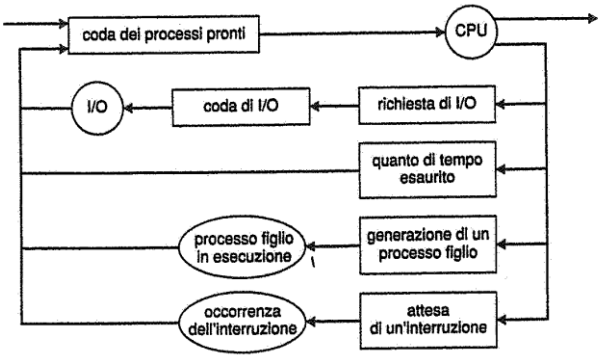
CAPITOLO 3 : I PROCESSI

1. CONCETTO DI PROCESSO

Una questione importante che sorge nell'analisi dei S.O. è la definizione delle attività. Un sistema a lotti esegue i propri job mentre un sistema a partizione del tempo esegue task. Se un sistema esegue un solo processo alla volta, ossia che richiama nella CPU un processo quando è terminato quello precedente, si chiama sistema monoutente, altrimenti multiprogrammazione. Un **processo** è un'entità attiva. Ossia è un programma in esecuzione. Un programma è un'entità passiva, è costituito da un file binario residente nel disco che contiene il codice. Un programma diventa un processo quando viene caricato in memoria ed attende di essere eseguito dalla CPU.
Un processo si può trovare in vari stati:
Nuovo: il processo viene creato
Attesa:il processo attende che si verifichi qualche evento
Esecuzione: il processo si trova nella CPU ed è eseguito
Pronto: il processo è pronto per essere assegnato alla CPU.
Terminato: il processo ha terminato la sua esecuzione.
Per tener traccia delle informazioni inerenti un processo, ogni processo viene rappresentato dal PCB. È un blocco che contiene molte informazioni sul processo quali il suo stato, lo scheduling della CPU, i registri della cpu, il program counter, informazioni sulla gestione della memoria, informazioni sulle risorse utilizzate dal processo e informazioni sullo stato di I/O.

2. SCHEDULING DEI PROCESSI

L'obiettivo della multiprogrammazione consiste nel massimizzare l'utilizzo della cpu in modo tale che risulti il maggior possibile operativa. L'obiettivo della partizione del tempo sta nel far eseguire un processo per un determinato tempo in modo tale che ad ogni utente risulti che stia utilizzando la cpu. Per questo motivo lo scheduler dei processi seleziona dalla coda dei processi un processo il quale viene posto in esecuzione nella cpu. Ogni processo è inserito nella coda dei processi, composta da tutti i processi del sistema. I processi presenti in memoria centrale che sono pronti e nell'attesa di essere eseguiti, vengono collocati nella coda dei processi pronti. Quando viene assegnato ad un processo la cpu, questo viene tolto dalla coda dei processi pronti ed eseguito finche non si verifichi o un'interruzione, o una richiesta di I/O oppure esso termini la sua esecuzione. Poichè nel sistema esistono molti processi, il disco può essere occupato con una richiesta di I/O di un altro processo. Per risolvere questo processo, ogni dispositivo ha una propria coda dei processi che attendono di ricevere informazioni da esso.
Un nuovo processo inizialmente si colloca nella coda dei processi pronti che attende di essere selezionato e gli viene assegnata la cpu. Una volta che gli viene assegnata la cpu egli vi rimane finche non si verifichi una di queste condizioni:
il processo richiede una operazione di IO e quindi viene inserito nella coda dei processi del dispositivo;
il processo può creare un nuovo processo ed attenderne la terminazione;
il processo può essere rimosso forzatamente dalla cpu a causa di un'interruzione ed essere inserito nella coda dei processi pronti.



Scheduler

Nel corso della sua esistenza, un processo si trova in varie code di scheduling. Il SO attraverso lo scheduler seleziona i processi dalle suddette code. Spesso accade che più processi vengono sottoposti in esecuzione di quanto se ne possono eseguire. Quindi questi processi vengono trasferiti in memoria secondaria, generalmente dei dischi dove si tengono fino al momento della loro esecuzione.

Lo scheduler a lungo termine provvede a scegliere questi lavori e a caricarli in memoria centrali e li inserisce nella coda dei processi pronti.

Lo scheduler a breve termine invece seleziona tra i processi pronti, i processi che devono essere assegnati alla cpu. Questi due scheduler si differenziano maggiormente per la frequenza in cui queste operazioni vengono effettuate, lo scheduler a breve temine seleziona processi molto velocemente, nell'ordine dei millisecondi mentre per lo scheduler a lungo termine possono passare anche diversi minuti prima che selezioni un nuovo processo. È importante che lo scheduler a lungo termine faccia un'accurata selezione dei processi che possono essere processo con prevalenza di IO (IO bounds) e processo con prevalenza di esecuzione (CPU bounds).

Il altri sistemi operativi, come quello a partizione del tempo, è stato introdotto un livello di scheduling intermedio. Questo scheduling a medio termine quando un processo viene tolto dalla cpu, viene tolto anche dalla memoria centrale posto in quella secondaria e successivamente viene ricaricato. Questo schema si chiama avvicendamento dei processi in memoria.

Cambio di contesto

A volte sono le interruzioni ad indurre il SO a sospendere un processo ed eseguire routine del kernel. In presenza di molte interruzioni, si avranno rispettivamente anche numerosi cambi di contesto. Nel momento in cui la cpu decide di terminare l'esecuzione del processo, il SO deve salvare ogni informazione relativa al processo in modo che quando viene ricaricata, l'esecuzione continua nel punto in cui era stata interrotta. Quindi nel momento in cui si deve effettuare un cambio di contesto, il sistema salva nel suo PCB il contesto del processo, il valore dei registri della cpu, il valore del program counter.

3. OPERAZIONE SUI PROCESSI

Creazione di un processo

Durante la propria esecuzione, un processo può crearne altri attraverso apposite chiamate di sistema. Si avrà quindi una distinzione tra processo padre e processo figlio. Ogni processo viene identificato attraverso un identificatore detto PID che può essere ottenuto attraverso la chiamata di sistema getpid(). Nel sistema UNIX per tener traccia di tutti i processi in esecuzione, basta lanciare nella shell il comando ps –el che permette di visionare tutte le informazioni sui processi. Quando si crea un nuovo processo, esso ottiene oltre alle varie risorse fisiche e logiche, i dati di inizializzazione che il processo padre passa al figlio. Quando un genitore crea un nuovo processo, per quel che riguarda l'esecuzione ci sono 2 possibilità:

- il genitore ed il figlio vengono eseguiti parallelamente
- il genitore aspetta la morte del figlio per continuare la propria esecuzione

Ci sono anche 2 possibilità per quel che riguarda gli spazi di indirizzi

il processo figlio è un duplicato del processo padre

nel processo figlio si carica un nuovo programma (attraverso la chiamata di sistema exec())

Nel momento in cui viene eseguita la chiamata di sistema fork(), questa restituisce un pid che vale 0 se si riferisce al figlio, valore > 0 se si riferisce al padre. Nel momento in cui viene effettuata la chiamata di sistema fork(), vengono copiati i dati e gli spazi di indirizzo nel figlio.

Terminazione di un processo

Un processo termina quando finisce l'esecuzione della sua ultima istruzione e inoltra la richiesta al sistema operativo di essere cancellato usando la chiamata return, exit o _exit. La terminazione di un processo può avvenire anche il atre situazioni:

- il processo figlio ha acceduto nell'uso di alcune risorse che gli sono state assegnate.
- il compito del processo figlio non è più richiesto

- il processo genitore termina e quindi come succede in alcuni SO come VMS, se il processo genitore termina, tutti i suoi figlio terminano anch'essi.

In UNIX se un processo padre termina mentre altri figli sono ancora in esecuzione, essi vengono affidati al processo init(), che assume il ruolo di nuovo genitore

4. COMUNICAZIONE TRA PROCESSI

I processi eseguiti concorrentemente, hanno bisogno di poter comunicare tra di loro per diversi motivi. Si ha una distinzione tra processi indipendenti i quali effettua la loro esecuzione senza dipendere da altri processi, oppure processi cooperanti che vengono influenzati o influenzano altri processi. La comunicazione è importante per diverse ragioni:

- Condivisione dell'informazione, poiché più utenti possono essere interessati alla stessa informazione
- Accelerazione del calcolo, se le attività vengono suddivise ed eseguite parallelamente
- Modularità
- Convenienza

Per lo scambio di informazioni tra processi, vi sono delle tecniche di comunicazione delle IPC. Tale tecniche prevedono diversi metodi per lo scambio di informazioni : attraverso memoria condivisa oppure attraverso la coda di messaggi. Lo scambio di messaggi è utile quando si devono trasmettere piccole quantità di informazioni, altrimenti è meglio usare la memoria condivisa.

Sistemi a memoria condivisa

La comunicazione tra i processi avviene sulla base di una memoria condivisa che attraverso opportune chiamate di sistema tale memoria viene collocata nello spazio degli indirizzo del processo e successivamente si può accedere a tale memoria attraverso il suo identificatore univoco. Altri processi che intendono usarla devono annetterla al proprio spazzine degli indirizzi. Ad esempio la memoria condivisa viene utilizzato dal problema del produttore/consumatore. Nella memoria condivisa vengono tenuti gli elementi prodotti dal produttore e consumati dal consumatore. Vi sono 2 tipi di buffer: buffer illimitato che non pone limiti mentre il buffer limitato pone una dimensione al numero massimo di elementi che si possono produrre.

Sistemi a scambio di messaggi

Un altro modo per la comunicazione tra processi è quello dello scambio di informazioni attraverso lo scambio dei messaggi. La comunicazione avviene senza la condivisione dello spazio degli indirizzi e tramite le chiamate di sistema send e receive. Vi possono essere diversi tipi di comunicazione:
comunicazione diretta o indiretta
comunicazione sincrona o asincrona
comunicazione automatica o esplicita.

Per comunicare si devono disporre della possibilità di far riferimento ad altri processi. Con la comunicazione diretta ogni processo che invia o riceve dati, oltre al messaggio deve specificare il processo con cui vuole comunicare. Questo tipo di comunicazione ha la seguente caratteristica: ogni processo deve chiamarsi a vicenda e viene instaurato un canale tra i 2 processi. Questo tipo di scambio è detta simmetrica e una variante detta asimmetrica avviene tramite la specificazione solo da parte del processo che manda il messaggio, la specifica del processo ricevente.
Con la comunicazione indiretta, lo scambio dei messaggi avviene tramite delle porte che sono degli oggetti in cui i processi prendono e mettono messaggi. La stessa porta deve essere condivisa da i processi che intendono usare questo tipo di comunicazione.
La comunicazione tra processi avviene attraverso chiamate delle primitive send e receive. Le chiamate possono essere bloccanti o non bloccanti. Invio sincrono- ricezione sincrona: il processo che invia/riceve il messaggio attende che l'altro processo riceve/manda il messaggio
Invio asincrono- ricezione sincrona: il processo che invia/riceve il messaggio continua la sua esecuzione.

