

Cambio di Definizione in base al contesto storico (quindi dipende dalle potenzialità della macchina).

- **Insieme di Programmi** che fungono da intermediario tra utente e hardware.

- **Gestione delle risorse.** (riproduzione di un filmato cosa che 30 anni fa non era possibile)

- **Gestione delle procedure della macchina.**

Obiettivo:

- Eseguire programmi di applicazione (ovvero lanciati dall'utente) e rendere il sistema conveniente da utilizzare ed efficace e versatile. (può essere utilizzato per filmati, audio o qualsiasi altra necessità dell'utente).

Fisicamente il nostro sistema di calcolo è strutturato in modo tale:

1. **Dispositivi fisici(hardware):** forniscono risorse di calcolo fondamentali (CPU, memoria, I/O, schede, tastiere). strato basso
2. **SO:** controlla e coordina l'uso dei dispositivi da parte dei programmi d'applicazione.
3. **Programmi d'applicazione:** definiscono il modo in cui sono utilizzate le risorse del sistema per risolvere i problemi degli utenti (compilatori, sistemi di basi di dati, videogiochi, applicativi per il business). qualsiasi programma che l'utente può usare.
4. **Utenti** (persone, macchine, altri calcolatori). l'utente può essere anche un altro sistema o programma che fa richiesta al nostro SO.

Esempio word fa delle chiamate al SO il quale deciderà quali risorse assegnare per ad esempio stampare, quindi quale stampante usare. Sistema Serbo-freno che impedisce che la macchina possa slittare, poiché premendo il pedale invece che frenare direttamente l'utente da un input di frenata che poi il serbo-freno gestisce e impedisce la frenata diretta ma creando una graduale frenata che la quale non perde aderenza al suolo. Analogamente il SO è un serbo-mecanismo che da all'utente la sensazione che le operazioni li compia lui stesso, ma in realtà è il SO che dialoga con la macchina interpretando il volere dell'utente.

SO = **Assegnatore di Risorse:** gestisce e assegna risorse nel modo migliore.

SO = **Programma di controllo** che conteggia l'esecuzione e impedisce che programmi prendano il controllo in maniera aggressiva sugli altri, bloccando la macchina o altri processi abusando di tutta la risorsa.

SO= **nucleo/Kernel** = programma associato alla macchina che funziona sempre mentre tutto il resto viene visto come programmi di applicazione.

Sistemi MainFrame:

Sono computer ad alte prestazioni con grande quantità di memoria e processori che elaborano tantissime operazioni con requisiti simili. (Database Commerciali, Server delle Transazione etc): riducono il tempo di elaborazione, tra un'operazione e l'altra e quindi maggiore velocità del passaggio da un'operazione a quella successiva.

Un pacchetto di programmi capace di dare una visione d'insieme e di controllo dell'intero sistema è detto monitor residente.

Sistema a lotti:

Configurazione della memoria in cui si hanno due blocchi di memoria distinti in cui in uno è dedicato al SO, mentre l'altra a tutto il resto (notiamo che l'SO è nella parte migliore della memoria, parte più esterna del disco o nelle celle con indirizzi con numerazione più piccola (celle privilegiate), nelle parti basse della memoria quindi accessibili più velocemente. Questo perché la macchina impiega tempo di elaborazione per gestire gli indirizzi quindi velocizzando questo tempo miglioriamo le prestazioni dell'intera macchina dato che l'SO gestisce tutto il resto.

(la parte esterna di un disco contiene una quantità maggiore di dati in un singolo ciclo, quindi a velocità maggiore.)

Sistema Multi-programmati:

Sono sistemi capaci di caricare più programmi(lavori) nello stesso momento in memoria che sono quindi mantenuti in un unico gruppo che sono già presenti sul disco e poi caricati nella memoria centrale.

Il sistema deve suddividere la memoria in base ai processi caricati e stabilirne una priorità secondo uno specifico algoritmo che poi definisce lo scheduling della CPU: Il sistema deve stabilire quale coda di esecuzione rispettare e quindi stabilire il primo tra i processi disponibili.

Sistemi a partizione del tempo d'elaborazione:

la CPU viene ripartita tra i vari lavori tenuti sul disco, seguendo un metodo del TIME SHARING ripartizione del tempo:

la CPU viene assegnata a un processo/utente per volta, ma a una velocità tale che ogni processo sembra essere eseguito in contemporanea, apparente esclusività. Quindi viene stabilito un tempo unico di elaborazione per ogni processo, esempio 5 processi 1 unità di tempo, una unità al primo poi una un'unità al secondo e così via. Quindi ogni utente ha una comunicazione diretta con il sistema e ciascun utente dispone di almeno un programma nella memoria.

Sistemi da scrivania:

personal computer, dispositivi I/O, Stampanti, comodi per l'utente e possono funzionare con diversi SO windows MacOS Unix, Linux)

Sistemi Paralleli:

Sistemi con più CPU in stretta Collaborazione, dove diversi processi vengono eseguiti contemporaneamente ma sono indipendenti tra di loro e condividono la stessa memoria e temporizzatori di clock. Memoria condivisa. Quindi sistemi fortemente connessi, tolleranza ai guasti, l'errore di un sistema non blocca gli altri sistemi (Fault Tolerant).

Si dividono in SMP multielaborazione simmetrica AMP multielaborazione Asimmetrica:

nella simmetrica ogni CPU (unità di elaborazione) ha una copia del SO e quindi ogni sistema può eseguire più processi contemporaneamente senza subire un calo delle prestazioni.

Nella asimmetrica a ogni CPU viene assegnato un compito specifico quindi in realtà ogni CPU dipende dagli altri.

Sistemi Distribuiti:

Sono basati su reti che sfruttano la capacità di comunicare per risolvere problemi di calcolo, ogni CPU ha la propria memoria locale (cache dove vengono caricati le informazioni sui dati caricati dal disco alla centrale), vantaggi condivisione di risorse e velocità. Richiedono LAN e WAN E MAN Local, Wide Area Network Metropolitan. Possono essere sistemi client server, asimmetrici alcune macchine che forniscono un servizio e altre macchine che sono clienti; invece, peer to peer coinvolgono punti specifici della rete.

LAN - Local Area Network (rete locale), che collega nodi relativamente vicini, nell'ambito tipicamente di uno stesso edificio

MAN - Metropolitan Area Network (rete metropolitana), con collegamenti a distanze massime di decine di chilometri

WAN - Wide Area Network (rete geografica), che collega nodi a distanza qualsiasi, anche planetaria.

Sistemi Cluster (batterie di Sistemi):

Più CPU unite per svolgere attività comuni, Batterie Asimmetriche: un calcolatore rimane in fase di attesa Attiva mentre l'altro esegue le applicazioni, simmetriche più CPU eseguono applicazione e si controllano reciprocamente.

Sistemi di elaborazione in tempo reale:

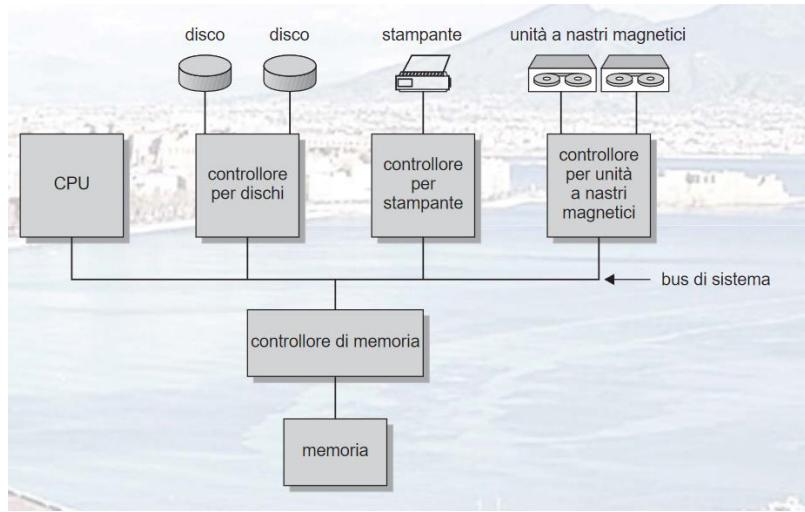
applicazioni che devono essere eseguite in tempo ristretto Hard o Soft real time esempio tempo reale forte (tempo di atterraggio di un aereo richiede il minimo tempo di elaborazione che non possono essere interrotte poiché delicatissime).

Soft la funzionalità viene garantita seppur con un leggerissimo ritardo tollerabile.

Sistemi Palmari:

per la dimensione hanno performance ridotte.

Elaborazione basata su web non avviene sulla macchina ma avviene su un server.



Bus di sistema è un canale di comunicazione, quindi le varie componenti comunicano tra di loro migrando i dati attraverso i bus.

Controller è un'interfaccia che ogni dispositivo o unità di memoria (disco) ha diversa velocità e quindi ognuno necessita di un controllore che compensa le differenze di velocità e stabiliscono le regole di comportamento dei dispositivi.

Questi dispositivi cooperano in maniera concorrente, gareggiano per divedersi le risorse, se più processi hanno bisogno della stessa risorsa iniziano a contendersela, trovare un insieme di formalismi che stabilisce l'ordine di utilizzo della risorsa.

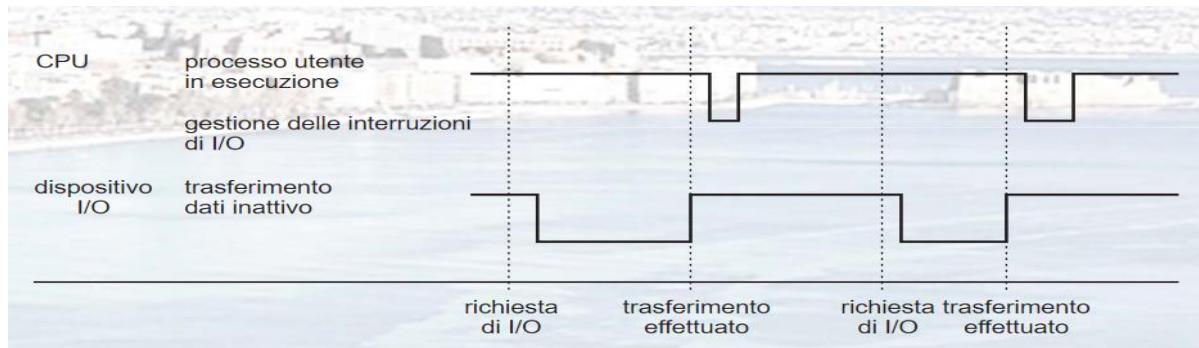
Ogni controller si occupa di un unico dispositivo fisico, ed ha un buffer locale in cui il dispositivo memorizza i dati e li rilascia alla velocità desiderata. Esempio la stampante ha uno "scaffale" su cui memorizzare i dati ricevuti immediatamente dal pc, e rilasciarli man mano che la stampante stampa alla velocità che desidera.

La cpu sposta i dati tra la memoria centrale e i vari buffer.

La I/O avviene dal dispositivo al buffer locale, e quando un controller prende in carica questi dati con segnali di inizio e fine dette interrupt per avvertire la cpu di un evento. L'architettura deve sapere gestire le interruzioni e anche salvare l'indirizzo dell'istruzione interrotta. Un segnale di eccezione detto trap è causato da un programma in esecuzione e richiede un intervento del sistema per gestire l'eccezione. Un moderno SO è interrupt driven attraverso un serie di interruzione si stabilisce la procedura di elaborazione.

La gestione dell'interruzione:

1. Memorizza gli indirizzi di ritorno nello stack di sistema. Se interrompiamo il processo dobbiamo ricordarci in quale punto dell'algoritmo ricominciare a elaborare il processo successivo.
 2. Determinare il tipo di interruzione: polling, interrogazione ciclica dei dispositivi (si perde molto tempo a interrogare e si perde molta energia per fare il ciclo) è conveniente in alte situazioni di traffico, alta congestione. Con il basso traffico preferisco il meccanismo di interruzione (esempio ragazzi che alzano la mano) vectored interrupt system, ogni interruzione viene gestita tramite l'implementazione di segmenti di codice prestabiliti.
- ogni interruzione ha un tempo di elaborazione dei dati che si traduce in un ritardo del trasferimento dei dati da un dispositivo alla cpu o viceversa.



Struttura I/O:

quando c'è una richiesta di trasferimento dati avviene attraverso una wait è una system call che agisce sul sistema operativo che permette il cambio di uno stato di sistema elencato nella tabella apposita degli stati del dispositivo.

Lo scambio di informazione può avvenire in modo sincrono e asincrono. Sincrono richiede che entrambi i dispositivi che comunicano devono essere entrambi disponibili per permettere il trasferimento dei dati esempio (chiamata telefono), asincrono non richiede questa partecipazione esempio (messaggio WhatsApp o e-mail).

Nel sincrono il processo richiedente deve fare un trasferimento dati, il processo si interrompe e il nucleo prende in carico questa chiamata e chiama il dispositivo tramite il driver che è un software che regola la predisposizione dell'I/O e gestisce le interruzioni, il driver con il gestore interrompe il dispositivo per poi trasferire i dati quando il processo è disponibile.

Nell'asincrono il processo non viene interrotto obbligatoriamente e i dati possono continuare ad essere trasferiti al dispositivo e soltanto alla fine quando sono resi disponibili vengono restituiti al processo richiedente. Esempio cpu e un disco non devono partecipare al trasferimento dei dati contemporaneamente; quindi, i processi possono rimanere in attesa o fare altro.

La tabella di stato dei dispositivi: permette di interrogare la disponibilità di un certo dispositivo.

Accesso diretto alla memoria:

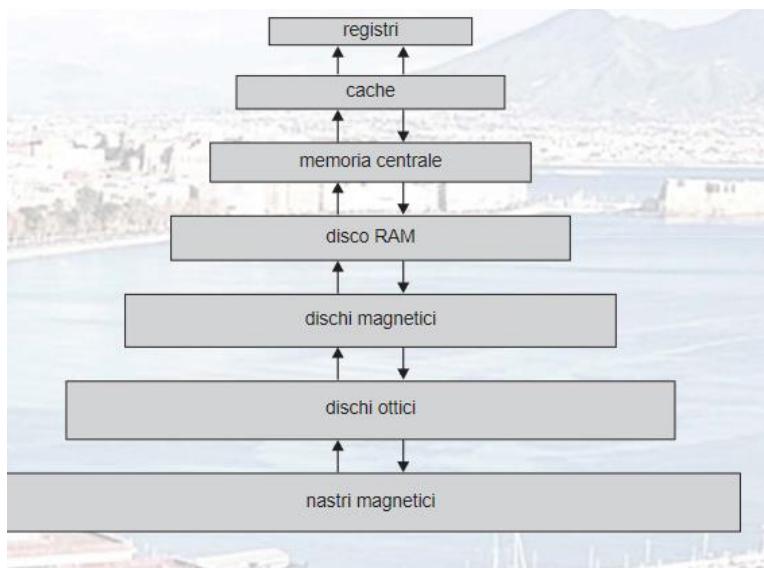
non è importante coinvolgere la cpu per lo scambio dei dati se no spreca tanto tempo e risorsa, poiché dei dati dopo essere letti possono essere elaborati nella memoria centrale e non più nella cpu. Tecnica DMA direct memory access quindi un dispositivo per non pesare sulla CPU legge un dato direttamente dalla memoria centrale.

La memoria centrale RAM è direttamente accessibile dalla cpu.

La memoria secondaria può conservare le informazioni in maniera permanente. Nel CD-ROM i dati sono scritti in una spirale mentre nell' hard disk i dati sono scritti in dei cerchi concentrici composti da una sostanza elettromagnetica, i dati vengono letti da un sistema di bracci posti su un attuatore detto anche pettine. Questo pettine si immerge nel disco composta da più piatti, così può accedere a più punti del disco, i piatti girano a velocità costante, 10000 RPM round per minute.

Si può creare una gerarchia di memoria in base alla velocità o costo o volatilità delle varie memorie:

Cache: copia temporanea di informazione in unità più veloce, la RAM si può considerare una cache per la memoria secondaria.



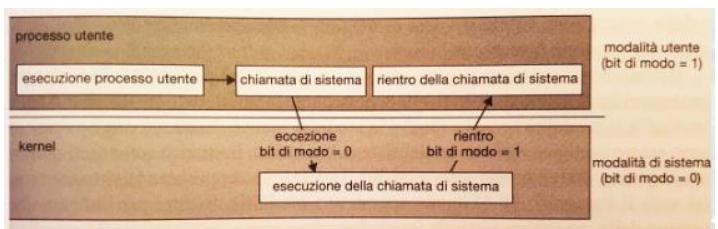
i registri sono all'interno della cpu estremamente veloci, cache della cpu permette di trattare i dati ad altissima velocità, i dati dalla RAM passano alla cache per calcoli veloci. Disco RAM è un disco che si può configurare come se fosse un unità Hardisk. Dischi magneti hard disk, dischi ottici cd rom. **La cache:** richiede una politica della gestione della cache poiché lo stesso dato viene copiato su più memorie e quindi si può creare un problema di coerenza della cache, e se il dato viene modificato in uno di questi per evitare errori va corretta anche per le sue copie. se un dato che si trova nel hard disk (disco magnetico) viene copiato in memoria centrale (RAM) e dunque diventa un processo (programma attivo) poi per essere eseguiti viene copiato dalla cache del processore è poi

l'elaborazione effettiva dei dati avviene nei registri hardware.

Architetture di protezione: la cpu, le I/O e la memoria per essere protette, la cpu tramite il SO può lavorare in dual mode:

abbiamo un segnale che ci informa se i dati trattati sono dati del sistema operativo (monitor mode/kernel) o che sono dell'utente. Questo segnale è il bit di modo, funzioni di sistema 0 o si sta trattando istruzioni utente 1.

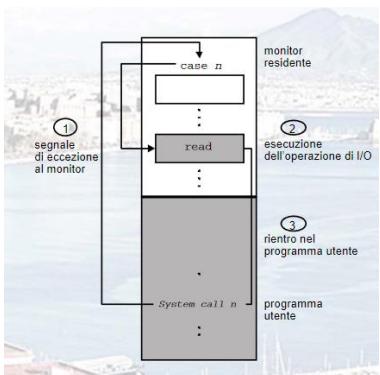
Esempio un processo utente 1 attraverso una chiamata di sistema 1 commuta il bit di modo a 0 il kernel esegue la chiamata di sistema e poi ritorno il bit di modo a 1 rientrando alla chiamata utente e ritorna al processo utente.



Protezione CPU

Passo alle varie modalità commutando il bit di modo. Le istruzioni della cpu sono tutte privilegiate non possono essere interrotte dall'utente. In questo modo posso proteggere le I/O, tutte le istruzioni dell'I/O sono privilegiate, per evitare che l'utente possa causare danni.

