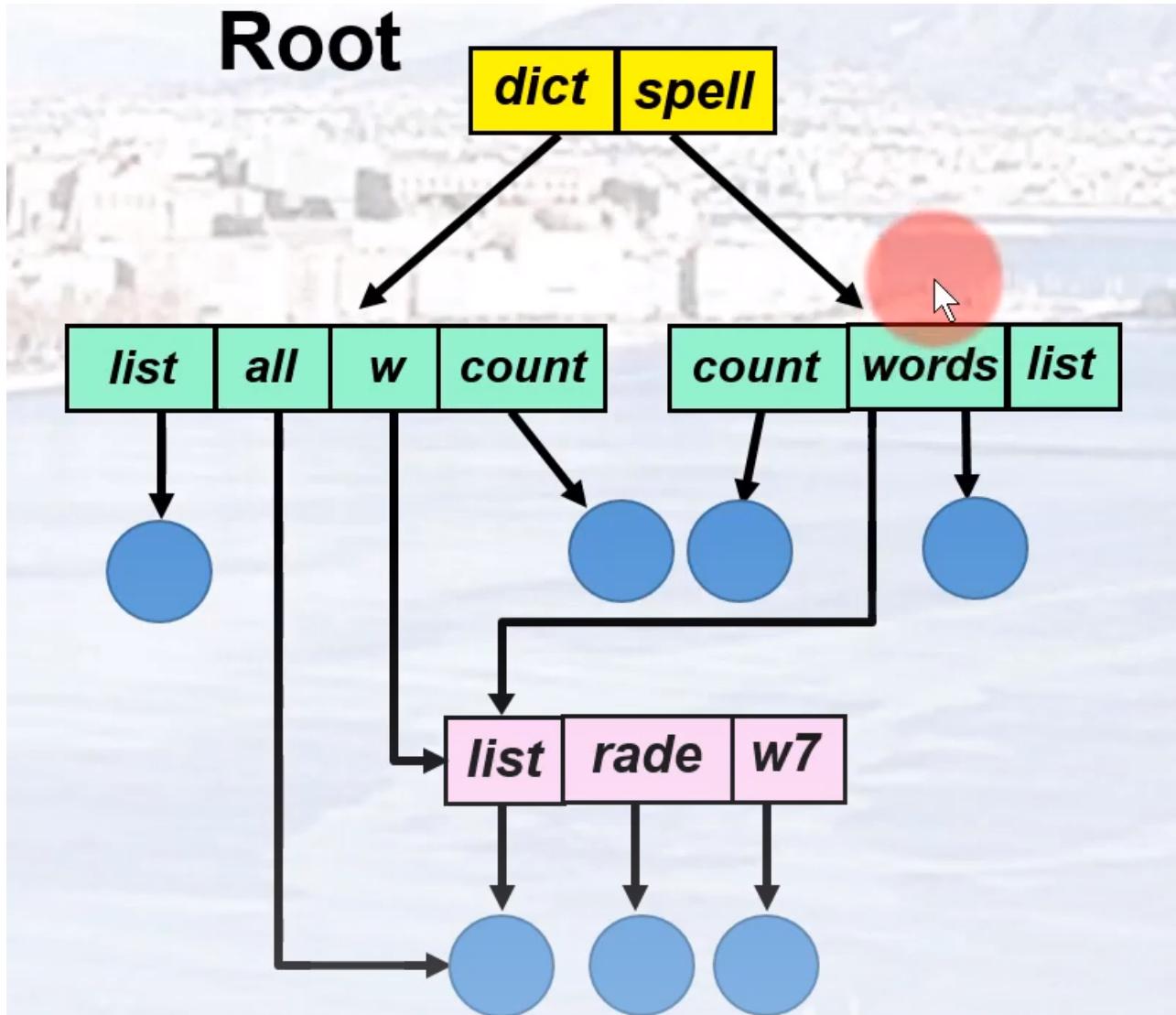
```
$drive
|
└─ $boot (boot partition, fat32)
|
└─ $system (system partition, LUKS2 + btrfs)
    |
    └─ @ (mounted as /)
        |
        └─ /home (mounted @home subvolume)
        |
        └─ /boot (mounted boot partition)
        |
        └─ /swap (mounted @swap subvolume)
        |
        └─ /.snapshots (mounted @snapshots subvolume)
        |
        └─ @home (mounted as /home)
        |
        └─ @swap (mounted as /swap, contains swap file)
        |
        └─ @snapshots (mounted as /.snapshots)
```

- Ricerca efficiente;
- Possibilità di raggruppamento;
- Directory corrente;

Comandi base

`rm`, `mkdir` per rimuovere un file e per creare una cartella (per rimuovere una cartella è necessario il flag `-rf` (ricorsivamente)).

21.14 Struttura di directory a grafo aciclico



Consente alla directory di avere sottodirectory e file condivisi (cosa che si può fare banalmente con un link di file e cartelle, ma a quanto pare nessuno sa c:))

- Nomi diversi che possono riferirsi allo stesso file (aliasing);
- Se a ogni operazione di cancellazione segue l'immediata rimozione del file, potrebbero restare puntatori a un file che esiste più

Per sopperire a questo problema:

- Cercare tutti i collegamenti e rimuoverli;

- Conservazione del file fino alla rimozione di tutti i suoi riferimenti.

21.15 Montaggio di un file system

Per essere reso accessibile, un file system deve essere montato (MA VA).

```
mount /dev/disk/by-uuid/BF57-4007 /mnt
```

21.16 Condivisione di file

Caratteristica nei sistemi multiutente fortemente desiderabile.

La condivisione può avvenire attraverso uno schema di protezione. Nei sistemi distribuiti, i file sono condivisi attraverso la rete.

Il file system di rete (NFS) è uno dei metodi più comuni per la condivisione.

21.17 Protezione dei file

Il **proprietario/creatore** di un file deve essere in grado di controllare:

- Che cosa possa essere fatto su un file e da chi.

Tipi di accesso:

- Lettura;
- Scrittura;
- Esecuzione;
- Aggiunta;
- Cancellazione;
- Elencazione.

drwxr-xr-x	- valentino 10 giu 16:36 config
drwxr-xr-x	- valentino 4 giu 23:20 Documenti
drwxr-xr-x	- valentino 14 mag 13:56 Dropbox

21.17.1 Modalità d'accesso

Lettura (R), scrittura (W) ed esecuzione (X).

Tre classi di utenti;

- Proprietario (7) → 111 (RWX);
- Gruppo (6) → 110 (RWX);
- Universo (1) → 001 (RWX)

22 Struttura del file system

22.1 Struttura del file

- Unità di memorizzazione logica;
- Raccolta di informazioni correlate.