5. COMUNICAZIONE CLIENT SERVER

Una socket è definita come l'estremità di un canale di comunicazione. Una coppia di processi che comunica attraverso una rete usa una coppia di socket, una per ogni processo e ogni socket è identificata attraverso un indirizzo IP concatenato a un numero di porta. Il cliente comunica con il server creando una socket e collegandosi per suo tramite alla porta su cui il server è in ascolto. Stabilita la connessione, il client può leggere dalla socket tramite le ordinarie istruzioni di IO. I messaggi vengono suddivisi in pacchetti ordinati correttamente ed inviati sulla socket. Si indirizzano ad un demone il quale provvederà a comporre il messaggio originale.

CAPITOLO 4: I THREAD

1. INTRODUZIONE

Un thread è un percorso di controllo d'esecuzione all'interno di un processo.
Un thread è l'unita di base d'uso della Cpu e comprende un identificatore di thread, un contatore di programma, un insieme di registri e una pila. Condivide con gli altri thread che appartengono allo stesso processo la sezione del codice, i fati, altre risorse. Un processo tradizionale, chiamato anche processo pesante, è composto da un solo thread. Un processo multithread è composto da più thread. Molti programmi per i moderni pc sono predisposti per essere eseguiti da processi multithread. Ad esempio una pagina web può assegnare un thread per la richiesta dei dati, uno per la visualizzazione a video, ecc.
I thread hanno anche il ruolo primario nei sistemi che impiegano le RPC; si tratta di un sistema che permette la comunicazione tra processi, fornendo un meccanismo di comunicazione simile alle normali chiamate di sistema. Molti kernel di sistemi operativi sono multithread.

Vantaggi

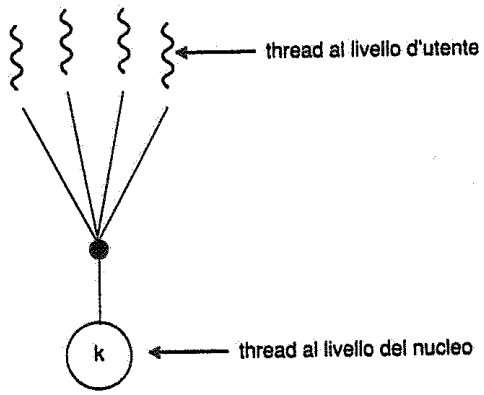
I vantaggi della programmazione multithread si possono classificare secondo 4 fattori principali:

- Tempo di risposta: rendere multithread un'applicazione interattiva può permettere a un programma di continuare la sua esecuzione, anche se una parte di esso è bloccata o sta eseguendo una operazione particolarmente lunga.
- Condivisione delle risorse: normalmente i thread condividono la memoria e le risorse del processo cui appartengono. Il vantaggio della condivisione del codice consiste nel fatto che un'applicazione può avere molti thread di attività diverse, tutti nello stesso spazio degli indirizzi
- Economia: assegnare memoria e risorse per la creazione di nuovi processi è costoso: poiché i thread condividono le risorse del processo cuoi appartengono, è molto più vantaggioso creare thread e gestirne i cambi di contesto
- Uso di sistemi multiprocessore: i vantaggi della programmazione multithread aumentano notevolmente nelle architetture multiprocessore dove i thread si possono eseguire in parallelo

2. MODELLI DI PROGRAMMAZIONE MULTITHREAD

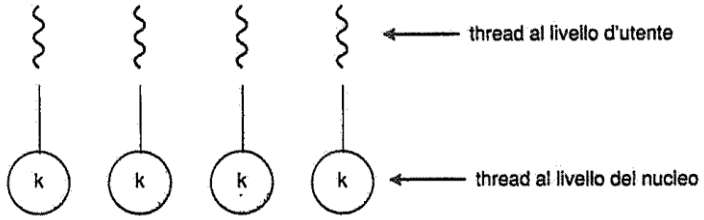
I thread possono essere distinti in thread a livello utente e thread a livello kernel: i primi sono gestiti senza l'aiuto del kernel, i secondi sono gestiti direttamene dal SO.

Modelli da molti a uno



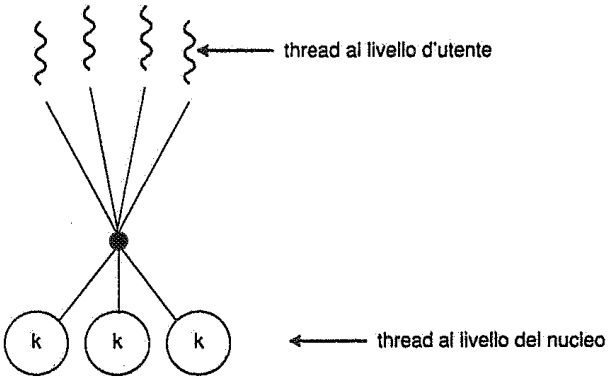
Il modello da molti a uno fa corrispondere molti thread a livello utente a un singolo thread a livello kernel. Poiché si svolge nello spazio utente, la gestione dei thread risulta efficiente, ma l'intero processo rimane bloccato se un thread invoca una chiamata di sistema di tipo bloccante. Inoltre poiché un solo thread alla volta puoi accedere al kernel, è impossibile eseguire thread multipli in parallelo in sistemi multiprocessore.

Modello da uno a uno



Il modello da uno a uno mette in corrispondenza ciascun thread a livello utente con un thread a livello kernel. Questo modello offre un grado di concorrenza maggiore rispetto al precedente, poiché anche se un thread invoca una chiamata di sistema bloccante, è possibile eseguire un altro thread. L'unico svantaggio di questo modello è che la creazione di ogni thread a livello utente comporta la creazione del corrispondente thread a livello kernel.

Modello da molti a molti



Il modello da molti a molti mette in corrispondenza di più thread a livello utente con un numero minore o uguale di thread a livello kernel; quest'ultimo può essere specifico per un carta applicazione o per un particolare calcolatore. Nonostante il modello da molti a uno permette di creare tanti thread a li vello utente quanti ne desiderino, non viene garantita una concorrenza reale. Il modello da molti a molti non presenta difetti come quelli da uno a uno e da molti a uno. I programmatori possono creare liberamente i thread che ritengono necessari e i corrispondenti thread a livello kernel si possono eseguire in parallelo nelle architetture multiprocessore. Inoltre se un thread impiega una chiamata di sistema bloccante, il kernel può fare in modo che si esegua un altro thread.

3. LIBRERIE DEI THREAD

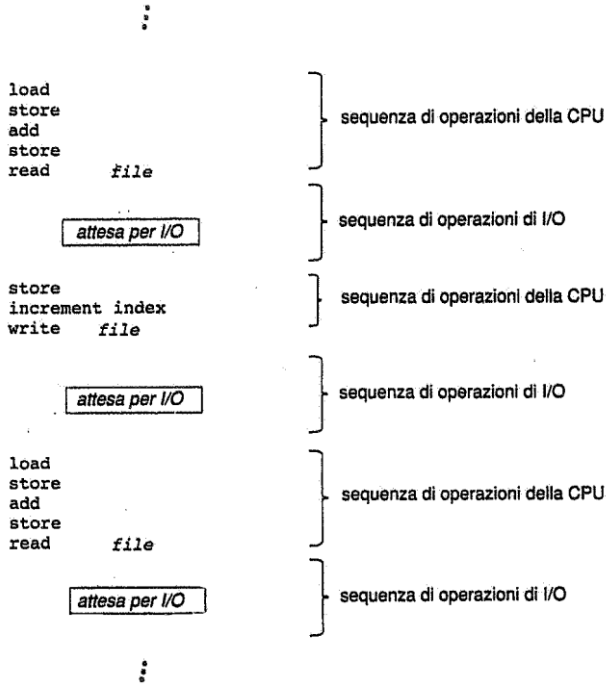
La libreria dei thread fornisce al programmatore una api per la creazione e la gestione dei thread. I metodi con cui implementare una libreria dei thread sono essenzialmente due. Nel primo la libreria è collocata interamente a livello utente, senza far ricorso al kernel. Il codice e le strutture dati per la libreria risiedono tutti nello spazio degli indirizzi. Questo implica che invocare una funzione della libreria si traduce in una chiamata locale a una funzione nello spazio degli utenti. Il secondo metodo consiste nell'implementare una libreria a livello kernel, con l'ausilio diretto del sistema operativo. In questo caso il codice e le strutture dati per la libreria si trovano nello spazio del kernel. Innovare una funzione della api per la libreria provoca una chiamata di sistema al kernel.

CAPITOLO 5: SCHEDULING DELLA CPU

1. CONCETTI FONDAMENTALI

Lo scheduling della CPU è alla base dei sistemi operativi multiprogrammati: attraverso la commutazione del controllo della CPU tra i vari processi, il sistema operativo può rendere più produttivo il calcolatore.
In un sistema monoprocessore si può eseguire al massimo un processo alla volta, gli altri processi devono aspettare che la CPU sia libera e possa essere nuovamente sottoposta a scheduling. L'idea della multiprogrammazione è relativamente semplice. Un processo è in esecuzione finche non deve attendere un evento, generalmente una richiesta di IO. Durante l'attesa la CPU resterebbe inattiva e tutto il tempo d'attesa sarebbe sprecato. Con la multiprogrammazione si cerca di rendere questi tempi produttivi, tenendo in memoria più processi e nel momento in cui quello che è in esecuzione riceve un'interruzione, questo viene messo in memoria e salvato il suo PCB e un nuovo processo viene assegnato alla CPU.

Ciclicità delle fasi d'elaborazione e di IO



L'esecuzione di un processo consiste in un ciclo di elaborazione ed un ciclo di completamento. I processi si alternano in questi due cicli fino al loro completamento. Comincia con una sequenza effettuata dalla CPU (CPU burst), seguita da una sequenza di operazioni di IO (IO burst) e cosi via fino a quando non viene inviata una richiesta di terminazione.
Un programma con prevalenza di IO produce molta sequenza di operazioni della CPU a breve durata. Un programma con prevalenza di operazioni di elaborazione, produce poche sequenze di operazioni della CPU molte lunghe. Quindi deve sussistere un buon algoritmo per la scelta di queste operazioni.

