Sommario

Sistemi Operativi

Collocato al terzo livello Il sistema operativo è quello strato software che avvolge la macchina che permette l'utilizzo ottimizzato ed astratto delle risorse che la compongono (ovvero gestisce le risorse del hardware). grazie a ciò i moderni sistemi operativi sono in grado di eseguire più programmi contemporaneamente inoltre tra i suoi compiti troviamo:

1. il tenere traccia di quali processi stanno utilizzando quali risorse
2. garantire che ogni processo abbia accesso alle risorse di cui dispone di autorizzare
3. gestire i conflitti derivanti dall'accesso condivise (regione critica)

Struttura di un sistema operativo

esistono 6 tipi di sistemi operativi:

1) **sistemi monolitici**
   - è l'approccio più comune, viene eseguito un unico programma binario che ha al suo interno delle procedure collegate tra loro
   - la presenza di migliaia di procedure rende il sistema di difficile comprensione
2) **sistemi a livello**
   - il sistema operativo a livelli è costituito a livelli uno sotto quello sottostante
   - **multics** è una struttura a livelli a forma di anelli dove gli anelli più interni hanno maggiori privilegi rispetto a quelli esterni
3) **sistemi a microkernel**
   - il sistema operativo a microkernel sono utilizzati in quelle applicazioni mission critical (business) che hanno requisiti di alta affidabilità
   - inizialmente tutti i livelli erano nel kernel ma con approccio a strati si poteva scegliere di definire il confine tra kernel e utente.
   - il lato negativo e che il difetto di un codice dipende dalla dimensione del modulo e dalla sua età
   - **NB:** sia nei sistemi a livelli che nel microkernel si cerca di avere lo spazio occupato dal kernel più piccolo possibili in quanto un bug nel kernel può causare danni molto gravi
4) **sistemi client-server**
   - una piccola variante del microkernel è nel distinguere due classi di processi:
     1. i server che forniscono un servizio
     2. i clienti che utilizzano i servizi
   - il livello più basso è il microkernel
   - la comunicazione tra server e clienti avviene tramite scambio di messaggi
5) **sistemi a macchina virtuale**
   - l'idea della virtualizzazione è stata ignorata fino a pochi anni fà
   - ad oggi anche gli utenti finali possono eseguire due o più sistemi operativi contemporaneamente sulla stessa macchina
6) **sistemi exokernel**
   - il suo compito è quello di allocare alle macchine virtuali le risorse e di controllare i tentativi di impiego, in modo che ogni virtual machine utilizzi le proprie risorse

le api Windows

Windows e UNIX hanno un modello di programmazione diverso: un programma Windows è guidato dagli eventi mentre un programma UNIX richiede servizi tramite le chiamate di

sistemi (anche Windows ha le chiamate di sistema). le procedure api win32 servono ai programmatori per ottenere i servizi del sistema operativo

il linguaggio C

I sistemi operativi sono scritti in c a causa dei puntatori che sono una variabile che contiene l'indirizzo di memoria di un'altra variabile o l'inizio di una struttura dati (riducendo lo spreco di memoria è i tempi di esecuzione). i programmatori infatti utilizzano l'espressione "la variabile che punta a"

Header file

- i file di intestazione (o header) hanno estensione (.H) e contengono dichiarazioni, definizioni e macro, quest'ultima è dichiarata attraverso la parola chiave **#define**
- un file sorgente **C** ha gestione (.C) e può includere uno o più file di intestazione che utilizzano la direttiva **#include**

Kernel

costituisce il nucleo o core di un sistema operativo, ovvero il software che fornisce un accesso sicuro e controllato dell'hardware ai processi in esecuzione sul computer. (ultimo livello software prima del hardware)

User mode

E la parte dedicata all'utente e ha accesso esclusivo ad un sottoinsieme delle risorse della macchina

Exokernel

il compito è di allocare le risorse alle macchine virtuali e controllarne i tentativi di utilizzo,per evitare che una macchina tenti di accedere alle risorse di un'altra

costituisce il nucleo o core di un sistema operativo, ovvero il software che fornisce un accesso sicuro e controllato dell'hardware ai processi in esecuzione sul computer. (ultimo livello software prima del hardware). Esso ha accesso tutte le risorse dell'hardware

Microkernel

Confine utente kernel

inizialmente tutti i livelli erano kernel ma con l'approccio a strati si poteva scegliere di definire il confine tra kernel e utente

Processo

Il processo è una istanza del programma oppure un programma in esecuzione. I processi competono per le risorse perciò sono divisi in diverse priorità secondo il tipo di scheduling Essi si trovano nello spazio utente

Creazione di un processo

1) Inizializzazione del sistema
2) System call create
3) Richiesta utente
4) Inizio job in modalità batch

Stati del processo

Pronto, in esecuzione bloccato

Regione critica

regione critica, è una porzione di codice che accede a una risorsa condivisa tra più processi in competizione

Race condition

si verifica quando due o più processi tentano di scrivere/leggere un qualche dato condiviso,ed il risultato finale dipende dall'ordine in cui vengono eseguiti i processi

Thread

Sono sotto processi o segmenti di processi i quali esistono dentro i processi e condividono le risorse tra di loro (spazi di indirizzamento, variabili globali, file aperti, figli sotto thread). Essi non condividono PC (program counter), registri, stato,stack. i Thread non hanno protezioni poiché non competono per le risorse. Possono essere collocati sia nello spazio utente che nel kernel o un ibrido dei due

Perché usare i Thread

1) Semplifica i programmi
2) Non vanno in competizione
3) Possono lavorare in modo asincrono
4) Sono facili da creare e da eliminare

5) Le CPU moderne supportano i multithreading

Differenza tra policy e meccanismo

L'algoritmo di scheduling deve essere parametrico ma i parametri possono essere forniti da processi utente. questo facilità lo scheduling in caso di processi con gerarchie diverse

Deadlock

indica una situazione in cui due o più processi o azioni si bloccano a vicenda, aspettando che uno esegua una certa azione (es. rilasciare il controllo su una risorsa come un file, una porta input/output ecc.) che serve all'altro e viceversa.

Condizioni necessarie a un Deadlock

In un *deadlock* si verificano sempre queste condizioni, dette anche di Havender:

1. Mutua esclusione: almeno una delle risorse del sistema deve essere 'non condivisibile' (ossia usata da un processo alla volta oppure libera).
2. Accumulo incrementale: i processi che possiedono almeno una risorsa devono attendere prima di richiederne altre (già allocate ad altri processi).
3. Impossibilità di prelazione: solo il processo che detiene la risorsa può rilasciare.
4. Attesa circolare: esiste un gruppo di processi $\{P_0,P_1,...,P_n\}$ per cui P_0 è in attesa per una risorsa occupata da P_1, P_1 per una risorsa di P_2, ecc. P_n per una risorsa di P_0.

Semafori

Idea alla base dei semafori e di contare un numero di risvegli prefissati al processo.

Con **down()** o Il semaforo viene decrementato. Se, dopo il decremento, il semaforo ha un valore negativo, il task viene sospeso e accodato, in attesa di essere riattivato da un altro task.

Con **up()** Il semaforo viene incrementato.Se uno o più processi erano sospesi sul semaforo, uno di essi verrà scelto(es. a caso)per poter completare la sua **down.**

Semaforo a mutua esclusiva

E un semaforo che fa in modo che un solo processo per volta entra nella sezione critica.

Mutex

Sono una versione semplificata dei semaforo, possono assumere solo due stati locked (es:0) o unlocked (es:1)

Istruzione TSL

l’istruzione TSL(Test and Set Lock) funziona così:
TSL RX,lock
mette il contenuto della parola di memoria lock nel registro RX, e poi memorizza un valore diverso da zero all’indirizzo di memoria lock
La CPU che esegue TSL blocca il bus di memoria per impedire ad altre CPU di accedere alla memoria finchè non ha finito

Monitor

È un insieme di procedure, variabili e dati strutturati tutti raggruppati in un particolare tipo di modulo o pacchetto.

Il monitor può essere utilizzato da due o più processi o thread per rendere mutuamente esclusivo l'accesso a risorse condivise. Il vantaggio nell'utilizzo del monitor deriva dal fatto che non si deve codificare esplicitamente alcun meccanismo per realizzare la mutua esclusione, giacché il monitor permette che un solo processo sia attivo al suo interno

Busy waiting

comportamento che fa si che la CPU controlli sempre una variabile; molto svantaggioso poichè spreca tempo di CPU

Mutua esclusiva con busy waiting

Essa è possibile utilizzando il busy waiting. La mutua esclusiva è quando mentre un processo e nella regione critica gli altri processi devono aspettare.

Scambio di messaggi

Questo metodo usa due primitive, ***send e receive***, che sono system call

la prima spedisce un messaggio ad una determinata destinazione, la seconda riceve il messaggio.

Esse possono essere bloccanti o non, a seconda della tipologia di primitiva utilizzata(***sincrona-asincrona***)

Es.

send(destinazione,&messaggio)

receive(sorgente,&messaggio)

Barriere

meccanismo di sincronizzazione per gruppi di processi.

le applicazioni sono divise in fasi, e **nessun processo può andare avanti fino a che tutti non sono pronti *(hanno raggiunto la barriera)***

quando un processo raggiunge la **barriera**, viene bloccato finché tutti non l'hanno raggiunta

La soluzione di Peterson

G.L. Peterson scoprì un algoritmo per garantire la mutua esclusione. Esso consiste in 2 procedure scritte in linguaggio ANSI C(cioè con prototipo per procedure)

L'algoritmo assicura la corretta ***sincronizzazione***, impedendo lo stallo (***deadlock***) ed assicurando che soltanto un processo alla volta possa eseguire una ***sezione critica***

Inizio:

1)

   prima di accedere alla regione critica, ciascun processo richiama il metodo ***entra_nella_regione*** con il proprio numero, **0** o **1** come parametri(indicano se il processo deve aspettare o no)

2)

   dopo aver finito di lavorare con le variabili condivise,il processo chiama ***lascia_la_regione***, per indicare che ha finito e permettere ad un'altro processo di entrare.

3)

   inizialmente, nessun processo è nella sez. critica,poi il proc.0 chiama ***entra_nella_regione*** e indica il suo interesse,impostando il proprio elem, nel vettore e mettendo *turno* a 0:Se il proc. 1 chiamasse ***entra_nella_regione*** ora, rimarrebbe in attesa fino a quando *interessato[0]* non diventa **FALSE**

4)

   se entrambi i processi chiamano ***entra_nella_regione*** contemporaneamente:

   entrambi memorizzavano il loro n.di processo nella variabile *turno*, e qualunque cosa venga memorizzata, il secondo valore conta, il primo verrà sovrascritto

Scheduling

Lo scheduler dei processi è quel componente del sistema operativo che si occupa di decidere quale processo va mandato in esecuzione. Nell'ambito della multiprogrammazione

è pensato per mantenere la CPU occupata il più possibile, ad esempio avviando un processo mentre un altro è in attesa del completamento di un'operazione di I/O.

Gli algoritmi di scheduling possono dividersi in 2 categorie

Non preemptive

viene scelto un processo e semplicemente viene lasciato in esecuzione finché non viene bloccato, oppure finchè quest'ultimo non rilascia volontariamente la CPU

Preemptive

viene scelto un processo e lasciato in esecuzione per un massimo di tempo prestabilito.

se questo è ancora in esecuzione allo scadere del tempo, viene sospeso e ne viene scelto un altro

Preemption

Il pre-rilascio o prelazione (in inglese preemption) è, in informatica, l'operazione in cui un processo viene temporaneamente interrotto e portato al di fuori della CPU, senza alcuna cooperazione da parte del processo stesso, al fine di permettere l'esecuzione di un altro processo. Il processo interrotto viene in genere ripristinato una volta che quello a priorità maggiore ha terminato il suo lavoro (a meno che non ci siano ancora altri processi a priorità più alta, nel qual caso il meccanismo si ripete). Tale scambio è noto come context switch o cambiamento di contesto. Il pre-rilascio può avvenire tramite uno scheduler, che ha il compito di interrompere e/o ripristinare i processi presenti nel sistema operativo a seconda del loro stato; in tal caso si parla di preemptive scheduling (o scheduling con prerilascio).

Algoritmi di scheduling nei sistemi batch

Nei sistemi batch non ci sono utenti impazienti in attesa di una risposta rapida al loro breve quesito. questi sistemi sono ancora usati nel mondo bancario, assicurativo, per il calcolo delle buste paga, l'inventario, calcolo degli interessi e gestione dei reclami.