Il file system risiede permanentemente nella memoria secondaria. Ha una struttura stratificata.

Blocco di controllo (FCB, *file control block*), struttura di memorizzazione che contiene informazioni sui file (proprietà, permessi e posizione)

22.2 File system stratificato

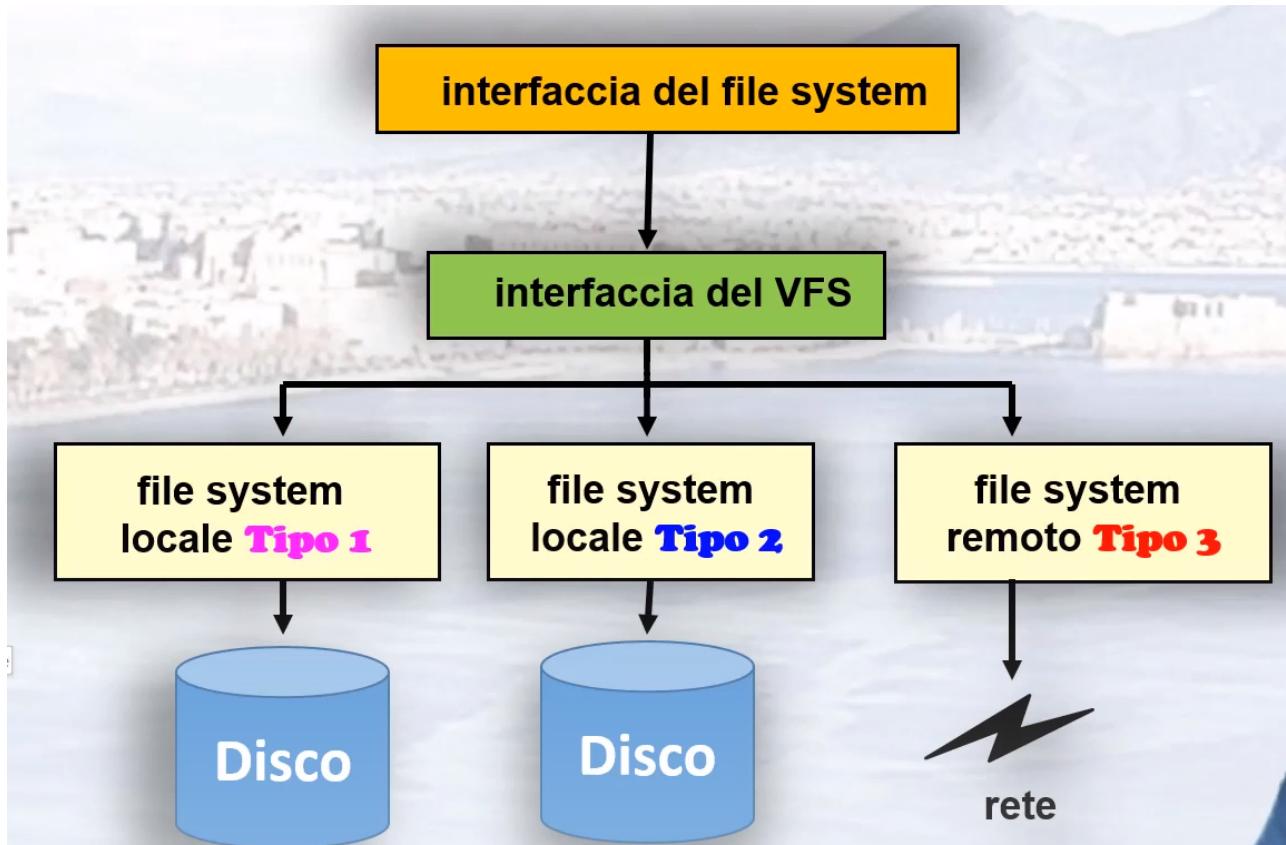
Programmi di applicazione → file system logico → modulo di organizzazione dei file → file system di base → controllo dell' I/O → dispositivi.

22.3 File system virtuali (VFS)

Fornisce una tecnica OO per la realizzazione del file system. Separa le operazioni generiche del file system dalla loro realizzazione definendo un'interfaccia VFS uniforme.

Il VFS è basato su una struttura di rappresentazione dei file detta `vnode` che contiene un indicatore numerico per tutta la rete di ciascun file.

22.3.1 Schema di un file system virtuale



22.4 Realizzazione delle directory

Lista lineare contenente i nomi dei file con puntatori ai blocchi di dati:

- Metodo di facile programmazione, ma onerosa in termini di tempo.

Tabella hash → Lista lineare con struttura di dati hash:

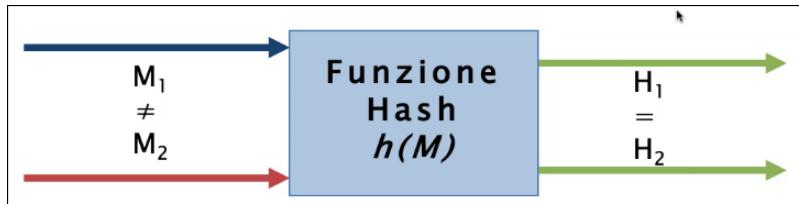
- Diminuisce il tempo di ricerca nella directory;
- Si va in contro a collisioni (dimensione fissa).

22.5 Funzioni di Hash

22.5.1 Collisione

$$h: \Sigma^* \rightarrow \Sigma^n$$

$$h(m_1) = h(m_2)$$



Esistono infinite collisioni

22.5.2 Proprietà

- One-Way → dato un hash y è computazionalmente difficile trovare M : $y=h(M)$;
- Sicurezza debole (2nd pre-image) → dato M è computazionalmente difficile trovare una variazione di M , M^1 tale che:

$$h(M) = h(M^1)$$

- Sicurezza forte (collision resistance) → computazionalmente difficile trovare 2 diversi messaggi con lo stesso valore hash.

Una One-Way Hash Function:

- Verifica le proprietà pre-image e 2nd pre-image resistant;
- Viene detta weak one-way has function.

Una Collision Resistant Hash Function:

- Verifica la proprietà di collision resistance;
- Viene detta strong one-way has function.

Una funzione f è One-Way se:

- Per ogni x nel dominio di f è facile calcolare $y=f(x)$, ma dato y , è computazionalmente inammissibile trovare x tale che $y=f(x)$.