Scheduler della CPU

Ogniqualvolta la CPU passa nello stato d'inattività, il sistema operativo sceglie per l'esecuzione uno dei processi presenti nella coda dei processi pronti. In particolare lo scheduler a breve termine che sceglie tra i processi presenti in memoria ad assegnarli alla CPU.

Scheduling con diritto di prelazione

Le decisioni riguardanti lo scheduling della CPU si possono prendere nelle seguenti circostanze:

- Un processo passa dallo stato di esecuzione in uno stato di attesa
- Un processo passa dallo stato di esecuzione allo stato di pronto

- Un processo passa dallo stato di attesa allo stato di pronto
- Un processo termina

Il primo e il 4 caso non comporta alcuna scelta di scheduling; a essi segue la scelta di un nuovo processo da eseguire. Una scelta si deve fare nel caso 2 e 3.
Quando lo scheduling interviene solo nei casi 1 e 4 è detto senza diritto di prelazione altrimenti è detto con diritto di prelazione. Nel primo caso, quando si assegna la CPU ad un processo pronto, questo rimane in possesso della CPU finche non ha terminato oppure al passaggio nello stato di attesa.

Dispatcher

Un altro elemento coinvolto nella funzione di scheduling della CPU è il dispatcher: si tratta di un modulo che passa effettivamente il controllo della CPU ai processi scelti dallo scheduler a breve termine. Questa funzione riguarda:

- Il cambio di contesto
- Il passaggio alla modalità utente
- Il salto alla giusta posizione del programma utente per riavviarne l'esecuzione

Poiché si attiva ad ogni cambio di contesto, deve essere il più rapido possibile.

2. CRITERI DI SCHEDULING

Esistono diversi algoritmi di scheduling della CPU e hanno proprietà differenti e possono favorire una determinata classe di processi. Nella scelta dell'algoritmo di scheduling bisogna tener conto:

- Utilizzo della cpu: la CPU deve restare il più a lungo possibile in attività.
- Produttività: la CPU è attiva quando svolge del lavoro. Una misura del lavoro svolto è data dal numero dei processi completati nell'unita di tempo
- Tempo di completamento: ossia considerare il tempo che intercorre tra la sottomissione del processo e il completamento della sua esecuzione
- Tempo di attesa: l'algoritmo di scheduling della CPU non influisce su tempo impiegato per l'esecuzione di un processo o di un'operazione di IO. Il tempo di attesa è la somma degli intervalli d'attesa passati nella coda dei processi pronti.
- Tempo di risposta: il tempo di risposta è il tempo che intercorre tra la sottomissione di una richiesta e la prima risposta prodotta.

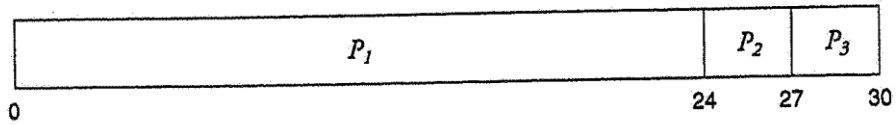
3. ALGORITMI DI SCHEDULIG

Scheduling in ordine di arrivo FCFS

Il più semplice algoritmo di scheduling della CPU è quello della scelta dei processi in base all'ordine di arrivo. Tale algoritmo noto anche come FCFS sceglie il processo in testa alla coda FIFO. Quando un processo entra nella coda dei processi pronti, si collega il suo PC all'ultimo elemento della coda. Quando la CPU è libera si assegna al processo che si trova in testa alla coda. È spesso molto lungo. Un problema che si può verificare con la scelta di tale algoritmo è l'effetto del convoglio: tutti i processi attendono che un lungo processo liberi la CPU mentre nella coda ci sono processi a breve termine che attendono un lungo periodo. È senza prelazione, una volta che la CPU viene assegna ad un processo, questo lo trattiene fino al suo rilascio.

Processo	Durata della sequenza
P_1	24
P_2	3
P_3	3

Se i processi arrivano nell'ordine P_1, P_2, P_3 e sono serviti in ordine FCFS, si ottiene il risultato illustrato dal seguente schema di Gantt:



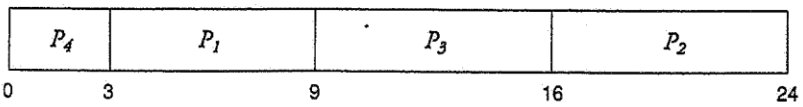
Il tempo di attesa medio è di 17 millisecondi (0+24+27)/3=17

Scheduling per brevità SJF

Un criterio diverso è quello dello scheduling della CPU per brevità noto anche come SJF. Questo algoritmo associa per ogni processo la lunghezza della successiva sequenza di operazioni. Ogni volta si assegna la CPU al processo che ha la più breve sequenza. Se 2 processi hanno la stessa sequenza allora in questo caso si applica l'algoritmo FCFS.

Processo	Durata della sequenza
P_1	6
P_2	8
P_3	7
P_4	3

Con lo scheduling SJF questi processi si ordinerebbero secondo il seguente diagramma di Gantt:



In questo caso il tempo di attesa è (3+16+9+0)/4=7.

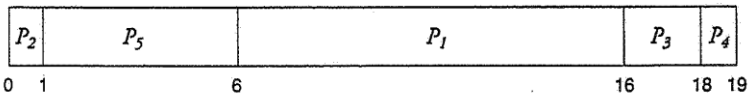
Si può dimostrare che è ottimale ma la difficoltà reale implica nel conoscere la durata della successiva richiesta di CPU. Tale algoritmo può essere sia con diritto di prelazione che senza. Nel caso di senza l'algoritmo continua la sua esecuzione così come è stato presentato, mentre con diritto se durante l'esecuzione di un processo arriva un nuovo processo in cui la sua sequenza è minore della sequenza restante del processo in esecuzione, allora tale processo viene posto nella coda dei processi pronti e quello appena arrivato viene assegnata la Cpu.

Scheduling per priorità

L'algoritmo SJF è un caso particolare del più generale algoritmo di scheduling per priorità. Tale algoritmo oltre a presentare la durata della sequenza, si associa una priorità a ogni processo e si assegna la CPU al processo con priorità più alta. Un algoritmo SJF non è altro che un algoritmo con priorità dove tale priorità è l'inversa della lunghezza.

Processo	Durata della sequenza	Priorità
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Usando lo scheduling per priorità, questi processi sarebbero ordinati secondo il seguente schema di Gantt:



Il tempo medio di attesa è di 8.2 millisecondi.

Le priorità si possono usare sia internamente che esternamente. Quelle internamente usano una o più quantità misurabili per calcolare la priorità del processo. Quelle esterne invece riguardano criteri esterni al sistema operativo. L'algoritmo per priorità è sia con diritto che senza diritto di prelazione.

Un problema importante è il problema della starvation (attesa indefinita). Un processo pronto per l'esecuzione che non dispone della CPU si può considerare bloccato nell'attesa nel caso questo abbia una priorità bassa e non venga mai eseguito.

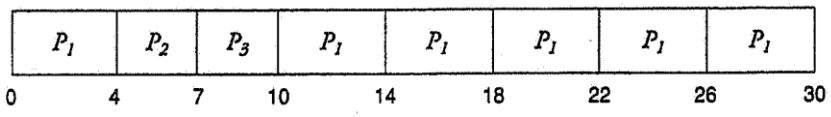
Una soluzione a questo problema è costituita dall'invecchiamento. Ogni qualvolta passa un periodo di tempo un processo che sta nell'attesa della CPU ottiene una aumento graduale della priorità facendo cosi che venga prima o poi eseguito.

Scheduling circolare Round Robin

L'algoritmo circolare noto anche come Round Robin è stato progettato appositamente per i sistemi a partizione del tempo. Viene scelto un quanto di tempo che varia tra i 10 e i 100 millisecondi e

trascorso questo lasso di tempo il processo viene interrotto e inserito nuovamente nella coda dei processi pronti. La coda dei processi viene gestita come una coda FIFO circolare.
Nell'algoritmo RR la CPU si assegna a un processo per non più di un quanto di tempo per volta. Le prestazioni sono molto dipendenti alla scelta del quanto di tempo in quanto se è troppo lungo si avvicina all'algoritmo FCFS se troppo breve il criterio RR si chiama condivisione della CPU in quanto ad ogni utente sembra di disporre di una propria CPU ma che sia 1/n della velocità reale della CPU. Bisogna considerare il cambio di contesto che se vengono effettuate continuamente si porta ad uno spreco di tempo nei cambi di contesto. Il quanto di tempo deve quindi essere nettamente superiore al cambio di contesto in modo che ci possa essere una buona quantità di operazioni prima che il processo viene stoppato.

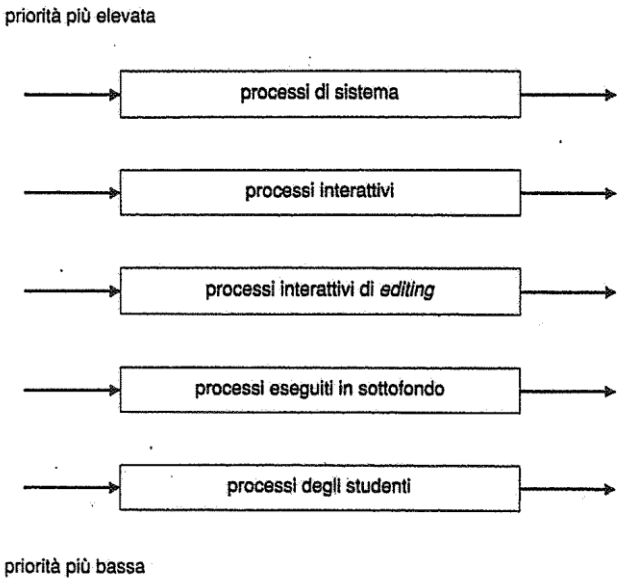
Processo	Durata della sequenza
P_1	24
P_2	3
P_3	3



Il tempo d'attesa medio è di 17/3 = 5,66 millisecondi.