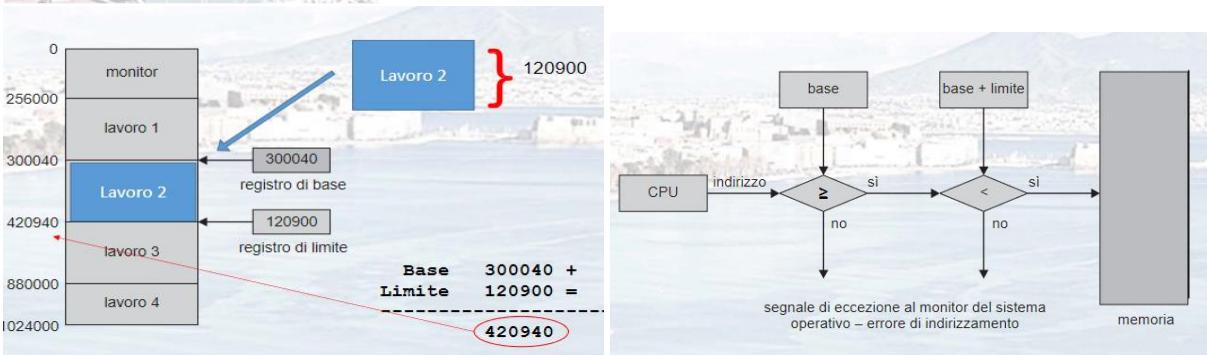
Il programma dell'utente esegue una system call mediante un segnale di eccezione al monitor, l'esecuzione del programma viene sospesa e il controllo passa al monitor residente, in base all'eccezione il monitor decide quale procedura applicare per questa system call, poi viene eseguita un'interruzione e si ritorna al programma utente.



Protezione I/O

La memoria si protegge utilizzando degli indici che mi indicano l'inizio di una procedura e denotando la sua lunghezza.

Usando un registro di base e un registro limite, la prima in quale locazione di memoria iniziare il job, la seconda dove finisce. Una successione di uno o più processi lanciati da un singolo comando viene detto job. Occorre effettuare una protezione degli indirizzi (virtuali cioè non corrispondono all'effettiva posizione fisica della memoria, dipende Hardware)



Indirizzi della RAM sono fisici, indirizzi della cpu sono virtuali. Se l'indirizzo del dato rispetta l'intervallo si può accedere alla RAM e gestire il dato.

Protezione Hardware avviene nel modo di sistema, quindi è il SO che governa i conflitti tra processi, un utente non può accedere all'hardware.

Protezione della cpu tramite Timer temporizzatore, si stabilisce un tempo massimo per cui un processo può sfruttare una risorsa in questo caso la cpu e oltre quel limite il processo viene interrotto. Esempio meccanismo Time Sharing.

Le reti locali possono essere congiunte tramite un gateway, struttura di passaggio a diverse sottoreti.

Componenti del sistema

Capitolo 3

IL SISTEMA OPERATIVO SI OCCUPA DI DIVERSE GESTIONI

Gestione processi: processo è un programma in esecuzione, un programma è solo un insieme di istruzioni.

all'interno di un programma che carica in memoria possono corrispondere più processi

Il SO è responsabile della creazione e cancellazione di processi, sia utente che sistema, oppure sospensione e ripristino, sincronizzazione dei processi e comunicazione tra processi.

Gestione della memoria centrale:

è un area/vettore di dimensione grande composta da celle con ognuna il proprio indirizzo, la RAM essendo una memoria volatile in mancanza di corrente perde i dati salvati al suo interno, per questo motivo occorre memorizzare alcune informazioni sull' hard disk. Il SO tiene traccia di quali parti della memoria sono usate e da che cosa, decide quali processi caricare in memoria se c'è spazio, assegnare e revocare lo spazio di memoria secondo le necessità.

Gestione dei file:

i file possono essere sia dati che programmi che in forma sorgente o in forma compilata(eseguibile). Il nome non determina il tipo di file, è solo un'indicazione per indicare quale programma eseguire per fare uso del file.

Quindi gestisce sia la creazione che la cancellazione di file e directory, le directory ci permettono di strutturare i file in cartelle che costituiscono un albero la cui prima cartella è detta root. Può creare delle copie di backup che sono fotografie computazionali per poterci ricordare lo status informativo di una macchina.

Gestione del sistema di I/O e dunque del sistema di buffer-caching (elabora i dati secondo la velocità desiderata dal dispositivo) si interfaccia con i dispositivi tramite i driver (software che costituiscono l'insieme di regole di comunicazione con un dispositivo) i produttori per questo motivo oltre all'hardware devono fornire anche il driver per comunicare con il dispositivo, che possono essere già presenti sul firmware (programma integrato direttamente alla macchina).

Gestione della memoria secondaria: il SO si occupa del trasferimento di dati da memoria RAM al disco e viceversa, Hardisk o solid state disk, attraverso lo scheduling (pianificazione di entrata e uscita) del disco e gestione dello spazio libero e di assegnazione.

Reti: sistema distribuito è un insieme di unità di elaborazione che non condividono memoria, dispositivi periferici o clock.

Sono collegate tra loro tramite una rete di comunicazione, comunicano tra di loro sottoforma di regole (**protocolli**) di comunicazione, ogni dispositivo dispone di una specifica serie di protocolli ad hoc, le reti permettono l'accelerazione di calcolo, aumento disponibilità delle reti e un incremento dell'affidabilità

sistemi di protezione: sistema di meccanismi che definiscono l'accesso delle risorse e impedisce agli utenti di abusarne proteggendo la cpu, memoria, I/O. per garantire che una risorsa non venga monopolizzata da un utente/programma/sistema. Usi autorizzati e no. Definiscono gli specifici controlli.

Interprete di comandi: interfaccia che dispone di un insieme di comandi che permettono di comunicare con la macchina. Esempio cmd, powershell. Quindi legge e interpreta le istruzioni, sistemi unix bash, tcsh, sh.

Rilevamento errori, Assegnazione delle risorse e la contabilizzazione delle risorse: contabilizzazione è addizionale, ci permette di contare gli usi di determinate risorse e quanto vengono impiegate.

Le system call: Sono chiamate che permettono di colloquiare con le parti basse del sistema, es. accesso alla memoria, sovrascrittura di un disco, quindi abbiamo strati elaborazioni,

passaggio dei parametri: Ogni volta che si effettua una chiamata del sistema non ha un valore assoluto, dipende da chi l'ha chiamato quindi ci sono dei parametri che dipendono dal chiamante e dal tipo di operazione che si intende fare.

Quando si fa la system call si passa l'indirizzo del blocco dove risiedono i dati che vogliamo trattare, a un registro.

Le system call si distinguono per processi, file, dispositivi e informazione. Categorie*

MS-DOS Microsoft Disk Operating System primo Sistema a larga diffusione del SO, girava su dischetto, command.com era l'interprete si lanciava windows con "win". Il processo veniva caricato nello spazio di memoria libero.

UNIX l'interprete può trovarsi in qualsiasi parte della memoria, quindi la memoria non è tutta addensata in unica sezione contigua.

Modelli di comunicazione: I processi per comunicare hanno bisogno di regole per gestire lo scambio di informazioni, due modi:

scambio di messaggi: Processo A deposita un messaggio al nucleo, e il nucleo lo rispedisce al processo B quando ne fa richiesta. Vantaggio di non usare altra memoria, va bene per dati piccoli

memoria condivisa: più utile per dati di grandi dimensioni, quindi una cella di memoria può essere condivisa da più processi.

Programmi di sistema: funzionalità che permettono di creare un ambiente di lavoro, gestione file, informazione di stato, modifica dei file, caricamento ed esecuzione programmi, programmi applicazione.

L'interfaccia col sistema operativo è definita dai programmi di sistema. MS DOS (massima funzionalità nel minimo spazio).

Non era un sistema modulare, e spesso molti programmatore scrivevano righe di codice in Assembler per risparmiare spazio.

DOS riusciva a comunicare direttamente col BIOS con gli strati bassi della macchina.

UNIX a una struttura più solida, controllano file system, cpu e memoria etc.

Metodo stratificato si suddivide il SO, lo strato più basso (0) è lo strato fisico quello più alto è l'interfaccia utente. livello i fa system call/usa funzionalità a livelli inferiori, e fornisce servizi all'utente a livelli superiori. Quindi ogni strato ha delle funzionalità. **OS/2** altro sistema stratificato.

Architettura microkernel:

Si preferisce avere più nuclei piccoli facilmente gestibili e veloci piuttosto che un unico mega nucleo; quindi, non è importante caricare in memoria tutte le funzionalità, ma solo quelle necessarie. Es. se devo stampare non è una funzionalità obbligatoria e quindi va caricata solo quando l'utente ne necessita, non appena avviamo la macchina.

Macchine virtuali:

facciamo un'astrazione della risorsa, esempio estendiamo la memoria fisica in una memoria virtuale, dischi virtuali etc.

posso potenzialmente far girare una macchina virtuale all'interno di un'altra macchina, ad esempio all'interno di windows posso fare avviare un altro windows/mac etc. vantaggio posso far girare più macchine sullo stesso hardware.

Molti provider che vendono macchine virtuali, macchine in cloud nella rete, contro sono un po' più lente.

Quindi il SO crea un'illusione che un processo disponga della propria CPU con la propria memoria virtuale. Tutto avviene grazie al time sharing. Macchina virtuale buona per testare software o SO perché ogni macchina è isolata dalle altre.

Macchina virtuale Java:

servono a far eseguire delle piccole applicazioni sul web, indipendentemente dal tipo di SO o di macchina. Quindi per ottenere questa versatilità creo uno strato interfaccia JVM che raccoglie queste applicazioni in byte code e vanno in esecuzione sulla macchina che si trova al di sotto della JVM.

Scopi della progettazione: per l'utente sicuramente la comodità la facilità la versatilità, dal punto di vista del sistema deve essere facile la progettazione, realizzazione e manutenzione. Deve garantire meno errori possibili.

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

DOS scritto in Assembler, adesso in C o C++ vantaggio la portabilità (porting) non dipendono dalla macchina, si possono ottenere eseguibili in qualunque macchina.

SYSGEN configura e genera il sistema per ciascuna situazione. Booting: avviamento di un calcolatore attraverso il caricamento del nucleo, bootstrap program, piccolo segmento di codice memorizzato in una Rom che individua il nucleo e lo carica nella memoria e avvia l'esecuzione.

Processi

Capitolo 4

Processo è l'unità di elaborazione più semplice e non va confuso con il programma, un programma può corrispondere a più processi. Possiamo avere due tipi di elaborazione di processi, sistemi a lotti batch system che esegue lavori job, oppure time sharing. Nel batch tutti i processi vengono passati all'elaboratore tutti insieme e non c'è possibilità di interagire con questi processi. Nella partizione del tempo comporta una stessa slice di tempo concesso singolarmente a tutti i processi caricati, in maniera ciclica, avviene in maniera così rapida che il processo ha l'idea di contemporaneità con gli altri processi.

In situazioni di alto traffico il batch è più performante mentre il time sharing in situazione di basso traffico.

Un processo è un programma in esecuzione.

Comprende: un program counter (variabile che contiene l'indirizzo dell'ultima istruzione di un programma. Così se viene sospeso sappiamo da dove riprendere), uno stack (è una struttura dati con accesso in testa) e una sezione dati.

STATI DEL PROCESSO: 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 a un'unità di elaborazione. **TERMINATO:** il processo ha terminato l'esecuzione e deve rilasciare tutte le risorse che stava utilizzando.



il fatto che un programma è pronto non garantisce l'esecuzione, poiché deve essere scelto in base ai requisiti dell'algoritmo che gestisce lo scheduling dei processi.

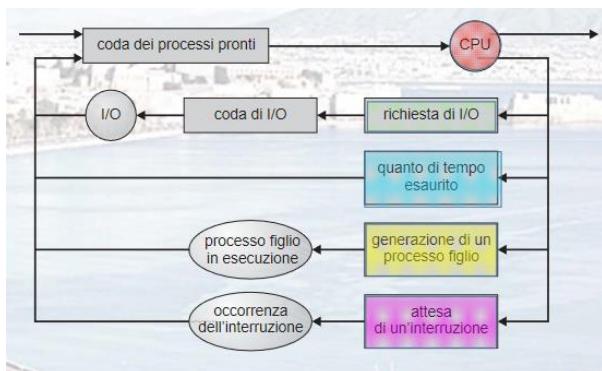
Lo stato di attesa indica che un processo ha bisogno di dati, risorse, non può continuare l'esecuzione. Quando riceve i dati torno allo stato di pronto in attesa di essere scelto dall'elaboratore. Se il processo comporta una interruzione momentanea di un processo dovuto alla presenza di un altro processo che necessita di essere eseguito, il processo iniziale torna allo stato di pronto in attesa di scelta e quindi eseguito (non è in attesa, aspetta il suo turno con l'algoritmo di scheduling).

Il processo termine quando esegue l'ultima istruzione e rilascia le risorse, dati, dischi memoria, canali di comunicazione...

Quando nasce un processo c'è bisogno di una serie di informazione contenute nel PCB: **Process Control Block**.

Che possiede lo stato e il numero del processo, il program counter, registri della cpu chiamati in causa, scheduling della cpu, informazioni su gestione della memoria informazione di contabilizzazione delle risorse, informazioni stato I/O;

La CPU può commutare tra processi ovvero durante una chiamata di sistema, può esserci una nested interrupt, che può richiamare un altro processo innestato. E si crea una struttura nidificata. Abbiamo due processi, il processo 0 in esecuzione riceve un'interruzione, l'SO salva lo stato in PCB 0, poi ripristina lo stato di PCB 1 che poi va in esecuzione, quando termina o viene sospeso, l'SO salva lo stato in PCB 1 e ripristina lo stato PCB 0. I PCB si trovano nel kernel del SO. Il tuning si fa la messa appunto del sistema per capire quanti processi può gestire contemporaneamente. I processi vengono elaborati seguendo una linked list, si creano delle code di processi, di processi pronti, coda del dispositivo I/O. si può passare da una coda all'altra in base alle necessità.



Esistono 3 tipi di scheduler a medio e **lungo termine**, a breve termine della CPU (oggetto esame).

Nel lungo termine preleva i processi che devono essere caricati nella coda dei processi pronti. **AMMESSO**

Possono passare minuti, poiché viene invocato molto frequentemente e gestisce il grado di multiprogrammazione (quanti processi può far passare alla coda dei processi pronti).

Nel breve termine fa la selezione dei lavori pronti per essere eseguiti e assegna la cpu a ognuno di essi. **DISPATCH**

MODULO DISPATCH è un altro sistema operativo che fa tutti i settings per fare eseguire un processo pronto, questo processo ha delle latenze di dispatch (tempo di elaborazione per sospendere un processo precedente e iniziare un nuovo) +basso meglio. Procedure molto critiche poiché viene chiamata la cpu che deve compiere scelte in millisecondi

Medio termine corrisponde all'interruzione, ha il compito di rimuovere i processi dalla memoria quindi dalla contesa della CPU, non è critico. Se un processo viene rimosso dalla cpu, o ritorna nella coda dei processi pronti o nella coda d'attesa I/O oppure termina. Può alleggerire il compito dello scheduler a breve termine, riducendo i processi da eseguire.

INTERRUZIONE

I processi possono essere 1 con prevalenza di I/O oppure 2 con prevalenza di elaborazione: 1 impiega la maggior parte del tempo nell'esecuzione di operazione di I/O. 2 richiede poche operazioni di I/O e impiega il resto del tempo nelle elaborazioni della CPU.

CAMBIO DI CONTESTO passando dalla computazione di un processo a un altro comporta un cambio di contesto che viene ottimizzato dai costruttori presenti nei software che permettono di minimizzare questo tempo.

LA CREAZIONE DI UN PROCESSO segue una struttura ad albero, dove processi genitore figlio possono condividere risorse, e possono essere eseguiti in maniera concorrente, il genitore attende che uno o tutti i figli terminano prima di riprendere l'esecuzione. Se viene eliminato un processo padre muoiono a cascata tutti i processi figli.

Gli spazi degli indirizzi di un figlio è un duplicato del processo genitore, nel processo figlio si carica un programma.

in UNIX possiamo distinguere due tipi di creazioni di processi figlio, la FORK crea un nuovo processo, la EXEC dopo una fork sostituisce lo spazio di memoria del processo precedente con un nuovo programma.

Un processo TERMINA quando terminata l'ultima istruzione e chiede al SO di essere cancellato usando la system call exit e tutte le risorse del processo vengono liberate (memoria, cpu, canali di comunicazione...).

Un figlio può riportare alcuni dati al processo genitore attraverso la system call wait. Un genitore tramite il comando abort può terminare un processo figlio per eccesso di uso di risorse, per assegnazione ad un altro figlio, o perché termina.

Il figlio, quindi, non può continuare il processo padre dopo che muore poiché anch'esso muore (terminazione a cascata).

Processi cooperanti

Processo Indipendente se non può influire su altri processi nel sistema o subirne l'influsso.

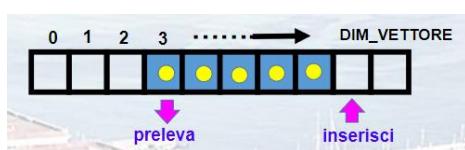
Cooperante se influenza o può essere influenzato da altri processi in esecuzione nel sistema.

Vantaggi condivisione dati, accelerazioni di calcolo, modularità.

PRODUTTORE CONSUMATORE

Esistono processi che producono, e processi che consumano, avendo una memoria limitata c'è un limite di produzione mentre i consumatori hanno bisogno di informazioni riguardanti a ciò che possono consumare, quando e come.

Una possibile soluzione è la memoria condivisa, sfrutto degli indici (preleva e inserisci) se inserisci == preleva condizione di vuoto, se ((inserisci +1) % dim_vettore) == preleva, allora il vettore è pieno. Max vettore = dim_vettore -1 perché inserisci punta sempre a una casella vuota.



while (1) forma volgare di un semaforo, scopo didattico

Item appena_prodotto;

while (1) {

```
while ((inserisci +1) % dim_vettore) == preleva); /* non fa niente */
```

```
vettore[inserisci] = appena_prodotto;
```

```
inserisci = (inserisci +1) % dim_vettore;}
```

Item da_consumare; **INSERISCI E PRELEVA SONO VARIABILI CONDIVISE DA ALTRI PROCESSI E QUINDI POSSONO ESSERE MODIFICATE DALL'ESTERNO DEL PROCESSO**

while (1) {

```
while (inserisci == preleva); /* non fa niente */
```

```
da_consumare = vettore[preleva];
```

```
preleva = (preleva +1) % dim_vettore; }
```

I processi comunicare tramite funzioni **IPC** Inter Process Comunication, attraverso due operazioni send e receive in un canale di comunicazione condiviso.

Comunicazione può essere diretta o indiretta.

Diretta: abbiamo send (P, Messaggio) invia un messaggio al processo p, receive (Q, messaggio) riceve un messaggio dal processo Q. necessita di sapere a priori delle informazioni e parametri che permettono la comunicazione diretta es nome processo e quale messaggio. **Indiretta:** i messaggi vengono inviate a delle porte(**mailbox**) che sono ben localizzate a tutti i processi. Send (A, messaggio)

Receive (A, messaggio) invia e riceve il messaggio alla porta A. Lo scambio di messaggi e le primitive send e receive possono essere sincrono(bloccante) e asincrono (non bloccante).