1) First-come first-served

- no preemptive
- il primo che arriva è il primo ad essere servito
- la CPU è assegnata ai processi in ordine di arrivo
- c'è unica coda dei processi pronti
- in un sistema con processi eterogenei per tempo di esecuzione potrebbero rallentare i processi veloci in assenza di preemption

2) Shortest job first

- no preemption
- i job entrano nella coda in ordine ma lo scheduling sceglie quello che ha il tempo di esecuzione più breve

- i tempi di esecuzione dei processi sono noti in anticipo
- questo algoritmo è ottimale solo se i job sono tutti disponibili al momento della scelta

3) Shortest remaining time next

- si preemption
- lo scheduler sceglie il processo a cui manca meno tempo al termine dell' esecuzione
- se arriva un nuovo job che impiega meno tempo per terminare rispetto al job corrente quest'ultimo è sospeso e viene eseguito il nuovo arrivato

Algoritmi di scheduling nei sistemi interattivi

Nei sistemi interattivi la preemption è essenzialmente per evitare che un processo utente monopolizzi la CPU bloccando gli altri utenti. la preemption è essenziale anche per i server che gestiscono normalmente molti utenti remoti

1) Scheduling round-robin

- ad ogni processo è assegnato un "quantum" di tempo di CPU per l'esecuzione allo scadere del quale viene interrotto e si passa, con in modo circolare e paritario al successivo processo
- la scelta del "quantum" è un fattore critico poiché se troppo breve vi sono troppe interruzioni dei processi che causano un **overhead** mentre un valore troppo alto provoca una coda di attesa eccessiva

2) Scheduling con priorità

- come nel round- robin si assegna un quantum di tempo ma in più si assegna una priorità a ciascun processo e lo scheduling tiene conto della priorità nel momento in cui deve scegliere il processo da mandare in esecuzione
- per evitare che i processi ad alta priorità girino per un tempo indeterminato lo scheduling può:
    1. diminuire la priorità del processo in esecuzione ad ogni ciclo di clock
    2. utilizza un quantum di tempo massimo di CPU

3) Shortest process next

- come si cerca di come nell'algoritmo shortest job next (usato nei sistemi batch) di ordinare i job dal minor tempo di esecuzione al maggior tempo di esecuzione
- la difficoltà risiede nel calcolare i tempi di esecuzione i quali nei sistemi interattivi non sono sempre li stessi
- un approccio è di misurare i tempi di esecuzione utilizzati per le stime di successive attraverso una media pesata tenendo conto che le misure più recenti sono più attendibili rispetto a quelle passate (**aging**)

4) Scheduling garantito

- lo scheduling fa promesse sulle performance reali dei processi utenti per poi mantenerle

- se ci sono **n** utenti collegati dovrebbero ricercare **1/n** del tempo della CPU
- per mantenere la questa promessa il sistema memorizza quanta CPU ogni processo ha avuto fin dalla sua creazione e calcolare il il rapporto tra avuto e promesso
- se un qualsiasi processo ha un valore più basso di tale rapporto e il successivo candidato ad essere messo in esecuzione fino a quando il suo rapporto non è il più basso tra quelli pronti

5) Scheduling a lotteria

- si assegna un biglietto della lotteria ai vari processi
- ogni volta che lo scheduling deve fare una scelta pesca a caso un biglietto e il processo che ha quel biglietto si aggiudica la risorsa
- i processi che cooperano, se lo desiderano possono scambiarsi i biglietti che detengono per alterare le priorità di esecuzione
- può essere utilizzato per problemi di difficile che sono di difficile risoluzione es: lo streaming video tanti sono i biglietti quanti sono i frame automatici della CPU
- di solito si utilizzano 100 biglietti distribuiti tra tutti i processi
- 20 biglietti = 20% di possibilità

6) Scheduling fair-share (quota equa)

- schedulato per proprio conto indipendentemente dal suo proprietario
- dove se un utente avvia 4 processi (A,B,C e D) ed un altro utente un solo processo (E) se il primo utente si prenderà l'80% del tempo della CPU
- per evitare questo situazione prima di qualsiasi schedulazione di controllo il proprietario del processo viene assegnato in base agli utenti collegati una porzione della CPU (in questo caso spartita al 50%)

Algoritmi di scheduling nei sistemi real-time

Nei sistemi real-time è essenziale la tempistica, la preemption potrebbe non essere necessaria perché i processi saranno che devono dare dei risultati in tempi brevi quindi il tempo gioca un ruolo essenziale e si suddividono in:

1) hard real-time
    - le scadenze devono essere sempre rispettate
2) soft real-time
    - la scadenza deve essere rispettata, ma comunque tollerabile

con eventi:

1) **periodici**
    - quando si verificano a intervalli di tempo regolari
2) **non periodici**
    - quando si verificano in modo imprevedibile

In un sistema soft real-time con eventi periodici è sostenibile se riesce a far fronte agli eventi stessi. ovvero se riesce a trattare un evento prima che ne arrivi un altro. se ci sono **n** eventi periodici e ogni evento avviene con una frequenza pari **1/P** supponendo che ciascun evento

richiede **C** secondi di tempo di CPU per gestirlo allora il sistema è in grado di reggere il carico se e solo se

$$\sum_{i=0}^{n} = \frac{C_i}{P_i} < 1$$

inoltre lo scheduling per i sistemi real-time può essere:

1) scheduling statico
    - le decisioni di scheduling sono prese dal algoritmo prima che il sistema inizi a funzionare. questo è applicabile solo quando ci sono informazioni disponibili in anticipo riguardo il lavoro da svolgere e le scadenze da rispettare
2) scheduling dinamico
    - le decisioni di scheduling sono prese dal algoritmo in fase di esecuzione del sistema. qui non ha bisogno di conoscere in anticipo alcuna informazione sul compito da svolgere

Scheduling dei thread

differisce sostanzialmente a seconda che siano supportati thread a livello utente o livello kernel

livello utente:

Scheduling Round-Robin e a priorità sono quelli più comuni.

il solo vincolo è ***l'assenza*** di clock, per interrompere un processo in esecuzione da troppo tempo

livello kernel:

il kernel sceglie un thread particolare da eseguire, non tiene in considerazione a quale processo appartiene(**anche se potrebbe farlo**)

una ***differenza sostanziale*** tra thread a livello utente e kernel sono le prestazioni:

fare un context-switch tra thread *livello utente* richiede *poche istruzioni macchina*,mentre con i thread a *livello kernel ci vuole un completo context-switch,modificando la mappa di memoria e invalidando la cache*

Gestione della memoria

Spazio degli indirizzi

nei sistemi operativi possono risiedere più programmi in memoria allo stesso tempo ma richiedono una protezione oppure si può virtualizzare la memoria questo consente al processo di vedere uno spazio di indirizzo unico anche se parte dei dati sono collocati nella memoria principale (Ram) mentre altri sui dischi (Rom)

Swapping

Si prende un processo e lo si esegue per un certo tempo, quando ha terminato si salva sul disco questo tramite la creazione di buchi nella memoria che permettono inserimento e/o la rimozione di processi dalla memoria. I processi inattivi sono memorizzati sul disco in modo da non occupare la memoria quando non sono in esecuzione. si tratta di una operazione onerosa per la CPU

Bitmap

La bitmap è uno dei due modi per tenere traccia dell'utilizzo della memoria.

la memoria viene divisa in unità di allocazione che possono essere lunghe poche parole o arrivare a qualche kilobyte. A ciascuna unità viene associato un bit della bitmap, se è zero unità è libera se è uno è occupata

Liste collegate

E altro metodo per gestire la memoria.

Si mantiene una lista concatenata dei segmenti di memoria liberi, dove un segmento è un processo oppure un buco tra due processi.

Ciascun elemento specifica un processo (p) o un buco(h o L) Si possono usare diversi algoritmi (ipotizzando di sapere quanti blocchi occupano)

Algoritmi

1) **First fit**

Scorre la lista finché non trova uno spazio libero abbastanza grande da contenere il processo

2) **Next fit**

E una variante del first fit funziona allo stesso modo del first fit ma tiene traccia della posizione dove ha trovato ultimo spazio libero e la prossima volta che è chiamato a cercare uno spazio libero avvia la ricerca dalla posizione dove si era interrotto ultima volta

3) **Best fit**

Scorre tutta la lista alla ricerca del più piccolo spazio che riesce a contenere il processo richiesto

4) **Worst fit**

Funziona con una logica inversa al **best fit** cerca di prendere il più grande buco disponibile

5) **Quick fit**

Si basa sul trovare uno spazio libero della dimensione desiderata (es:4 KB, 8 KB)

Memoria virtuale

Consente ai programmi di poter essere eseguiti anche quando sono parzialmente nella memoria principale. L'idea alla base è che la dimensione di programma.dati.stack può eccedere la dimensione della memoria fisica per essi disponibile. IL SO mantiene in memoria le parti che sono in uso in un certo momento, mentre le altre vengono mantenute sul disco.

L'idea alla base della memoria virtuale è che ogni programma ha il proprio spazio degli indirizzi personali, suddiviso in pezzi chiamati pagine. Ogni pagina è un intervallo di indirizzi contigui. Queste pagine sono mappate sulla memoria fisica, ma non tutte le pagine devono stare nella memoria fisica per eseguire il programma.

Registri base e registri limite

Questa soluzione si basa su una versione particolarmente semplice della rilocazione dinamica. Quello che fa è mappare lo spazio degli indirizzi di ogni processo su di una parte diversa di memoria fisica in un modo semplice. La soluzione classica è di dotare ogni CPU con due registri hardware speciali, solitamente chiamati registro base e registro limite. Quando sono usati questi registri, i programmi sono caricati in posizioni di memoria consecutive dovunque vi sia spazio e senza riposizionamento durante il caricamento. Al momento dell'esecuzione di un processo, il registro base è caricato con l'indirizzo fisico dove comincia il suo programma in memoria e il registro limite è caricato con la lunghezza del programma.Ogni volta che un processo consulta la memoria, sia per prelevare un'istruzione sia per leggere o scrivere una parola dati, prima di inviare l'indirizzo sul bus di memoria l'hardware della CPU aggiunge automaticamente il valore di base all'indirizzo generato tramite il processo. Contemporaneamente controlla se l'indirizzo offerto sia uguale o maggiore del valore nel registro limite, nel cui caso è generato un errore l'accesso viene interrotto.

Paginazione

La maggior parte dei sistemi di memoria virtuale usa una tecnica chiamata paginazione o paging. Su qualsiasi computer i programmi referenziano un insieme di indirizzi di memoria.

Gli indirizzi possono essere generati usando l'indicizzazione, i registri base, i registri segmento o in altri modi, Questi indirizzi generati dal programma sono chiamati indirizzi virtuali e formano lo spazio virtuale degli indirizzi. Sui computer senza memoria virtuale, l'indirizzo virtuale è messo direttamente sul bus di memoria e provoca la lettura o la scrittura della parola della memoria fisica con lo stesso indirizzo. Quando è usata la memoria virtuale, gli indirizzi virtuali non vanno direttamente al bus di memoria, ma a una MMU (memory management unit) che mappa gli indirizzi virtuali sugli indirizzi di memoria fisica

pagina = unità dello spazio di indirizzamento virtuale che si trova nella memoria virtuale.

page fault

Si verifica quando la MMU si accorge che la pagina **non è mappata**, e fa in modo che la CPU esegua una **trap** al SO

Tabelle delle pagine

L'indirizzo virtuale è diviso in numero di pagina virtuale (i bit più significativi) e un offset (i bit meno significativi). Il numero di pagina virtuale è utilizzato come un indice nella tabella delle pagine per trovare la voce per quella pagina virtuale. Il numero di frame (se esiste) è rintracciato dalla voce della tabella delle pagine. Il numero di frame è allegato alla parte finale più significativa dell'offset, rimpiazzando il numero di pagina virtuale, per formare un indirizzo fisico che può essere inviato alla memoria. In questo modo lo scopo della tabella delle pagine è di mappare le pagine virtuali sui frame delle pagine.