Scheduling a code multiple

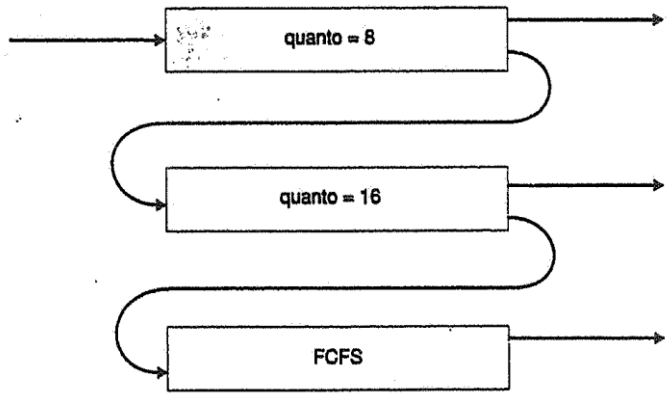
E' stata creata una classe di algoritmi di scheduling adatta a situazioni in cui i processi si possono classificare facilmente in gruppi diversi. Una distinzione diffusa è per esempio quella che si fa tra i processi che si eseguono in foreground e quelli in background. Un algoritmo a code multiple suddivide la coda dei processi pronti in code distinte, i quali vengono assegnati in modo permanente ad una coda ed ogni coda ha un proprio algoritmo di scheduling.
Ogni coda ha la priorità assoluta sulle code di priorità più bassa. Nessun processo della coda dei processi in sottofondo può iniziare l'esecuzione finche le code per i processi di sistema non siano tutte vuote.



Scheduling a code multiple con retroazione

Di solito in un algoritmo di scheduling a code multiple i processi vengono assegnati ad una coda e non possono quindi cambiarla. Lo scheduling a code multiple con retroazione permette ai processi di spostarsi tra le code. L'idea consiste nel separare i processi che hanno caratteristiche diverse nelle

sequenze delle operazioni della CPU. Si attua una forma di invecchiamento che impedisce il verificarsi di un'attesa indefinita.



Lo scheduler fa eseguire tutti i processi della coda 0 attuando l'algoritmo RR con quanto=8, non appena si svuota esegue quelli in coda 1 con RR a quanto=16, infine quelli nella coda 2 con algoritmo FCFS. All'inizio tutti vengono collocati nella coda 0, i processi che non terminano vengono collocati nella coda 1 che a loro volta se non terminano vengono collocati nella coda 2.
La definizione di scheduling a code multiple con retroazione costituisce il più grande criterio di scheduling della CPU, che nella fase di progettazione si può adeguare a un sistema specifico.

4. SCHEDULING PER SISTEMI MULTIPROCESSORE

Soluzioni di scheduling per multiprocessore

Se sono disponibili più unita d'elaborazione, anche il problema dello scheduling è proporzionalmente più complesso.
Una prima strategia di scheduling della CPU per i sistemi multiprocessore affida tutte la decisione ad un CPU detta master e tutti i processi utenti alle altre cpu. Quando invece ciascun processore provvede al proprio scheduling, si parla di multielaborazione simmetrica SMP.

Predilezione per il processore

Si consideri cosa accade alla memoria cache non appena che un processo abbia terminato la sua esecuzione nella CPU: i dati che ha trattato per ultimo permango nella cache, se il processo viene continuato da un altro processore, i contenuti della memoria cache devono essere invalidati sul processore di partenza e convalidati sull'altro. A causa di alti costi di svuotamento di memoria cache, si applicano delle regole impedendo ad un processo di passare da un processore ad un altro.

Bilanciamento del carico

Sui sistemi SMP è importante che il carico sia distribuito uniformemente tra i vari processori in modo da non permettere che un processore sia saturo un altro sia inattivo. Il bilanciamento del carico quindi tenta di ripartire il carico di lavoro uniformante tra i processi. Ci sono 2 tecniche di bilanciamento: migrazione guidata e migrazione spontanea. La prima prevede che un processo controlli costantemente il carico di ogni processore e quando verifica che uno sia saturo, sposta i processi su altri processori. La 2 invece fa si che un processore inattivo sottragga spontaneamente processi a processori saturi.

Multithread simmetrico

I sistemi SPM consentono l'esecuzione contemporanea, su processori fisici multipli di numerosi thread. Una possibile alternativa è quello che ha fatto l'intel noto come multithread simmetrico SMT che fornisce diversi processori logici. L'idea che caratterizza la SMT è la creazione di più processori logici basati sullo stesso processore fisico cosi da presentare al sistema operativo una serie di processori anche in presenza di uno solo fisico. Bisogna sapere che la SMT è a livello fisico, no software.

5. SCHEDULING DEI THREAD

Sui sistemi operativi che prevedono la presenza dei thread, il sistema pianifica l'esecuzione dei thread a livello kernel, non dei processi. I thread a livello utente sono gestiti da una libreria, il kernel non è consapevole della loro esistenza. Di conseguenza per eseguire i thread a livello utente occorre associare loro dei thread a livello kernel.

6. VALUTAZIONE DEGLI ALGORITMI

Ci si può chiedere quale algoritmo di scheduling della CPU bisogna applicare. Per scegliere un algoritmo occorre innanzitutto stabilire l'importanza relativa di queste misure. Bisogna rendere massimo l'utilizzo della CPU con il vincolo che il massimo tempo di risposta sia di 1 secondo e rendere massima la produttività in modo che il tempo di completamento si linearmente proporzionale al tempo d'esecuzione totale.

La valutazione analitica, secondo l'algoritmo dato e il carico di lavoro del sistema, fornisce una formula o un numero che valuta le prestazioni dell'algoritmo per quel carico di lavoro. Consiste nel valutare gli algoritmi su dati concreti, i risultati ottenuti sono numeri esatti che consentono il confronto tra gli algoritmi.

Reti di code

Il sistema di calcolo si descrive come una rete di server, ciascuno con una coda d'attesa. La CPU è un server con la propria coda dei processi pronti e il sistema di IO ha le sue code dei dispositivi. Se sono noti l'andamento degli arrivi e dei servizi, si possono calcolare la lunghezza media delle code e il tempo medio d'attesa e cosi via.

Simulazioni

Per riuscire ad avere una valutazione più precisa degli algoritmi di scheduling, ci si può servire di simulazioni. Le simulazioni implicano la programmazione di un modello del sistema di calcolo; le strutture dati rappresentano gli elementi principali del sistema.

Durante l'esecuzione della simulazione si raccolgono i dati che vengono stampati e confrontati. Poiché per effettuare una buona simulazione richiedono diverse ore del tempo di d'elaborazione, le simulazioni possono essere tuttavia molto onerose. Una simulazione dettagliata da risultati molto precisi ma richiede anche una grande quantità di tempo.

Realizzazione

Persino una simulazione ha dei limiti per quel che riguarda la precisione. L'unico modo assolutamente sicuro per valutar un algoritmo di scheduling consiste nel codificarlo, inserirlo nel sistema operativo e osservarne il comportamento nelle reali condizioni di funzionamento. Le spese sono dovute alla codifica dell'algoritmo e alle modifiche da fare al sistema operativo e alle reazioni degli utenti sulle modifiche del sistema operativo.

CAPITOLO 6: SINCRONIZZAZIONE DEI PROCESSI

1. INTRODUZIONE

Un processo cooperante è un processo che può influenzarne un altro in esecuzione nel sistema o anche subirne l'influenza. I processi cooperanti possono condividere direttamente uno spazio logico di indirizzi.

Parlando del problema del produttore/consumatore e rifacendoci al concetto di buffer. Come sappiamo la nostra memoria può condividere massimo DIM_BUFFER -1 elementi. Quindi per tenere traccia del numero di elementi si aggiunge una variabile, contatore, inizializzata a 0 che incrementa ad ogni produzione e decrementa ad ogni consumo.

```
while (1){
     /* produce un elemento in appena_prodotto */
     while (contatore == DIM_VETTORE)
        ; /* non fa niente */
     vettore[inserisci] = appena_prodotto;
     inserisci = (inserisci + 1) % DIM_VETTORE;
     contatore++;
}
```

Il codice per il processo consumatore si può modificare come segue:

```
while (1){
     while (contatore == 0)
        ; /* non fa niente */
     da_consumare = vettore[preleva];
     preleva = (preleva + 1) % DIM_VETTORE;
     contatore--;
     /* consuma un elemento in da_consumare */
}
```

Le procedure del produttore e del consumatore possono non funzionare altrettanto correttamente se si eseguono in modo concorrente. Si supponga per esempio che il valore della variabile contatore sia attualmente 5, e che i processi produttore e consumatore eseguano le istruzioni contatore++ e contatore- - in modo concorrente. Si può avere attraverso le istruzioni del calcolatore un risultato diverso da quello che dovrebbe essere. Per evitare situazioni di questo tipo, il cui più processi accedono e modificano gli stessi dati in modo concorrente e i risultati dipendono dall'ordine degli accessi occorre assicurare che un solo processo alla volta possa modificare la variabile contatore. Questa condizione richiede una forma di sincronizzazione dei processi.

2. PROBLEMA DELLA SEZIONE CRITICA

Si consideri un sistema composta di n processi (P0, P1, …,PN-1), ciascuno avente un segmento di codice chiamata sezione critica in cui il processo può modificare variabili comuni, aggiornare una tabella, scrivere in un file. Quando un processo è in esecuzione nella propria sezione critica, non si deve consentire a nessun altro processo di essere in esecuzione nella propria sezione critica. Quindi, l'esecuzione delle sezioni critiche da parte dei processi è mutuamente esclusiva nel tempo. Il problema della sezione critica si affronta progettando un protocollo che i processi possono utilizzare per comprare. Ogni processo deve chiedere il permesso per entrare nella propria sezione critica. La sezione di codice che realizza questa richiesta è la sezione d'ingresso. La sezione critica può essere seguita da una sezione d'uscita e la restante parte del codice è detta sezione non critica. Una soluzione del problema della sezione critica deve soddisfare i tre requisiti:

- **Mutua esclusione**. Se il processo Pi è in esecuzione nella sua sezione critica, nessun altro processo può essere in esecuzione nella propria sezione critica.
- **Progresso**. Se nessun processo è in esecuzione nella sua sezione critica e qualche processo desidera entrare nella propria sezione critica, solo i processi che si trovano fuori delle

rispettive sezioni non critiche possono partecipare alla decisione riguardante la scelta del processo che può entrare per primo nella propria sezione critica.

- **Attesa limitata**. Se un processo ha gia 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.

Soluzione per 2 processi

I processi sono indicati con P0 e P1. una volta introdotto un processo Pi l'altro è Pj dove j=1-i.

```
do{
    sezione d'ingresso
        sezione critica
    sezione d'uscita
        sezione non critica
} while (1);
```

Struttura generale di un tipico processo P_i.

Algoritmo 1

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

Il primo tentativo di soluzione consiste nel far condividere ai processi una variabile intera, turno, inizializzata a 0. se turno==i si permette al processo Pi di entrare nella propria sezione critica. Questa soluzione assicura che in un dato momento un solo processo può trovarsi nella propria sezione critica. Tuttavia la soluzione non soddisfa il requisito di progresso poiché richiede una stretta alternanza dei processi nell'esecuzione della sezione critica. Se ad esempio turno==0, P1 non può entrare nella propria sezione critica, anche se P0 si trova nella propria sezione non critica.