Sincrono richiede la presenza sia del processo mandante che ricevente, **asincrono** no. Esempio chiamata telefono/e-mail.

Code di messaggi(buffering)

Queste code usufruiscono di quest'area di appoggio, che può essere **capacità zero** (il ricevente deve essere pronto a ricevere altrimenti viene bloccato il messaggio del trasmittente) **capacità limitata** si presume che c'è un'area di appoggio che se satura il trasmittente deve attendere che si svuota. **Capacità illimitata** il trasmittente non si ferma mai.

Possiamo avere uno scambio di messaggi tra macchine diverse. **La modalità socket**, è identificato da un indirizzo IP e il numero finale corrisponde a una porta. Quindi ci può essere comunicazione tra una coppia di socket dove la porta di un calcolatore comunica con una porta di un webserver.

Possiamo avere una comunicazione tramite **RPC** Remote Process Call permettono di astrarre il meccanismo della chiamata di procedure, quindi, permette a macchine differenti di comunicare a livello più dettagliato. Poi abbiamo RMI funzioni di metodi JAVA che da un programma in una JVM invocano oggetti remoti presenti in una altra JVM.

Thread **capitolo 5**

Sono percorsi di esecuzione, thread=filo, un programma quando viene caricato in memoria si può convertire in uno o più processi. Ad ogni processo sono associati uno o più thread (non il contrario) quindi sono sempre sicuramente più dei processi.

Una macchina multithread, significa che il processore è unico, ma è composto da più nuclei di elaborazione fisici. La macchina può creare anche diversi processori logici(virtuali). Un processo può generare un o più thread, Linux & Unix creano un albero dei Thread (simile ad albero di processi).

Processo o task ha un suo spazio di indirizzamento e di stato, se la scheda del pcb è completa allora è detto **processo pesante**.

Mentre il thread per costruzione condivide lo spazio del processo; quindi, è più snello rispetto al processo, dove il codice di esecuzione è definito nello stesso flusso del processo. Quindi possiamo considerarlo un **processo leggero**.

Il parallelismo: se in un processore abbiamo più core elaboriamo più processi contemporaneamente su unita di elaborazione diverse. nei sistemi monoprocesso il parallelismo (il multithread) viene simulato tramite la concorrenza; quindi, assegnando ad ogni processo una certa quantità di tempo (time sharing), allo scadere di questo tempo abbiamo un **context switch**, nei thread il cambio di contesto è veloce perciò vengono considerati leggeri.

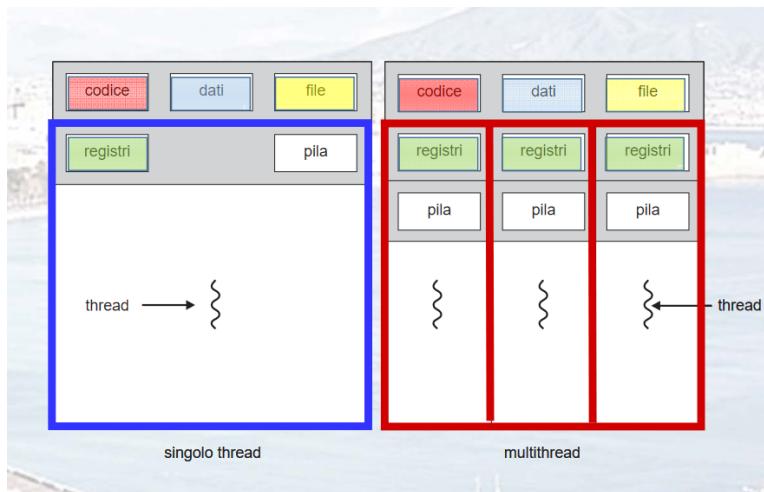
Sistemi preemptive (prelazionabili)

In un sistema non prelazionabile il cambio di conteso avviene o quando un processo o un thread interrompono la propria esecuzione volontariamente oppure terminano; in un sistema prelazionabile, posso prelazionare(preferire) un processo rispetto a un altro per maggiore priorità (comporta una maggiore complessità ma più performante).

La schedulazione (pianificazione dell'ordine dei processi da eseguire) può seguire diversi algoritmi esempio Round Robin.

Motivazione dei thread

Diverse operazioni possono essere ripetute nel tempo, es. ogni immagine di un sito viene caricata da un thread specifico senza bloccare la GUI del programma, posso visionare un'immagine senza aspettare il caricamento dell'intera pagina. I thread possono essere anch'essi in attesa di input o messaggi di un altro programma. In alternativa al multithread vi è anche il **polling** (concedo risorsa in base a chi me ne fa richiesta in quel momento interrompendo i processi precedenti).



lo scopo del thread è condividere codice dati e file di un processo.

Vantaggi tempo di risposta ridotto, condivisione e maggiore risparmio di risorse, parallelo, sforzo maggiore di rendere tutto sincronizzato. Alto rischio di ingorghi se non sincronizzati bene.

Thread a livello utente sono gestiti tramite una libreria di funzioni gestiti dall'utente. (lib POSIX, C-threads).

A livello di nucleo gestiti direttamente dal kernel del sistema operativo es: windows95-2000, solaris, linux. + complessi.

Modelli di programmazione multithread tra thread liv utente e liv nucleo molti a uno, molti a molti, uno ad uno.

Molti thread utente a un singolo thread nucleo, sistemi che non gestiscono i thread a livello nucleo. C'è un selettore che decide il flusso di processi da eseguire.

Uno ad uno vi è corrispondenza diretta tra thread nucleo a thread utente.

Molti a molti, uno numero di thread utente in corrispondenza con un numero **minore o uguale** di thread a livello nucleo.

Questo perché più thread a livello nucleo che gestiscono lo stesso thread comporterebbe uno spreco di risorse, dato che ne basta uno per ogni thread utente.

un programma multithread impiega le API (Application Program Interface) **Pthreads** che definiscono il comportamento di un thread secondo l'interfaccia **POSIX Portable Operating System Interface Unix**.

Oppure le API **Win32**. In java è stata creata la classe Thread e si possono creare thread sovrascrivendo il metodo run di quella classe, gestiti dalla JVM.

Le system call fork ed exec, fork crea un nuovo spazio per il nuovo thread mentre l'exec sostituisce il thread chiamante.

Bisogna gestire la cancellazione in vari casi, i segnali per coordinarsi tra loro, gruppi di thread che snelliscono la gestione.

Scheduling della CPU

Capitolo 6

CPU essendo l'unità più importante richiede una perfetta organizzazione del tempo e della modalità di elaborazione.

IL BUONO è MEGLIO DELL'OTTIMO, è meglio trovare una soluzione buona in generale in tempi brevi, piuttosto che una risposta perfetta in tempi lunghissimi. Bisogna trovare quindi una **STRATEGIA** adatta. La strategia varia dal punto di vista, a cosa vogliamo dare priorità.

L'obiettivo della multiprogrammazione è massimizzare l'uso della CPU, esempio in un ufficio postale sono contenta se tutti gli impiegati si dividono le code equamente in modo da snellire il carico di lavoro ed evitare effetto convoglio(**deadlock**). La **CPU** gestisce i passaggi di stato dei processi da pausa ed elaborazione. Un processo appena viene mandato in esecuzione avviene un CPU burst di calcolo, si ha un picco di operazioni di elaborazione alternato poi a momenti di attesa o assestamento.

Lo scheduler della CPU deve decidere come elaborare i processi nelle code.

Bisogna stabilire quando la CPU deve agire:

- Se il sistema è non preemptive la cpu può agire dopo che un processo termina o va in attesa (decisioni prese dal processo e non dal SO)
- Se il sistema è preemptive la cpu agisce quando si passa da pronto a esecuzione oppure da attesa a pronto, quindi può decidere quale processo far eseguire in base alle priorità.

Il Dispatcher è il modulo che manda in esecuzione il nuovo processo, comprende il context switch, il passo al modo utente, il salto alla posizione giusta da cui riprendere il programma che aveva lasciato in attesa l'utente.

il tempo per fermare un processo e avviare un altro è detto **latenza di dispatch**.

Esistono vari **criteri di scheduling**:

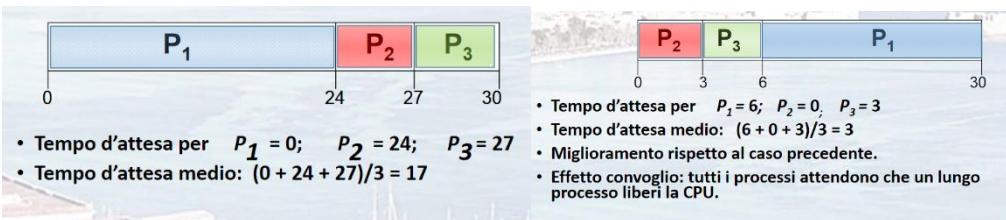
Max utilizzo della cpu, Max produttività numero di processi in un'unità di tempo, min tempo di completamento dei processi, min tempo d'attesa dei processi pronti, min tempo di risposta da una richiesta alla sua effettiva prima risposta.

Fattore extra la varianza minima/costanza di tempo: un tempo medio per tutti processi.

ALGORITMI DI SCHEDULING

First come first served: (FIFO).

Tempo di attesa medio: (somma tutti i tempi) / (numero di processi) primo processo chiamato ha tempo 0. Diagrammi di Gantt:



deadlock

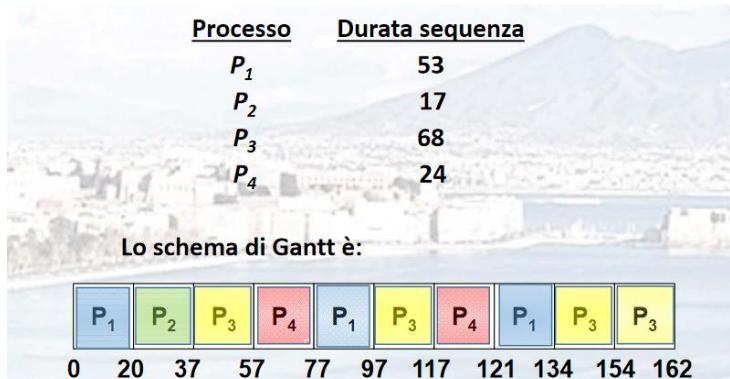
SJF: shortest job first.

Sceglie i job che richiede meno tempo per l'esecuzione totale. Due varianti: prelazionabile e non. In quello con prelazione se arriva un processo con tempo di elaborazione minore a quello attuale, lo sostituisce. Senza prelazione i processi non possono essere interrotti e vengono sempre terminati. Ottimo tempo di attesa, svantaggio non possiamo sapere a priori quale processo impiegherà meno tempo. Negli esercizi assumiamo che il costo di cambio contesto è sempre uguale e irrisorio ma nella realtà il suo costo varia a seconda dei processi che vengono cambiati e costano un determinata quantità di tempo. Quindi troppi cambi di contesto comportano a un non effettivo miglioramento delle tempistiche di elaborazione (quindi non sempre conviene come algoritmo). Si applica un principio di località per fare la predizione della lunghezza dei processi. Problema di questi algoritmi: attesa indefinita (starvation) processi con bassa priorità possono rimanere in attesa per lunghissimo tempo. Soluzione Aging invecchiamento: aumento graduale della priorità del processo che viene escluso.

Scheduling circolare (Round Robin) RR

Si stabilisce un tempo fisso q , e lo si attribuisce a n processi in maniera circolare.

Ogni processo deve aspettare almeno $(n-1)q$ unità di tempo. Per un q che tende a infinito abbiamo FCFS(FIFO), mentre per un q breve abbiamo la condivisione della CPU abbiamo il **processor sharing** (ma se abbiamo troppi cambi di contesto e stressiamo la CPU). Esempio con $q = 20$.



RR funziona bene in casi di elevato numero di processi.

Se aumenta q diminuiscono i cambi di contesto e viceversa. **Esercizi in video lezione so 6.**

Scheduling a code multiple: Si possono divider in code di processi in primo piano con scheduling RR o code in sottofondo con scheduling FCFS.

Si necessita uno scheduling tra queste code. Uno scheduling con retroazione (multilevel feedback queue scheduling) varia il quanto di tempo passando da una coda ad un'altra. Quindi le politiche di gestione dei processi cambiano continuamente.

Sistemi con più cpu si dividono in diverse tipologie, se lavorano in modo simmetrico ovvero sono interscambiabili tra di loro si ha una **multielaborazione simmetrica**, se una unità prevale e gestisce le altre e gestisce le attività di sistema è **asimmetrica (master server)**. Obiettivo caricare al massimo le cpu. Se le unità sono identiche, in relazione alle loro funzioni, allora abbiamo un **sistema omogeneo**.

Scheduling in tempo reale stretto o debole, Differenziamo i sistemi in **real time hard** capaci di garantire il completamento di funzione critiche entro un tempo definito. **Real time soft** sistemi nei quali i processi critici hanno una priorità maggiore dei processi ordinari, quindi, non garantisce in quanto tempo riuscirà a completarli. Reale stretto esempio missile.

Valutare gli algoritmi: Modello deterministico: un algoritmo testato per un certo carico di lavoro e con prestazioni definite. Si fanno statistiche.

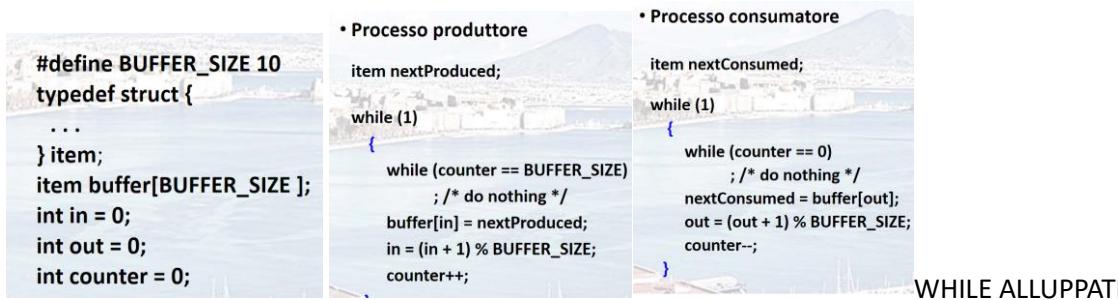
Reti di code o Simulazioni. Nei test si ha il tape trace, è un log ciclico di parametri del sistema, che permette di determinare le prestazioni di un algoritmo di scheduling.

cambiando l'ordine dei processi cambia il tempo di attesa medio, ma il totale è sempre 30.
Algoritmo estremamente rapido ma si può creare una situazione convoglio (slide a sinistra).

La sincronizzazione tra processi per l'uso delle risorse è più complesso più sono i processi da gestire.

Nei sistemi multi-programmati (attuali) vi è un enorme accesso alle risorse in maniera asincrona: possono essere condivisi sia dati, che spazi di memoria (indirizzi). Servono quindi dei meccanismi per gestire questo accesso alle risorse.

Sfruttando la soluzione della memoria limitata con la **memoria condivisa** (inserisci/preleva) con vettore coda circolare, supponiamo che vogliamo usare tutto il vettore usando la variabile counter che conta gli item prodotti e decrementato se consumati. Esempio panettiere



Problematiche: questi codici non sono univoci, diversi processi possono avere questi blocchi di codice, quindi counter è una variabile condivisa, se abbiamo counter++ e counter— istruzioni atomiche (non più scindibile in sotto-operazioni e non può essere interrotta) si possono codificare:

L'istruzione “**count++**” si può codificare in linguaggio macchina come:

```

register1 = counter
register1 = register1 + 1
counter = register1

```

L'istruzione “**count - -**” può essere realizzata come:

```

register2 = counter
register2 = register2 - 1
counter = register2

```

l'operazione avviene nei registri. (ALU)

Non è quindi in realtà un'istruzione atomica poiché scindibili in altre tre sub istruzioni; quindi, il problema nasce poiché il produttore e consumatore cercano di aggiornare il registro in maniera concorrente, dove quindi le istruzioni possono risultare intercalate (**interleaved**). Esempio di errore interleaved, le istruzioni non avvengono in maniera sincronizzata ed ordinata (prima producer o consumer, NO) e portano a un risultato errato:

Si supponga che il valore della variabile counter sia inizialmente 5. Una sequenza è:

producer: register1 = counter	(register1 = 5)
producer: register1 = register1 + 1	(register1 = 6)
consumer: register2 = counter	(register2 = 5)
consumer: register2 = register2 - 1	(register2 = 4)
producer: counter = register1	(counter = 6)
consumer: counter = register2	(counter = 4)

Il valore della variabile potrebbe essere 4 o 6, ma il risultato corretto dovrebbe essere 5.

Quindi si crea una vera e propria gara per le risorse (**race condition**): vince chi arriva ultimo a usare la risorsa.

Si risolve considerando una **regione critica**, ovvero una regione di risorse dove un processo ha un uso esclusivo e gli altri processi non possono utilizzarle/modificarle. Problema: non bisogna consentire agli altri processi di stare nella propria sezione critica se un processo si trova già nella propria.

Nella gestione della sezione critica bisogna garantire 3 proprietà: **1 mutua esclusione, 2 Progresso, 3 Attesa limitata.** 1 Se il processo P i è in esecuzione nella sua sezione critica, nessun altro processo può essere in esecuzione nella propria sezione critica. (program counter è il cursore che tiene traccia se un processo si trova nella sezione critica, delimitata da sezione di ingresso e uscita, test di condizione per entrare)

2 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 entrare in sezione critica e tale decisione non può essere ritardata indefiniteamente.

3 Se un processo ha già richiesto 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.

- Si suppone che ogni processo sia eseguito a una velocità diversa da zero
- Non si può fare alcuna ipotesi sulla velocità relativa degli n processi. *nessun processo deve rimanere in attesa infinita.

come risolvere il problema della sezione critica:

do {

(sezione d'ingresso)

sezione critica

(sezione d'uscita)

sezione non critica} while (1);

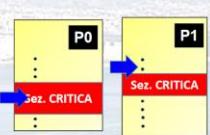
le sezioni dei processi sono diverse ma condividono le stesse risorse.

Possibili soluzioni Algoritmo 1

- Variabili condivise:
 - int turno;
 - inizialmente turno = 0
 - turno = i \Rightarrow P_i entra nella propria sezione critica

Processo P_i
do {
 while (turno != i);
 sezione critica
 turno = j;
 sezione non critica
} while (1);

• Soddisfa la mutua esclusione
ma non il requisito di progresso



- Variabili condivise:
 - boolean pronto[2];
 - inizialmente pronto [0] = pronto [1] = false.
 - pronto [i] = true \Rightarrow P_i pronto a entrare nella sua sezione critica

Processo P_i
do {
 pronto[i] := true;
 while (pronto[j]);
 sezione critica
 pronto[i] = false;
 sezione non critica
} while (1);

• Soddisfa la mutua esclusione
ma non il requisito di progresso.

Algoritmo 2

- Combiene le variabili condivise degli algoritmi 1 e 2.

Processo P_i
do {
 pronto[i] = true;
 turno = j;
 while (pronto[j] && turno = j);
 sezione critica
 pronto[i] = false;
 sezione non critica
} while (1);

• Soddisfa tutti i tre requisiti; risolve il problema della sezione critica per i due processi.