MMU

Quando si usa la memoria virtuale, gli indirizzi virtuali non finiscono direttamente nel bus di memoria, ma vanno a finire nell'**unità di gestione della memoria(MMU,***Memory Management Unit*), che mappa gli indirizzi virtuali sugli indirizzi fisici

Translation lookaside buffer

è un piccolo dispositivo hardware che serve a mappare gli indirizzi virtuali sugli indirizzi fisici senza passare dalla tabella delle pagine. Questo dispositivo viene chiamato **TLB**(anche **memoria associativa**)

SI trova dentro la MMU e contiene un piccolo numero di elementi ,fino a 64:

ciascun elemento contiene informazioni che riguardano la pagina, in particolare:

n. pagina virtuale

bit che viene messo ad 1 se la p è stata modificata

il codice di protezione

pagina fisica in cui si trova la pagina virtuale

Il modo normale per gestire una TLB miss è quello di andare nella tabella delle pagine e di eseguire operazioni di indicizzazione per localizzare la pagina riferita

Tabella delle pagine invertite

In questa tabella è presente un elemento per ogni pagina fisica nella memoria reale,invece che un elemento per pagina nello spazio di indirizzamento virtuale..
La traduzione da virtuale a fisico diventa molto più difficile. quando il processo n riferisce la pagina p, l'hardware non può trovare la pagina fisica utilizzando p come indice nella tabella delle pagine, dovrà invece ricercare nell'intera tabella delle pagine invertite un'elemento del tipo (n,p).Questa ricerca deve essere fatta per ogni riferimento in memoria, rendendo il calcolatore molto lento
Il modo per uscire da questo dilemma è di utilizzare il TLB

Algoritmi di sostituzione delle pagine

Dopo una page fault, il SO deve scegliere una pagina da rimuovere dalla memoria, per fare spazio alla pagina che deve essere caricata.

Algoritmo ottimo di sostituzione delle pagine

é facile da descrivere ma impossibile da realizzare.
Funziona così:
nel momento che si verifica un fault di pagina,in memoria si trova un certo numero di pagine e a una di queste si farà riferimento proprio nella prossima istruzione.
Ogni pagina può essere **etichettata** con il numero di istruzioni che saranno eseguite prima che a quella pagina si faccia riferimento per la prima volta

alla fine, *semplicemente si dovrà rimuovere la pagina con l'etichetta più alta*

L'unico problema è che è irrealizzabile: al momento del page fault,il SO non ha nessun modo di sapere quando si farà riferimento a ciascuna delle pagine.

Not recently used (NRU)

Ci sono 2 bit che controllano lo stato della pagina:

R - pagina riferita

M - pagina modificata

il bit R viene messo ad 1 ogni volta che la pagina viene riferita;M viene messo ad 1 quando la pagina viene modificata

Quando avviene un page fault, il SO ispeziona tutte le pagine e le divide in 4 categorie:

1) **Classe 0(non usata,non modificata) (R=0,M=0)**
2) **Classe 1(non usata,modificata) (R=0,M=1)**
3) **Classe 2(usata,non modificata) (R=1,M=0)**

4) **Classe 3(usata,modificata) (R=1,M=1)**

Il NRU rimuove una pagina a caso della classe non vuota di numero più basso

Meglio rimuovere una pagina modificata a cui non si è fatto riferimento per un ciclo di clock**(R=0,M=1),** piuttosto che una pagina pulita che viene usata frequentemente**(R=1,M=0)**

First-in first-out (FIFO)

Il SO tiene una lista(**stack**) di tutte le pagine correttamente in memoria,dove la pagina in testa è la più vecchia e la pagina in coda è la più giovane.
Quando si verifica un fault di pagina,la pagina in testa viene rimossa e la pagina in coda viene aggiunta allo stack.
FIFO non viene quasi mai utilizzato nella sua forma più pura

Second Chance

è **una semplice modifica** dell algoritmo FIFO, che controlla il bit R della pagina più vecchia.

Se è a 0, la pagina è sia vecchia che non usata,e può essere rimpiazzata immediatamente

Se vale 1, viene posto a 0,la pagina viene sposata alla fine della coda dello stack, ed il suo tempo di caricamento viene aggiornato come se la pagina fosse appena arrivata in memoria

Da questo il nome **seconda opportunità**

Potrebbe risultare abbastanza efficiente

Clock

vengono organizzate le pagine in una lista organizzata a forma di orologio, una lancetta punta alla pagina più vecchia

Quando si verifica un page fault, viene controllata la pagina puntata dalla lancetta.

Se il bit R=0, la pagina viene scaricata e la pagina nuova viene inserita al suo posto, e la lancetta viene spostata di uno in avanti

Se il bit R=1,viene messo a 0 e la lancetta viene spostata di uno in avanti

Last recently used (LRU)

quando si verifica un page fault, *viene rimossa la pagina che non viene usata da più tempo*.

Questo algoritmo è una buona approssimazione dell'algoritmo ottimo, l'unica differenza è che si può effettivamente creare

L'implementazione di questo algoritmo non è economica.

Uno modo per implementare questo algoritmo tramite hardware è quello di dotare l'hardware di un contatore C di 64 bi, che viene incrementato automaticamente ad ogni istruzione

Quando si verifica un page fault, il SO esamina tutti i contatori nella tabella delle pagine per trovare quella con il valore più basso,la p corrispondente è quella meno utilizzata e di conseguenza può essere rimossa.

Per una macchina con n pagine fisiche,l'hardware LRU può tenere una matrice n*n bit,inizialmente tutti a 0;ogni volta che viene riferita la pagina k, vengono impostati prima i bit della riga tutti a 1, e i bit della colonna tutti a 0.

Alla fine della sequenza di pagine da verificare, la pagina con il valore di riga minore viene rimossa

Not Frequently Used(NFU)

è una variante del LRU,che può essere implementata via software.

l'algoritmo richiede un contatore associato ad ogni pagina, inizialmente a 0.
ad ogni interruzione di clock,il SO esamina tutte le pagine in memoria e per ogni pagina il bit R viene sommato al contatore.
Quando si verifica un page fault, viene scelta la pagina con il contatore più basso
C'è il rischio che il SO rimuova le po utili invece che p che non vengono più utilizzate

Modifica del NFU

Un'alternativa è l'algoritmo di invecchiamento(**aging**),
La modifica avviene in due fasi:

1) Prima di sommare il bit R al contatore, quest'ultimo viene shiftato a destra di un bit
2) il bit R viene sommato al bit più a sinistra invece che quello a destra.

Working Set

è l'insieme delle pagine che vengono utilizzate correntemente da un processo(chiamato anche **insieme di lavoro**); se la memoria disponibile è troppo piccola per poter contenere tutto il working set, il processo causerà molti page fault e girerà molto lentamente.
un programma che genera fault di pagina dopo poche istruzioni viene detto in **situazione di trashing**
Ogni sistema cerca di tener traccia degli insiemi di lavoro dei processi, e si assicurano che l'insieme di lavoro sia caricato in memoria. Questo approccio è detto **modello dell'insieme di lavoro.**

WSClock

è una miglioria dell'algoritmo clock, che utilizza anche i principi del working set(per questo **WSClock**)
La struttura dati è simile al clock,ma ogni elemento contiene, oltre ai bit R e M, anche il campo *Tempo ultimo utilizzo*
Funzionamento:

1) Se la pagina puntata ha bit R=1, viene posto a 0 e si va avanti di 1
2) se la pagina ha bit R=0,età>età max e M=0, la p non è nel working set e quindi può essere rimossa

3) Se il bit R=0,età>età max e M = 1, la pagina non può essere rimossa, e viene schedulata una scrittura sul disco
4) Se la lancetta torna alla pos di partenza:
    a) è stata schedulata una scrittura
    b) non è stato schedulata una scrittura

Se a, allora viene rimossa la prima pagina non modificata che si incontra
Se b, allora viene selezionata la pagina corrente e rimossa.

Differenze tra algoritmi locali e globali

Uno dei grossi problemi associati alla scelta dell'algoritmo di sostituzione delle pagine , sta nel decidere come deve essere allocata la memoria fra i processi in competizione
Ci sono 2 tipi di algoritmi:

1)**Locale**
2)**Globale**

Algoritmi globali

Gli algoritmi globali sono quelli che allocano dinamicamente le pagine fisiche fra tutti i processi eseguibili;il numero di pagine fisiche varia nel tempo
Viene controllata la dimensione del Working Set tramite i bit dell'invecchiamento, ma **non** viene comunque garantita l'assenza di trashing

Un'altro metodo sarebbe quello di far cominciare ogni processo con un numero di pagina proporzionale alla dimensione della pagina,ma l'allocazione deve essere aggiornata dinamicamente
Un modo per gestirla è utilizzando l'algoritmo **PFF(**Page Fault Frequency**,frequenza dei fault di pagina**)

Algoritmi locali

Gli algoritmi locale sono quelli che assegnano ad ogni processo una quantità fissa di memoria
Se il Working Set cresce : si potrà avere del trashing
si restringe: l'algoritmo locale spreca memoria

Dimensione della pagina

la dimensione della pagina viene scelta dal sistema operativo.
ES. anche se l'hardware fosse progettato con pagine da 512 byte, il SO potrebbe facilmente trattare le pagine come pagine da 1KB
Determinare la dimensione ottima delle pagine richiede il bilanciamento di due fattori:

1)Un segmento non occupa un numero di pagina intero
2)In media, metà dell'ultima pagina sarà vuota, e lo spazio in più verrà sprecato

lo spreco di questo spazio viene detto **frammentazione interna**

Spazio-I e Spazio-D

Molti calcolatori hanno uno spazio di indirizzamento unico,che contiene sia programmi che dati.
Se è abbastanza grande okay, ma se è relativamente piccolo,si potrebbero creare dei problemi quando si dovranno inserire i valori all'interno.
Una soluzione è quella di avere 2 spazi separati, lo **spazio-I**(per gli indirizzi) e lo **spazio-D**(per i dati)
Entrambi vanno da 0 a qualche massimo,tipicamente $2^{16} - 1$ oppure $2^{32} - 1$
Ogni spazio può essere paginato in maniera indipendente dall'altro;ognuno ha la sua tabella delle pagine con la propria mappa delle pagine virtuali in pagine fisiche

Pagine condivise

Un problema relativo all'implementazione è quello della condivisione
per evitare di avere più copie della stessa pagine in memoria, risulta più efficiente **condividere** le pagine
Non tutte le pagine possono essere condivise(*pagine di testo SI,pagine di dati NO*)
La condivisione dei dati non è impossibile.
In un sistema paginato si fornisce ad ogni processo la propria tabella delle pagine,che puntano allo stesso insieme di pagine, evitando così la copiatura della pagine.
Entrambe le pagine dei dati sono mappate come READ-ONLY
Questa situazione va bene finchè un processo non modifica un valore in memoria;
se avviene, la violazione della protezione causerà una trap al SO. Verrà quindi fatta una copia della pagina, in modo che ogni processo abbia la sua copia privata.Entrambe le copie verranno messe in mod. READ-WRITE
Questo approccio viene chiamato **copy-on-write**

Memoria Condivisa Distribuita

Un'altra tecnica avanzata di gestione della memoria è l'utilizzo della **DSM(Distribuited Shared Memory)**
L'idea alla base è quella di consentire a diversi processi in rete di condividere un insieme di pagine, come un singolo spazio di indirizzamento lineare condiviso
Quando un processo riferisce ad una pagina che non è mappata in memoria,esso genera un page fault, ed il gestore del page fault localizza la macchina e le spedisce un messaggio, chiedendo di togliere la mappa alla pagina e di spedirla sulla rete.
Quando la pagina arriva, essa viene mappata e l'istruzione che ha causato il page fault viene reinizializzata

Trattamento dei page fault

Quando si verifica un page fault, avvengono i seguenti eventi :

1. l'hardware provoca una trap al SO, salvando il PC sullo stack
2. Viene fatta partire una procedure in assembler per salvare i registri generali
3. Il SO scopre del page fault e cerca di capire quale pagina virtuale è richiesta
4. Il SO controlla se l'indirizzo virtuale della pagina è un indirizzo valido e che i diritti di protezione siano compatibili con l'accesso
5. Se la pagina viene modificata,la si schedula per essere trasferita su disco e si ha un contex-switch
6. Se la pagina non è modificata,il SO cerca l'indirizzo su disco e schedula una richiesta per caricarla. IL processo è sospeso,in attesa
7. Quando la pagina arriva, le tabelle vengono aggiornate
8. L'istruzione che aveva causato il fault viene ripristinata,il PC viene ripristinato
9. Il processo che aveva causato il fault viene schedulato e il SO restituisce il controllo alla procedura assembler
10. Questa procedura recupera i registri e restituisce il controllo allo spazio utente, come se non fosse successo nulla

Segmentazione

la macchina viene dotata di tanti spazi di indirizzamento completamente indipendenti,chiamati **segmenti**
(il segmento è un'entità logica,può contenere una matrice,una procedura,uno stack ecc…;*NON* contiene un misto di variabili)

Ciascun segmento consiste di una sequenza lineare di indirizzi, che vanno da 0 ad un max;la lunghezza dei segmenti va da 0 al max;segmenti diversi possono avere lunghezze diverse
Per specificare un indirizzo nella memoria segmentata, il programma deve fornire un indirizzo a due parti:

1. **numero di segmento**
2. **indirizzo all'interno del segmento**

una memoria segmentata presenta vari vantaggi,oltre a semplificare la gestione delle strutture dati:se ciascuna procedura occupa un segmento distinto,il collegamento di procedure compilate separatamente risulta grandemente semplificato
La segmentazione facilita anche la condivisione delle procedure o dei dati
ES. le **librerie condivise**
Segmenti diversi possono avere tipi di protezione di tipo diverso;dato che avrà un solo tipo di dato, il segmento avrà la protezione adeguata al quel tipo di dato

Implementazione della segmentazione pura

L'implementazione della segmentazione pura differisce dalla paginazione per un solo motivo: le pagine hanno dimensione fissa, i segmenti no

Dopo che il sistema avrà lavorato un pò,la memoria sarà divisa in porzioni, alcune segmenti e altri buchi. Questo fenomeno viene chiamato **checkerboarding (frammentazione esterna)**

e spreca memoria. Vi si pone rimedio con la compattazione.