Algoritmo 2

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

L'algoritmo 1 non possiede informazioni sufficienti sullo stato di ogni processo; ricorda solo il processo cui si permette l'ingresso nella propria sezione critica. Per questo motivo si sostituisce la variabile turno con un vettore boolean pronto [2] che viene inizializzato a false. Se pronto[i] è true significa che Pi può entrare nella sezione critica, altrimenti entra Pj. Inizialmente viene posto pronto[i]=true permettendo al processo Pi di entrare nella sezione critica verificando che Pj non sia pronto per entrare nella propria sezione critica. Questa soluzione soddisfa il requisito di mutua esclusione ma no quello di progresso. Nel momento in cui sia pronto[i] che pronto[j] diventano true, Pi e Pj entrano nel ciclo infinito. Questo accade quando nello stesso momento vengono poste sia pronto[i] che pronto[j] a true.

Algoritmo 3 o soluzione di PETERSON

La soluzione di Peterson si applica a 2 processi, ognuno dei quali esegue alternativamente la propria sezione critica e la sezione rimanente. Vengono introdotte la variabile:
int turno (segnala di chi è il turno)
boolean pronto[2] (indica se un processo sia pronto ad entrare nella propria sezione critica)

```
do {
    pronto[i] = true;
    turno = j
    while (pronto[j] && turno == j);

        sezione critica

    pronto[i] = false;

        sezione non critica

} while (1);
```

Per dimostrare che la mutua esclusione è preservata si osserva che Pi acceda alla propria sezione critica solo se pronto [j]=false oppure turno=i.
Poiché tuttavia Pi non modifica il valore della variabile turno durante l'esecuzione dell'istruzione while, Pi entrerà nella sezione critica (pregresso) dopo che Pj abbia effettuato non più di un ingresso (attesa limitata).

Soluzione per più processi: algoritmo del fornaio

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

        sezione critica

    numero[i] = 0;

        sezione non critica

} while (1);
```

L'algoritmo 3 risolve il problema della sezione critica per due processi, mentre l'algoritmo del fornaio lo risolve per n processi. È basato su uno schema di servizio usato nella panetteria dove si deve evitare la confusione dei turni. Al suo ingresso nel negozio ogni cliente riceve un numero. Si serve progressivamente il cliente con il più basso. A parità di numero si serve il cliente con il nome minore.
Boolean scelta [n]
Int numero
(a,b)<(c,d) se a<c oppure se a=c e b<d

3. HARDWARE PER LA SINCRONIZZAZIONE

In generale si può affermare che qualunque soluzione al problema richiede l'uso di un semplice strumento detto lock. Il corretto ordine degli accessi alle strutture dati del kernel è garantito dal fatto che lezioni critiche sono protette da lock. In altri termini un processo per accedere alla propria sezione critica deve ottenere il permesso di un lock.
In un sistema dotato di una singola CPU tale problema si potrebbe risolvere semplicemente se si potessero interidre le interruzioni. Non si potrebbe eseguire nessunaltra istruzione quindi non si potrebbe apportare alcuna modifica inaspettata alle variabili condivise. È questo l'approccio seguito dal kernel senza diritto di prelazione.
Le interruzioni portano uno spreco del tempo e per questo motivo molte architetture offrono particolari istruzioni che permettono di controllare e modificare il contenuto di una parola di memoria oppure di scambiare il contenuto di due parole di memoria.

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

L'istruzione TestAndSet è eseguita atomicamente, cioè come un'unita non soggetta ad interruzioni.

```
do {
    while (TestAndSet(blocco));
        sezione critica
    blocco = false;
        sezione non critica
} while (1);
```

L'istruzione Swap agisce sul contenuto di 2 parole di memoria, anche'essa eseguita atomicamente. La mutua esclusione si garantisce dichiarando e inizializzando a false una variabile globale lock.

```
void Swap(boolean &a, boolean &b) {
    boolean temp = a;
    a = b;
    b = temp;
}
```

```
do {

    chiave = true;
    while (chiave == true)
        Swap(blocco,chiave);

    sezione critica

    blocco = false;

    sezione non critica

} while (1);
```

Il seguente algoritmo sfrutta l'istruzione TestAndSet per soddisfare tutti e tre i requisiti desiderati.

```
do {

    attesa[i] = true;
    chiave = true
    while (attesa[i] && chiave)
        chiave = TestAndSet(blocco);
    attesa[i] = false;

    sezione critica

    j = (i + 1) % n;
    while ((j != i) && !attesa[i])
        j = (j + 1) % n;
    if (j == i)
        blocco = false
    else
        attesa[j] = false;

    sezione non critica

} while (1);
```

Le strutture dati sono:
boolean attesa [n]=false
boolean lock=false.
Per soddisfare il requisito di mutua esclusione Pi entra nella propria sezione critica solo se attesa[i]=false oppure chiave=false.éer dimostrare che soddisfi il requisito del pregresso basta notare che un processo che esce dalla sezione critica imposta lock al valore false oppure attesa [j] a false entrambe consentono a un processo in attesa l'ingresso nella propria sezione critica.

4. SEMAFORI

Le varie soluzioni hardware al problema della sezione critica basate su istruzioni quali TestAndSet e Swap complicano l'attività del programmatore. Per ovviare questo problema si fa uso dei semafori.

Un semaforo S è una variabile intera che si può accedere, escludendo l'inizializzazione, solo tramite due operazioni atomiche: wait e signal.

```
wait(S) {
     while(S <= 0)
        ;//non-op;
     S--;
}

 signal(S) {
      S++;
 }
```

Tutte le modifiche del semaforo sono contenute nelle operazioni wait e signal.

Uso dei semafori

Si usa distinguere tra semafori contatore il cui valore è illimitato e semafori binari il cui valore è 0 o 1. I semafori sono utilizzati per risolvere il problema della sezione critica con n processi che condividono un semaforo chiamato mutex inizializzato a 1.

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

I semafori contatore vengono utilizzati nel controllo dell'accesso a una data risorsa. I processi che vogliono utilizzare il semaforo usano wait decrementando il valore. Quando restituiscono la risorsa effettuano la signal incrementando il valore.

Realizzazione

Il principale svantaggio dell'uso dei semafori è che richiede una condizione di attesa attiva. Mentre un processo si trova nella propria sezione critica, qualsiasi altro processo che tendi di entrarvi si trova sempre nel ciclo del codice della sezione di ingresso. Questo costituisce per un sistema a multiprogrammazione poiché l'attesa attiva spreca cicli alla CPU. Per superare l'attesa attiva si possono modificare signal e wait . quando un processo deve attendere anziché entrare nella attesa attiva si blocca se stesso e quindi la CPU sceglie un altro processo. Un processo bloccato che attende a un semaforo si riavvia attraverso l'operazione wakeup.

La lista rappresenta i processi in attesa a un semaforo. L'operazione signal preleva un processo da tale lista e lo attiva. L'operazione block sospende il processo e l'operazione wakeup pone in stato di pronto per l'esecuzione il processo bloccato. Se il valore del semaforo è negativo, la sua dimensione è data dal numero dei processi che attendono a quel semaforo. I semafori devono essere eseguiti in modo atomico. Nei sistemi multiprocessore è necessario disabilitare le interruzioni di tutti i

processori perché altrimenti le istruzioni dei diversi processi in esecuzione su processori distinti potrebbero interferire fra loro. Le operazioni wait e signal non eliminano completamente l'attesa attiva ma si limitano a rimuoverla dalla sezione di ingresso.

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

void wait(semaforo S) {
     S.valore--;
     if (S.valore < 0) {
        aggiungi questo processo a S.L;
        block();
     }
}

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

Stallo e attesa indefinita

La realizzazione di un semaforo con coda d'attesa può condurre a situazioni in cui ciascun processo di un insieme di processi attende indefinitamente un evento che può essere causato solo da uno dei processo dello stesso insieme. Quando si verifica una situazione di questo tipo ci troviamo in una situazione di stallo. Un insieme di processi è in stallo se ciascun processo dell'insieme attende un evento che può essere causato solida un altro processo dell'insieme. Un'altra questione connessa alle situazioni d istallo è quella dell'attesa indefinita.

5. PROBLEMI TIPICI DI SINCRONIZZAZIONE

Produttori consumatori con memoria limitata

Il problema dei produttori/consumatori con memoria limitata si usa generalmente per illustrare la potenza delle primitive di sincronizzazione. Si supponga di disporre di una certa quantità di memoria rappresentata da un buffer con n posizioni, ciascuna capace di contenere un elemento. Il semaforo mutex garantisce la mutua esclusione degli accessi al buffer ed è inizializzato al valore 1. i semafori vuole e piene conteggiano rispettivamente il numero delle posizioni vuote e il semaforo piene si inizializza a 0.

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

Figura 7.12 Struttura generale del processo produttore.

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

Figura 7.13 Struttura generale del processo consumatore.

Problema dei lettori-scrittori

Si consideri una base di dati da condividere tra numerosi processi concorrenti. Alcuni processi possono richiedere solo la lettura del contenuto mentre altri possono scriverci. Qualsiasi può effettuare letture concorrenti ma nel momento in cui uno scrive, l'altro non può ne leggere ne scrivere altrimenti vi è una incoerenza di dati. Per impedire tale problema è necessario che gli scrittori abbiano accesso esclusivo alla base dati condivisa. Vi possono essere 2 soluzioni a tale problema: il primo che nessun lettore attenda a meno che uno scrittore abbia gia la risorsa, la seconda che lo scrittore non attenda.
Entrambe le soluzioni presenta il problema della starvation sullo scrittore nel primo caso e nel lettore nel secondo. La soluzione prevede l'utilizzo di un semaforo.

```
wait(scrittura);
    ...
   esegui l'operazione di scrittura
    ...
signal(scrittura);
```

Struttura generale di un processo scrittore.