Algoritmo 3

Problema Algo 1 non è garantita l'alternanza tra processi (non sappiamo sempre se i processi si alternano senza ripetersi, Ping-pong; un processo con questo Algo non può ripetere la sezione critica, deve per forza alternarsi con un altro).

Problema Algo 2 non impone un'alternanza però non risolve il progresso, se immagino un processo P_j corrispettivo a P_i , e la prima riga pronto[i]:= true e pronto[j] := true entrambi entrano nella sezione critica nello stesso momento e si ha un loop(ogni processo blocca l'altro)

Nella Algo 3 viene aggiunto la variabile turno all'Algo 2 risolvendo tutti e 3 i requisiti.

Algoritmo 3 soddisfa i requisiti solo per 2 processi, quello del **fornaio** è una versione generale: bigliettini per ordine di accesso il processo riceve un numero prima di entrare nella zona critica, il numero più basso entra;

se due processi hanno lo stesso numero si sfrutta l'ordine lessicografico del nome. Questo algoritmo genera biglietti in ordine crescente. Blocchi verdi rappresentano sezione d'ingresso (condizioni per entrare in critica)

Il numero(biglietto) viene assegnato cercando il massimo tra tutti i processi in attesa +1. Scelta[i] flag che permette di capire se un processo è in attesa di essere eseguito.

```
do {
    scelta[i] = true;
    numero[i] = max(numero[0], ..., numero[n - 1]) + 1;
    scelta[i] = false;
    for (j = 0; j < n; j++) {
        while (scelta[j]);
        while ((numero[j] != 0) &&
               ((numero[j] < numero[i]) || 
                ((numero[j] == numero[i]) && (j < i)))
               );
    }
    sezione critica
    numero[i] = 0;
    sezione non critica
} while (1);
```

while(scelta[j]) **alluppa** finché tutti i processi non hanno un numero

Poi si fa entrare nella sezione critica, il processo con il numero più piccolo o con il nome che viene prima.

numero[i] = 0; è la sezione di uscita poiché rimuove il biglietto dal processo.

È un esempio di architettura di sincronizzazione. Ma bisogna snellire questa struttura con altre architetture sincronizzate migliori come, ad esempio, il **TestAndSet**: **test = recuperare** una variabile **set = settare a vero** una variabile.

```

1. boolean TestAndSet (boolean &obiettivo)           istruzione atomica
{
    boolean valore = obiettivo;          //prelevo il valore di obiettivo
    obiettivo = true;                  //setto a true
    return valore;                    //ritorno il valore iniziale
}

```

ESEMPIO VERSIONE MIGLIORATA

Dati condivisi:

boolean blocco = false;

- **Processo Pi**

```

do {
    while (TestAndSet(blocco)); //il while viene superato se inizialmente il blocco era false
        sezione critica      //adesso "blocco" è settato true quindi se entra un altro processo
    blocco = false;           //quel processo rimane alluppato e non entra nella critica
        sezione non critica //appena il processo nella critica termina pone "blocco" a false
    } while (1)              //facendo entrare quello rimasto nel while del test rimasto
inattesa

```

void Swap (boolean &a, boolean &b)

```

{
    boolean temp = a;
    a = b;
    b = temp;
}

```

Versione meno distruttiva nel caso volessi salvare il vecchio valore in una variabile.

istruzione sempre atomica quindi non sono intercalabili, quindi, non sono interrompibili

MUTUA ESCLUSIONE CON SWAP

Dati condivisi (inizializzati a false):

boolean blocco;

boolean waiting[n];

- **Processo P i**

```

do {
    chiave = true;
    while (chiave == true)
        Swap (blocco, chiave);
        sezione critica
    blocco = false;
        sezione non critica
} while (1)

```

SEMAFORI

Semafori evitano **attese attive** (attesa attiva quando un processo chiede costantemente se può entrare nella sezione critica facendo i vari test e quindi consumando risorsa della CPU) dobbiamo evitare gli **spinlock** ovvero processi che consumano sempre risorsa ciclicamente in esecuzione.

Il semaforo è una variabile accessibile attraverso primitive specifiche, MACRO (insieme di comandi)

wait (S):

```

while S≤ 0 do no-op; //non far niente
S--;

```

signal (S):

```

S++;

```

ESEMPIO SEZIONE CRITICA CON SEMAFORO:

Dati condivisi:

semaphore mutex; //inizialmente mutex = 1

• Processo Pi:

```
do {
    wait(mutex);
        sezione critica
    signal(mutex);
        sezione non critica
} while (1);
```

vuote=n piene=0 mutex=1

REALIZZAZIONE SEMAFORO:

typedef struct {

```
int valore;
struct processo *L;
} semaforo;
```

nell' esempio di prima sembrano sempre in attesa attiva ma in realtà il SO implementa due funzioni, **block** che sospende il processo che la invoca e **wakeup(P)** che riprende l'esecuzione, che vengono usate insieme ai semafori per evitare i spinlock. Sono due funzioni di test richiamate un'unica volta , definite dal costruttore.

wait(S):

```
S.valore--;
if (S.valore < 0) {

    aggiungi questo processo a S.L;
    block;}
```

signal(S):

```
S.valore++;
if (S.valore <= 0) {
    togli un processo P da S.L;
    wakeup(P);}
```

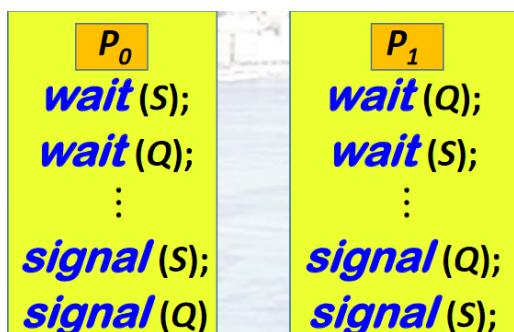
Usare un semaforo come strumento di sincronizzazione generale

Implementiamo un semaforo flag = 0, l'esecuzione di più processi è forzata dall'ordine di wait e signal; poiché un processo che inizia con una wait e il flag è a 0 rimane alluppata e aspetta quindi l'esecuzione di una signal di un altro processo. Quindi in ogni caso verrà eseguito sempre il processo i.

STALLO (DEADLOCK) CAUSATO DAI SEMAFORI

STALLO: situazione in cui due o più processi attendono indefinitamente un evento che può essere causato solo da uno dei processi in attesa. Avendo S e Q inizializzati a 1: se eseguo prima la wait(S) di P₀ e poi wait(Q) di P₁

Nessuno dei due processi può continuare e si ha uno stallo perché entrambi i semafori aspettano una signal che non arriverà mai. In questo caso si avrà una starvation, una situazione in cui il semaforo attende all'infinito.



Con un semaforo non può avere deadlock ma con almeno due si

Tipi di semaforo:

contatore: esempio un semaforo decrementato più volte da una wait ci permette di contare i processi in coda.

Binario: il suo valore è 0 o 1.

è possibile sfruttare più semafori binari e fare un controllo su una variabile condivisa dal semaforo contatore, è settare i semafori a 0 o 1

in base al valore del contatore.

produttore

```
do {
    ...
    produce un elemento in appena_prodotto
    ...
    wait(vuote);
    wait(mutex);
    ...
    inserisci appena_prodotto in vettore
    ...
    signal(mutex);
    signal(piene);
} while (1);
```

consumatore

```
do {
    wait(piene)
    wait(mutex);
    ...
    rimuovi un elemento da vettore e
    inseriscilo in da_consumare
    ...
    signal(mutex);
    signal(vuote);
    ...
    consuma l'elemento contenuto in
    da_consumare
    ...
} while (1);
```



Strutture dati:

semaforo_binario S1, S2;

int C;

• Inizializzazione:

S1 = 1

S2 = 0

C = valore iniziale del semaforo contatore S

Operazione wait

```
wait(S1);
C--;
if (C < 0) {
    signal(S1);
    wait(S2);
}
signal(S1);
```

• Operazione signal

```
wait(S1);
C++;
if (C <= 0)
    signal(S2);
else
    signal(S1);
```

PROBLEMI TIPICI DELLA SINCRONIZZAZIONE

Problema produttori e consumatori con memoria limitata: Buffer da gestire(quando si può produrre oppure consumare, c'è un limite di produzione e bisogna consumare quando c'è qualcosa).

Problema lettori e scrittori: se tutti i processi leggono non ci sono conflitti, mentre se piu di uno vuole scrivere bisogna sincronizzare i processi di scrittura. Si risolve con dei semafori appositi es. scrittura.

Problema dei cinque filosofi: Un tavolo, cinque sedie, 5 bacchette disposte in circolo tra i posti, una a sinistra un a destra per posto(si può mangiare solo con 2 bacchette)(due filosofi vicini non possono mangiare se tutti prendono le bacchette) : se tutti prendono una bacchetta stallo.

Soluzioni: 1. se solo quattro filosofi di siedono anche se qualcuno aspetta tutti possono mangiare.

2.un filosofo può prendere le bacchette solo se sono tutte e 2 disponibili(va eseguita nella sezione critica)

Rompo la condizione di possesso e attesa di una risorsa. **O TUTT O NIENT**

3.Soluzione asimmetrica: un filosofo in posizione pari prende prima la sinistra poi la destra, uno in posizione pari prende prima quella di destra e poi di sinistra.

Esempio caso 2: Tutti i semafori bacchetta di ogni filosofo posti a 1

```
do{
    wait(bacchetta[i])                      //sezione ingresso
    wait(bacchetta[(i+1)%5])
    ...
    mangia
    ...
    signal(bacchetta[i]);
    signal(bacchetta[(i+1)%5]);           //sezione uscita
    ...
    pensa
    ...
} while (1);
```

in caso di deadlock si decide ammazzare un processo.

Costrutto per gestione di regioni critiche: Costrutto di sincronizzazione ad alto livello

•Una variabile condivisa v di tipo T è dichiarata come:

v: shared T

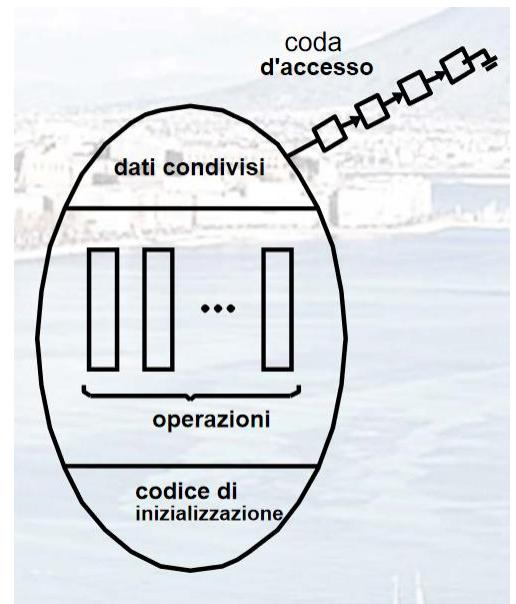
• Alla variabile v si può accedere solo dall'interno di un'istruzione: "**region v when B do S**" dove B è un'espressione booleana. • Mentre si esegue l'istruzione S nessun altro processo può accedere alla variabile.

Quando un processo vuole accedere alla variabile condivisa v nella regione critica, si valuta l'espressione booleana B, se risulta vera si esegue l'istruzione S; altrimenti il processo rilascia la mutua esclusione ed è ritardato fino a che B diventa vera e nessun altro processo si trova nella regione associata a v.

Un costrutto più ad alto livello:

Monitor

```
monitor nome_monitor
{
dichiarazioni di variabili condivise
procedure body P1 (...) {
...
}
procedure body P2 (...) {
...
}
procedure body Pn (...) {
...
}
codice d'inizializzazione
}
}
```



STALLO DEI PROCESSI

CAPITOLO 8

Le condizioni dello stallo sono i casi di possesso/attesa, nei filosofi per esempio lo stallo si crea quando tutti i filosofi(processi) hanno una bacchetta (risorsa) e ne attendono un'altra.

Nel caso dei 2 semafori entrambi ad 1, si crea lo stallo se ci sono due wait in contemporanea dove ogni semaforo aspetta l'altro.



Esempio ponte : una restrizione delle risorse può portare a uno stallo quindi come soluzione possibile usiamo il **rollback**: un ripristino di un processo ad uno stato di computazione sicuro precedente. Un possibile rischio è la starvation dei processi.

Per gestire questi problemi agiamo su due entità, le risorse e gli **utilizzatori** delle **risorse**.

Esistono vari tipi di **risorse**: quando parliamo di risorse parliamo di **classi** di risorse, ogni classe possiede un certo numero di istanze di quella risorsa.

Ogni risorsa ha tre fasi : **Richiesta**, **Uso**, **Rilascio**. Per evitare gli stalli possiamo intervenire in una di queste fasi. Per esempio, nell'uso sfruttiamo la prelazione(strappiamo le risorse a un processo) e il rollback. Nella fase di richiesta richiede uno sforzo di calcolo delle criticità. Nella fase di rilascio è pressoché inutile perché rilascia le risorse.

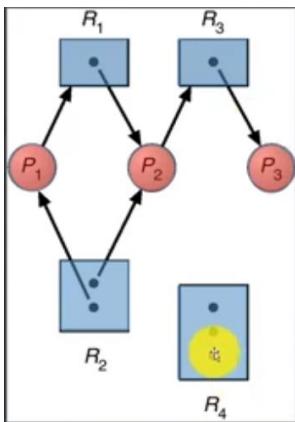
Quattro condizioni necessarie purché si crei una situazione di stallo: (se evito una di queste condizioni allo posso risolvere lo stallo)

- **Mutua esclusione**: solo un processo alla volta può utilizzare una risorsa.
- **Possesso e attesa**: un processo in possesso di almeno una risorsa attende di acquisire risorse già in possesso di altri processi.
- **Impossibilità di prelazione**: una risorsa può essere rilasciata dal processo che la possiede solo volontariamente, dopo aver terminato il proprio compito.
- **Attesa circolare**: deve esistere un insieme $\{P_0, P_1, \dots, P_n\}$ di processi tale che P_0 attende una risorsa posseduta da P_1 , P_1 attende una risorsa posseduta da P_2 , ..., P_{n-1} attende una risorsa posseduta da P_n , e P_0 attende una risorsa posseduta da P_0 .

Gli strumenti per aiutarci con questi stalli si uso **l'algoritmo del grafo**: identifichiamo V vertici(nodi) e E archi.

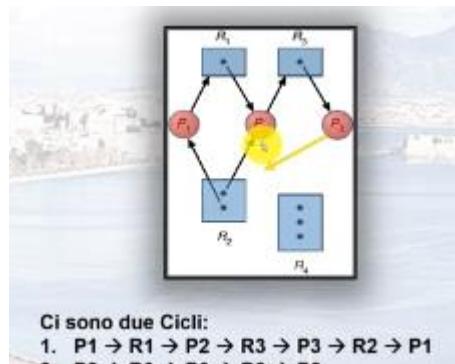
Distinguiamo P (processi) e R(classi di risorsa) come vertici, e Archi di richiesta ($P \rightarrow R[\text{classe}]$) e assegnazione ($R[\text{istanza}] \rightarrow P$). * il SO decide quale istanza di risorsa assegnare.



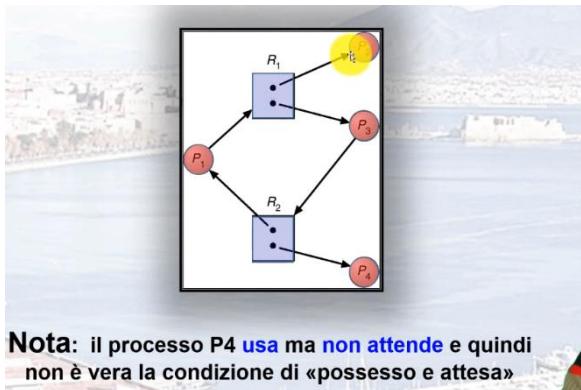


Caso senza cicli(sinistra).Per evitare casi di stallo dobbiamo evitare di creare dei cicli (caso a destra)

Non sempre il ciclo conduce a uno stallo, se Ci sono diverse istanze che ci permettono Di uscire dal ciclo (caso sotto).



Ci sono due Cicli:
 1. $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
 2. $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$



Nota: il processo P4 **usa ma non attende** e quindi non è vera la condizione di «possesso e attesa»

Se il grafo non contiene cicli non si verificano stalli, se il grafo contiene un ciclo e c'è solo un' istanza per un tipo di risorsa allo si verifica lo stallo, se ci sono più istanze c'è la **possibilità** che lo stallo non si verifichi.

METODI PER GESTIRE GLI STALLI

- 1.Assicurare che il sistema non entri in stallo(UTOPIA);
2. consentire gli stalli , individuarli e ripristinare;
3. ignorare lo stallo(UNIX).

Prevengo gli stalli, quindi, posso rompere le condizioni di mutua esclusione(comprare maggiore risorsa evitando la condivisione) e possesso e attesa (caso dei filosofi, causa basso utilizzo delle risorse e possibile starvation) Impossibilità di prelazione (uso la prelazione e possibile contro perdo dati importanti elaborati da un processo) Attesa circolare(impongo un ordine di utilizzo delle risorse che può comportare a uno spreco di elaborazione).

Evito le situazioni di stallo chiedendo il numero massimo di risorse che può chiedere, e definire uno stato delle risorse disponibili e assegnate dalle richieste massime dei processi. (questo richiede un gran numero di controlli, quindi costa molto in termine di elaborazione).

Bisogna definire uno stato sicuro: se evito uno di queste condizioni sono in uno stato sicuro. La sequenza di processi $\langle P_1, P_2, \dots, P_n \rangle$ è sicura se per ogni P_i , le richieste che P_i può ancora fare si possono soddisfare impiegando le risorse attualmente disponibili ed in + le risorse possedute da tutti i P_j , con $j < i$. * evito una circolarità. Se le risorse necessarie al processo P_i non sono immediatamente disponibili, allora P_i può attendere che tutti i P_j abbiano finito. • A quel punto, P_i può ottenere tutte le risorse di cui ha bisogno, completare il compito assegnato, restituire le risorse assegnate, e terminare. • Quando P_i termina, P_{i+1} può ottenere le risorse richieste, e così via. Quindi in uno stato sicuro non si verificano stalli. Non sicuro possibilità di stallo.

Algoritmo con grafo di assegnazione: arco di reclamo $P \rightarrow R_j$ un processo PUO' richiedere la risorsa R_j quando la richiede diventa un arco di richiesta $P \rightarrow R_j$. le risorse devono essere reclamate a priori(non sempre possibile perciò UTOPIA).

Algoritmo del Banchiere(non spiegato a lezione) *starvation non implica un deadlock

Necessita = massimo – assegnate. Vedi slide si tien genj, svantaggio non si può conoscere a priori le informazioni.