Segmentazione Intel Pentium

la memoria include sia paginazione che segmentazione, come nel MULTICS
Il pentium ha 16K segmenti,ciascuno dei quali può contenere 1000 miliardi di parole di 32 bit
Il cuore della memoria virtuale è dato da 2 tabelle :

1. **LDT(Local Descriptor Table,**tabella dei descrittori locali**)**
2. **GDT(Global Descriptor Table,**tabella dei descrittori globali**)**

LDT descrivono i segmenti locali a ciascun programma, mentre GDT descrive i segmenti di sistema.

Il registro CS contiene il selettore per il segmento di codice ed il registro DS quello per il segmento dei dati
Uno dei selettori dice se il segmento è globale o locale
altri 13 bit specificano il numero di elemento della LDT o GDT
2 bit hanno a che fare con la protezione
il bit 0 è proibito: serve ad indicare se quel registro di segmento è disponibile o no in quel momento
per formare **l'indirizzo lineare** si aggiunge all offset il campo(da 32 bit) *Base*

Input/Output

dispositivi input/output

i dispositivi I/O possono dividersi in 2 categorie :

1. **dispositivi a blocchi**
2. **dispositivi a caratteri()**

un dispositivo a blocchi memorizza le informazioni in blocchi di lunghezza fissa,ognuno con un proprio indirizzo. la proprietà essenziale è che i blocchi possono essere letti/scritti in maniera indipendente dagli altri.

un dispositivo a caratteri,invece, invia o riceve un flusso di caratteri senza tener conto di alcuna struttura di blocco. non è indirizzabile, e non può eseguire alcuna operazione di posizionamento

alcuni dispositivi non rientrano in queste classifiche, come i clock.

controllore dell I/O

le unità di I/O sono costituite da una parte meccanica e una elettronica.
la parte elettronica prende nome di **controllore dell I/O.**
La scheda del controllore contiene un connettore, nel quale può essere inserito il cavo proveniente dal dispositivo.
L'interfaccia tra controllore e dispositivo è a bassissimo livello
Ciò che esce realmente dal dispositivo è un flusso di bit che inizia con un **preambolo** e finisce con un **codice correttore di errore(ECC)**

I/O mappato in memoria

ogni controllore possiede dei registri che servono per comunicare con la CPU.
oltre ai registri, molti dispositivi sono dotati di un buffer dati, che può essere letto o scritto dal SO
La CPU comunica con i registri del controllore e il buffer dati in 2 modi:
nel primo approccio, ogni registro del controllore è caratterizzato da un numero **di porta I/O**
In questo schema, gli spazi di indirizzamento per l'I/O e la memoria sono divisi.
Il secondo approccio consiste nel mappare i registri del controllore nello spazio di indirizzamento della memoria centrale:
ad ogni registro del controllore viene assegnato un indirizzo di memoria unico, a cui non corrisponde nessuna parola di memoria (**I/O mappato in memoria**)

Il Pentium usa un'architettura ibrida, ovvero utilizza sia l'I/O mappato in memoria sia porte I/O

Vantaggi e svantaggi di I/O mappato in memoria

I vantaggi dell I/O mappato in memoria sono :

1. riduce l'overhead causato dalle chiamate alle procedure assembler
2. i registri sono salvati in variabili, questo rende possibile scrivere il controllore dell'I/O in linguaggio C, senza passare per l'assembler (punto 1)
3. non occorrono meccanismi di protezione speciali per impedire ai processi utente di eseguire I/O; quello che il SO deve fare è semplicemente evitare di mettere la parte di indirizzamenti contenente i registri nello spazio virtuale utente
4. ogni istruzione che fa riferimento in memoria, può fare riferimento anche ai registri dei controllori.

gli svantaggi sono :

1. ogni moderno calcolatore ha una qualche forma di memoria cache, sarebbe disastroso mettere i registri nella cache (per impedire questa situazione, l'hardware deve essere dotato di una funzionalità per la disabilitazione della cache)
2. se c'è un solo spazio di indirizzamento, tutti i moduli della memoria e tutti i device I/O devono esaminare tutti i riferimenti alla memoria e decidere a quali rispondere.

DMA(Direct Access Memory)

Per utilizzare la tecnica DMA, l'hardware deve essere provvisto del controllore DMA.
questa tecnica permette alla CPU di disinteressarsi del trasferimento dati e svolgere altre attività in parallelo
il controllore DMA è composto così:

1. contiene vari registri che possono essere letti/scritti dalla cpu(un registro di indirizzo di memoria,registro contatore di byte,uno o più registri di controllo).
2. i registri di controllo specificano la porta I/O,le direzioni di trasferimento ,l'unità di trasferimento e il numero di byte

Interrupt Rivisitati

Ogni volta che l'I/O termina, causa un interrupt, impostando un segnale sulla linea di bus che gli è stata assegnata.
Tale segnale è rilevato dal controllore che decide cosa fare
Se non ci sono altre interruzioni sospese, il controllore gestisce l'interruzione immediatamente;se invece c'è un'interruzione sospesa, il dispositivo viene momentaneamente sospeso.
Per gestire l'interruzione, il dispositivo manda sulle linee di indirizzamento un numero, che specifica quale dispositivo vuole richiamare l'attenzione

il numero mandato sulle linee di indirizzamento viene usato come indice in una tabella detto **vettore delle interruzioni,**per cercare un nuovo puntatore del PC. Questo puntatore punta all'inizio dell **ISR**
Ci sono 2 tipi di interrupt:

1. **interrupt precisi**
2. **interrupt imprecisi**

Interrupt precisi

sono tutti quegli interrupt che lasciano la macchina in uno stato ben preciso.
devono rispettare queste proprietà:

1. il PC è salvato in un posto noto
2. Tutte le istruzioni precedenti a quella puntata dal PC sono state eseguite completamente
3. Nessuna istruzione successiva a quella puntata da PC è stata eseguita
4. Lo stato di esecuzione dell'istruzione puntata dal PC è noto

Interrupt imprecisi

Un interrupt impreciso è un interrupt che non rispecchia le proprietà scritte prima
le macchine con interrupt imprecisi riversano una grande quantità di dati interni nello stack, per far capire al SO cosa sta succedendo.
Questo causa lentezza nel ripristino e nelle istruzioni.

Azioni software per la gestione dell'ISR(p. 425 architettura)

le azioni software per la gestione degli interrupt sono :

1. la ISR inizia con il salvare(su stack o in una tabella di sistema) tutti i registri che utilizza per poterli ripristinare
2. in genere ogni vettore di interrupt è condiviso con tutti i device dello stesso tipo,perciò non identifica univocamente il terminale che ha causato l'interrupt
3. viene letta ogni informazione sull'interrupt.
4. nel caso di errore I/O, viene gestito ora.
5. le variabili globali prt e count vengono aggiornate;la prima viene incrementata,la seconda decrementata
6. se richiesto, viene inviato un cod. speciale al device per informarlo che l'interrupt è stato trattato
7. ripristino dei registri salvati
8. esecuzione dell'istruzione IRET che ripristina la modalità e lo stato della CPU

Scopi del software I/O

I principali scopi del software I/O sono :

1. **indipendenza dal dispositivo**(deve essere possibile scrivere programmi che si interfaccino con ogni dispositivo I/O)
2. **denominazione uniforme (**il nome di un file o device dovrebbe essere semplicemente una stringa di caratteri (o un numero intero) indipendenti dal device)
3. **Trattamento degli errori** (gli errori dovrebbero essere trattati più vicino possibile all'hardware)
    a. molti errori sono transitori (come la lettura da testina), scompaiono se si ripete l'operazione
4. **Trasferimento sincrono/asincrono (**i trasferimenti possono essere bloccanti (sincroni) o gestiti dagli interrupt (asincroni)**)**
    a. La maggior parte dell'I/O è asincrono. Tuttavia è più semplice creare programmi con primitive bloccanti (es. ***read()***), quindi sta al SO dare l'illusione di creare programmi con primitive sincrone, quando in realtà i device sono asincroni.
5. **Bufferizzazione** (i dati che escono da un device non possono essere portati immediatamente nella loro destinazione finale)
    a. Per es. quando arriva un pacchetto dalla rete, il SO non sa a chi deve andare finché non viene immagazzinato ed esaminato
6. **Condivisione (**dato che alcuni device (es. dischi) sono condivisi con più utenti, si dovrebbero introdurre dei dispositivi dedicati (non condivisi). Anche se la loro implementazione potrebbe portare a dei problemi, sta al SO gestirli, insieme a quelli condivisi**)**

Tipologia di I/O

Ci sono 3 tipologie di I/O:

1. **I/O programmato**
2. **I/O con DMA**
3. **I/O con interrupt**

I/O programmato

Questa tipologia lascia alla CPU tutto il lavoro da fare
Facciamo un esempio:
Consideriamo un processo che voglia stampare una stringa di caratteri; il processo metterà la stringa in un buffer.
A questo punto il processo acquisisce la stampante in scrittura, tramite una system call.
Il SO copia il buffer in un array dello spazio kernel e controlla se la stampante è libera.
Appena si libera, copia il primo carattere dal buffer e lo mette nel registro dati della stampante, utilizzando l'I/O mappato in memoria
appena copia il primo, il SO controlla se la stampante è pronta per un'altro, e così via.
Le azioni che esegue il SO sono quindi:

1. mettere il buffer nello spazio kernel
2. poi con un ciclo emette i caratteri uno alla volta

La caratteristica peculiare è che dopo il primo carattere, il SO controlla continuamente se il dispositivo è pronto ad accettare un'altro carattere. Questo comportamento è detto **polling o attesa attiva**
Questa tipologia è semplice, ma ha lo svantaggio di tener occupata la CPU per tutta la durata di esecuzione del processo

I/O guidato dalle interruzioni

Un'altro esempio è se la stampante invece che prendere i dati dal buffer, li riceve man mano che arrivano
Ogni volta che viene eseguita una system call per la stampa, il primo carattere viene spedito dal buffer; a questo punto la CPU richiama lo scheduler e il processo che ha avviato la stampa viene sospeso
Quando la stampante è pronta a ricevere un'altro carattere, genera un'interruzione che ferma il processo corrente e ne salva lo stato; viene richiamato l'ISR della stampante
Se non ci sono altre interruzioni, il processo utente viene liberato, altrimenti continua in un ciclo
Uno svantaggio di questo metodo è che per ogni carattere viene generata un'interruzione. Questo richiede tempo e spreca tempo CPU

I/O tramite DMA (NO CPU)

Una soluzione allo svantaggio di I/O con interruzioni è quello di usare il DMA.