```
wait(mutex);
numlettori++;
if (numlettori == 1)
    wait(scrittura);
signal(mutex);
    ...
    esegui l'operazione di lettura
    ...
wait(mutex);
numlettori--;
if (numlettori == 0)
    signal(scrittura);
signal(mutex);
```

Struttura generale di un processo lettore.

Semaforo mutex, scrittura
Int numlettori;
i semafori mutex e scrittura sono inizializzati a 1, numlettori a 0. scrittura è comune a entrambi i processi. Il semaforo mutex si usa per assicurare la mutua esclusione. La soluzione del problema dei lettori-scrittori sono state generalizzate su alcuni sistemi in modo da fornire lock di lettura e scrittura.

Problema dei 5 filosofi

Cinque filosofi siedono ad una tavola rotonda con un piatto di spaghetti davanti, una forchetta a destra e una forchetta a sinistra (bastoncini cinesi secondo un'altra versione). Ci sono dunque cinque filosofi, cinque piatti di spaghetti e cinque forchette.

Si immagini che la vita di un filosofo consista di periodi alterni di mangiare e pensare, e che ciascun filosofo abbia bisogno di due forchette per mangiare, ma che le forchette vengano prese una per volta. Dopo essere riuscito a prendere due forchette il filosofo mangia per un po', poi lascia le forchette e ricomincia a pensare. Il problema consiste nello sviluppo di un algoritmo che impedisca lo stallo (deadlock) o la morte d'inedia (starvation). Il deadlock può verificarsi se ciascuno dei filosofi tiene in mano una forchetta senza mai riuscire a prendere l'altra. Il filosofo F_1 aspetta di prendere la forchetta che ha in mano il filosofo F_2, che aspetta la forchetta che ha in mano il filosofo F_3, e così via in un circolo vizioso. La situazione di starvation può verificarsi indipendentemente dal deadlock se uno dei filosofi non riesce mai a prendere entrambe le forchette. La presa di forchette è analoga al blocco di risorse limitate nella programmazione reale, situazione nota con il nome di *concorrenza*. Una semplice soluzione è quella di rappresentare ogni bacchetta con un semaforo. Questa soluzione garantisce che 2 vicini non mangino contemporaneamente.

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

Figura 7.17 Struttura del filosofo *i*.

6. MONITOR

Benché i semafori costituiscono un meccanismo pratico ed efficace per la sincronizzazione dei processi, il loro uso scorretto può generare errori difficili da individuare in quanto si manifestano solo in determinate situazioni

Se invece per errore di un programmatore o errore negligenza di programmazione vengo scambiati i semafori, allora il problema non si risolverà mail.

Uso del costrutto monitor

Un tipo di dato astratto che incapsula dati privati e dispone di metodi propri è il monitor. Al suo interno vengono definite una serie di operazioni che sono contraddistinte dalla muta esclusione.

```
monitor nome_monitor
{
    dichiarazioni di variabili condivise

    procedure body P1 (...) {
       ...
    }
    procedure body P2 (...) {
       ...
    }
       .
       .
       .
    procedure body Pn (...) {
       ...
    }
    {
       codice d'inizializzazione
    }
}
```

Figura 7.19 Sintassi di un monitor.

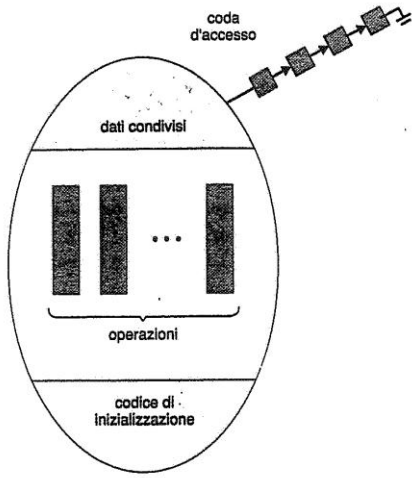


Figura 7.20 Schema di un monitor.

Pertanto una procedura al suo interno ha accesso unicamente alle variabili dichiarate localmente situate nel monitor. Il costrutto monitor assicura che all'interno di un monitor possa esser attivo un solo processo alla volta. Un programmatore definisce una o più variabili condizionali: le uniche operazioni su tali variabili sono le operazioni di signal e wait. Ci sono 2 possibilità:

segnalare e attendere: P attende Q che lasci il monitor o attende su un'altra variabile condition

segnalare e proseguire: Q attende che P lasci il monito o attende su un'altra variabile condition.

Soluzione al problema dei 5 filosofi per mezzo del monitor

I concetti relativi al costrutto monitor presentando una soluzione esente da stallo del problema dei 5 filosofi. Esistono vincoli nel quale un filosofo può prendere la forchetta solo nel caso in cui abbia entrambe le forchette disponibili.

La distribuzione delle bacchette è controllata dal monitor. Ciascun filosofo prima di cominciare a mangiare deve invocare la funzione prende; ciò può determinare la sospensione del filosofo. Completata con successione l'operazione, il filosofo può mangiare. Infine posa le bacchette e si mette a pensare.

```
monitor fc
{
    enum {pensa, affamato, mangia} stato[5];
    condition auto[5];

    void prende(int i) {
       stato[i] = affamato;
       verifica(i);
       if (stato[i] != mangia)
          auto[i].wait();
    }

    void posa(int i) {
       stato[i] = pensa;
       verifica((i + 4) % 5);
       verifica((i + 1) % 5);
    }

    void verifica(int i) {
       if ((stato[(i + 4) % 5] =! mangia) &&
          (stato[i] == affamato) &&
          (stato[(i + 1) % 5] != mangia)) {
             stato[i] = mangia;
             auto[i].signal();
       }
    }