Differenze con OWHF:

- Non ci sono limitazioni sul codominio;
- Non è richiesta la sicurezza debole

22.5.3 Costruzione della funzione

- Il messaggio input M viene diviso in k blocchi di lunghezza fissa m_1, m_2, \dots, m_k ;
- I blocchi vengono trattati in modo
 - **Serial**/Iterato → una collisione per $h(M)$ implica una collisione di f (padding);
 - Parallel → resistente alle collisioni se lo è la funzione h .

Modello HASH a cascata

$$h(M) = H1(M) * H2(M)$$

È dato dal prodotto di 2 funzioni hash → una collisione per $h(M)$ vuol dire trovare una collisione sia per $H1$ che per $H2$

22.5.4 Cifrari a blocchi

- Cifrario a blocchi E_k per input ad n bit;
- Funzione g che da n bit produce una chiave;

22.6 Metodi di assegnazione

Indica il modo in cui i blocchi di disco vengono assegnati ai file:

- Assegnazione contigua;
- Assegnazione concatenata;
- Assegnazione indicizzata.

22.6.1 Assegnazione contigua

Ciascun file deve occupare un insieme di blocchi contigui nel disco

Semplice → è richiesto solo l'indirizzo del primo blocco e la sua lunghezza

Contro → spreco di spazio

22.6.2 Assegnazione concatenata

Ciascun file è composto da una lista concatenata di blocchi del disco i quali possono essere sparsi in qualsiasi punto del disco stesso

blocco = puntatore

Semplice → richiede solo l'indirizzo di partenza.

Sistema di gestione dello spazio libero:

- Nessuno spreco di spazio;
- Nessun accesso casuale.

Una variante del metodo di assegnazione consiste nell'uso della tabella di assegnazione dei file.

22.6.3 Assegnazione indicizzata

Raggruppa tutti i puntatori in una sola locazione (blocco indice).

- Necessita quindi della tabella indice;
- Accesso casuale e dinamico senza frammentazione esterna.

Ogni file deve avere un blocco indice, quindi è auspicabile che questo sia quanto più piccolo possibile (non troppo perchè altrimenti non può contenere i puntatori sufficienti per un file di grosse dimensioni).

Necessita quindi di un meccanismo per la gestione di questa situazione:

- Schema concatenato;
- Indice a più livelli;
- Schema combinato.

22.7 Gestione dello spazio libero

Vettore di bit (n blocchi)

$\text{bit}[i] = 1 \rightarrow \text{blocco}[i]$ libero

$\text{bit}[i] = 0 \rightarrow \text{blocco}[i]$ assegnato

Il numero del primo blocco libero è dato dalla seguente espressione:

(numero di bit per parola) x (numero di parole di valore 0) + scostamento del primo bit 1

22.8 Efficienza e prestazioni

L'efficienza dipende da:

- Algoritmi usati per l'assegnazione del disco;
- Gestione delle directory

Prestazioni:

- Cache del disco;
- Rilascio indietro (*free-behind*) e lettura anticipata (*read-ahead*) → tecniche di ottimizzazione degli accessi sequenziali;
- Per migliorare le prestazioni nei PC si riserva e si gestisce una sezione della memoria come un disco virtuale o disco RAM.

22.9 I/O senza una buffer cache unificata

Tecniche di memoria virtuale per la gestione dei dati dei file come pagine anzichè come blocchi di file system

22.10 Ripristino

Verifica della coerenza dei dati (si cerca di correggere ogni incoerenza).

Backup di dati come metodologia per ovviare a incoerenze di dati. In caso contrario c'è il restore.

22.11 NFS

Uno dei suoi scopi era quello di operare in un ambiente eterogeneo di calcolatori, sistemi operativi e architetture di rete.

Questa indipendenza si ottiene usando primitive RPC costruite su un protocollo di rappresentazione esterna dei dati (XDR) usato tra 2 interfacce indipendenti dalla realizzazione.

Presenta un montaggio classico e uno a cascata.

22.12 Protocollo di montaggio

Stabilisce una connessione logica iniziale tra server e client.

Comprende il nome della directory remota da montare e il nome del calcolatore server in cui tale directory è memorizzata.

Quando il server riceve una richiesta di montaggio conforme alla propria lista di esportazione, riporta al client una maniglia di file (*file handle*) da usare come chiave per ulteriori accessi al file.

La maniglia di file contiene tutte le informazioni di cui ha bisogno il server per gestire un proprio file.

22.13 Protocollo NFS

Offre un insieme di RPC per operazioni su file remoti che svolgono le seguenti operazioni:

- Ricerca di un file in una directory;
- Lettura di un insieme di elementi di una directory;
- Manipolazione di collegamenti e di directory;
- Accesso ad attributi di file;
- Lettura e scrittura di file.

Assenza dell'informazione di stato → ogni richiesta deve fornire un insieme completo di argomenti.

I dati modificati si devono riscrivere nei dischi del server prima che i risultati siano riportati al client.

Il protocollo NFS non fornisce meccanismi per il controllo della concorrenza.

22.14 Architettura NFS

Interfaccia del file system UNIX (basata su chiamate open, read, write, e close e sui descrittori di file)

File system virtuale → identifica i file locali da quelli remoti e invoca l'appropriata operazione del file system.

Vantaggio → client e server sono identici (un calcolatore può essere client, server o entrambi).

22.15 Traduzione dei nomi di percorso

Si compie suddividendo il percorso stesso in nomi componenti ed eseguendo una chiamata lookup nell'NFS separata per ogni coppia formata da un nome componente e un vnode di directory.

Una cache per la ricerca dei nomi delle directory, nel sito del client, conserva i vnode per i nomi delle directory remote; in questo modo si accellerano i riferimenti ai file con lo stesso nome di percorso iniziale.