il controllore DMA emette i caratteri uno alla volta, senza interpellare la CPU
In questo caso il controllore DMA è l'I/O programmato, con la differenza che il lavoro lo fa lui e non la CPU
Il vantaggio è che si riduce drasticamente il numero di interrupt, da 1 per carattere a 1 per buffer
LO svantaggio è che il controllore DMA è più lento della CPU, e se il controllore DMA non riesce a far andare a massima velocità il device, o la CPU non ha altro da fare, l'I/O programmato e quello con interruzioni potrebbero essere meglio

Driver dei dispositivi

Ogni device I/O deve essere controllato da un codice.
Questo codice è detto **driver dei dispositivi**
Ogni driver controlla un solo device, al più una sola classe di device.
Deve far parte del kernel, per poter accedere all'hardware del dispositivo
CI sono 2 categorie:

- **device a blocchi**
- **device a caratteri**

I driver devono essere **rientranti**, cioè si devono aspettare di essere richiamati prima di finire l'esecuzione corrente
In un sistema "a caldo", gli utenti possono staccare il dispositivo mentre ancora in esecuzione
I driver devono assicurare l'integrità delle strutture dati kernel, abortendo il processo di trasferimento I/O, poi devono rimuovere ogni richiesta pendente sul device, ed infine deve avvisare gli utenti di tale cancellazione

Interfacciamento uniforme per i driver

un punto focale in un SO è cercare di rendere tutti i device e i driver più simili possibile.
Un'aspetto di questo problema è l'interfaccia tra driver e SO
Avere un'interfaccia diversa per ogni dispositivo, causerebbe uno sforzo a livello programmazione molto elevato, perché le funzioni del driver cambiano da driver a driver
Se invece si utilizza un'interfaccia standard per tutti i driver, lo sforzo a livello programmazione è drasticamente ridotto, ed inoltre diventa più semplice collegare nuovi driver
UN'altro aspetto di avere un'interfaccia comunque è il modo in cui sono denominati i file
in UNIX, per esempio, il nome del device contiene esattamente l'i-node di un file speciale, che contiene un **numero principale di dispositivo** usato per trovare il driver appropriato.
Contiene anche un **numero secondario di dispositivo** usato per specificare l'unità su cui leggere/scrivere.

Bufferizzazione

un'altro aspetto fondamentale è la bufferizzazione
Il miglior metodo è quello della **bufferizzazione doppia**
In questo schema si hanno 2 buffer nel kernel e uno nello spazio utente.

Quando il primo buffer kernel si riempie, se ne fa una copia nello spazio utente(se è stato richiesto), e nel mentre si utilizza il secondo buffer kernel.
In questo modo i due buffer fanno a turno.
La bufferizzazione è molto usata, ma ha i suoi svantaggi.
Se viene usata troppo, le prestazioni ne risentiranno.

Riportare errori

Gli errori I/O sono molto più frequenti di quanto si pensa.
ci sono varie classi di errori I/O.
la prima classe è quella degli errori di programmazione.
Essa si verifica quando il processo I/O chiede qualcosa di impossibile,
Le azioni da fare sono semplici, riportare un codice di errore al chiamante
un'altra classe è quella di errori I/O veri e propri, per esempio quando si vuole scrivere su un disco rovinato. In questo caso il driver decide cosa fare
La decisione viene presa in base all'ambiente e alla natura dell'errore:

1. Se è un'errore di lettura, e ce un utente disponibile, viene visualizzata una finestra di dialogo, l'utente deciderà
2. Se non c'è un utente disponibile, la system call fallirà e verrà restituito un codice di errore

Software I/O a livello utente

sebbene la maggior parte I/O si trovi all'interno del SO, una piccola parte di esso si compone di librerie collegate ai programmi utente,e da interi programmi che risiedono al di fuori del nucleo operativo

Spooling

non tutto il software I/O a livello utente è composto da librerie
Un sistema di **spool** ne è un esempio.
Lo spooling è una tecnica particolare di trattare i dispositivi dedicati in un ambiente multiprogrammato
Un esempio di device che usa lo spooling:stampante
quello che si fa è creare un processo speciale, detto **demone**,e una directory speciale, detta **directory di spool**
Per stampare un file, un processo prima genera tutto il file da stampare, e poi lo scrive nella directory di spool, In seguito il demone,stampa i file presente nela directory
La tecnica di spooling viene usata anche in altre situazioni, non solo nella stampante

I dischi

i più comuni sono i dischi magnetici, che sono caratterizzati dal fatto che la lettura e scrittura sono ugualmente veloci

i dischi magnetici

tutti i dischi sono organizzati in cilindri,ognuno dei quali contiene tante tracce quante sono le testine. le tracce sono divise in settore,ed il numero dei settori che si trovano su una traccia va da 8 32 sui dischetti e fino a diverse centinaia sui dischi fissi
su i dischi **IDE,** il driver contiene un microcontrollore che fa parte del lavoro,e lascia al controllore vero e proprio il compito di emettere comandi a livello più alto
Una caratteristica importante è la possibilità che un controllore esegua operazioni di posizionamento(seek) su due o più dischi contemporaneamente.
questo fatto è noto come **posizionamento sovrapposti(**overlapped seeks**):** mentre il controllore e il software stanno aspettando il completamento di un operazione, il controllore può cominciare un operazione di posizionamento su un altro disco

Formattazione del disco

Prima di usare un disco , ogni piatto deve ricevere una **formattazione di basso livello,** eseguita via software
la formattazione consiste in una serie di tracce concentriche,ognuna contenente alcuni numeri di settore, con piccoli intervalli tra i settori(**intersection gap**)
Il preambolo comincia con un certo pattern di bit che permette all'hardware dio riconoscere l'inizio di un settore
la dimensione della porzione dati è determinata dal programma di formattazione
Il campo ECC contiene informazioni ridondanti che possono essere utilizzate per ripristinare il disco dopo un errore di lettura.
con la formattaziona a basso livello,la posizione del settore 0 in ogni traccia si sposta di una certa distanza rispetto al settore 0 della traccia precedente.Questa distanza è detta **pendenza del cilindro** e viene fatta per aumentare le prestazioni.
Il passaggio da una testina all'altra richiede tempo, quindi esiste anche una **pendenza della testina**
Con la formattazione a basso livello, lo spazio disponibile si riduce, di circa il 20%
Esempio
un drive non formattato da $20x10^9$ byte, dopo la formattazione forse avrà $17{,}2x10^9$byte di spazio disponibile.Il So probabilmente riporterà solo 16GB piuttosto che 17,2, perché considera 1GB come 2^{30} byte e non 10^9byte

Algoritmi di scheduling del disco

Il tempo necessario per leggere/scrivere un blocco dal disco è dato da :

1. il tempo di posizionamento della testina(seek time)
2. il ritardo di rotazione
3. il tempo di trasferimento

Per la maggior parte dei sistemi, il seek time è quello predominante
Un algoritmo che si può utilizzare è il **FCFS**(molto simile al FIFO).(**First Come First Served**) con questo algoritmo non si può fare molto per quanto riguarda le prestazioni

Una variante del FCFS utilizzata nei dischi è l'algoritmo **SSF(Shortest Seek First)**
Questo algoritmo, invece che eseguire le richieste cosi come arrivano, esegue prima quelle più vicine alla posizione della testina corrente. Questo algoritmo favorisce le posizione centrali del disco rispetto a quelle più esterne
RIduce di circa la metà gli spostamenti sul braccio, rispetto al FCFS

Una variante del SSF è l'algoritmo dell'ascensore.
Questo algoritmo funziona con un bit di controllo.***UP o DOWN***
Quando viene soddisfatta una richiesta, io software controlla questo bit:

1. Se è UP, il braccio sis sposta verso la richiesta più bassa disponibile, in un unica direzione
    a. se non c'è nessuna richiesta, la direzione viene invertita
2. Se è DOWN, accade la stessa cosa solo nella direzione opposta

Questo algoritmo è leggermente migliore dell SSF, solo in pochi casi però. In generale è peggiore.
Una proprietà interessante è che il numero di spostamenti del braccio è fisso:
è uguale al doppio del numero di cilindri

Memorizzazione Stabile

Talvolta i dischi possono generare errori,diventando inutilizzabili.
Per alcune applicazione è essenziale che i dati non siano mai persi ne corrotti, per questo si usa un sottosistema del disco con questa proprietà:
Quando si effettua una scrittura su disco, o avviene in modo corretto, o non avviene proprio
Questo sistema si chiama **memorizzazione stabile**
Questa tecnica utilizza una coppia dischi identici dove i blocchi corrispondenti lavorano insieme per formare un blocco libero da errori
per raggiungere questo obiettivo vengono definite 3 operazioni:

1. **scritture stabili**
   consiste nello scrivere prima il blocco sul drive 1, e poi rileggerlo per verificare che sia tutto okay, in caso di errore viene riscritto e riletto per un massimo di n volte,finchè non viene completato con successo.al completamento del'op sul drive 1, il blocco del drive 2 viene scritto e letto, come prima. AL termine di tutto, se non ci sono crash della CPU,il blocco è stato scritto correttamente su entrambi i dischi
2. **letture stabili**
   consiste nel leggere un blocco dal drive 1; se questo porta un ECC, viene riletto per un massimo di n volte.Se tutte le letture portano ad un ECC, allora si tenta la lettura sul drive 2. l'op di lettura viene sempre completata con successo.
3. **ripristino del crash**
   dopo un crash, un programma di ripristino analizza entrambi i blocchi sui drive.Se una coppia di blocchi è buoan okay, se uno dei due riporta un ECC errato, il suo contenuto viene sovrascritto con quello dei blocco buono, se entrambi sono buoni ma diversi, il blocco del drive 2 viene sovrascritto con quello del drive 2

I clock

i **clock**(detti anche **timer**) sono essenziali per le operazioni in un sistema multiprogrammato. Fra le altre cose, mantengono anche la data e l'ora e impediscono che un processo occupi tutto il processore.Il software del clock prende la forma di driver del dispositivo, sebbene non sia né un dispositivo a caratteri ne uno a blocchi

Hardware del clock

I clock sono di due tipi.
quello più semplice è legato alla linea di alimentazione a 110 o 220 V, e genera un'interruzione ogni ciclo del voltaggio, a 50 o 60 Hz
l'altro tipo ha 3 componenti :

1. un oscillatore a cristallo
2. un contatore
3. registro di caricamento

quando un cristallo di quarzo viene tagliato e messo sotto pressione, può essere forzato a generare un segnale periodico di alta precisione. (proprietà ***piezo-elettrica***)
Con una opportuna elettronica, questo segnale base può essere moltiplicato per ottenere fino a 1000 MHz o anche più
questo segnale viene mandato al contatore per far sì che esso conti all'indietro.QUando il contatore raggiunge lo zero,cause un'interrupt di CPU
i clock programmabili presentano diversi modi di funzionamento.
Nel modo **a colpo singolo**(**one-shot-mode**) quando viene fatto partire,il clock copia il valore del registro di caricamento nel contatore,e poi lo decrementa ad ogni impulso del cristallo
Nel modo **ad onda quadra**(**square-wave-mode**) dopo essere arrivato allo zero ed aver causato l'interruzione, il registro di caricamento viene copiato automaticamente nel registro contatore,e tutto si ripete all'infinito. queste interruzioni vengono dette *tic* del clock

Software del clock

i compiti esatti del driver di clock variano da sistema a sistema, ma generalmente essi includono:

1. mantenere data e ora
2. evitare che un processo sia più attivo del normale
3. tenere una contabilità del processore (CPU accounting)
4. gestire le system call **alarm** da parte dei processi utente
5. mettere a disposizione i **timer watchdog** per parti del SO
6. eseguire il profiling,monitoraggio e raccolta statistiche

La prima funzione(detta anche **tempo reale**) richiede semplicemente di incrementare il contatore a ogni tic

la seconda funzione il clock deve controllare che ogni processo rispetti il quanto di tempo assegnatogli dallo scheduler; ad ogni interruzione, il clock decrementa un contatore(inizialmente = quanto)di una unità

la terza funzione è la contabilità del processore.per realizzare questo lavoro, viene fatto partire un secondo timer, diverso dal primo principale, ogni volta che un processo viene eseguito. A fine esecuzione, si può leggere questo timer per vedere per quanto tempo è stato in esecuzione il processo.
alcune parti del SO richiedono l'inizializzazione dei timer, detti timer watchdog
il meccanismo usato dal driver del click per trattare i watchdog timer è lo stesso di quello per i segnali utente;la sola differenza è che quando un timer arriva a 0,anzichè provocare un segnale, il driver chiama una procedura fornita dal chiamante(può fare qualunque cosa ritenga necessaria),sebbene a livello del nucleo di interruzioni non siano conveniente e non esistano i segnali

l'ultimo punto della lista riguarda il profiling:alcuni SO mettono a disposizione un meccanismo con il quale un processo utente può chiedere al sistema di costruire un **istogramma(profile)** del suo contatore di istruzioni
quando il profiling è ammesso, ad ogni tic il driver del clock controlla se il processo attualmente in esecuzione sotto profiling, e in questo caso calcola l'intervallo di indirizzi corrispondente al contatore di istruzioni attuale