Algoritmo di Rilevamento Stalli: Algoritmo grafo d'attesa, se ci sono cicli allora li posso rilevare(pesante)

si preferisce quindi ignorare gli stalli e cercare di ripristinare i processi e non usare il rilevamento . L'uso di questo algoritmo dipende dalla frequenza di stalli e dal numero di processi. **Per ripristinare** ci basta terminare i processi in stallo, spegnere il sistema, oppure uno alla volta fino allo sblocco.(selezione vittima) poi dopo rollback(comporta starvation). **Combo prevenire + evitare + rilevare consente un approccio ottimale.**

La memoria è il luogo dove viene caricato un programma e diventa attivo (processo), più memoria più performante. Input queue = coda di processi in attesa di essere caricati in memoria centrale. Bisogna stabilire quando un indirizzo in memoria virtuale in un indirizzo reale concreto. Esistono diversi momenti:

fase di compilazione = si sa dove risiederà il processo in memoria e si genera l'**Absolute code**. Se successivamente viene cambiata la locazione è necessario ricompilare il codice. Il programma non può essere spostato poiché vincolato a una cella di memoria precisa.

Caricamento = se non è possibile sapere la locazione in compilazione il compilatore deve generare il codice rilocabile **relocatable code**. Gli indirizzi vengono generati quando il programma viene caricato in memoria.

Esecuzione = durante l'esecuzione mantengo dei simboli astratti, che mi permettono di stabilire un indirizzo fisico reale. Necessita di specifiche caratteristiche dell'architettura

SPAZIO INDIRIZZI LOGICI E FISICI

La visibilità dell'utente è astratta, diversa da quello fisico, serve un formalismo di traduzione tra queste visioni.

Indirizzi logici (prodotti dalla CPU) **indirizzi fisici (presenti nella RAM)**. La conversione viene fatta da un modulo chiamato **MMU (Memory Management Unit)** sfrutta un indirizzo base è una soglia di spostamento (**offset**) che ci conduce dal logico al fisico. 14000 base 346 offset.

CARICAMENTO DINAMICO (su richiesta) economia memoria*

Non tutto il programma viene caricato in memoria centrale, ma solo quelli richiesti dalla RAM.

Il collegamento viene differito fino al momento dell'esecuzione, lo stub è un codice di riferimento che indica come localizzare la libreria residente nella memoria.

SOVRAPPOSIZIONE DI SEZIONI (OVERLAY)

Possiamo sostituire i processi in memoria nella stessa sezione di memoria .

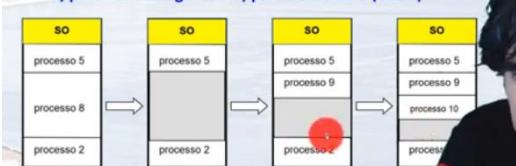
SWAPPING Uso un memoria ausiliaria di scambio (**backing store**) che mi permette di sostituire aree con altre aree con fasi di **ROLL IN E ROLL OUT**.

STRATEGIE DI ASSEGNAZIONE DI MEMORIA

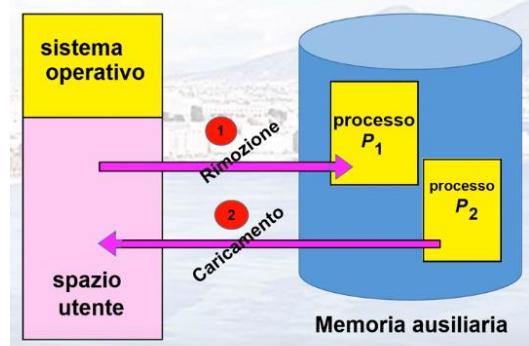
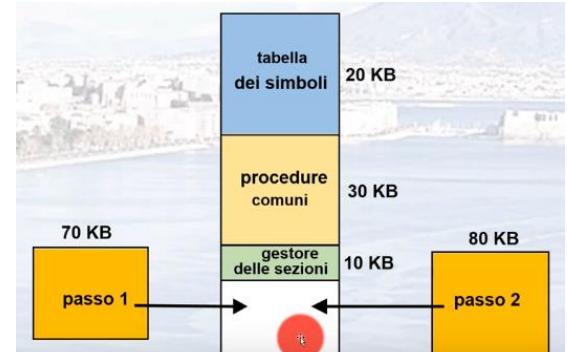
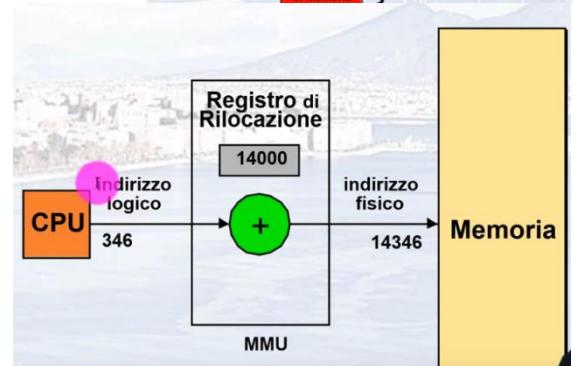
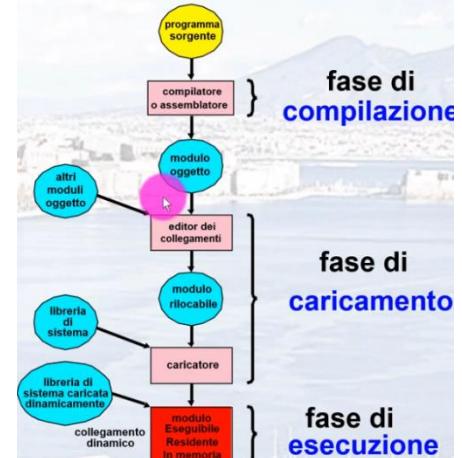
Assegnazione contigua, ha luogo quando carico l'intero programma in una porzione contigua di memoria (unico blocco contiguo). Devo garantire di sapere dove finisce un processo e dove inizia il successivo, sfruttando un registro di rilocazione (indirizzo base) e un registro limite (offset). Il processo deve essere caricato nel range prestabilito base-base+limite

• Assegnazione su più partitioni

- Buco (**hole**): blocco di memoria disponibile; ve ne sono di varie dimensioni, sparsi all'interno della memoria.
- Quando si carica un processo che necessità di memoria, occorre cercare un buco sufficientemente grande da contenere.
- Il sistema operativo tiene traccia di:
a) partitioni assegnate b) partitioni libere (buchi)



Ho bisogno quindi di una politica che mi permetta di capire come occupare i buchi in maniera ottimizzata.



Esistono diversi tipi di buchi:

First-fit: si assegna il primo buco abbastanza grande.

Best-fit: si assegna il buco più piccolo in grado di contenere il processo; occorre compiere la ricerca su tutta la lista, sempre che non sia ordinata per dimensione. Produce le parti di buco inutilizzate più piccole, ed è pertanto noto come best-fit.(posso perdere il disavanzo perché troppo piccolo)

Worst-fit: si assegna il buco più grande; anche in questo caso occorre esaminare tutta la lista. Produce le parti di buco inutilizzate più grandi. (disavanzo più grande)

First-fit e best-fit funzionano meglio di **worst-fit** poiché riducono il tempo e l'utilizzo della memoria.

FRAMMENTAZIONE

La frammentazione è una indice che misura quanto la memoria libera sia disseminata nella memoria centrale in maniera non contigua(**esterna**) una possibile soluzione e la **compattazione** di tutti questi spazi in una singola area, possibile solo se la rilocazione è dinamica e si compie nella fase di esecuzione.

La frammentazione interna mi dà informazione sul disavanzo tra ciò che chiedo e ciò che ottengo(il SO ha misure prefissate e quindi può assegnare più spazio di quanto richiesta per questione di architettura).

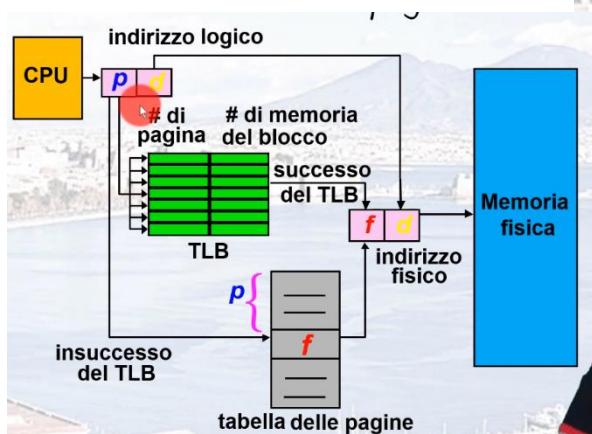
PAGINAZIONE

La memoria può essere gestita secondo un partizionamento di grandi blocchi uguali chiamati **pagine o frame**. Ogni indirizzo generato dalla CPU è diviso in **numero di pagina (p)**(indirizzo base nella memoria fisica) e **scostamento di pagina(d)**(indirizzo che combinato alla base mi permette di puntare a un parte della pagina) F sta per FRAME. Con la pagina non ho quindi più bisogno della memoria contigua, poiché posso spezzare un programma in parti uguali su varie pagine salvandomi i vari indirizzi nella **tabella delle pagine**.

ARCHITETTURA DELLA PAGINAZIONE

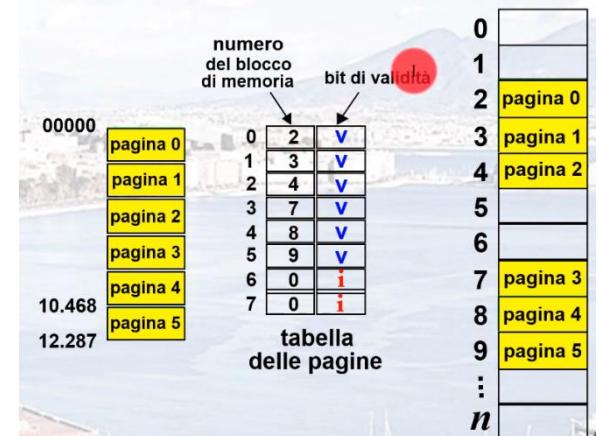
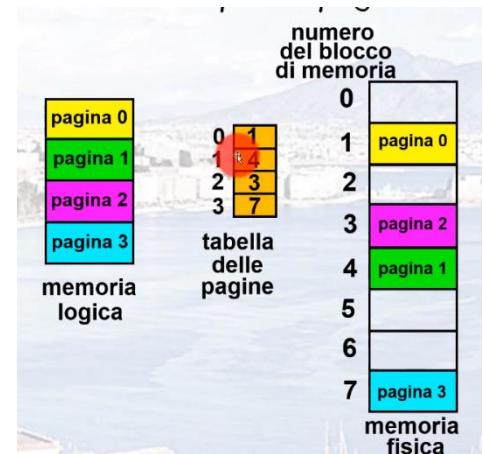
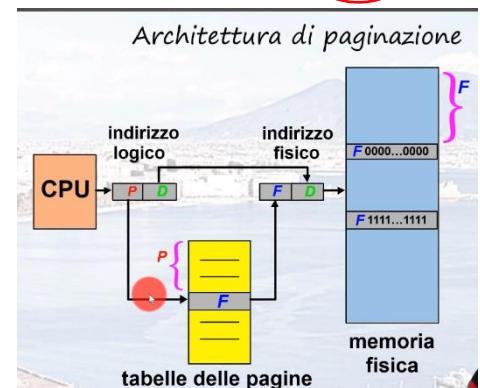
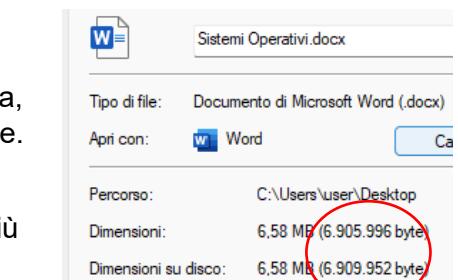
La tabella delle pagine si trova in memoria principale, e dispone di un registro della base **PTBR** e un registro della lunghezza della tabella **PTLR**. **Page table base lenght register**.Poi abbiamo il **TLB** che permette Velocizzare i tempi di accesso.

Un buffer che ad uno specifico numero di pagine corrisponde un blocco di memoria



hit ratio percentuale di volte che un numero pagina si trova nel TLB. Ci da info se conviene usare il TLB.

non sempre un numero di pagina si trova nel TLB** in caso va trovato nella tabella delle pagine



PROTEZIONE DELLA MEMORIA

Oltre al controllo sul range dell'indirizzo abbiamo il bit di validità 0 e 1 ci informa se un indirizzo logico nella tabella delle pagine corrisponde effettivamente a un indirizzo nella memoria fisica.

PROBLEMA CON PAGINAZIONE

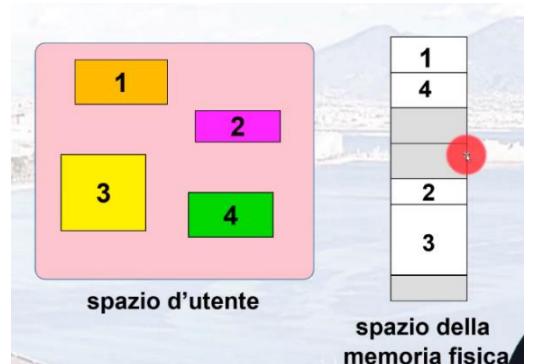
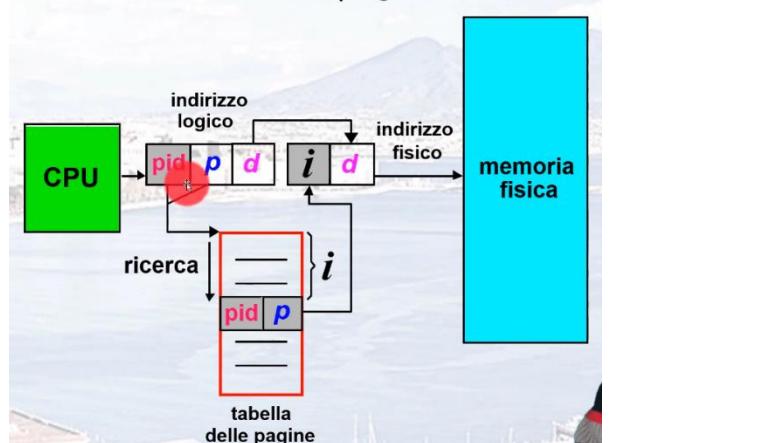
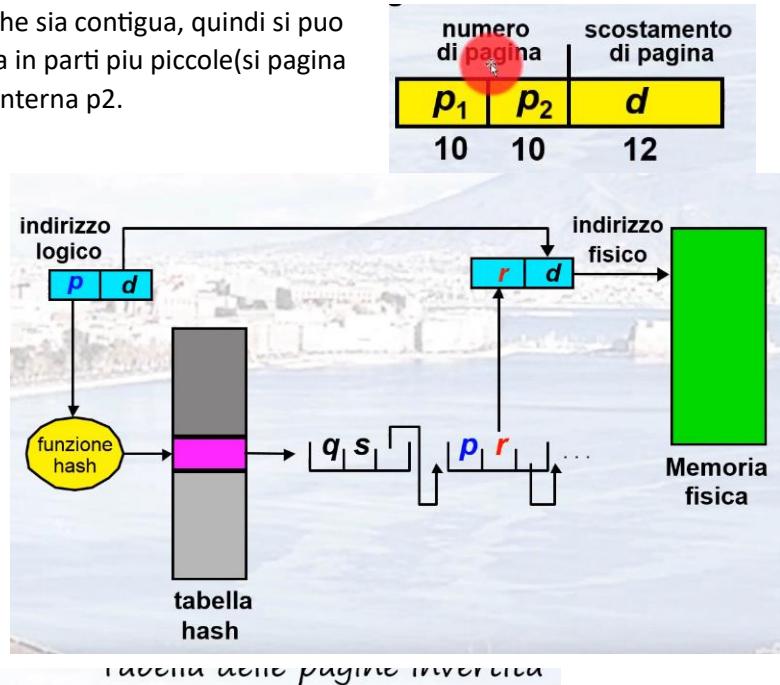
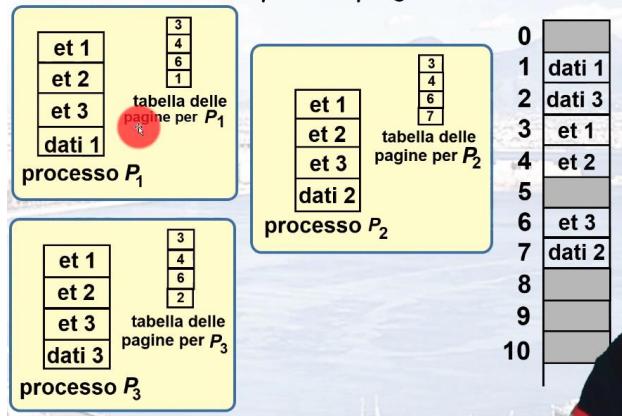
La tabella delle pagine può essere molto lunga e non è detto che sia contigua, quindi si può usare una paginazione **gerarchica**(a più livelli): divide la tabella in parti più piccole(si pagina la stessa tabella). Ho due tabelle una più esterna p1 e un più interna p2.

Posso usare il criterio **HASH(tritare)** è una funzione che dato un valore restituisce un altro valore. Vedo il risultato della funzione hash e dalla tabella hash vedo quel valore a che indirizzo fisico corrisponde.

Il problema dell' hash sono le collisioni(i casi in cui due o più valori iniziali dopo la funzione di hash possono restituire lo stesso valore, le collisioni si trovano tutte nella bucket list che conserva tutti i possibili valori di ritorno)

Un ulteriore criterio è quello della tabella delle pagine invertita, invece di usare tante tabelle delle pagine quanti sono i processi mappo la tabella delle pagine dalle pagine fisiche, avendo un'unica tabella delle pagine. Quindi oltre alla pagina e lo scostamento ho il **PID** process identifier per effettuare una ricerca ottimizzata nella tabella.

Gestisce al meglio la condivisione



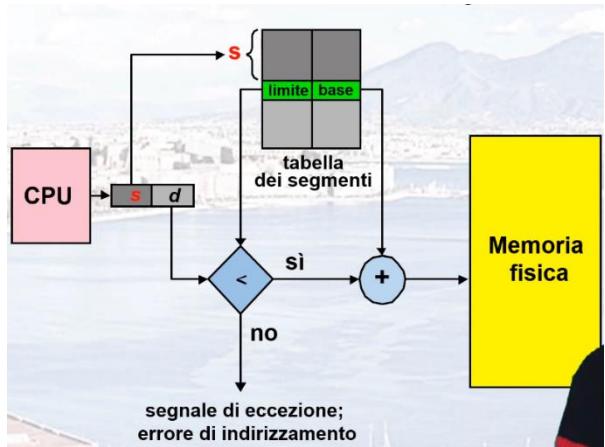
SEGMENTAZIONE

è il criterio di divisione di un oggetto, non in parti uguali, ma seguendo la natura dell'oggetto, posso fare pagine di lunghezza diversa del programma. Vantaggio che il programma è diviso in maniera più logica per un utente, le operazioni in unico spazio contiguo sono più facilitate.

ARCHITETTURA DI SEGMENTAZIONE(paginazione variabile)

È composta da un indirizzo logico composto da **numero di segmento e scostamento**. Abbiamo sempre due registri uno che punta alla tabella dei segmenti e uno che indica il numero di segmenti nella tabella, **STBR ,STLR segmente table base/length register**.

se lo spiazzamento d è minore di limite allora sommo la base ed ottengo l'indirizzo fisico. Può esserci una condivisione dei segmenti tra processi.