23 Sistemi di I/O

23.1 Architetture e dispositivi di I/O

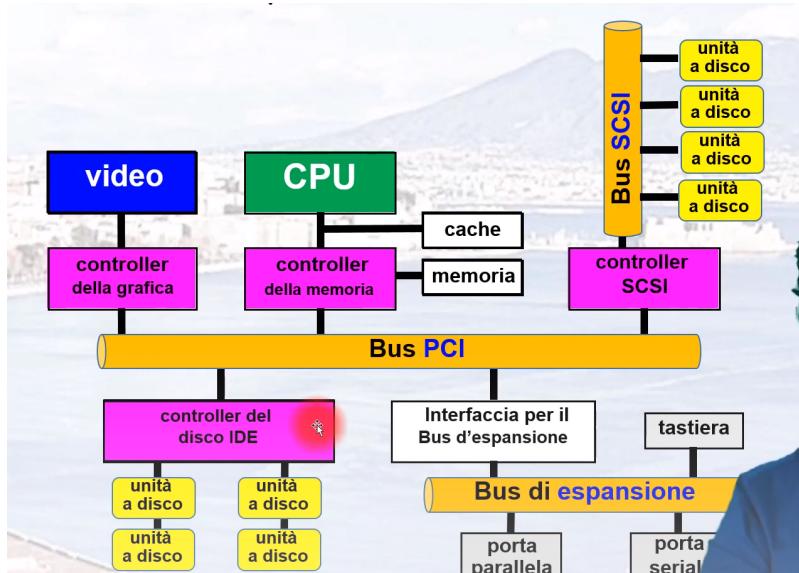
- Grande numero di tipi di dispositivi;
- Concetti comuni
 - Porta;
 - Bus (collegamento a margherita o accesso diretto condiviso);

- Controller.
- Dispositivi di controllo delle istruzioni di I/O;

I dispositivi hanno indirizzi usati da:

- Speciali istruzioni di I/O;
- I/O associato alla memoria.

23.2 Tipica struttura del bus di un PC



23.3 Indirizzi delle porte dei dispositivi di I/O nei PC (elenco parziale)

Indirizzi per l' I/O	Dispositivo
000-00F	Controller DMA
020-021	Controller delle interruzioni
040-043	Temporizzatore
200-20F	Controller dei giochi
2F8-2FF	Porta seriale (secondaria)
320-32F	Controller del disco
378-37F	Porta parallela
3D0-3DF	Controller della grafica
3F0-3F7	Controller dell'unità a dischetto
3F8-3FF	Porta seriale (principale)

23.4 Interrogazione ciclica (polling)

Determina lo stato del servizio:

- Command-ready;
- Busy;
- Error.

L'interrogazione ciclica è in sè un'operazione efficiente; tale tecnica diviene però inefficiente se le ripetute interrogazioni trovano raramente un dispositivo pronto per il servizio mentre altre utili elaborazioni attendono la CPU.

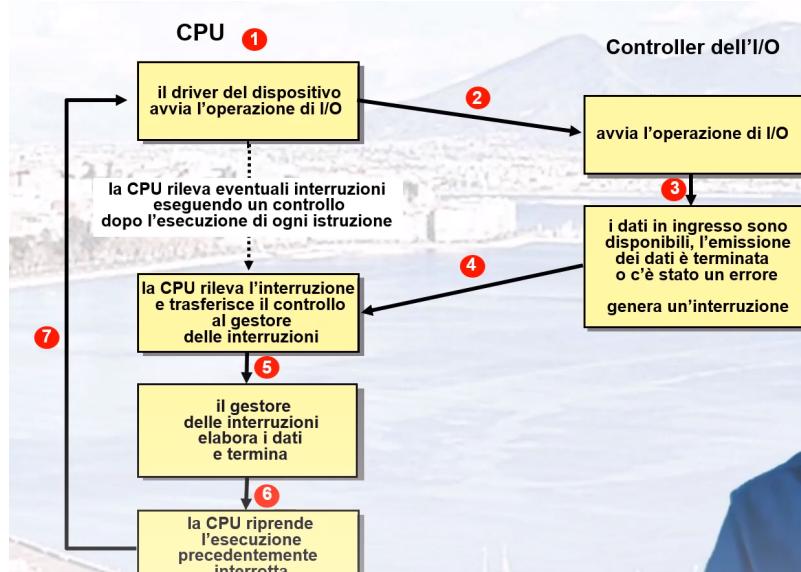
23.5 Interrogazione (interrupt)

I segnali d'interruzione sono usati diffusamente dai SO moderni per gestire eventi asincroni e per eseguire nel modo supervisore le procedure del nucleo.

I controller dei dispositivi, gli errori e le chiamate del sistema generano segnali d'interruzione al fine di innescare l'esecuzione di procedure del nucleo.

Poichè le interruzioni sono usate in modo massiccio per affrontare situazioni in cui il tempo è un fattore critico, è necessario avere un'efficiente gestione delle interruzioni per ottenere buone prestazioni del sistema.

23.6 Cicli di I/O basato sulle interruzioni



23.7 Accesso diretto alla memoria (DMA)

Usato per l'evitare l' I/O programmato per trasferimento di grandi quantità di dati. Richiede un controller DMA.

Questo agisce direttamente sul bus della memoria, presentando al bus gli indirizzi di memoria necessari per eseguire il trasferimento senza aiuto da parte della CPU.

23.8 Interfaccia di I/O per le applicazioni

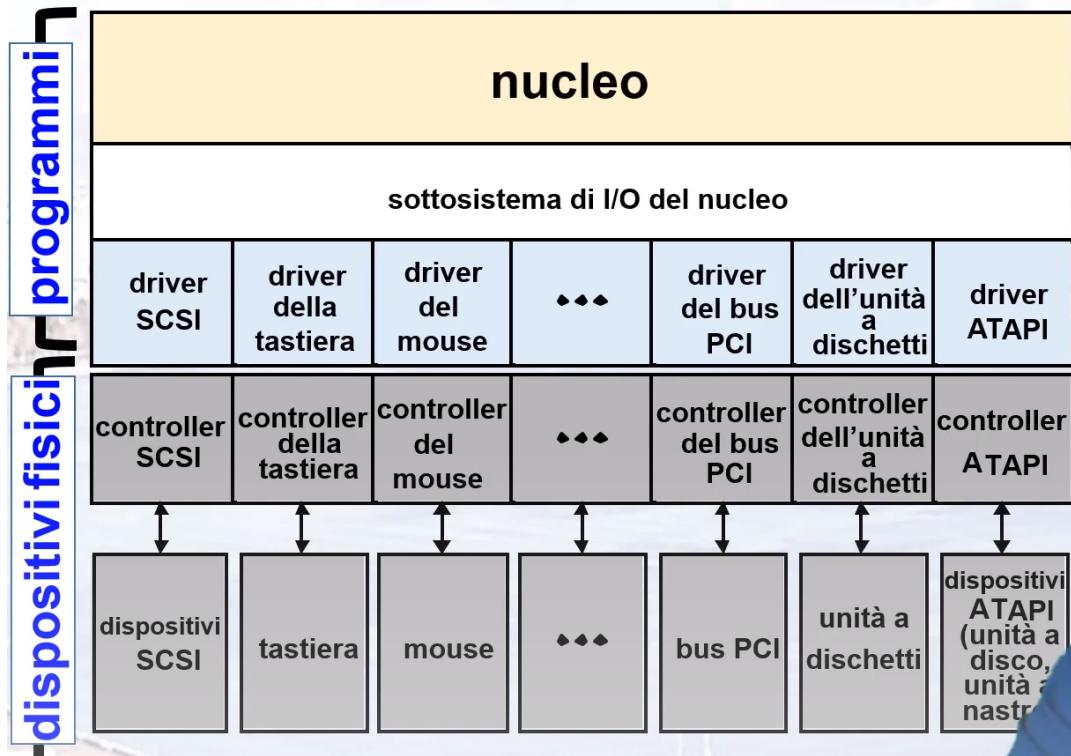
Le chiamate del sistema di I/O encapsulano il comportamento dei dispositivi in alcune classi generiche.