I timer soft

la maggior parte dei computer ha un secondo clock programmabile che può essere impostato in modo da causare delle interruzioni di temporizzazione a qualunque frequenza richiesta dal programma
i **timer soft** eliminano le interruzioni: mentre il kernel è in esecuzione per un qualsiasi motivo, prima di ritornare in modo utente, controlla il clock real-time per vedere se il timer soft è scaduto;in questo caso esegue l'evento programmato,senza bisogno di passare al modo kernel dato che il sistema è già in tale modo
alla fine dell'esecuzione, il timer soft viene reimpostato per una scadenza successiva.

i timer soft funzionano o meno a seconda dell'utilizzo del kernel, Alcuni esempi di utilizzo del kernel sono :

1. system call
2. TLB miss(accessi mancanti alla TLB)
3. page fault
4. I/O interrupt
5. CPU idle(che si mette in attesa)

I File system

è la parte del SO che gestisce i file ed è in maniera gerarchica nella quale ciascun nodo intermedio è un contenitore di altri file (directory) in cui elenco della directory che occorre attraversare dalla radice (root) per raggiungere il file chiamato (path del file)

Shell

Il sistema operativo è il codice che realizza le chiamate di sistema (system call) di conseguenza non fanno parte del sistema operativo ma gli editor, i compilatori, gli assemblatori e l'interprete si

NB: La **GUI** (graphic user interface è un applicazione o programma che sta sopra al sistema operativo ma non ne fa parte

File

i file sono un unità logiche di informazioni create dai processi, i quali possono creare i file e leggere quelli già esistenti. le informazioni memorizzate nei file sono persistenti e non dipendono dalla vità del processo, ma dovrebbero scomparire solo quando il suo proprietario lo rimuove. infine sono gestiti dal sistema operativo. i principali argomenti per la progettazione di un sistema operativo che i file abbiano come caratteristiche: la struttura, la denominazione, la acesso, l'utilizzo, la protezione, la realizzazione e la gestione

- **Nomi dei file**

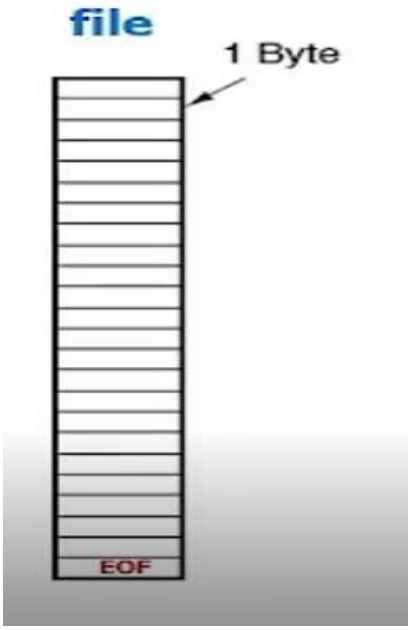
  le regole con cui i nomi dei file sono adottati dipende da sistema a sistema, ma tutti i sistemi considerano i nomi validi fino a 8 caratteri che non siano però meta caratteri usati dal file system:(" # % & * : < > ? / \ { | } …).

  in alcuni casi sono ammessi i caratteri speciali (!) o lettere accentate (à è ì ò ù….)

  inoltre il nome è di norma affiancato dal suo tipo (o estensione del file) es: .py .C .jpg .pdf Etc…

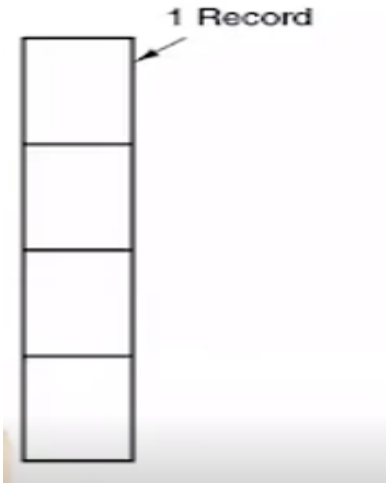
Estensione	Significato
file.bak	File di backup
file.c	Programma sorgente in linguaggio C
file.gif	Immagine in Compuserve Graphical Interchange Format
file.hlp	File di aiuto
file.html	Documento HTML (world wide web hypertext markup language)
file.jpg	Immagine codificata con lo standard JPEG
file.mp3	Musica codificata in formato audio MPEG layer 3
file.mpg	Filmato codificato in formato audio MPEG standard
file.o	File oggetto (output da compilatore, non ancora linkato)
file.pdf	Documento in formato Adobe PDF (portable document format)
file.ps	File PostScript
file.tex	Input per il programma di formattazione TEX
file.txt	File di testo generico
file.zip	Archivio compresso

- **struttura dei file**
  1) **sequenza a byte**
     - in questo caso il sistema operativo non sa che cosa è contenuto nel file in quanto vede solo una sequenza di byte
     - è una struttura con la massima flessibilità poiché i programmi utente possono mettere tutto ciò che vogliono nei loro file
     - tutte le versioni di UNIX, MS-DOS e Windows utilizzano questo modello di file



  2) **sequenza di record**

- il sistema operativo vede una sequenza di record di lunghezza fissata con una struttura ben definita
- le operazioni di lettura e scrittura avvengono sui record
- i primi mainframe utilizzano record da 80 caratteri così come le colonne di una scheda perforata



3) **albero di record**
   - e una struttura composta da un albero di record che non necessitano d'essere della stessa grandezza dove ogni record contiene un campo chiave in posizione fissa
   - albero è ordinato sul campo chiave per velocizzare le ricerca di una chiave particolare
   - l'operazione base non è di ottenere il record successivo, ma rintracciare il record con una determinata chiave
   - questo modello è utilizzato sui mainframe in alcune applicazioni commerciali
   - i record possono essere aggiunti al file, con il sistema operativo, l'utente non può decidere dove posizionare i record

- **Tipi di file**

  i sistemi operativi supportano diversi tipi di file

1) **i file normali**
   - sono quelli che contengono i dati degli utenti
   - sono generalmente ASCII o binario
   - i file ASCII consistono di righe di testo non necessariamente della stessa lunghezza
   - i file ASCII sono facili da visualizzare e stampa e possono essere aggiornati con un semplice editor di testo
   - i programmi che utilizzano file ASCII per l'input/output possono essere interfacciati con facilità (es: pipe)
   - i file binari sono sequenze di bit (non interpretati come caratteri) e hanno una struttura interna nota soltanto ai programmi che li utilizzano (non possono essere stampati e maneggiati da editor di testi)

2) **le directory**
    - sono i file di sistema usati per mantenere la struttura dei file
3) **i file speciali a caratteri**
    - sono legati all'input/output e sono utilizzati per modellare i dispositivi di I/O seriali come terminali stampati e reti
4) **file speciali blocchi**
    - sono utilizzati per modellare i dischi

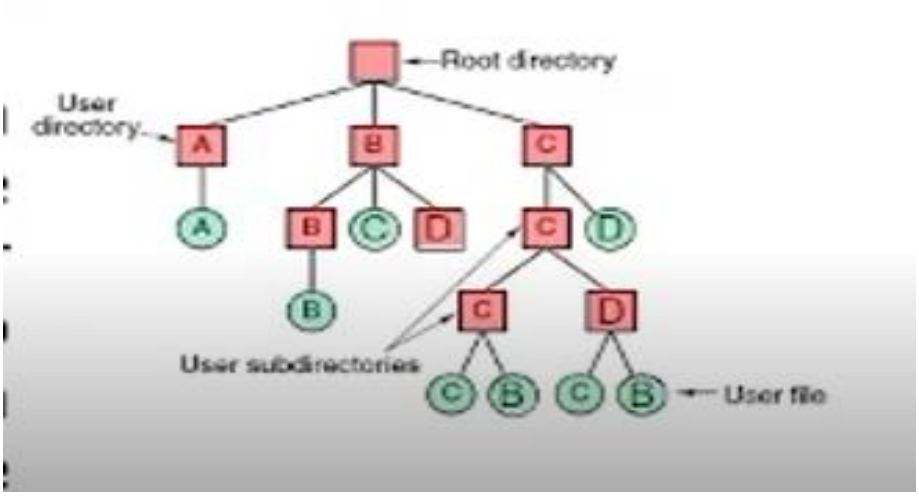
- **accesso ai file**
    - i primi sistemi operativi avevano un solo tipo di accesso ai file l'**accesso sequenziale** dove i byte vengono letti nell'ordine in cui sono scritti, cominciando dall'inizio
    - i file sequenziali erano perfetti quando l'archiviazione era su nastro magnetico piuttosto che su disco
    - con i dischi è possibile leggere i byte o i record in qualsiasi ordine (da cui file ad accesso casuale) o accedere ai record tramite una chiave
    - il metodo **read()** permette di leggere i dati a partire da una posizione speciale nel file
    - il metodo **seek()** permette di impostare la posizione corrente nel file in modo da poterlo leggere sequenzialmente da quella posizione
- **attributi dei file**
    - ogni file ha un nome, un contenuto (i dati) e alcuni **attributi** (**meta-dati**): il timestamp di creazione, la dimensione, gli utenti autorizzati ad accedere al file, ….
    - gli attributi variano da sistema a sistema ma ne esistono di comuni tra tutti

Attributo	Significato
Protezione	Chi può accedere al file e in che modalità
Password	Password necessaria per accedere al file
Creatore	ID della persona che ha creato il file
Proprietario	Proprietario attuale
Flag di sola lettura	0 per lettura/scrittura; 1 per sola lettura
Flag di file nascosto	0 per normale; 1 per non visualizzare negli elenchi
Flag di file di sistema	0 per file normali; 1 per file di sistema
Flag di file archivio	0 per già sottoposto a backup; 1 per file di cui fare il backup
Flag ASCII/binario	0 per file ASCII; 1 per file binari
Flag di accesso casuale	0 per accesso sequenziale; 1 per accesso casuale
Flag temporaneo	0 per normale; 1 per cancellare il file alla fine del processo
Flag di file bloccato	0 per non bloccato; non zero per bloccato
Lunghezza del record	Numero di byte nel record
Posizione della chiave	Offset della chiave in ciascun record
Lunghezza della chiave	Numero di byte del campo chiave
Data e ora di creazione del file	Data e ora di quando il file è stato creato
Data e ora di ultimo accesso al file	Data e ora di quando è avvenuto l'ultimo accesso al file
Data e ora di ultima modifica al file	Data e ora di quando è avvenuta l'ultima modifica al file
Dimensione attuale	Numero di byte nel file
Dimensione massima	Numero di byte di cui può aumentare il file

- **operazioni dei file**
    - i file nascono per memorizzare informazioni per poterle rileggere in un secondo momento
    - esistono varie operazioni disponibili sono le chiamate di sistema più comuni sono:
        1) **create()** crea un file vuoto con alcuni attributi impostati
        2) **delete()** rimuove il file e libera lo spazio su disco
        3) **open()** prima di poter utilizzare un file, un processo deve aprirlo. la chiamata di sistema memorizza in memoria gli attributi e l'elenco degli indirizzi sul disco per un accesso rapido nelle successive chiamate
        4) **close()** chiude il file liberando la memoria dai riferimenti al file perchè non verrà più utilizzato. molti sistemi incoraggiano l'uso della chiamata imponendo un numero massimo di file aperti per i processi, poiché il disco è scritto a blocchi, la chiusura di un file forza la scrittura dell'ultimo blocco del file
        5) **read()** legge i dati dal file. soltanto i byte vengono letti dalla posizione corrente. occorre specificare il numero di dati da leggere e il buffer in memoria ove inserirli
        6) **write()** scrive i dati nel file, dalla posizione corrente
        7) **append()** aggiunge dati alla fine del file
        8) **seek()** cambia la posizione corrente nel file ad accesso casuale. dopo questa chiamata di sistema i dati possono essere letti o scritti a partire dalla nuova posizione
        9) **stat()** in UNIX viene utilizzato per gli attributi dei file
        10) **rename()** che viene utilizzato per rinominare un file

le directory

- sono il modello astratto di dati che nascondono le peculiarità dell'hardware utilizzati nei nastri, dischi e nei dispositivi I/O
- i file system normalmente organizzano i file in contenitori denominati **directory** o cartelle
- la forma più semplice è la **directory principali** (o **root directory**) una sola directory (chiamata root) contenente tutti i file
- ogni directory può contenere altre directory (o file), con il vincolo che per ciascuna di esse esista sempre un solo contenitore(o genitore) questo porta ad una struttura gerarchica (padre e figlio)
- gli utenti possono avere una directory di inizio (**home**) della propria gerarchia (**path name)**