    void inizializzazione() {
       for (int i = 0; i < 5; i++)
          stato[i] = pensa;
    }
}
```

CAPITOLO 7: STALLO DEI PROCESSI

In un ambiente multiprogrammazione più processi possono competere per ottenere un numero finito di risorse. Se le risorse richieste sono trattenute da altri processi a loro volta nello stato di attesa, il processo potrebbe non cambiare più il suo stato. Situazioni di questo tipo sono chiamate di stallo.

1. MODELLO DEL SISTEMA

Un sistema è composto da un numero finito di risorse da distribuire tra i vari processi in competizione. Le risorse sono suddivise in tipi differenti, ciascuno promanato da un certo numero di istanze identiche. Cicli di CPU, spazio di memoria, file e dispositivi sono tutti tipi di risorsa. Se un sistema ha due unita d'elaborazione, tale tipo di risorsa ha due istanze. Se un processo richiede un'istanza relativa a un tipo di risorsa, l'assegnazione di qualsiasi istanza di quel tipo può soddisfare la richiesta. Prima di adoperare una risorsa, un sistema deve richiederla e dopo averla usata deve rilasciarla. Un processo può richiedere tutte le risorse necessarie per eseguire il compito assegnatoli, anche se il numero delle risorse richieste non può superare il numero totale delle risorse presenti nel sistema. Nelle ordinarie condizioni di funzionamento un processo per servirsi di una risorsa deve:

- **Richiesta.** Se la richiesta non si può soddisfare immediatamente, il processo deve attendere
- **Uso.** Il processo può adoperare la risorsa
- **Rilascio.** Una volta terminato l'uso della risorsa, deve rilasciarla

La richiesta e il rilascio avvengono attraverso chiamate di sistema quali request e release. Una tabella del sistema memorizza tutti gli stati di ogni risorsa e se questa viene assegnata, indica il processo relativo. Un gruppo di processi entra in stallo quando tutti i processi del gruppo attendono un evento che può essere causato solo da un altro processo che si trova nello stesso stato di attesa.

2. CARATTERIZZAZIONE DELLE SITUAZIONI DI STALLO

In una situazione di stallo, i processi non terminano mai l'esecuzione e le risorse del sistema vengono bloccate impedendo l'esecuzione di altri processi.

Condizioni necessarie

So può avere una situazione di stallo solo se si verificano contemporaneamente queste 4 condizioni:

- **Mutua esclusione.** Almeno una risorsa deve essere non condivisibile, vale a dire che è utilizzabile da un solo processo alla volta. Se un altro processo richiede tale risorsa si deve ritardare il processo finche la risorsa non diventi disponibile
- **Possesso e attesa.** Un processo in possesso di almeno una risorsa attende di acquisire risorse gia in possesso di altri processi.
- **Impossibilita di prelazione.** Non esiste un diritto di prelazione sulle risorse vale a dire che una risorsa può essere rilasciata volontariamente solo da processo che la sta utilizzando
- **Attesa circolare.** Deve esistere un insieme di processi (P0,P1,…Pn-1) tale che P0 attende una risorsa da P1, P1 una di P2, …, Pn-1 una di P0.

Grafo di assegnazione delle risorse

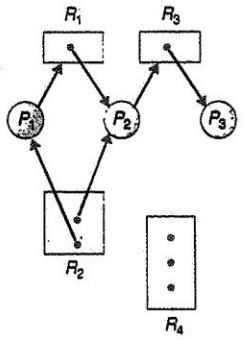
Le situazioni di stallo si possono descrivere con maggior precisione avvalendosi di una rappresentazione detta grafo di assegnazione delle risorse.

Si tratta di un insieme di vertici V e un insieme di archi E. un arco diretto dal processo Pi alla risorsa Rj significa che il processo ha richiesto la risorsa. Un argo dalla risorsa al processo significa che la risorsa è assegnata al processo. Un arco dal processo alla risorsa è detto arco di richiesta, un arco dalla risorsa al processo è detto arco di assegnazione. Quando un processo richiede l'uso di una risorsa, si inserisce l'arco dal processo alla risorsa e se viene assegnata subito, l'arco immediatamente diventa di assegnazione e al suo rilascio viene cancellato.

Gli stati dei processi possono essere:

il processo P1 possiede un'istanza del tipo di risorsa R2 e attende una di R1 e cosi via…

- insiemi P, R ed E,
  - $P = \{P_1, P_2, P_3\}$,
  - $R = \{R_1, R_2, R_3, R_4\}$,
  - $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_1, R_2 \rightarrow P_2, R_3 \rightarrow P_3\}$;



Se ciascun tipo di risorsa ha esattamente un'istanza, allora l'esistenza di un ciclo implica la presenza di uno stallo; se il ciclo riguarda solo un insieme di risorse ciascuno dei quali ha solo un'istanza, allora si è verificato uno stallo.

3. METODI PER LA GESTIONE DELLE SITUAZIONI DI STALLO

La situazione di stallo si può affrontare in 3 modi:

- Si può usare un protocollo per prevenire le situazioni di stallo
- Si può permettere al SO di entrare in stallo individuarlo e ripristinarlo
- Si può ignorare che il problema di stallo fingendo che non esistono.

La 3 soluzione è quella adottata dalla maggior parte dei sistemi operativi.
Per assicurare che non si verifichi mai uno stallo, il sistema può servirsi di metodi di prevenzione o di metodi per evitare. **Prevenire le situazioni di stallo** significa far uso di metodi atti ad assicurare che non si verifichi almeno una delle condizioni necessarie. **Evitare le situazioni di stallo** occorre che il sistema abbia in anticipo informazioni aggiuntive riguardante le risorse che un processo richiederà ed userà. Se un sistema non attui una prevenzione o un algoritmo per evitare le situazioni di stallo, allora esse possono verificarsi senza capire cosa sia successo.

4. PREVENIRE LE SITUAZION DI STALLO

Affiche si abbia uno stallo si devono verificare 4 condizioni necessarie; perciò prevenire il verificarsi di uno stallo assicurando che almeno una non si verifichi.

. **Mutua esclusione**

Deve valere la condizione mutua esclusione solo per le risorse non condivisibili. Le risorse condivisibili non causano mai uno stallo quindi non bisogna mai attendere.

Possesso e attesa

Per assicurare che la condizione di possesso e attesa non si presenti mai nel sistema, occorre garantire che un processo che richiede una risorsa non ne possegga altre. Si può usare un protocollo che ponga la condizione che ogni processo prima di iniziare la propria esecuzione richieda tutte le risorse. Questo è uno spreco perché una risorsa può essere utilizzato quasi alla fine e quindi resterebbe per un lungo periodo inattiva. Un altro protocollo è quello che un processo per poter richiedere ed utilizzare una risorsa, deve rilasciare tutte quelle che ha.
Un processo che richiede più risorse molto usate può trovarsi nella condizione di attenderne indefinitamente la disponibilità.

Impossibilita di prelazione

La terza condizione necessaria prevede che non sia possibile avere la prelazione su risorse gia assegnate. Si può impegnare un protocollo che se un processo richiede una risorsa e questa non è disponibile, allora egli le rilascia tutte per poi richiederle quando sono tutte disponibili. In alternativa quando un processo richiede alcune risorse, si verifica la disponibilità di queste ultime:

se sono disponibili vengono assegnate, se non lo sono si verifica se sono assegnate a un processo che attende altre risorse. Se le risorse non sono disponibili ne assegnate ad altri il processo deve attender. Questo protocollo è adatto a risorse il cui stato si può salvare e recuperare facilmente in un secondo momento.

Attesa circolare

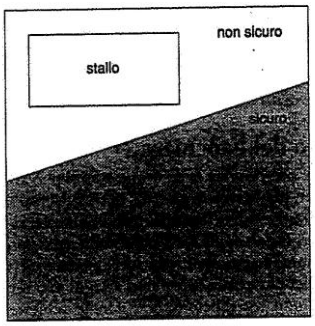
La quarta e ultima condizione necessaria per una situazione di stallo è l'attesa circolare. Un modo per assicurare che tale condizione d'attesa non si verifichi consiste nell'imporre un ordinamento totale all'insieme di tutti i tipi di risorse e un ordine crescente di numerazione per le risorse richieste da ciascun processo. Per prevenire si può usare un protocollo: ogni processo può richiedere risorse solo seguendo un ordine crescente di numerazione.

5. EVITARE LE SITUAZIONI DI STALLO

Gli algoritmi per prevenire le situazioni di stallo si trattano sul controllo della modalità della richiesta cosi da assicurare che non si possa verificare almeno una delle condizioni necessarie perché si abbia uno stallo. Un metodo alternativo per evitare le situazioni di stallo consiste nel richiedere ulteriori informazioni sulla modalità di richiesta, ossia l'ordine con cui utilizzerà le risorse. Una volta acquisita la sequenza completa delle richiese e dei rilasci di ogni processo, si può stabilire per ogni richiesta se il processo debba attenderne o meno per evitare una possibile situazione di stallo. Il modello più semplice e più utile richiede che ciascun processo dichiari il numero massimo delle risorse di ciascun tipo di cui necessita. Questo algoritmo definisce un metodo per evitare lo stallo ed esamina dinamicamente lo stato di assegnazione delle risorse per garantire che non posso esistere una condizione di attesa circolare.

Stato sicuro

Uno stato si dice sicuro se il sistema è in grado di assegnare risorse a ciascun processo in un certo ordine e impedire il verificarsi di uno stallo. Più formalmente un sistema si trova in uno stato sicuro solo se esiste una sequenza sicura. Uno stato sicuro non è di stallo. Viceversa uno stato di stallo no è sicuro ma non tutti gli stati non sicuri sono di stallo.



	Richieste massime	*Unità possedute*
P_0	10	5
P_1	4	2
P_2	9	2

Il sistema ha 12 unita a nastro. All'istante t0 il sistema si trova in uno stato sicuro (ha 3 unita libere). Se il processo P2 richiede 2 risorse il sistema entra in uno stato di stallo in quanto in questo modo nessuno riesce a portare a termine la propria esecuzione. L'idea è semplice: assicurare che il sistema rimanga sempre in uno stato sicuro. Ogni volta che un processo richiede una risorsa il sistema stabilisce se si verifica uno stallo o meno e si soddisfa la richiesta solo se non si verifichi lo stallo.

Algoritmo con grafo di assegnazione

Per evitare le situazioni di stallo si può far uso di una variante del grafo di assegnazione delle risorse. Oltre agli archi di richiesta e di assegnazione si introduce un nuovo arco, l'arco di reclamo disegnato con una linea tratteggiata. Questo arco ha la stessa direzione dell'arco di richiesta e nel momento che il processo richiede la risorsa essa diventa un arco di richiesta. Ciò significa che gli archi di reclamo devono essere gia inseriti prima che il processo Pi inizi. Quando un processo rilascia una risorsa, l'arco di rilascio diventa arco di reclamo.

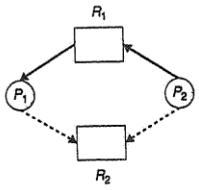


Figura 8.5 Grafo di assegnazione delle risorse per evitare le situazioni di stallo.

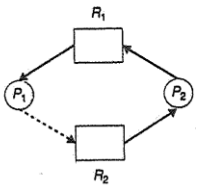


Figura 8.6 Uno stato non sicuro in un grafo di assegnazione delle risorse.

6. RILEVAMENTO DELLE SITUAZIONI DI STALLO

Se un sistema non si avvale di un algoritmo per prevenire o evitare situazioni di stallo, è possibile che si verifichi effettivamente- in tal caso il sistema deve fornire i seguenti algoritmi:

- Un algoritmo che esamini lo stato del sistema per verificare che si sia verificato
- Un algoritmo che ripristini il sistema dalla condizione di stallo

Istanza singola di ciascun tipo di risorsa

Se tutte le risorse hanno una sola istanza si può definire un algoritmo di rilevamento di situazione di stallo che fa uso di una variante del grafo di assegnazione delle risorse, detta grafo d'attesa, ottenuta dal grafo di assegnazione togliendo i nodi dei tipi di risorse e componendo gli archi tra i processi. Come in precedenza, nel sistema esiste uno stallo se e solo se il grafo d'attesa contiene un ciglio. Per evitare le situazioni di stallo, il sistema deve conservare in grafo d'attesa e invocare periodicamente algoritmo che cerchi un ciclo all'interno del grafo.

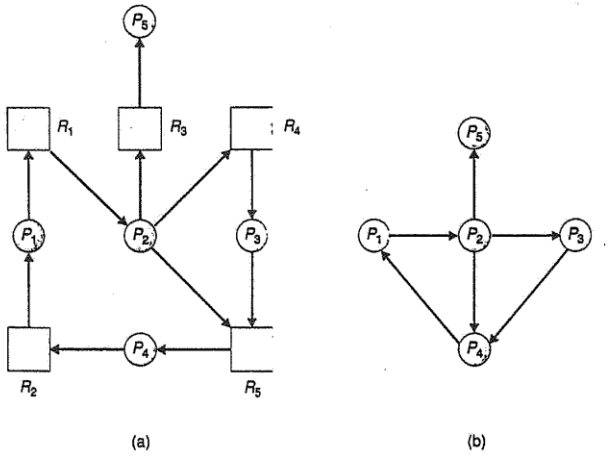


Figura 8.7 (a) Grafo di assegnazione delle risorse; (b) grafo d'attesa corrispondente.