Lo scopo dello strato dei driver dei dispositivi è di nascondere al sottosistema di I/O del nucleo le differenze fra i controller dei dispositivi.

I dispositivi possono ovviamente differire in molti aspetti:

- Trasferimento a flusso di caratteri o a blocchi;
- Accesso sequenziale o diretto;
- Sincroni o asincroni;
- Condivisibili o riservati;
- Velocità di funzionamento;
- Lettura e scrittura, solo lettura o solo scrittura.

23.9 Struttura relativa all' I/O di un nucleo



23.10 Dispositivi con trasferimento a blocchi o a caratteri

I dispositivi con trasferimento a blocchi includono le unità a disco:

- Le istruzioni comprendono `read`, `write` e `seek`;
- I/O a basso livello (`raw I/O`) o accesso tramite file system;
- Possibile accesso al file associato alla memoria.

I dispositivi con trasferimento a caratteri includono tastiere, mouse, porte seriali:

- I comandi comprendono `get`, `put`;
- È possibile costruire servizi aggiuntivi quali l'accesso riga per riga.

23.11 Dispositivi di rete

Interfaccia di rete `socket` → separa il protocollo di rete dalle operazioni di rete (include la funzione `select`).

Gli approcci variano ampliamente (pipe half-duplex, code FIFO full-duplex, STREAMS, code di messaggi e socket).

23.12 Orologi e temporizzatori

Segnano l'ora corrente, segnalano il tempo trascorso, regolano il temporizzatore.

Il dispositivo che misura la durata di un lasso di tempo e che può avviare un'operazione si chiama **temporizzatore programmabile**.

`ioctl` tratta gli aspetti dell' I/O quali orologi e temporizzatori.

23.13 I/O bloccante e non bloccante

Bloccante → si sospende l'esecuzione dell'applicazione.

Non bloccante → sovrappone elaborazione e I/O.

Chiamate del sistema asincrone → restituiscono immediatamente il controllo al chiamante senza attendere che l'I/O sia stato completato.

23.14 Sottosistema per l' I/O del nucleo

Scheduling → fare lo scheduling di una serie di richieste dell' I/O significa stabilirne un ordine d'esecuzione efficace.

Memorizzazione transitoria → un buffer è un'area di memoria che contiene dati mentre vengono trasferiti tra 2 dispositivi o tra un'applicazione e un dispositivo:

- Necessita di gestire la differenza di velocità fra il produttore e il consumatore di un flusso di dati;
- Gestione dei dispositivi che trasferiscono dati in blocchi di dimensioni diverse;
- Realizzazione della *semantica delle coppie*.

Cache → regione di memoria veloce per copie di dati.

Code (spooling) → memoria di transito contenente dati per un dispositivo che non può accettare flussi di dati intercalati.

23.15 Gestione degli errori

Un SO che usa la protezione della memoria può proteggersi da molti tipi di errori dovuti ai dispositivi o alle applicazioni.

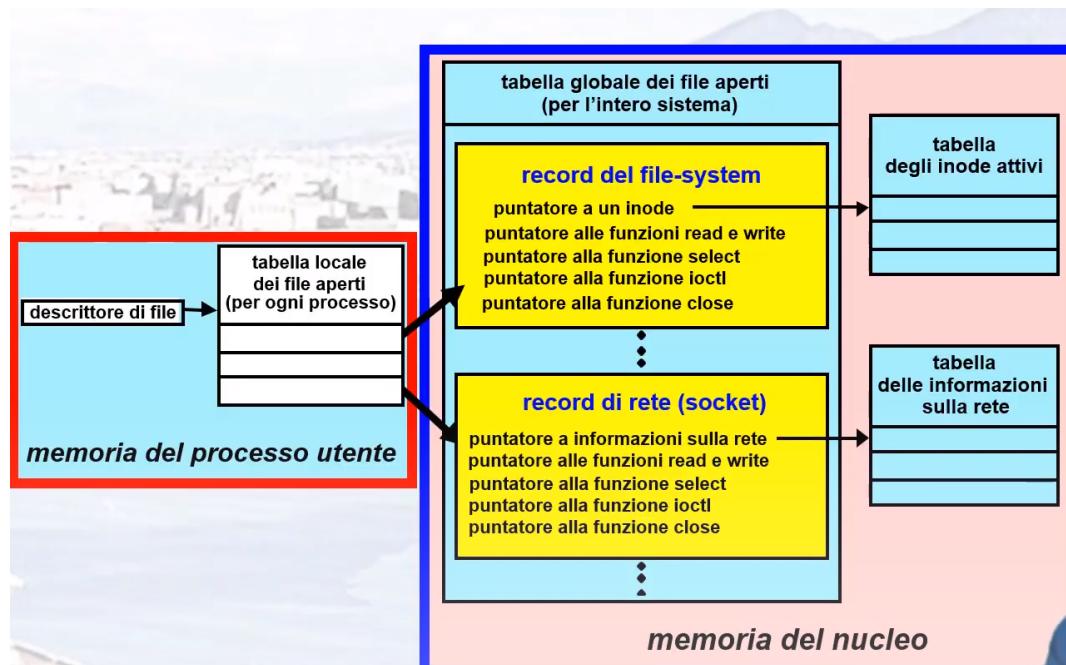
Di norma una chiamata del sistema per l' I/O riporta un bit d'informazione sullo stato d'esecuzione della chiamata.

Molti dispositivi SCSI mantengono alcune pagine di informazioni su errori avvenuti; queste pagine possono essere richieste dalla macchina (accade raramente).