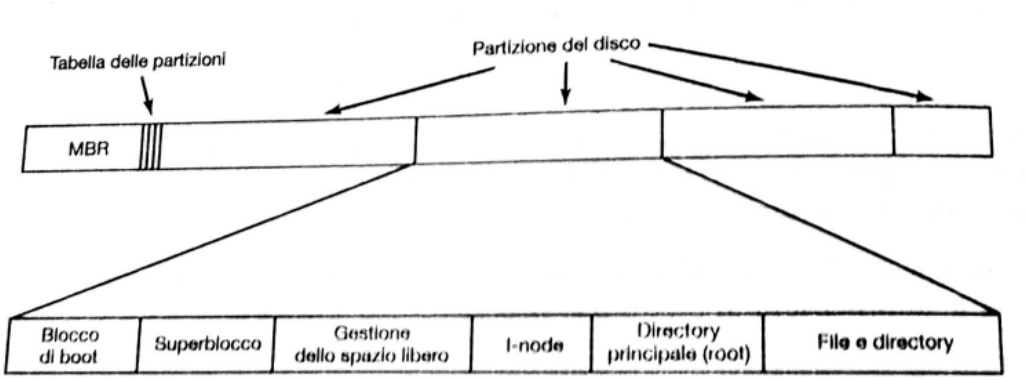
- le chiamate di sistema per la gestione delle directory hanno qualche variante da sistema a sistema rispetto a quelle per la gestione dei file in UNIX abbiamo:
    - **mkdir()** crea una directory vuota [ad eccezione di “.” (directory corrente) e “..” (la directory genitore) che sono automaticamente inserite]
    - **rmdir()** rimuove una directory vuota (deve contenere solo “.” e “..”)
    - **opendir()** carica in memoria tutti i riferimenti di localizzazione sul disco prima della lettura (così come avviene per i file)
    - **readdir()** restituisce la successiva voce di una directory aperta.
    - **closedir()** chiude la directory al termine della lettura e libera il corrispondente spazio in memoria
    - **rename()** rinomina una directory esistente
    - **link()** crea un **link hard** tra un file esistente e il **pathname** indicato
    - **unlink()** rimuove il collegamento nella directory se il file è unico è cancellato dal file system altrimenti rimosso solo un collegamento

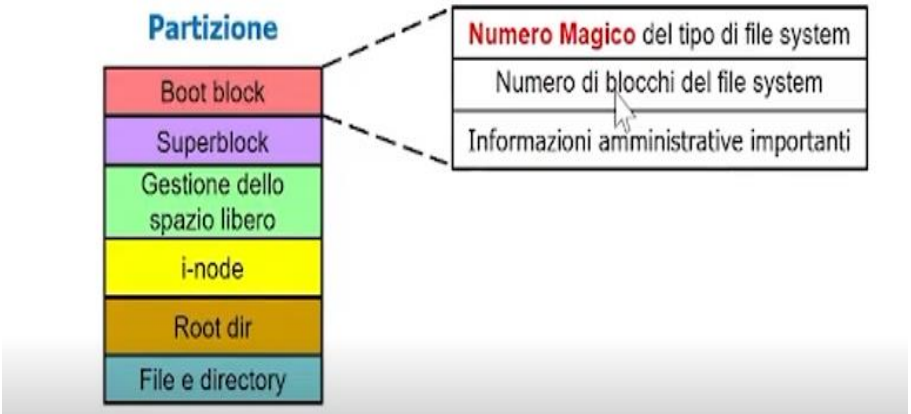
Path name

- quando un file è organizzato come un albero per identificare in modo univoco ogni oggetto della struttura (directory e file) sono utilizzati due metodi
    1) **path name assoluto**
        - ogni file(o directory) viene identificato dal percorso necessario per raggiungerlo, a partire dalla directory principale
        - il percorso è composto dalla sequenza ordinata di directory separate da un carattere speciale: “/” in UNIX , “\” in Windows, “>” in MULTICS
        - se il primo carattere del nome percorso è il separatore, allora il riferimento è assoluto
    2) **path name relativo**
        - il file o la directory viene individuato utilizzando un punto relativo sull’albero(la **directory corrente,** la **home** dell’utente,...)
        - la maggior parte dei sistemi operativi hanno in ciascun directory due file speciali:
            1. “.” è la directory corrente
            2. “..” è il suo genitore (tranne per root che fa riferimento a se stesso)

Layout del file system

- i file system sono memorizzati sui dischi
- la maggior parte dei discho possono essere divisi in una o più partizioni con file system indipendenti su ogni partizione
- **settore 0** del disco e chiamato **MBR** (**Master Boot Record**) e viene utilizzato per avviare il computer
- la fine del **MBR** contiene la **tabella delle partizioni** dove sono indicati l'inizio e la fine di ogni partizione
- nella tabella una sola partizione è contrassegnata come attiva
- quando il computer è avviato, il BIOS legge ed esegue il **MBR**
- **MBR** localizza la partizione attiva, legge ed esegue il primo blocco (**blocco di boot** o **boot clock**) sulla partizione
- il programma nel **blocco di boot** carica il sistema operativo contenuto nella partizione
- Ogni partizione inizia con un **blocco di boot** anche se non contiene un operativo avviabile perché potrebbe contenere uno in futuro



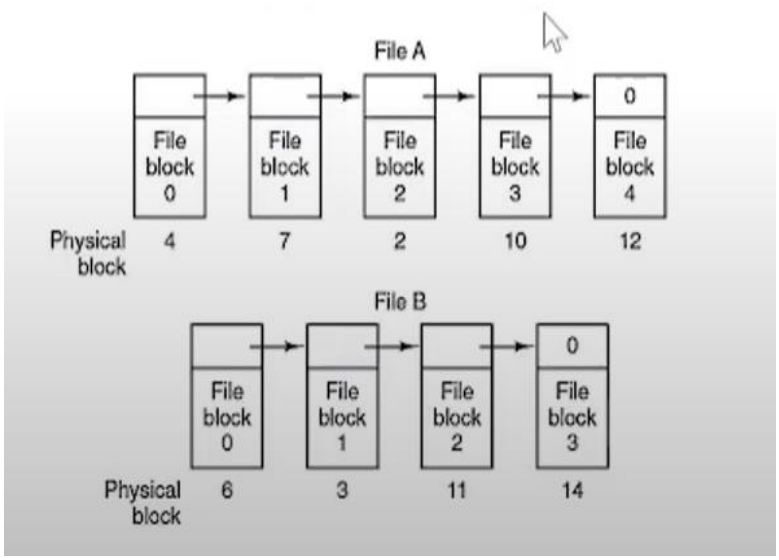
- **contenuto della partizione**
  - anche se in generale il contenuto delle partizioni cambia da file system a file system, uno schema ricorrente è:
  - 

- il **blocco di boot** contiene tutti i parametri fondamentali relativi al file system, viene letto in memoria quando il computer è avviato o il file system viene usato per la la prima volta

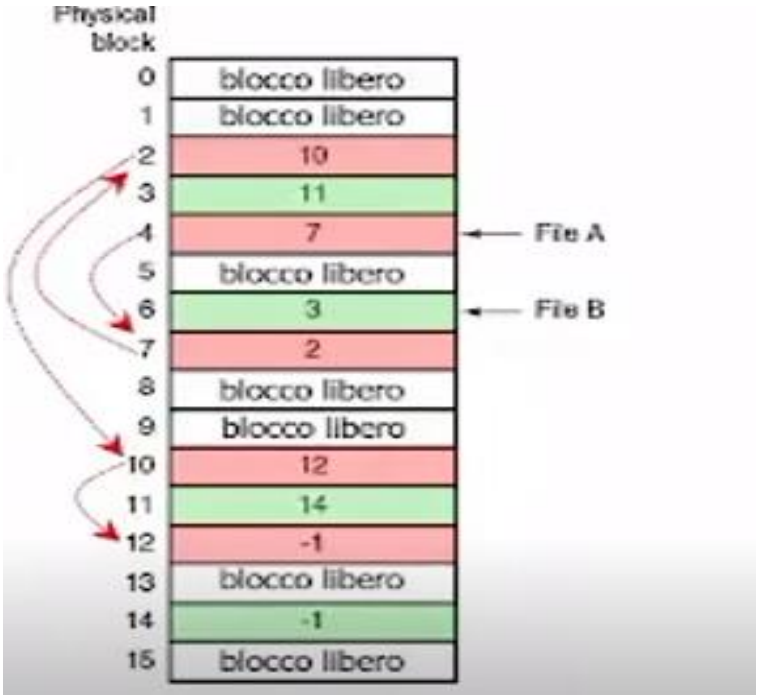
Allocazione dei file

Un quesito importante nella memorizzazione di un file è tenere traccia di quali blocchi del disco sono associati a ciascun file. esistono vari metodi utilizzati in diversi sistemi operativi:

- **Allocazione contigua**
  - è lo schema di allocazione più semplice dove i file vengono memorizzati come blocchi adiacenti al disco
  - poiché la dimensione del file sarà difficilmente multiplo della dimensione del blocco si crea un piccolo spreco di spazio interno all'ultimo blocco
    1. **Svantaggi**
       1) piccolo spreco di spazio nell'ultimo blocco
       2) frammentazione a causa della rimozione dei file (gli spazi vuoti tra file non riescono ad essere impiegati per i nuovi)
    2. **vantaggi**
       1) semplice da realizzare: è sufficiente conoscere l'indirizzo del disco del primo blocco e il numero di blocchi nel file
       2) alte prestazioni: l'intero file può essere da disco in una sola operazione (tempo di seek e la latenza domina il tempo di accesso del disco
- **Allocazione con liste collegate**
  - si utilizza una lista concatenata di blocchi del disco per memorizzare i file (le liste non necessitano di essere adiacenti)
  - la prima parola di ogni blocco viene usata come puntatore alla successiva. il resto del blocco è dati
    1. **svantaggi**
       1) piccolo spreco nell'ultimo blocco
       2) l'accesso causale è **estremamente lento** prima di arrivare ad un blocco occorre accedere ai puntatori dei blocchi che lo precedono
       3) poiché la dimensione del blocco dati non è una potenza di due a causa dello spazio riservato al puntatore (pochi byte) ci potrebbe essere una inefficienza dovuta ai programmi che leggono/scrivono quantità di dati in 2^n
    2. **vantaggi**
       1) possono essere utilizzati tutti i blocchi del disco (non c'è spazio libero inutilizzato tra i file)
       2) per ciascuna entry nella directory è sufficiente memorizzare solo l'indirizzo del primo blocco del disco
       3) la lettura dei file in modo sequenziale è facile ma non può essere svolta in un sol colpo prima