Virtualizzare sta a simboleggiare un livello di astrazione, in questo caso della memoria.

la memoria virtuale rappresenta la separazione dalla visione della memoria da parte dell'utente(logica) a quella fisica della macchina. Solo una parte del programma si trova dentro la memoria virtuale, lo spazio di indirizzamento logico è potenzialmente più grande rispetto a quello effettivamente fisico. La virtualizzazione ha tanti vantaggi di prestazioni e condivisione dei processi ma può richiedere maggior elaborazione nella passaggio da virtuale a fisico.

Abbiamo due tipi di gestione della memoria virtuale: **demand paging** e **demand segmentation**, paginazione e segmentazione su richiesta.

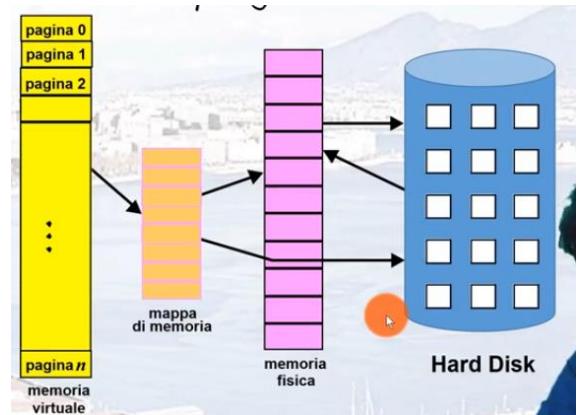
Nella memoria fisica cerco sempre di mantenere i processi attivi e liberarmi dei processi inattivi passandoli all'hard disk **

Nella paginazione su richiesta non si carica mai nella memoria una pagina non necessaria, minor tempo di avvicendamento(meno cose caricate più velocizza le prestazioni), minore quantità di memoria fisica, risposta più veloce e più utenti che possono gestire la memoria.

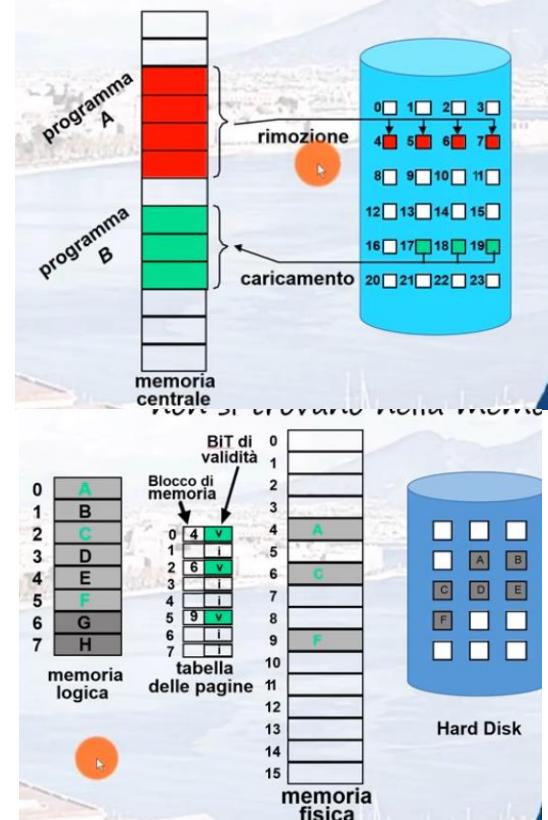
Il metodo che mi permette di ripescare dalla memoria i vari pezzi di un processo dalla memoria, sfrutto una tabella delle pagine con il meccanismo di protezione del bit di validità (1 in memoria, 0 No). Il bit all'inizio è 0. Di norma la prima pagina di un programma è la più importante perché sono presenti le direttive e implementazioni più importanti, quindi dalla memoria se voglio rimuovere una parte del processo di sicuro non la rimuovo perché potrebbe essere ricaricata per dei riferimenti del codice del resto del programma.



I pezzi di un programma in una memoria logica vengono caricati a secondo della necessità (A, C, F) (vedi destra) per vedere se nella memoria fisica ho dei pezzi di un programma vedo il bit di validità dalla tabella delle pagine. Se per qualche motivo mi viene richiesto di caricare un blocco di memoria (A) viene caricato in questo esempio dall'HDD alla memoria centrale e quando non serve viene liberata la memoria centrale.



Trasferimento di una mem nelllo spazio contiguo d

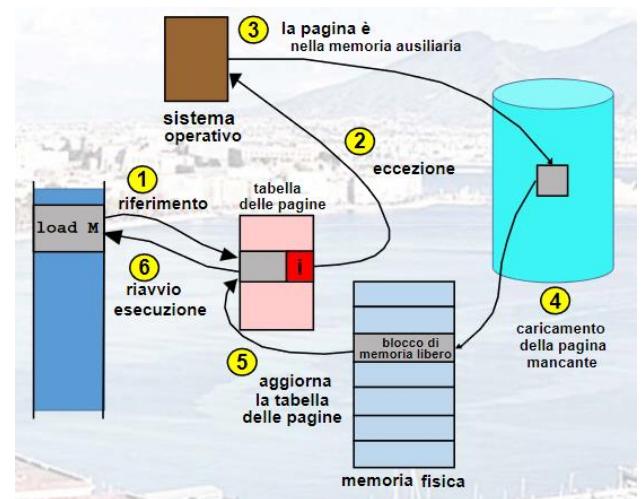


Esecuzione di pagina mancante

(1)Se un processo tenta di accedere a una pagina non caricata si ha il **page fault trap** eccezione di pagina mancante.

(2)L'architettura di paginazione, traducendo l'indirizzo, nota che il bit non è valido e invia un segnale d'eccezione al sistema operativo. Il sistema operativo deve decidere:

- (3a)errore di indirizzo non valido \Rightarrow abort.
- (3b)insuccesso del sistema operativo nella scelta delle pagine da caricare in memoria. • (3c)trovata la pagina mancante sull' HDD.
- (4)Si individua un blocco di memoria libero nella mem fisica.
- (4)Si trasferisce la pagina desiderata nel blocco di memoria libero. • (5)Si aggiornano le tabelle, bit di validità = 1.
- (5)Si riavvia l'istruzione che era stata interrotta del segnale di eccezione.



*load M è istruzione che si trova nella memoria logica (virtuale)

la memoria virtuale offre la **copy on write** (quando devo modificare una pagina effettuo una copia della pagina, se un processo genitore o figlio scrive un pagina condivisa, il sistema crea una copia, quindi solo le pag modifica vengono copiate) e anche il **memory mapping**(estendo la memoria fisica all'Hard Disk, così posso utilizzare le system call per usare la classe del RAM o del HDD).

Associazione dei file alla memoria (snellisce l'uso delle primitive)

L'associazione alla memoria di un file consiste nel permettere che una parte dello spazio degli indirizzi virtuali sia associata logicamente a un file.

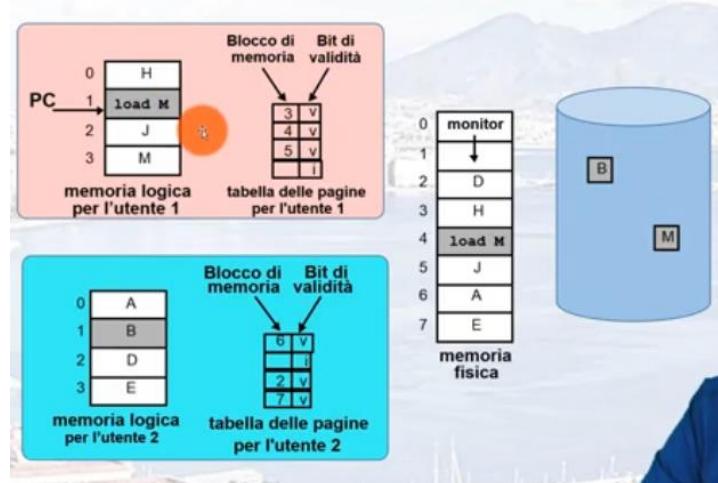
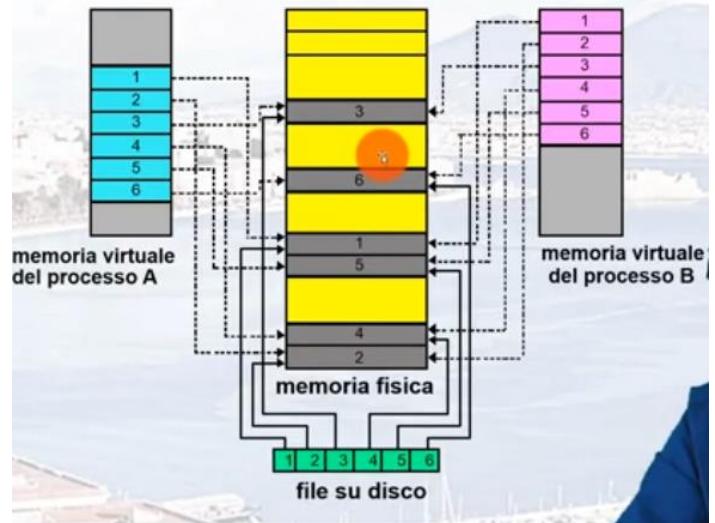
- L'accesso iniziale al file avviene per mezzo dell'ordinario meccanismo di paginazione su richiesta, che determina un'assenza di pagina cui segue il caricamento in una pagina fisica di una porzione di file della dimensione di una pagina, leggendola dal file system. Le operazioni successive di lettura/scrittura sul file si gestiscono come normali accessi alla memoria.

• In questo modo si semplifica l'accesso e l'uso dei file, si permette la manipolazione degli stessi attraverso la memoria e si evita il carico dovuto alle chiamate del sistema `read()` e `write()`.

- Si può permettere l'associazione dello stesso file alla memoria virtuale di più processi, allo scopo di condividere dati.

POSSO SIMULARE UNA MEMORIA PIÙ GRANDE sostituendo le pagine inutilizzate con quella da utilizzare, per scegliere la pagine da sostituire (si utilizza **un dirty bit** (solo le pagine modificate vengono scritte su disco).

Si individua uno spazio libero, se c'è va bene, altrimenti serve un algoritmo per scegliere la vittima da sostituire, si libera lo spazio viene caricata la pagina e riavviato il processo per andare avanti.



Questi algoritmi di sostituzione della pagina si valutano attraverso la **reference string** dove si calcola il numero di assenze di pagine in quella successione e indicano la successione dei riferimenti alle pagine che vanno caricate in memoria.

Esiste un rapporto tra pagine mancanti page fault e lo spazio disponibile cioè i numeri di blocchi di memoria. Più spazio c'è quindi meno ci dobbiamo preoccupare di liberare e sostituire aree di memoria. Alcuni programmi se non c'è abbastanza spazio non

vengono caricati.

PRIMO ALGORITMO FIFO

è buono perché è il più semplice da implementare, (butta fuori la prima che è entrata)

Successione di riferimenti: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5				
3 blocchi di memoria		4 blocchi di memoria		
9 assenze di pagine				10 assenze di pagine
1	1	4	5	1 1 5 4
2	2	1	3	2 2 1 5
3	3	2	4	3 3 2 4
1				3

nel caso di 3 blocchi di memoria dalla string scrivo 1,2,3 poi rimuovo la pagina 1 è inserisco la 4 e così via seguendo la reference string. I passaggi sono: *primo entrato da sostituire

123 → 423 → 413 → 412 → 512 → 512 → 512 → 312 → 342 → 345

Con 4 blocchi di memoria ho un **anomalia di belady** ovvero ho avuto un aumento di assenza di pagine (strano, non rispecchia il grafico sopra) il FIFO in pratica non butta fuori quella che meno ci serve ma butta la prima, quindi quella che quasi sicuramente viene ricaricata in memoria (per questo c'è una pagina in più).



ALGORITMO OTTIMALE OPT

si sostituisce la pagina che non si userà per il periodo più lungo di tempo. Il **4** è quello che dopo più tempo viene richiamato e verrà sostituito. Il problema è che non è sempre possibile conoscere a priori la successione dei riferimenti(guardare il futuro).

ALGORITMO LRU LEAST RECENTLY USED

Utilizziamo un'approssimazione, eliminiamo la pagina usata più lontanamente possibile(principio di località se ragiono su uno spazio di tempo breve, posso dire che l'immediato futuro è simile all'immediato passato), se una pagina non l'ho utilizzata recentemente molto probabilmente non la riuserò più nell'immediato futuro(secondo il principio di località LRU guarda al passato per prevedere il futuro). Esempio a destra se devo caricare la pagina 5 guardo indietro(a sinistra della stringa) e noto che la pagina 3 è l'ultima di quelle usate senza essere ricaricate più recentemente (come la 1 e 2) (la 4 è stata caricata dopo quindi è più nuova rispetto alla 3) quindi la 3 verrà sostituita per prima per far spazio alla 5. **Metodo 1:** Nelle pagine viene salvato un contatore più il contatore è piccolo più la pagina viene scelta, significa che è più vecchia cioè è stata chiamata prima. **Metodo 2:** si sfrutta uno stack, con una double linked list, in cima allo stack si trova sempre la pagina usata per ultima in fondo si trova la pagina usata recentemente. Si sfruttano i **bit di riferimento** a ciascuna pagina si usa un bit che sta a 1 ogni volta che ci si riferisce ad essa e si sostituisce quella che sta a 0 se esiste. Con **seconda chance**(o orologio) esegue una interrogazione doppia circolare dei bit di riferimento se viene confermato allora avviene la sostituzione. Poi abbiamo **LFU** sostituisce la pagina con il conteggio più basso o **MFU least/most frequently used** probabilmente la pagina con il contatore più basso è quella che è stata appena inserita.

INTERFACCIA DEL FILE SYSTEM

CAPITOLO 11

Non è una delle parti critiche per il calcolo, come la gestione CPU o Memoria.

FILE è un insieme di informazioni correlate e messe insieme nella memoria secondaria a cui è stato assegnato un nome.

Il nome non assicura il contenuto. Un .doc non assicura che sia un file di testo(è solo un'indicazione al SO su quale applicazioni usare per aprire il file). Un file può essere di diversi tipi *vedi slide.

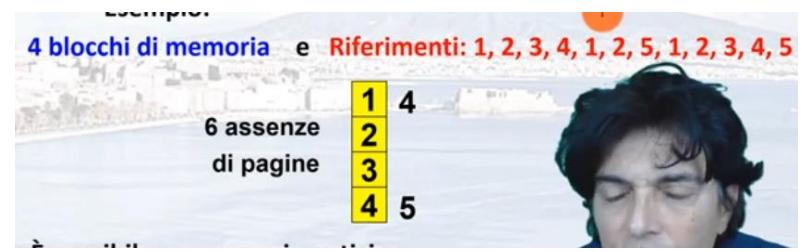
Le strutture del file possono essere **nessuna**: sequenza di byte; **semplici**: righe, lunghezza fissa o variabile(log file, file che registra tutti gli eventi di sistema, password, host); **complesse**: documento formattato(pdf html con oggetti grafici) e codici relocabili. Word , mkv, avi contenitori che possono avere file di diversa natura.

ATTRIBUTI DEI FILE

Dipende dal SO: un file dal DOS non ha l'attributo che indica il proprietario mentre Windows sì, spesso questi attributi sono segnati nella directory e non direttamente sul file. Es. nome, tipo, locazione, dimensione, protezione(lettura scrittura), ora data..

OPERAZIONI PRINCIPALI

Creazione, scrittura, lettura, riposizionamento, cancellazione, troncamento, **open** ricerca nella directory e carica il file in memoria, **close** rimuove il file dalla memoria e lo ripone nella directory.(ricorda il caricamento è parziale in base alla richiesta) **open** e **close** sono due primitive. **.bat** file batch(insieme di istruzioni)



- Successione dei riferimenti: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	5
2	
3	5
4	3

Metodo 1: Contatori

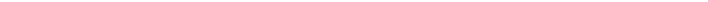
A ogni elemento della tabella delle pagine si associa un campo **del momento d'uso**, e alla CPU si aggiunge un contatore che si incrementa a ogni riferimento alla memoria.
Ogni volta che si fa un riferimento a una pagina, si copia il contenuto del registro contatore nel campo del momento d'uso nella tabella relativa a quella specifica pagina.

- Quando occorre sostituire una pagina, si sostituisce quella con il valore associato più piccolo.

4	7	0	7	1	0	1	2	1	2	7	1	2
7	7	0	7	1	0	0	1	2	1	2	7	1
4	4	4	4	4	4	4	4	4	4	4	4	4
4	4	4	4	4	4	4	4	4	4	4	4	4

In tal modo:

- in cima allo stack si trova sempre la pagina usata per ultima
- in fondo si trova la pagina usata meno recentemente



file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

METODI DI ACCESSO

1.Sequenziale, posso leggere solo in una certa posizione(nastro magnetico).

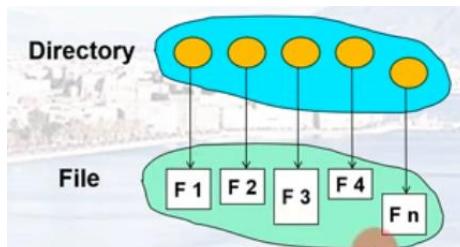


**read next
write next
reset
no read after last write
(rewrite)**

2.Diretto posso indicare la posizione esatta in cui voglio leggere o scrivere.

Posso simulare entrambi i metodi con l'altro.

I file possono essere memorizzati nella cartella seguendo un file di indice che mi manda ai file relativi veri e propri. Ci sono vari modi: il primo è la **directory** e l'elenco degli identificativi che puntano ai veri file. Possono esistere anche copie di backup su disco delle dir.



**read n
write n
position to n
read next
write next
rewrite n**

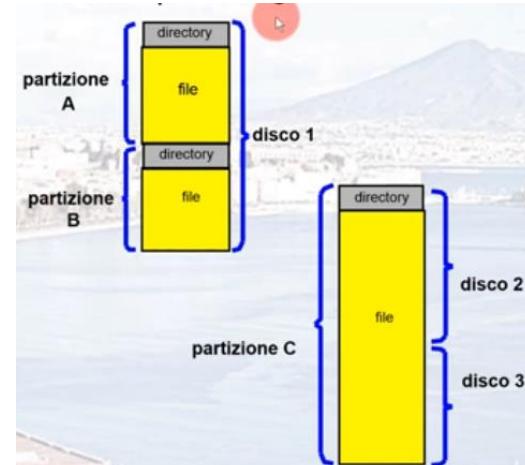
I file system **sono organizzati** principalmente su disco: il disco può essere diviso in partizioni diverse(A e B esempio per avere due SO).ogni partizione ha una sua directory. è possibile fondere più dischi come un'unica entità(vedi partizione C).