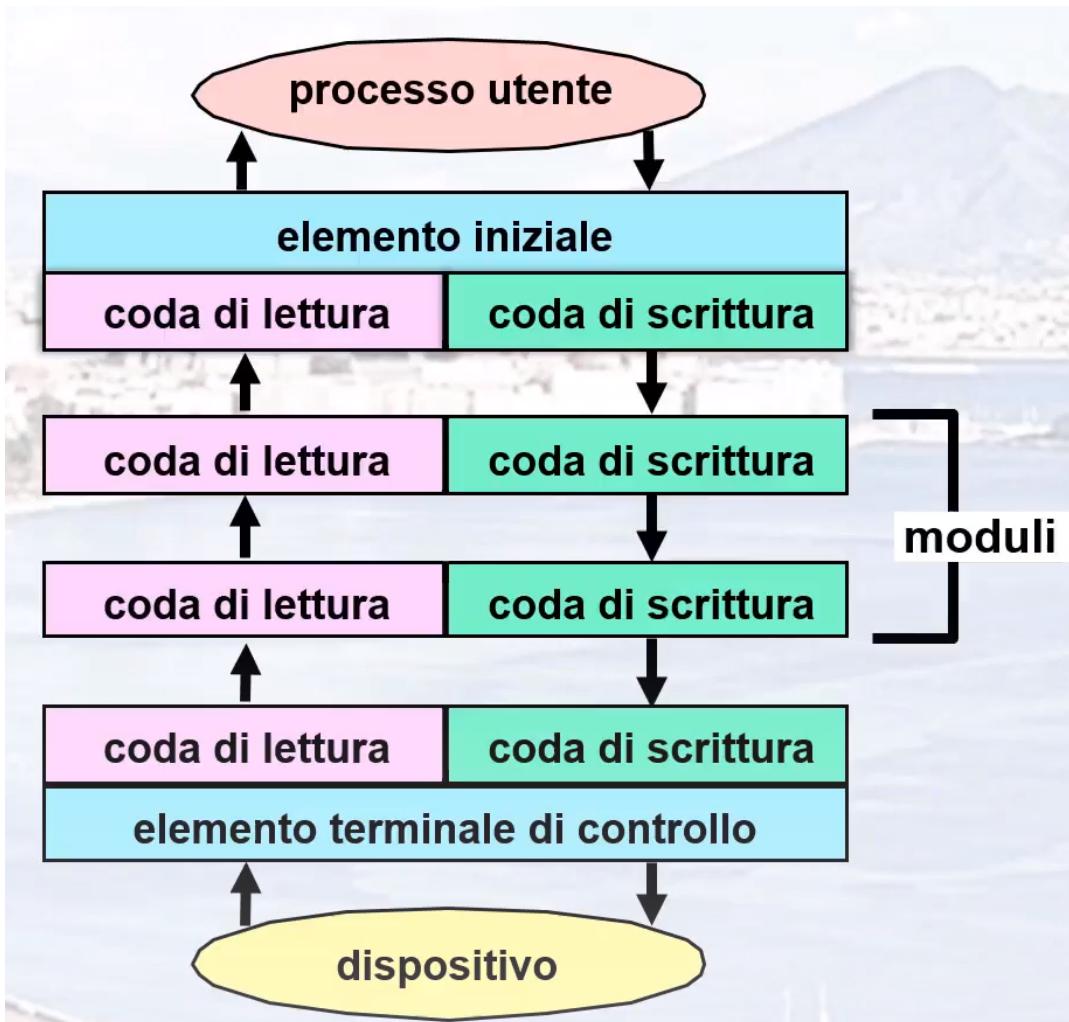
23.16 Strutture dati del nucleo

Il nucleo ha bisogno di mantenere informazioni sullo stato dei componenti coinvolti nelle operazioni di I/O.

23.17 Strutture dati del nucleo per la gestione dell' I/O in UNIX



23.18 Struttura di STREAMS

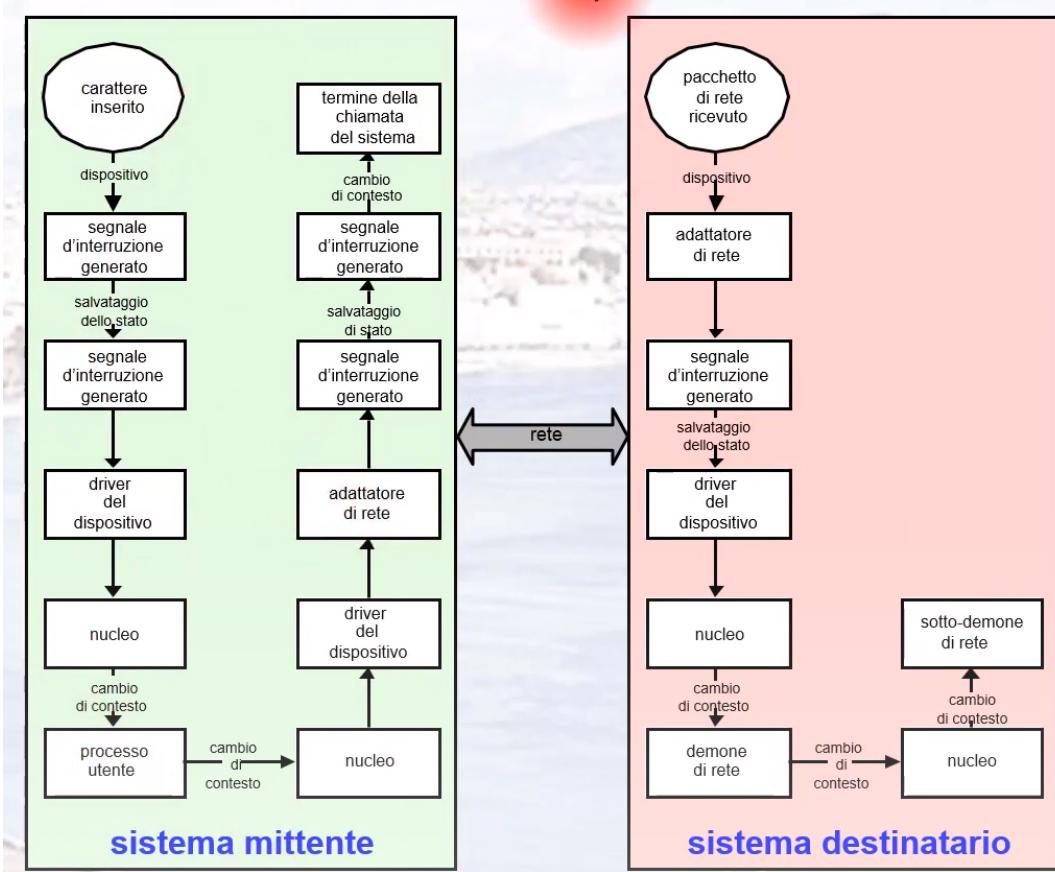


23.19 Prestazioni

L'I/O è uno tra i principali fattori che influiscono sulle prestazioni di un sistema:

- Richiede un notevole impiego della CPU per l'esecuzione del codice del driver e per uno scheduling equo ed efficiente;
- I risultanti context-switch sfruttano fino in fondo la CPU e le sue memorie cache;
- Copia dei dati;
- Traffico di rete.