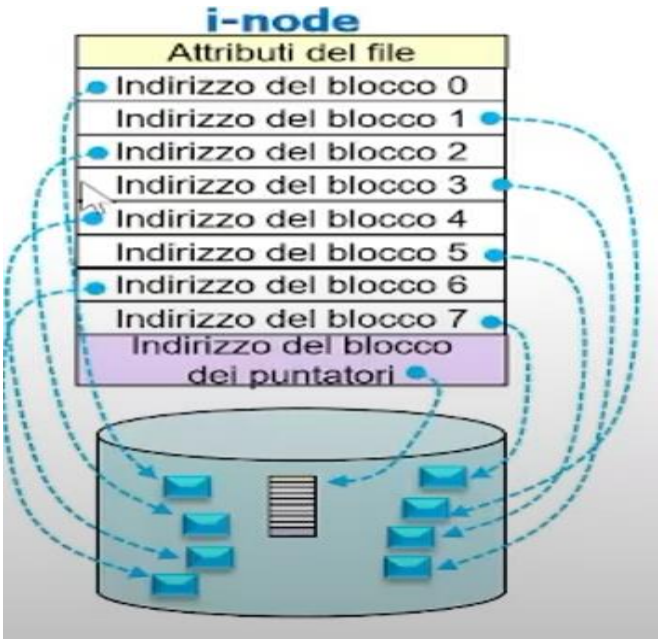


- **Allocazione con tabella in memoria**
  - la tabella in memoria sfrutta dei puntatori di ogni blocco del disco (**FAT File Allocation Table**)
  - le catene dei dati sono terminati con un numero speciale terminatore (-1)
  - l'intero blocco è disponibile per i dati e accesso casuale è facile
  - la entry nella directory mantiene il numero di blocco iniziale (un intero)
  - l'intera tabella deve essere in memoria (con un disco da 200GB e blocchi da 1 KB, la tabella 600 MB di RAM)
  - entrambi gli svantaggi dell allocazione a contigua e a liste collegate non sono presenti qui



- **i-nodes**
  - per ogni file viene creata una struttura di dati denominata **i-node (index-node)** che contiene gli attributi e gli indirizzi dei blocchi del disco

- poiché l'**i-node** ha un numero fissato di posizioni, per consentire l'incremento dei blocchi del file, l'ultima cella è riservata ad un altro blocco simile
- l'**i-node** è caricato in memoria solo quando il file viene aperto (la sua dimensione è più piccola di una FAT)
- mentre la Fat è proporzionale alla dimensione del disco (tante righe quanti i blocchi del disco), la dimensione dell'**i-node** dipende dal numero massimo di file che possono essere aperti



Riepilogo delle directory

- durante l'apertura del file, il sistema operativo utilizza il pathname per individuare la voce nella directory che fornisce:
  - l'indirizzamento del primo blocco del disco (allocazione contigua e gli schemi lista collegate) oppure
  - i numero di **i-node**
- la directory associa il numero il numero del file, gli attributi e il puntatore ai dati
- nel caso dell'**i-node** gli attributi possono essere memorizzati anche nell'**i-node** stesso

File Condivisi

Quando più utenti lavorano insieme su un progetto, hanno bisogno di condividere file.
Il collegamento che permette ad un utente di vedere in una sua directory il file condiviso è detto **link.** Il file system diventa un grafo diretto aciclico piuttosto che un albero

La condivisione è comoda, ma introduce dei problemi, se le directory contengono gli indirizzi del disco, quando un file è collegato occorre copiarli. A questo punto se uno dei due utenti appende dati, i nuovi blocchi sono visibili solo all'utente che li ha inseriti vanificando il concetto di condivisione, due soluzioni:

1. UNIX: i blocchi non sono elencati nella directory ma in una struttura dati associa al file (i-node);

2. Link simbolico: il collegamento crea un nuovo file di tipo link che permette al sistema operativo di accedere al file sorgente

Ciascuno di questi metodi ha i suoi svantaggi:
quando B collega il file di C, l'i-node registra C come proprietario del file e il contatore dei riferimenti al file a 2 (Count).
Se ora C rimuove il file, se si cancella l'i-node, B avrà una voce che punta ad un i-node non valido oppure, se l'i-node è stato riassegnato ad un altro archivio, ad un file scorretto.
L'unica cosa possibile è cancellare la voce dalla Directory di C e mantenere l'i-node in C (con Count=1 e Owner=C).
B è l'unico utente che ha una voce di directory per un file di proprietà di C.

Con i collegamenti simbolici questo problema non si verifica perché solo il vero proprietario ha un puntatore per l'i-node. Quando il proprietario rimuove il file, viene distrutto.

- I collegamenti simbolici richiedono più tempo di gestione.
- Il file contenente il path deve essere letto, quindi il percorso analizzato e seguito fino a raggiungere l'i-node.
- Tutte queste attività possono richiedere accessi supplementari al disco. Inoltre è richiesto un i-node aggiuntivo per ogni link simbolico.
- I link simbolici hanno il vantaggio che possono essere utilizzati per collegare facilmente file che risiedono su macchine dislocate ovunque nel mondo.
- I link (simbolici o hard), a causa della presenza di più voci nelle directory, favoriscono l'eccesso di copia durante le operazioni di duplicazione o di backup.

File system basati su log strutturati

L’idea è di soddisfare la maggior parte delle letture direttamente dalla cache del file system(senza avere accesso al disco)

File system basati su journaling

L'idea di base è quella di tenere un diario di ciò che il file system sta per fare prima che lo faccia, in modo che in caso di crash si possa recuperare il lavoro.
Tali file system sono chiamati file system journaling e sono ampiamente utilizzati:

- NTFS di Microsoft;
- ext3 di Linux;
- ReiserFS.

Affinche il journaling funzioni occorre che le operazioni nel log devono essere **idempotenti**: possono essere ripetute tutte le volte che serve senza produrre effetti collaterali.
Per migliorare 'affidabilità, un file system può introdurre il concetto di **transazione atomica.**

- più operazioni possono essere raggruppate in una entità unica e svolta in modo atomico.

File System Virtuali

I sistemi UNIX usano il Virtual File System (VFS): la parte comune dei file system è posta su un livello astratto più alto, mentre il codice specifico su un livello più basso.
Con questo approccio è semplice aggiungere nuovi file system:

- i progettisti prima analizzano le tipologie di funzioni che il VFS può ricevere e poi scrivono il layer sottostante che permette al VFS di soddisfarle;
- se il file system esiste già, occorre realizzare delle funzioni di wrapping verso il VFS.

Protezioni dei file

usando **UNIX** i file sono protetti da un codice di protezione da 9 bit costituito da 3 ambiti: il proprietario, utenti del proprietario, tutti gli altri utenti. ogni ambito e formato da 3 bit (**RWX** bits) che definiscono le operazioni consentite:

- R = Lettura
- W = scrittura
- X = esecuzione

System call

è una procedura speciale che viene eseguita in modalità kernel a seconda dell' argomento chiamato:

1) **System call per la gestione dei file**

   tra le più utilizzate troviamo la **read()** per leggere il file e la **write()** per scrivere nel file

2) **System call per la gestione della directory**
   - la chiamata **mkdir()** crea una directory (file).
   - la chiamata **rmdir()** cancella una directory vuota
   - la chiamata **link()** fa riferimento ad un file
   - la chiamata **mount()** fonde due filesystem
3) **System call per la gestione dei processi**
   - in **POSIX** la chiamata **fork()** crea una replica del processo
   - in **UNIX** ad ogni processo è assegnato uno spazio di indirizzo suddiviso in 3 segmenti distinti
     1) **stack** , gestisce le attivazioni delle procedure
     2) **dati**, spazio per la memoria dei dati
     3) **testo**, il codice del programma

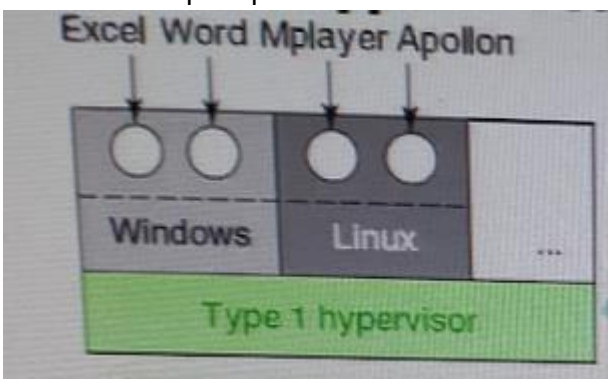
Virtualizzazione

- un sistema virtualizzato mantiene l'affidabilità di un sistema multicomputer ad un costo ridotto ed una maggiore semplificazione della manutenibilità
- un guasto sul server che gestisce le macchine virtuali produce un danno catastrofico rispetto a quello di una maggiore semplificazione della manutenibilità (la probabilità di un guasto naturale hardware è più bassa rispetto ad un guasto naturale software)

vantaggi della virtualizzazione

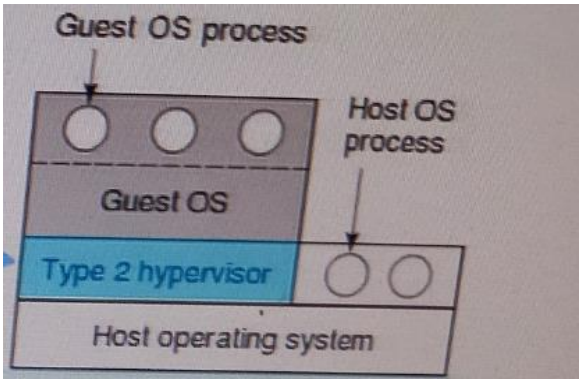
- un malfunzionamento su una macchina virtuale non inficia il comportamento delle altre
- la riduzione delle macchine fisiche riduce lo spazio, il consumo di energia, la produzione di calore quindi l'energia per il raffreddamento
- la creazione di checkpoint e la migrazione di macchine virtuali è più semplice rispetto ad un ambiente tradizionale
- si possono far girare applicazioni legacy su ambienti obsoleti che non funzionerebbero con gli attuali hardware
- i programmatori possono effettuare il test delle applicazioni su differenti SO senza disporre di hardware fisici e SO

Tipi di virtualizzazione

- esistono due differenti approcci per il monitor delle macchine virtuali: **hypervisor** di **tipo 1** e si **tipo 2**
    - **tipo 1**
      l'hypervisor è nel sistema operativo della macchina (**SO host**) e gira in modo kernel, il suo compito è di supportare le macchine virtuali (**SO guest**) così come accade per i processi e i thread
      

    - il tipo 1 è se e solo se tutte le istruzioni sono privilegiate

- la macchina virtuale è eseguita come un processo utente in modalità utente, non possono eseguire istruzioni sensibili anche se pensa di essere in modalità kernel (**modalità virtuale kernel**)
- quando il SO guest esegue una istruzione sensibile:
    1. se la CPU non ha la VT la virtualizzazione non è possibile
    2. altrimenti avviene una trap nel kernel e l'hypervisor può vedere se l'istruzione è stata inviata da:
        - ➢ una VM del SO guest, in questo caso la esegue
        - ➢ un programma utente in questo caso emula il comportamento dell'hardware reale
- **tipo 2**
  è un programma utente che interpreta le istruzioni della VM e le traduce sul SO della macchina reale: sono analizzati frammenti di codice insieme con lo scopo di incrementare le performance.



- **VMware** è un hypervisor di tipo 2 ed è eseguito come un programma utente su un qualsiasi SO host
- quando si avvia per la prima volta si comporta come un computer senza SO e cerca di installare il SO guest nel suo disco virtuale
- per eseguire un programma si utilizza una tecnica nota come **Traduzione binaria**
    - ➢ esegue la scansione del codice alla ricerca dei **blocchi base** cioè blocchi che terminano con istruzione di flusso (es: JMP, CALL, trap,…)
    - ➢ ricerca nei blocchi le istruzioni sensibili e le traduce con una procedura **VMware**
    - ➢ il blocco è messo nella chance di **VMware** e può essere eseguita alla velocità della macchina fisica (istruzione tradotte in parte)

Confronto tra hypervisor

- negli hypervisor tipo 2 tutte le istruzioni sensibili sono sostituite da chiamate a procedura che ne emulano il comportamento. il SO guest non invierà mai nessuna istruzione sensibile alla macchina fisica
- le performance delle CPU con VT sono enormemente superiori rispetto a quelle senza VT ?
    - purtroppo l'approccio "trap-and-emulare" adottato dagli hardware VT genera troppi trap ed un eccessivo overhead di gestione, mentre la traduzione delle istruzioni sensibili è più efficiente

Paravirtualizzazione

- gli hypervisor tipo 1 e 2 funzionano senza modificare il SO guest, ma con performance meno eccellenti
- un diverso approccio prevede la modifica del codice sorgente del SO guest, invece di eseguire istruzioni sensibili si effettuano chiamate di procedure definite dall hypervisor
- quindi l'hypervisor definisce un'interfaccia delle **API** (**Application Program Interface**), che i sistemi operativi guest possono attivare
- questo trasforma di fatto l'hypervisor in un microkernel e il SO guest modificato viene detto **paravirtualizzato**
- le performance ovviamente migliorano poiché le trap si trasformano in system call

Per architetture parallele vedi file "Architetture dei calcolatori"