Nelle directory abbiamo diverse info:

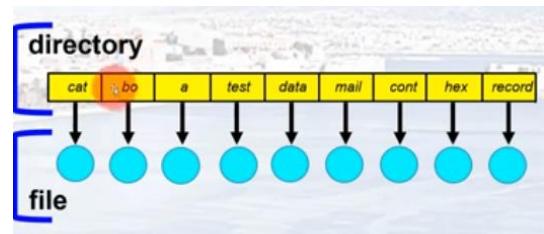
e diverse operazioni:

- Nome
- Tipo
- Indirizzo
- Dimensione corrente
- Dimensione massima
- Data dell'ultimo accesso
- Data dell'ultimo aggiornamento
- ID del proprietario
- Informazioni di protezione

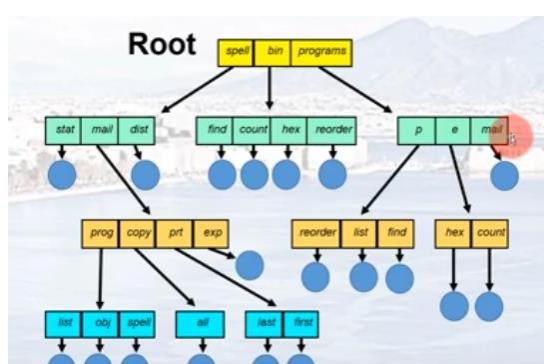
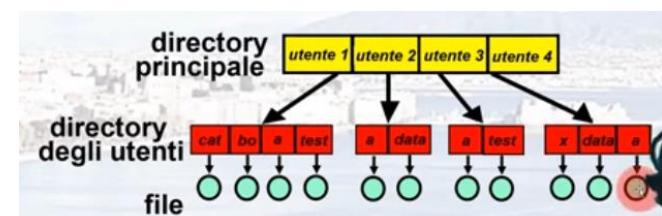
- Ricerca di un file
- Creazione di un file
- Cancellazione di un file
- Elencazione di una directory
- Ridenominazione di un file
- Attraversamento del file system(gestire la navigazione nelle directory da parte dell'utente, non tutto è acccesibile)



Organizzare un directory per aumentare l'**efficienza** e individuare subito un file, **nome** leggibile e intuitivo, **raggruppamento** per tipo di file. Uso le cartelle **a livello singolo** ovvero la directory raggruppa tutti i file per tipo è ho una singola lista e non più una struttura ad albero per tutti gli utenti che usano il sistema (schema a destra).

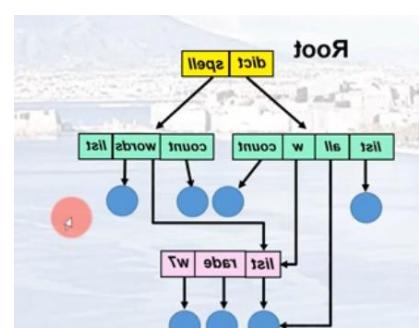


Directory **a due livelli** ho una directory degli utenti e ogni utente a il suo accesso a determinati file . (destra)



Abbiamo anche dei Path, se parte dalla radice "\ allora è **assoluto** , invece se parto dalla **current directory** è un percorso **relativo**. Operazioni rm remove file mkdir crea cartella, se cancella la cartella superiore a cascata si eliminano le sotto cartelle. Noi in realtà abbiamo un grafo aciclico quando parliamo di cartelle condivise.Nomi diversi a stesso file(**aliasing**).

La cancellazione di un file potrebbe causare l'avanzo dei puntatori nella directory.soluzione(cancello tutti i collegamenti al file e poi il file). Versione vecchie windows potevano creare un grafo ciclico. Si possono evitare con garbage collection e altri algoritmi.



Montaggio di un file system

Quando accendo il sistema si effettua il montaggio(mount point) quindi quando connetto un dispositivo il SO fornisce il punto di montaggio dove collegarsi alla directory. per la **condivisione dei file** servono **schemi di protezione**, ad esempio **NFS** network file system(strategie che permettono di distribuire i dati sulla rete ai vari utenti).

Quindi quando parliamo protezione dobbiamo stabilire un proprietario e i tipi di accesso:

- Lettura
- Scrittura
- Esecuzione
- Aggiunta
- Cancellazione
- Elencazione

Posso decidere per che cosa posso condividere a scelta arbitraria. Schema unix → gruppi dei proprietari → 3 bit → corrispondono a quale gruppo appartiene il tipo di accesso al file. Sistema Ottale(da 0 a 7) 7 6 1 numeri dei proprietari.

REALIZZAZIONE DEL FILE SYSTEM

CAPITOLO 12

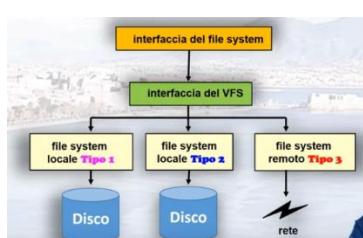
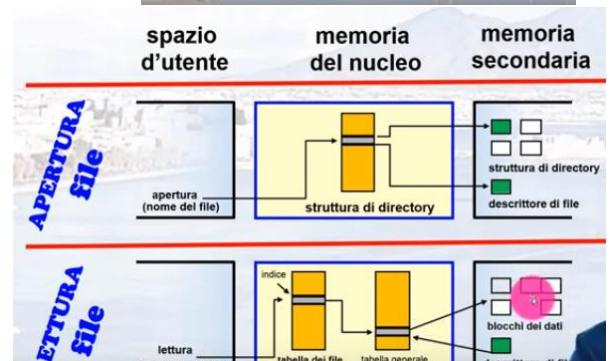
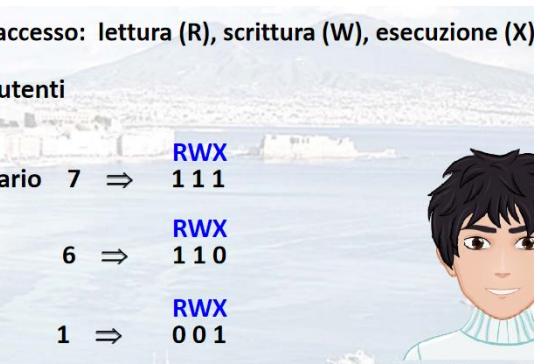
Struttura del file system: risiede nella memoria esterna secondaria permanente, ovvero anche in mancanza di energia il file permane nella memoria. È **stratificata** è a **livelli** e ogni livello a una funzione diversa. possiede un **FCB file control block** che contiene informazioni sui file come proprietà, permessi e posizione.

Più scendiamo giù di livello più usiamo delle primitive e formalismi più semplici e atomici per gestire i file system.

Il tipo descrittore di file contiene tutti i **permessi** data **proprietario dimensione e blocco di dati**. Quando accedo al file system, io accedo solo ai file indice e uso solo una parte del file in base alla necessità.

In apertura carico i file, mentre in lettura tramite l'indice leggo il file che voglio usare tra la tabella dei processi aperti.

I **VFS** file system virtuali , sono file system virtuali accessibili all'utente mascherando quella che è la struttura fisica del file vera e propria e tutti i suoi meccanismi in maniera semplificata e uniforme, i **vnode** sono nodi identificativi di un file che ne permettono il recupero. In rete le operazioni sono più lente di quando file system è in locale.



REALIZZAZIONE DIRECTORY

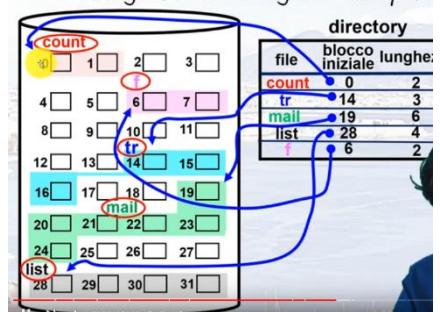
Lineare, lista dei nomi dei file e ogni nome punta al blocco di dati.

La strategia di puntamento è l'hash, diminuisce il tempo di ricerca perché non viene fatta scorrere una lista ma si accede ai file attraverso la codifica di un valore chiave. Contro **Collisioni** 2 chiavi stesso hash.

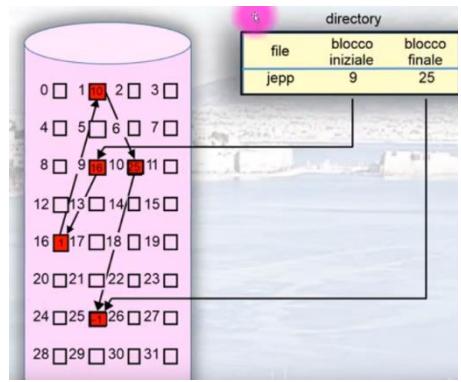
METODI DI ASSEGNAZIONE

Strategie di assegnazione dei dati a regioni della memoria secondaria.

Assegnazione contigua: semplice è richiesto solo l'indirizzo del primo blocco libero e la lunghezza in blocchi successivi liberi contigui. **Limite impiego dello Spazio, vantaggio l'accesso è rapido.**

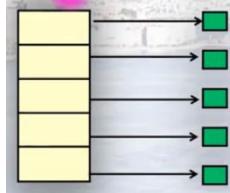


Assegnazione concatenata: tramite una linked list posso sparpagliare un file in memoria, ho la possibilità di non sprecare spazio. È necessita solo l'indirizzo di partenza. Non vi può essere accesso casuale perché devo scorrere la lista. Una variante è il **FAT** file allocation table ,una tabella dei file.

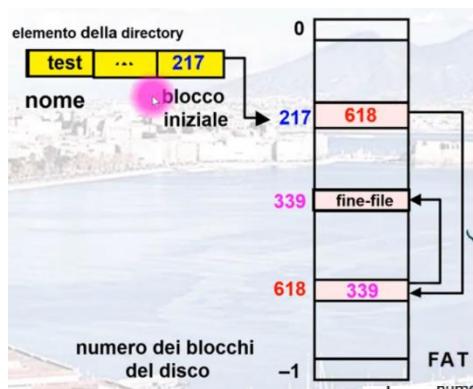
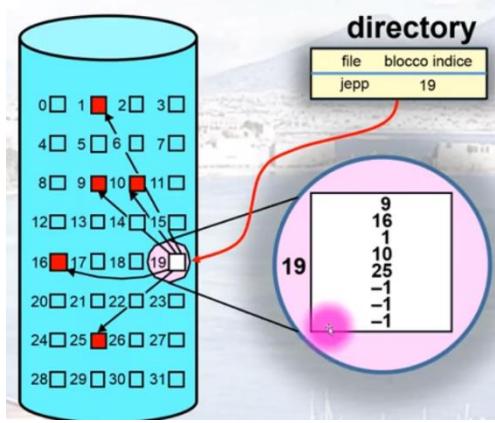


Assegnazione indicizzata:

raggruppa tutti i puntatori in un blocco indice, tabella degli indici. Pro: accesso casuale, non ha frammentazione esterna



Nel FAT, Dove non ci sono Puntatori metto -1.

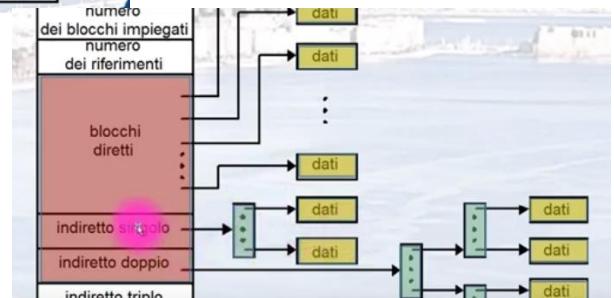


Se il file è piccolino uso dei puntatori diretti ai blocchi dati. Se il file è troppo grande uso un indiretto **singolo**, o **doppio** etc(accedo a una tabella che contiene i puntatori ai blocchi, oppure una tabella che contiene i puntatori ad altre tabelle fino ad arrivare ai dati).

Se la tabella degli indici è troppo grande si possono usare dei meccanismi:

schema concatenato, a piu livelli o schema combinato.

Il combinato sfrutta i vantaggi di tutti i metodi precedenti (Unix).



Gestione dello spazio libero

Metodo 1: Uso un **vettore di bit** che mi indica quali blocchi sono liberi(1) o occupati(0). Per trovare il primo blocco libero:

numero bit per parola(dipende dall'architettura) * numero parole di valore 0 + scostamento primo bit 1 (es. 1000 = scos= 3)

svantaggio questo vettore di bit deve essere contenuto nella memoria.

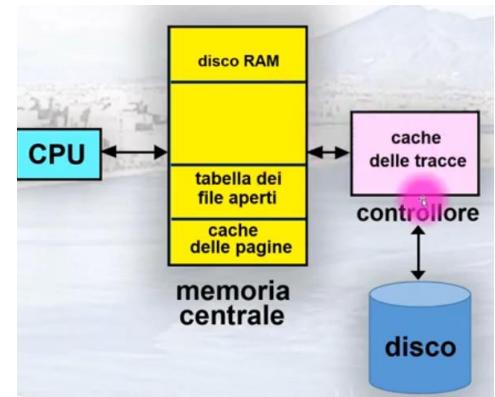
Per ovviare a questo svantaggio **metodo 2: lista concatenata** dei blocchi liberi.

L'efficienza della gestione dello spazio libera dipende da molti fattori, uno importante è la **(buffer)cache**: sono memorie temporanee con tempi di accesso più elevato, che permette al SO di evitare l'attesa della disponibilità dell' hard disk e appoggiare i dati in questo buffer e così può poi continuare l'elaborazione.

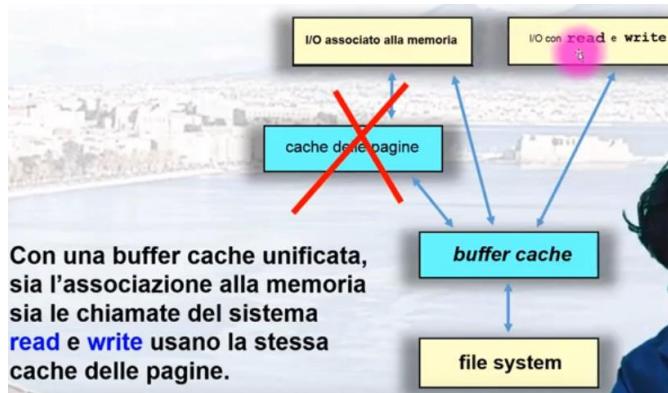


Quando si legge dal disco non si legge solo un blocco ma anche quelli successivi(**principio di località**) immediato passato=immediato futuro) quindi ci sono tecniche come la **lettura anticipata**.

Disco RAM , posso considerare un pezzo di ram come una memoria disco, esempio vi installo un programma per sfruttare la velocità della RAM.



Posso unire tutte le cache del sistema per creare un'unica **cache unificata**.



RIPRISTINO

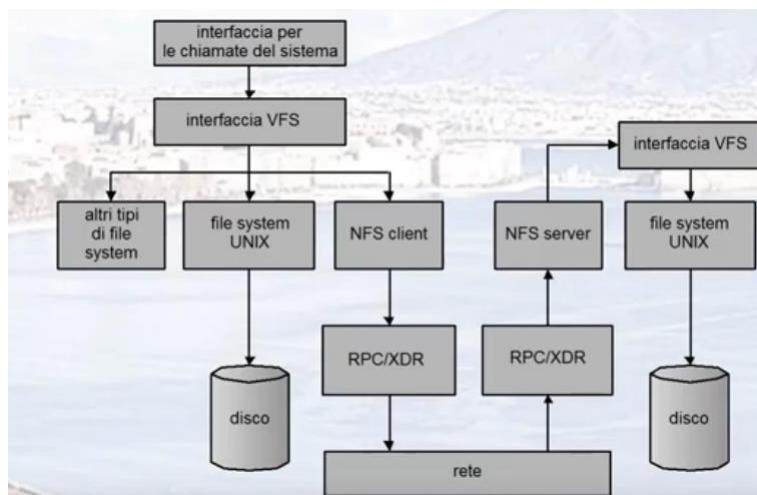
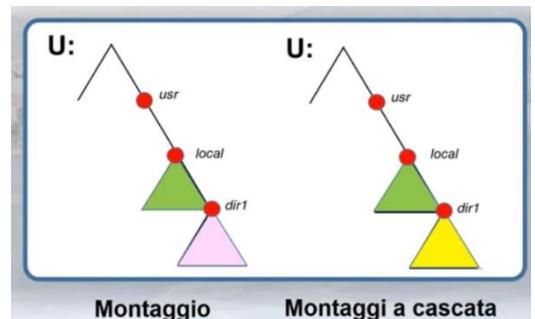
È una strategia per garantire la coerenza dei file system,

backup e restore. Backup creo un istantaneo del mio sistema e la salvo nell hard disk, e restore sfrutto questa instantanea per ripristinare il mio status di elaborazione come in quel momento.

NFS

Permette accesso a file remoti tra vari utenti, unisce più stazioni di lavoro(LAN,WAN) a tutti gli utenti questi file vengono trattati in maniera illusoria allo stesso modo, come se fossero a tutti in locale, si ha un montaggio NFS. Il protocollo NFS è basato sulle RPC(Remote process Call)insieme di primitive) che permettono svariate operazioni su file: ricerca ,lettura ,manipolazione accesso attributi, Lettura e scrittura.

Architetture NFS gestisce il VFS e gestisce operazioni di open read write e close.



quando si accede a file system su sistemi diversi esempio client-server, necessito di una traduzione dei nomi dei percorsi in modo tale che i file system condivisi vengano richiamati nell NFS tramite una chiamata lookup composta dal nome della componente il vnode di directory e creare una percorrenza coerente.

SISTEMI I/O

CAPITOLO 13

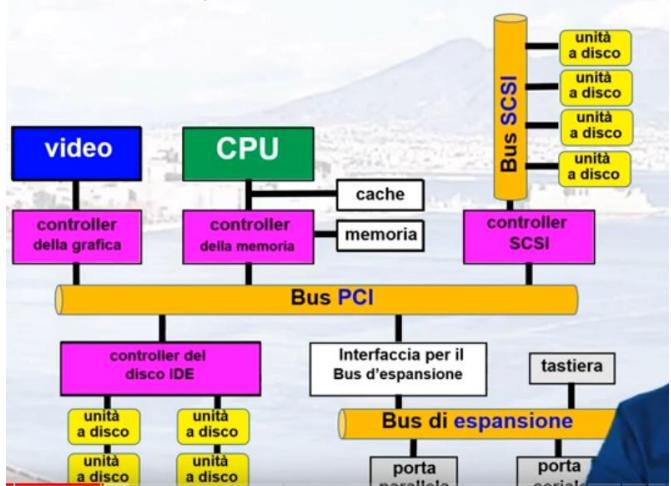
tanti dispositivi I/O comporta una gestione di tali dispositivi da parte dell SO. Questi dispositivi hanno concetti comuni:

porte per l'accesso, **bus** canali di comunicazione a margherita o accesso diretto condiviso,**controller** interfacce che collegano il bus I/O con i bus di Sistema.**driver** programmi di accesso al dispositivo(**black box** il costruttore del device crea questo software per facilitare l'accesso al dispositivo da parte del sistema e evitarne eventuali conflitti e peso). **Istruzione speciali, dispositivi di controllo e indirizzi di associazione alla memoria.**

STRUTTURA DEL BUS

PCI = bus di sistema **SCSI**= bus sottosistema, hanno il compito di fare un routing dei protocolli, ognuno ha il suo, i controller servono per regolamentare le frequenze di lavoro e interfacciare il dispositivo con il sistema. Sui controller abbiamo anche i vari **buffer** di memoria, come deposito per i dati.