23.20 Comunicazione fra calcolatori

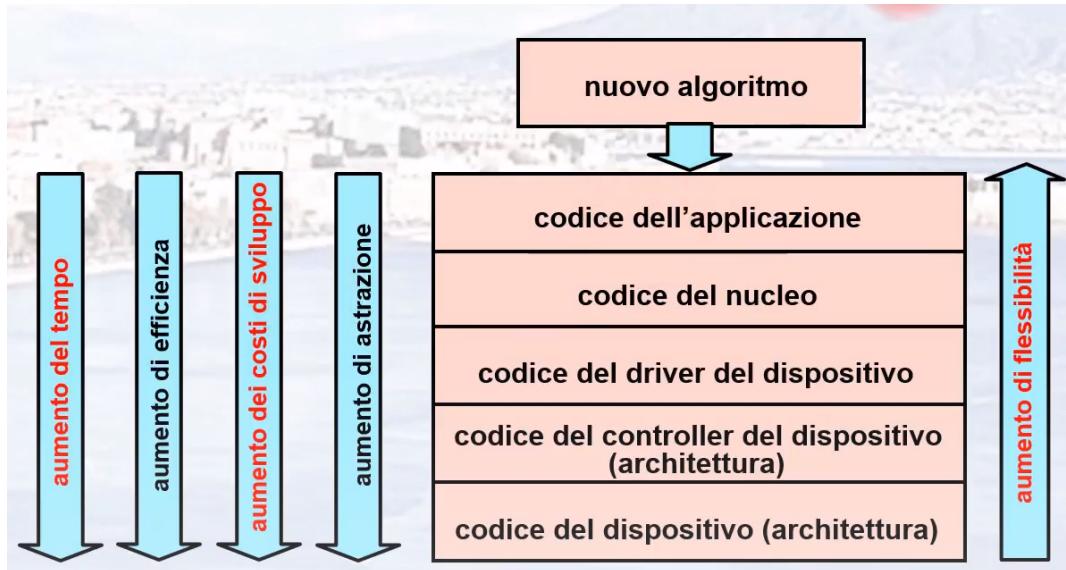


23.21 Migliorare le prestazioni

Per far ciò si possono applicare diversi principi:

- Ridurre il numero di context-switch;
- Ridurre la frequenza degli interrupt;
- Uso di controllori DMA intelligenti;
- Equilibrare le prestazioni della CPU, del sottosistema per la gestione della memoria, del bus e dell' I/O

23.22 Successione delle funzionalità dei servizi di I/O



24 Memoria Secondaria e terziaria

24.1 Struttura dei dischi

I dischi sono considerati un grande vettore monodimensionale di blocchi logici, dove un blocco logico è la minima unità di trasferimento.

Il vettore monodimensionale di blocchi logici corrisponde in modo sequenziale ai settori del disco:

- Il settore 0 è il primo settore della prima traccia sul cilindro più esterno;
- La corrispondenza prosegue ordinatamente lungo la prima traccia, quindi lungo le rimanenti tracce del primo cilindro, e così via, dall'esterno verso l'interno.

24.2 HDD vs SSD

	Costo	Capacità	Delicatezza	Velocità
HDD	Basso	Alta	Alta	Bassa
SSD	Alto	Bassa	Bassa	Alta

(Tabella non proprio vera ma dettagli)

24.3 Struttura HDD

L'insieme delle tracce dei dischi che compongono lo stesso HDD e che hanno lo stesso raggio formano quello che viene detto **cilindro**

24.4 Floppy-disk, CD-ROM, Dischi rimovibili, Dischi WORM, Nastri

(Perchè giustamente ci sta anche gente che li usa ancora)

CD-ROM → disco su cui i dati sono scritti su una lunga spirale che si propaga dall'interno verso l'esterno. Nel funzionamento questo ruota ad una velocità lineare costante.

Floppy-disk → costituito da una sostanza elettromagnetica distribuita in modo uniforme su tutta la superficie del disco. Nel funzionamento il disco ruota ad una velocità angolare costante. Le tracce sono aree circolari, numerate da 0 (traccia esterna) a N (traccia interna)

Dischi rimovibili → registrano i dati su un disco rigido ricorperto da materiale magnetico, non sfruttano il magnetismo ma materiali speciali che la luce laser può alterare in modo da creare punti relativamente chiari o scuri.

Dischi WORM (Write Once, Read Many) → dischi che possono essere scritti una sola volta e per questo durevoli e affidabili

Nastro → meno costoso di un disco, ma presenta un'accesso più lento. Vengono utilizzati nel caso di grosse quantità di dati, che non richiedano rapidi accessi diretti.

24.5 Hard disk Z-CAV

Considerando la forma dei cilindri, si è iniziato a progettare una nuova strategia nota con il nome di **Zone Bit Recording** (ZBR) il cui obiettivo è quello di memorizzare più settori nelle tracce esterne.

Poichè la tipologia usuale degli Hard Disk è di tipo CAV la strategia ZBR dovrà opportunamente gestire la velocità variabile della lettura e della scrittura (la velocità aumenta ovviamente nelle tracce esterne).

24.6 Partizionamento di un Hard Disk

Tra i vantaggi del partizionamento, con lo scopo di un migliore gestione del disco abbiamo:

- Necessità di separare al massimo aree di dati logicamente molti differenti tra loro;
- Evitare Seek-Time troppo elevati.

24.7 Scheduling dell'Hard Disk (tradizionale)

Per accedere in diversi punti del disco il braccetto impiega un tempo proporzionale allo spostamento compiuto; la spezzata che congiunge tutti i punti determina la lunghezza totale.

Gli obiettivi principale della gestione del disco è quella di minimizzare i tempi di accesso al dispositivo stesso.

Se occorre quindi accedere a dati che sono memorizzati in punti diversi, occorre allora stabilire una metodologia ottimale che determini in quale ordine sia migliore per accedere ai dati.

L'ordine di accesso ai dati ne determina il tempo totale di accesso.

24.8 Metodologie di scheduling

FCFS (First Come First Served) → le richieste vengono soddisfatte nell'ordine in cui esse sono pervenute (semplice da realizzare ma non ottimizza la distanza da percorrere).

SSTF (Shortest Seek Time First) → si seleziona la richiesta più vicina rispetto all'attuale posizione della testina (si può incorrere in situazioni di starvation).

SCAN → il braccetto parte da un'estremo e si sposta nella sola direzione possibile, servendo le richieste mentre attraversa i cilindri, fino a raggiungere l'altro estremo; a questo punto inverte la marcia e la procedura continua.

24.9 Scheduling per scansione circolare (C-SCAN)

Variante dello scheduling SCAN concepita con lo scopo di garantire un tempo di attesa meno variabile.

A differenza del classico SCAN, una volta raggiunta l'altro estremo, il braccetto ritorna immediatamente all'inizio del disco stesso, senza servire richieste durante il viaggio di ritorno.

L'algoritmo tratta il disco come una lista circolare, quindi come se il primo e l'ultimo cilindro fossero adiacenti.

24.10 Scheduling C-LOCK

Versione di C-SCAN in cui il braccio si sposta solo finchè ci sono altre richieste da servire in ciascuna direzione, dopo di che cambia immediatamente direzione, senza giungere all'estremo del disco.

24.11 Gestione dell'unità a disco

Prima che possa memorizzare dati, il disco deve essere suddiviso in settori che possano essere letti/scritti dal controllore.

Il SO deve registrare quindi le proprie strutture dati all'interno del disco. Ciò avviene in 2 passi:

1. Partizionare il disco;

- Creare il file system (ext4, btrfs, e se sei banale NTFS).

Il blocco di avviamento (*boot block*) inizializza il sistema. Il *bootstrap loader* è memorizzato nella ROM.

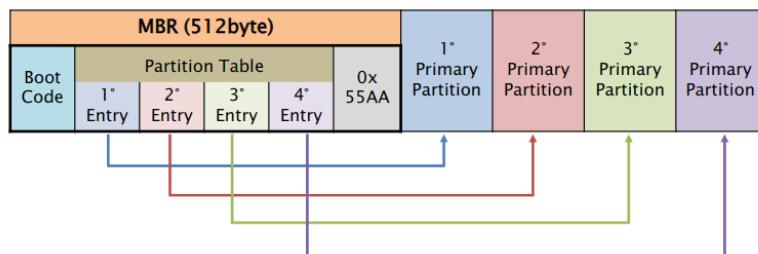
La formattazione fisica mette anche da parte dei settori di riserva non visibili al SO → si può istruire il controller affinchè sostituisca da un punto di vista logico un settore difettoso con uno dei settori di riserva non utilizzati. Questa strategia è nota con il nome di *sector sparing* (accantonamento di settori).

24.12 DOS Partition

Sistema di partizione più comune.

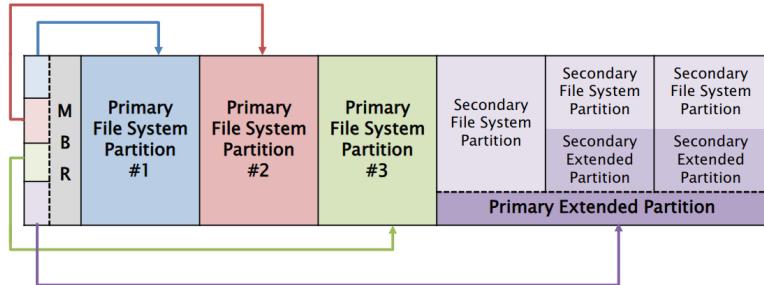
MBR (Master Boot Record) → presenta obbligatoriamente un primo settore di 512 byte:

- Boot Code;
- Partition Table (max 4)
 - Starting CHS address;
 - Ending CHS address;
 - Starting LBA address;
 - Numbers of sector in partition;
 - Type of partition;
 - Flags;
- Signature → 0x55AA



- **Primary File System Partition** → partizione primaria che contiene un file system
- **Primary Extended Partition** → partizione primaria che contiene altre partizioni;
 - Tabella di partizione;
 - **Secondary File System Partition** → partizione secondaria che contiene un file system (partizione logica);
 - **Secondary Extended Partition**

- * Tabella di partizione;
 - * Secondary File System Partition;
 - * Secondary Extended Partition
- . . .



24.12.1 Boot Code

Situato nei primi 466 byte del primo settore (MBR):

- Microsoft Boot Code → processa la tabella di partizione e ricerca ed identifica quella c.d. *bootable*, tramite FLAG;
- Possibile incapsulamento di virus;

Il settore MBR viene allocato all'inizio del *Disk Volume* e di ogni *Extended Partition*:

- EBR (Extended Boot Record) (512 byte)
 - La parte riservata al *Boot Code* è inutilizzata;
 - La parte riservata alle altre 2 entry nella *Partition Table* è vuota.

24.13 Gestione dell'area di swap

Molto comune sui sistemi Gnu/Linux (meno frequente su windows).

In questo modo la memoria virtuale usa lo spazio dei dischi come estensione della memoria centrale.

L'area di swap può essere ricavata all'interno del normale file system o, in una partizione separata del disco.

Molto spesso, in base alle performance del disco e del PC (tenendo conto anche dell'uso che l'utente ne vorrà fare) si sceglie tra:

- Swap Partition;
- Swap File;
- Zram.

24.14 Configurazione RAID (Redundant Array of Independent/inexpensive Disk)

Combinazione di più dischi con lo scopo di migliorare l'affidabilità del sistema, le prestazioni, o entrambe.

Può essere realizzato sia in maniera HW che SW.

RAID Level 0 → permette la distribuzione automatica dei dati su più dischi ma visto che i dati non sono replicati non garantisce la tolleranza ai guasti in caso di rottura di un disco.

RAID Level 1 → I dati vengono replicati e nel caso di rottura di un disco si passa automaticamente ad un altro disco senza perdita di dati.

RAID Level 3 → simile al RAID 0, ma dedica un hard disk al recupero automatico mediante il controllo di parità

RAID Level 5 → Usa *data striping* completo e un disco per la correzione d'errore (si usa in caso di una forte garanzia di elevata protezione dei dati).

24.15 Connessione dei dischi

- Tramite le porte di I/O;
- Per mezzo di un file system distribuito.

24.16 Compiti del sistema operativo

- Gestione dei dispositivi fisici, di cui il SO realizza 2 astrazioni
 - Dispositivo a basso livello → un semplice vettore di blocchi di dati;
 - File System → Il SO accoda e organizza le richieste provenienti da diverse applicazioni;
- Presentazione di una macchina virtuale alle applicazioni.