Controller **SATA** più nuovi*.



Questi dispositivi hanno indirizzi specifici di riferimento

indirizzi per l'I/O (in esadecimale)	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 dischetti
3F8-3FF	porta seriale (principale)

CICLO CON INTERRUPT →

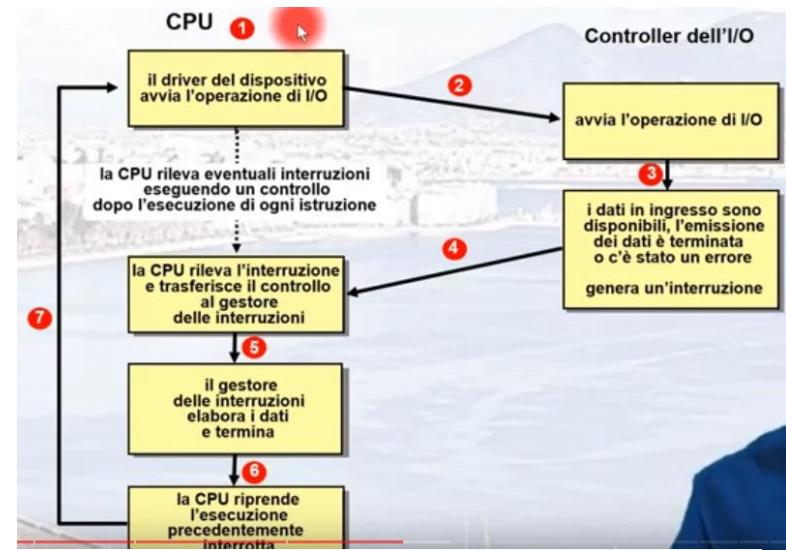
Le interruzioni vengono gestite in un vettore indicizzato, e ogni indice corrisponde a un tipo di interruzione.

indice del vettore	descrizione
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	divide not available
8	double fault
9	coprocessor segment over

METODI DI INTERROGAZIONE DEI DISPOSITIVI I/O

Polling = interrogazione ciclica per sapere se i dispositivi hanno bisogno di effettuare operazione i/o. determinato lo stato del servizio che puo essere :**command-ready, busy, error**. Ottimo in alto traffico.

Interrupting = gestisce eventi asincroni vere e proprie interruzione(alzate di mano) buono in situazione di basso traffico.



ACCESSO DIRETTO

Molti dispositivi hanno bisogno di caricare tanti dati in memoria, per evitare che la cpu governi questa funzionalità è possibile effettuare un trasferimento diretto dal dispositivo alla memoria centrale. Necessità di un direct memory access controller DMA.

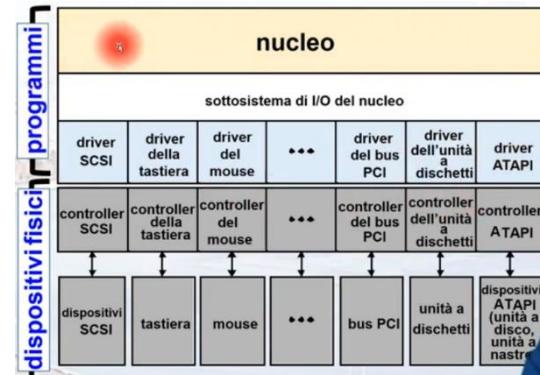
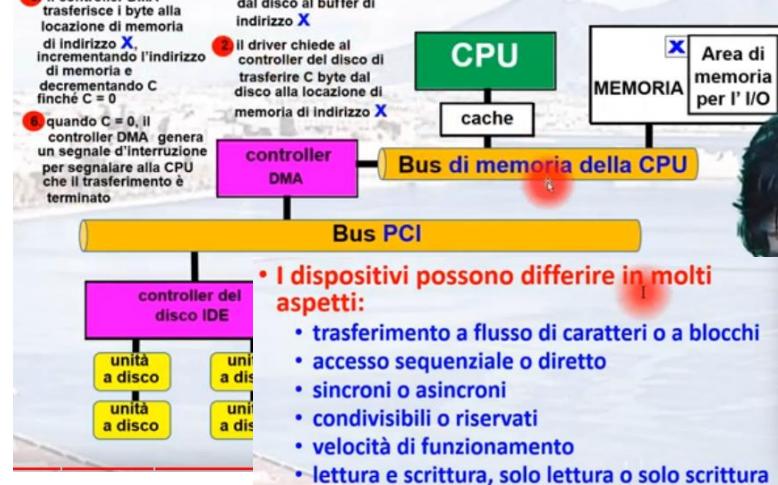
Interfaccia di I/O per le applicazioni

Nonostante i tanti tipi di dispositivi diversi, le operazione che si possono fare si possono racchiudere in **classi di metodi** (accesso lettura etc) la particolarizzazione di questo accesso è gestito dal driver. Lo scopo di questa interfaccia di sistema è di astrarre queste operazioni a basso livello, con dei comandi più comprensibili all'utente (indipendente dalla natura dei controller dei singoli dispositivi).

Sottosistema di I/O è un interfaccia per gestire tutti i driver dei singoli dispositivi senza entrarne in diretto contatto.

A destra tabella caratteristiche dei vari dispositivi

aspetto	variazione	esempio
modo di trasferimento dei dati	a caratteri a blocchi	terminale unità a disco
modo d'accesso	sequenziale casuale	modem lettore di CD-ROM
prevedibilità dell'I/O	sincrono asincrono	unità a nastro tastiera
condivisione	dedicato condiviso	unità a nastro tastiera
velocità	latenza tempo di ricerca velocità di trasferimento attesa fra le operazioni	
direzione dell'I/O	solo lettura solo scrittura lettura e scrittura	lettore di CD-ROM controller della grafica unità a disco



I dispositivi possono essere con trasferimento **a blocchi**, interi blocchi di istruzioni o codice, o unità... **trasferimento a caratteri**(tastiere mouse porte seriali)

Vengono passati dati carattere per carattere e ci sono comandi **get e put**. Posso pacchettizzare questi caratteri simulando dei **blocchi e definisco un accesso riga per riga**.

BISOGNA DIFFERENZIARE TRA ACCESSI LOCALI E REMOTI

I Dispositivi di rete hanno regole diverse, rispetto ai locali e vengono gestiti in maniera diverso.

Orologio segna l'ora corrente, **un temporizzatore** segna la durata per quanto tempo è stato utilizzato un dispositivo.

ioctl(UNIX) input output controller controlla gli orologi e i temporizzatori dei dispositivi.

Esistono due tipologi di I/O: **bloccante** – sospende l'esecuzione di un'applicazione (semplice ma inefficiente), **non bloccante** – si sovrappone l'elaborazione con l'I/O. (più complesso, usato per le chiamate asincrone, restituiscono il controllo al chiamante senza che l'I/O sia stato completato)

ANCHE I SOTTOSISTEMI DI I/O NECESSITANO DI SCHEDULING

Criteri: FIFO, SJF..., si necessita di una memoria transitoria detta **buffer** così gestiamo la velocità tra produttore e consumatore.

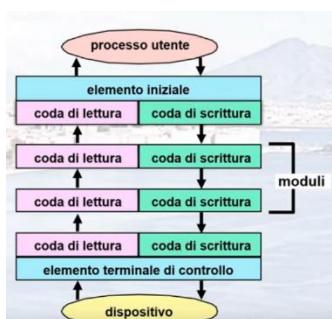
La cache è problematica nonostante è veloce, richiede una gestione delle copie, sincronizzazione/coerenza delle cache (tra la copia del sistema, e quella del dispositivo). **Veloce ma attenti alla sincronizzazione.** **Code spooling:** memoria di transito contenente dati per un dispositivo che non possono essere intercalati e quindi si gestisce una richiesta alla volta (stampante).

GESTIONE DEGLI ERRORI

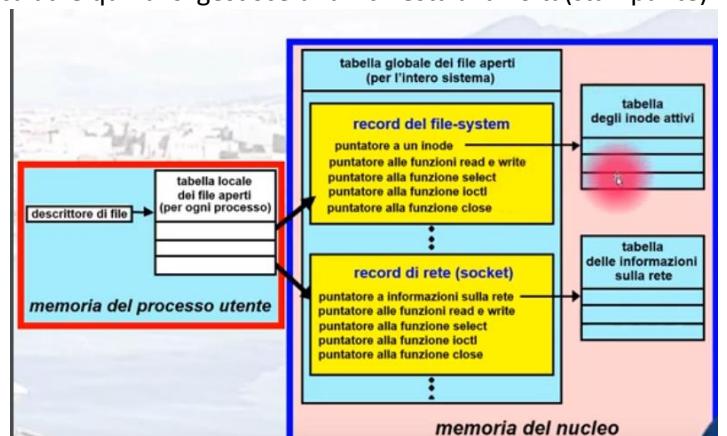
STRUTTURE DATI DEL NUCLEO

Il nucleo salva lo stato delle componenti coinvolti nelle operazioni di I/O. Si usano metodi orientati a oggetti.

Esempio a destra uso una tabella che accede a strutture dei file system o a strutture della rete (socket) in cui ho le tabelle delle risorse tenute attive.

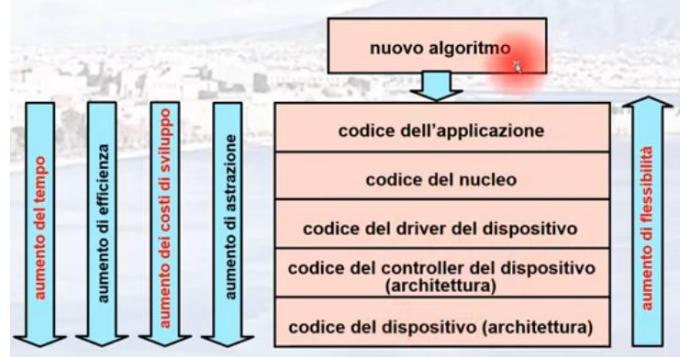


struttura streaming lettura dati da un disco (molto complesso):



Si consideri la lettura di un file da un'unità a disco:

- Si determina il dispositivo che detiene il file
- Si traduce il nome nella rappresentazione del dispositivo
- Si trasferiscono i dati dal disco al buffer



Le prestazioni dell'I/O non è tra i fattori critici per la gestione del sistema, una delle problematiche sono le copie di dati **sincronismo cache, traffico di rete**.

Per migliorare le prestazioni posso ridurre le copie dei dati, ridurre le interruzioni tramite grandi trasferimenti di dati, l'uso di controller intelligenti **DMA**.

Struttura del disco

Quando si trasferiscono dati sul disco, si scrive e si legge su blocchi e non singolo byte, sono considerati un unico vettore di blocchi logici che parte dall'esterno verso l'interno.

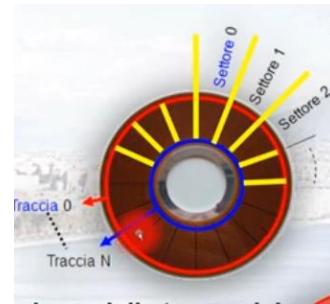
Hard disk è un dispositivo meccanico più lento rispetto SSD elettronico. Dischi ruotano a velocità elevate 10mila 15mila RPM.

Sono più dischi uno sopra l'altro e sopra vi è la testina, il tempo della testina per arrivare alla traccia desiderata **seek time**, l'attesa per cui la testina raggiunge il settore esatto nella traccia è detto **latency time**, calamite al neodimio fanno girare il disco. + rpm – latency. Gerarchia: **inboard memory(sulla scheda madre)** ci stanno registri cache e RAM, **outboard cd dvd hdd e ssd, offline storage nastro magnetico**.

SSD disco a stato solido

HDD	SSD
Basso	COSTO
Alta	Alto
Alta	CAPACITÀ
Bassa	DELICATEZZA
Bassa	VELOCITÀ
	Alta

Su ogni piatto possiamo disegnare dei cerchi concentrici dette **tracce di memorizzazione** e la riesce a polarizzare questa sostanza dei piatti e i vari 0 e 1. **L'attuatore** e l'asse che mantiene le a pettine. Questi cerchi vengono divisi in **settori**. A di diametro delle stesse tracce sui vari dischi ottengo **cilindro**, è un insieme di tracce con lo stesso raggio. queste coordinate posso scrivere incidere dati su HDD. Altri esempi cd-rom e floppydisk. CD usa una spirale e non una struttura come I HDD o Floppy disk. I CD si leggono dal centro in poi, quindi inizialmente il cd gira più velocemente più si allarga la spirale più rallenta. Negli HDD la velocità è costante.

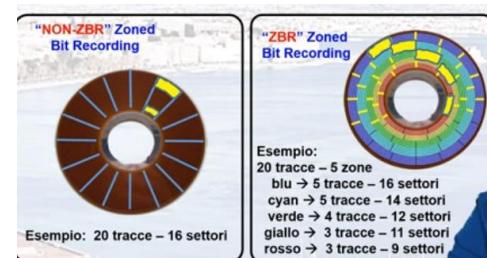


testina
leggere
testine
parita
il
Con

Quindi nel disco vi è una **Costant Linear Velocity CLV**, nell HDD e Floppy vi è un **Constant Angular Velocity CAV**.

DISCHI Z CAV

È una strategia **Zoned Bit Recording ZBR** dato che l'hard disk è composto da un sostanza ferromagnetica uniforme nel disco, quindi un traccia più interna ha un lunghezza minore e quella più esterna maggiore , quindi posso sfruttare questa maggiore percorrenza, dividendo la parte più esterna in numero di tracce superiore rispetto a quella interna. Questo mi permette di scrivere più dati sulla parte esterna.



Partizionamento di un disco

Diverse zone congrue tra di loro chiamate **partizioni**, conviene avere partizioni che abbiano tracce contigue/adiacenti tra di loro per evitare che la testina vado troppo su e giù. Per questo motivo il SO viene installato sulla parte più esterna poiche alla stessa velocità può elaborare un maggiore numero di dati. Conviene fare la **DEFrag deframmentazione** quindi ricomporre tutti i file system sparpagliati e compattarli all'esterno.



Dopo il defrag la testina si muoverà su e giù molto più poco quindi leggo il file più velocemente.

SCHEDULING DELL HARDDISK

Algoritmo dell'ascensore: stima della percorrenza dell'ascensore ideale,

l'ascensore corrisponde alla testina. Questo comporta sapere a priori già le richieste , quindi posso ottimizzare dando delle priorità. Gli algoritmi di scheduling sono **FCFS(SSD)** già spiegato, **SSTF(shortest seek time first)** (problema di starvation, si usa aging) prima le tracce più vicine tra loro rispetto alla testina, **SCAN(tocca le estremità 0 e 200, non soffre di starvation)** (problema, nelle zone dove sono appena passato è meno probabile che ci siano nuove richieste, la cosa migliore sarebbe non tornare all'indietro ma ripartire da 0 , tergilicristallo che sposta sempre da sinistra e va verso destra) il braccetto fa una scansione prima verso il centro e poi verso l'esterno, da un estremo all'altro(tergilicristalli). Esercizi vedi video lezione **SO14**.

In questo caso uso il **C-SCAN circular** e quindi faccio un salto, soffre di una malformazione, in ogni caso percorro il cilindro fino ad arrivare a toccare l'estremità seppur non ci sono richieste. Si risolve con il **C-LOOK** guardo se ci sono processi fino all'estremità se non ci sono faccio il salto.

Quale scegliere? SSTF è il più comune scan e c scan dipende dal contesto.

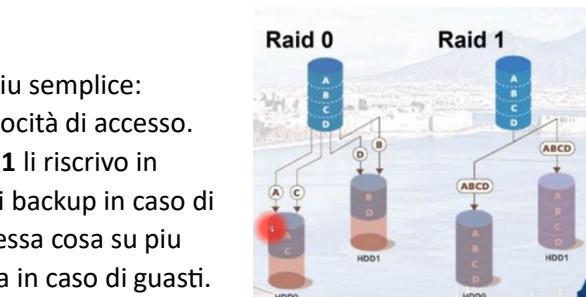
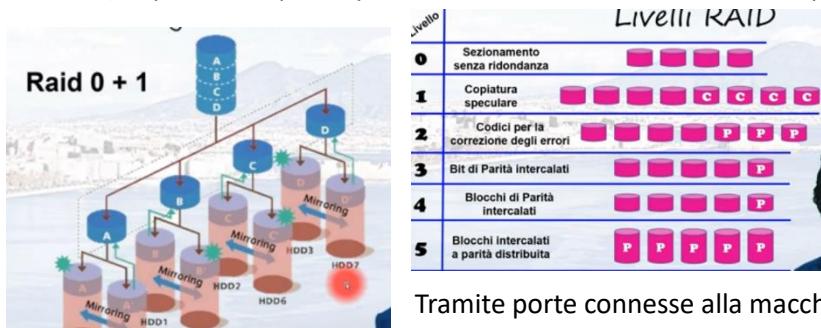
GESTIONE DELL'UNITÀ DISCO

Un disco nuovo viene prima partizionato(dividere in gruppi) e poi formattato(creare un file system) il blocco di avviamento boot block che avvia il sistema ed è caricato dal **bootstrap loader** memorizzato nella rom. In windows abbiamo MBR **Master Boot Record** che ha tabella delle partizioni e codice di avviamento e permette di gestire l'ordine avviamento delle partizioni.

L'area di avvicendamento swap space: la memoria virtuale usa lo spazio dei dischi come estensione della memoria centrale(RAM), può trovarsi all'interno dei file system oppure in una partizione separata del disco. Questa partizione si usa per caricare e scaricare pagine nella memoria centrale.

DISCHI RAID (Redundant Array of Independent Disk)

Fusione tra più hard disk , ogni combinazione ha un nome, **livello 0** più semplice: distribuisco automaticamente dei dati su più dischi aumentando la velocità di accesso. svantaggio non è tollerante ai guasti se una parte si corrompe , **livello 1** li riscrivo in parallelo su entrambi i dischi, la velocità è la stessa ma ho una copia di backup in caso di guasto. **Posso combinare entrambi i raid. Con il mirroring**(scrivo la stessa cosa su più harddisk) in parallelo, quadruplico la velocità di accesso è ho una copia in caso di guasti.



attraverso la ridondanza posso correggere gli errori.

La memoria dual channel sulla ram assomiglia al raid 0.

CONNESSIONE DISCHI

Tramite porte connesse alla macchina (SATA)**host attached storage** oppure con un sistema distribuito NAS network attached storage(memoria secondaria connessa alla rete) . **LA MEMORIA TERZIARIA SPESO OFFLINE**(CD O DISCHI RIMOVIBILI OTTICI)



WORM dischi incisi con un laser, NASTRI fanno schifo per l'accesso diretto.I compiti del **SO** ha il compito di astrarre e uniformare, la velocità di memoria terziaria non è critico, la latenza di accesso, l'affidabilità i dischi magnetici sono più fragili dei dischi, costi dell' hard disk calati nel tempo.