Per illustrare questo algoritmo, si consideri un sistema con cinque processi, da P_0 a P_4, e tre tipi di risorse: A, B, e C. Il tipo di risorsa A ha 7 istanze, il tipo B ha 2 istanze e il tipo C ha 6 istanze. Si supponga di avere, all'istante T_0, il seguente stato di assegnazione delle risorse:

	Assegnate	*Richieste*	*Disponibili*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Il sistema non è in stallo. Infatti eseguendo l'algoritmo per la sequenza $<P_0, P_2, P_3, P_1, P_4>$, risulta *Fine*[*i*] = *vero* per ogni *i*. Si supponga ora che il processo P_2 richieda un'altra istanza di tipo C. La matrice *Richieste* viene modificata come segue:

	Richieste
	A B C
P_0	0 0 0
P_1	2 0 2
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

Ora il sistema è in stallo. Anche se si possono reclamare le risorse possedute dal processo P0, il numero delle risorse disponibili non è sufficiente per soddisfare le richiesta degli atri processi.

7. RIPRISTINO DA SITUAZIONI DI STALLO

Una situazione di stallo si può affrontare in diversi modi. Una soluzione consiste nell'informare l'operatore della presenza dello stallo, in modo che possa gestirlo manualmente. L'altra soluzione lascia al sistema il ripristino in modo che interrompa o tutti i processi in stallo oppure interrompe man mano un processo finche non si risolva tale situazione.

Terminazione dei processi

Per eliminare le situazioni di stallo attraverso la terminazione dei processi si possono adoperare 2 metodi:

- **Terminazione di tutti i processi in stallo**. Chiaramente questo metodo interrompe lo stallo ma l'operazione è molto onerosa. Si pensi al caso in cui alcuni processi manca poco alla fine e venga terminato, tutto il lavora fatto andrebbe perso.
- **Terminazione di un processo alla volta fino all'eliminazione del ciclo di stallo**. Questo metodo causa un notevole carico poiché dopo aver terminato ogni processo, si devi impiegare un algoritmo di rilevamento per stabilire se lo stallo è ancora presente o meno.

Terminare un processo è un'operazione delicata, si pensi ad un processo che sta cambiando un file in scrittura, la sua terminazione resterebbe il file in uno stato scorretto. Per la terminazione parziale bisogna scegliere il processo da terminare. Le considerazioni sono soltanto economiche. La scelta dei processi è influenzata :

- la priorità dei processi
- il tempo trascorso dalla computazione il tempo necessario per completare
- la quantità e il tipo di risorsa impiegata
- il numero di processi che si devono terminare
- il tipo di processo; interattivi o a lotti.

CAPITOLO 8: MEMORIA CENTRALE

Uno dei risultati dello scheduling della CPU consiste nella possibilità di migliorare sia l'utilizzo della CPU sia la rapidità con cui il calcolatore risponde ai propri utenti. Per ottenere questo aumento delle prestazioni occorre tener il memoria parecchi processi: la memoria deve essere condivisa

1. **INTRODUZIONE**

La memoria è fondamentale nelle operazioni di un moderno sistema di calcolo; consiste in un ampio vettore di parole o byte, ciascuno con il proprio indirizzo. La CPU preleva le istruzioni dalla memoria sulla base del contenuto del program counter. Un tipico ciclo di esecuzione prevede che l'istruzione sia prelevata dalla memoria codificata ed eseguita, i risultati memorizzati in memoria. La memoria prevede solo un flusso di indirizzi, non sa come sono generati oppure a cosa servono.

Dispositivi essenziali

La memoria centrale e i registri incorporati nel processore sono le sole aree di memorizzazione a cui la CPU può accedere direttamente. Pertanto qualsiasi istruzione in esecuzione e tutti i dati utilizzati dalle istruzioni devono risiedere in uno di questi dispositivi per la memorizzazione ad accesso diretto. I registri incorporati nella CPU sono accessibili in genere nell'arco di un ciclo dell'orologio di sistema. Nei casi in cui l'accesso alla memoria richieda molti cicli di orologio, il processore entra necessariamente in stallo poichè manca dei dati richiesti per completare l'istruzione che sta eseguendo. Il rimedio consiste nell'interposizione di una memoria detta cache tra la CPU e la memoria centrale con lo scopo di ottenere le istruzioni dalla memoria centrale e trasferire alla cpu. È importante proteggere il sistema dall'accesso degli utenti. Innanzitutto bisogna assicurarsi che ciascun processo abbia uno spazio di memoria separato. Si può implementare il meccanismo di protezione tramite due registri, detti registri base e registri limite.

Per mettere in atto il meccanismo della protezione la CPU confronta ciascun indirizzo generato in modalità utente con i valori dei due registri. Qualsiasi accesso di un processo utente ad un'area del sistema operativo genera eccezioni oppure come errore fatale. Solo il SO può caricare i registri base e limite grazie a una istruzione privilegiata.

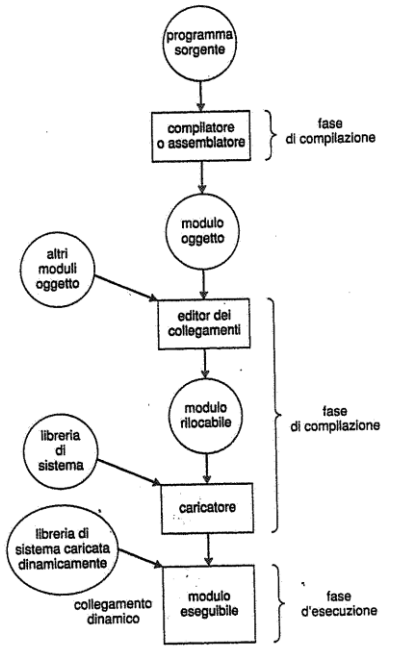
Associazione degli indirizzi

In generale un programma risiede in un disco sotto forma di file binario. Per essere eseguito, il programma va caricato in memoria e inserito all'interno di un processo. Secondo il tipo di gestione della memoria adoperato, durante la sua esecuzione il processo può trasferire dalla memoria al disco e viceversa. L'insieme dei processi presenti nei dischi e che attendono d'essere trasferiti in memoria per essere seguiti forma la coda di ingresso. La procedura normale consiste nello sceglier uno dei processi appartenenti alla coda di ingresso e caricalo in memoria.

Generalmente gli indirizzi del programma sorgente sono simbolici. Un compilatore di solito associa questi indirizzi a indirizzi rilocabili. L'editor dei collegamenti o il caricatore fa corrispondere a sua volta questi indirizzi rilocabili a indirizzi assoluti

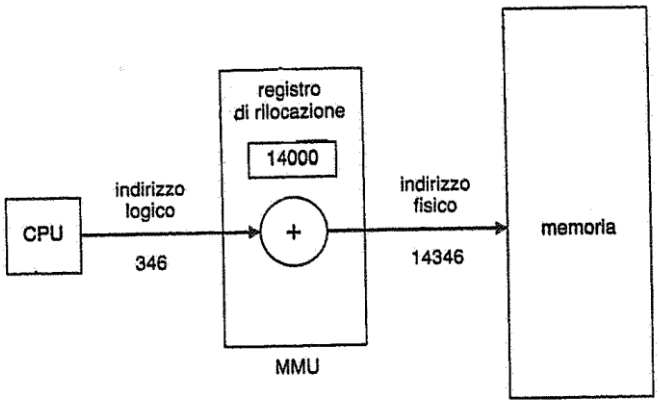
Generalmente l'associazione di istruzione e dati a indirizzi di memoria si può compiere in qualsiasi fase del seguente percorso.

- **Compilazione.** Se nella fase di compilazione si sa dove il processo risiederà in memoria, si può generare codice assoluto.
- **Caricamento.** Se nella fase di compilazione non è possibile sapere in che punto della memoria risiedere il processo, il compilatore deve generare codice rilocabile. Il questo caso si ritarda l'associazione finale degli indirizzi alla fase del caricamento. Se l'indirizzo iniziale cambia, è sufficiente ricaricare il codice utente per incorporare il valore modificato
- **Esecuzione.** Se durante l'esecuzione il processo può essere spostato da un segmento di memoria a un altro, si deve ritardare l'associazione degli indirizzi fino alla fase d'esecuzione.



Spazi di indirizzi logici e fisici a confronto

Un indirizzo generato dalla CPU di solito si indica come indirizzo logico, mentre un indirizzo visto dall'unita di memoria, cioè caricato nel registro dell'indirizzo di memoria di solito si indica come indirizzo fisico. I metodi di associazione degli indirizzi nelle fasi di compilazione e di caricamento producono indirizzi logici e fisici identici. Con i metodi di associazione nella fase d'esecuzione, invece, gli indirizzi logici non coincidono con gli indirizzi fisici. In questo caso si riferisce di solito agli indirizzi logici col termine di indirizzi virtuali. L'insieme di tutti gli indirizzi logici generati da un programma è lo spazio degli indirizzi logici, quelli degli indirizzi fisici è detto spazio degli indirizzi fisici. L'associazione nella fase d'esecuzione dagli indirizzi virtuali agli indirizzi fisici è svolta da un dispositivo detto unita di gestione della memoria MMU. Il registro base è ora denominato registro di rilocazione: quando un processo utente genera un indirizzo,, prima dell'invio all'unità di memoria, si somma tale indirizzo al valore contenuto nel registro di rilocazione.



Il programma utente non considera mai gli indirizzi fisici reali. Il programma crea un puntatore alla locazione 346, lo memorizza, lo modifica con gli altri indirizzi, tutto ciò semplicemente come un numero. Solo quando assume il ruolo di un indirizzo di memoria si riloca il numero sulla base del contenuto del registro di rilocazione.

Esistono 2 tipi di indirizzi: gli indirizzi logici (nell'intervallo da 0 a max) e indirizzi fisici (nell'intervallo da r+0 a r+max dove r è il registro base).

Caricamento dinamico

Per migliorare l'utilizzo della memoria si può ricorrere al caricamento diretto mediante il quale si carica una procedura solo quando viene richiamata; tutte le procedure si tengono in memoria

secondaria in un formato di caricamento rilocabile. Si carica il programma principale in memoria e quando durante l'esecuzione una procedura deve richiamarne un'altra si controlla innanzitutto che sia stata caricata, altrimenti si richiama il caricatore di collegamento rilocabile per caricare il memoria la procedura richiesta e aggiornare le tabelle degli indirizzi del programma in modo che registrino questo cambiamento. Il vantaggio dato dal caricamento dinamico consiste nel fatto che una procedura che non si adopera non viene caricata.

Caricamento dinamico e librerie condivise

Alcuni SO consentono solo il collegamento statico, in cui le librerie di sistema del linguaggio sono trattate come qualsiasi altro modulo oggetto e combinate dal caricatore nell'immagine binaria del programma. Senza questo strumento tutti i programmi di un sistema dovrebbero disporre all'interno dell'immagine eseguibile di una copia della libreria di linguaggio e tutto ciò richiede spazio nei dischi e in memoria centrale.

Con il collegamento dinamico invece per ogni riferimento a una procedura di libreria si inserisce all'interno dell'immagine eseguibile una piccola porzione di codice di riferimento che indica come localizzare la giusta procedura nella libraria residente in memoria o che deve essere caricata. Durante l'esecuzione il codice di riferimento controlla se la procedura richiesta è in memoria altrimenti provvede a caricarla. Con questo metodo tutti i processi che usano una libreria del linguaggio si limitano a eseguire la stessa copia del codice della libreria e in caso di aggiornamento di libreria basta solo sostituire la vecchia con la nuova senza cambiare nulla nei codici. Solo i programmi compilati con la nuova libreria subiscono gli effetti delle modifiche, gli altri continuano ad avvalersi della vecchia libreria. A differenza del caricamento dinamico, il collegamento dinamico richiede generalmente l'assistenza del sistema operativo.

Sovrapposizioni di sezioni

Per permettere ad un processo di esser più grande della memoria che gli si assegna, si può usare una tecnica chiamata sovrapposizione di sezioni (overlay). Il concetto di sovrapposizione si sezioni si basa sulla possibilità di mantenere nella memoria soltanto le istruzioni e i dati che si usano con maggiore frequenza. Quando sono necessarie altre istruzioni queste si caricano nello spazio precedentemente occupato dalle istruzioni che non sono più in uso.

es.

Passo 1	70 KB
Passo 2	80 KB
Tabella dei simboli	20 KB
Procedure comuni	30 KB

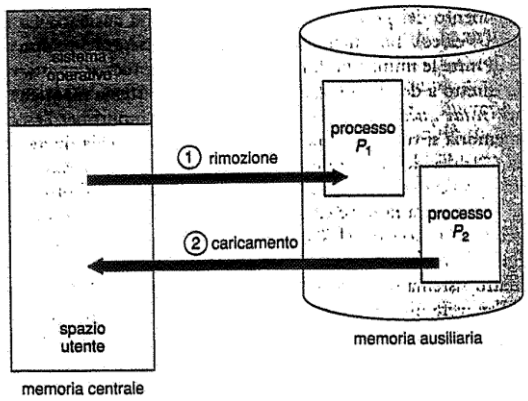
Per caricare il tutto c'è bisogno di 200 KB. Se sono disponibili solo 150 il processo non può essere eseguito. Per essere eseguito è necessario che passo1 e passo2 non si trovino contemporaneamente in memoria. Terminato il passo uno il controllo passa al passo2 che si sovrappone al passo1.

Questa tecnica non richiede alcun intervento del So ma può essere realizzata direttamente dall'utente per mezzo di semplici strutture di file, copiandone il contenuto nella memoria e quindi trasferendo il controllo a quest'ultima per eseguire istruzioni appena lette.

2. AVVICENDAMENTO DEI PROCESSI (SWAPPING)

Per essere eseguito un processo deve trovarsi in memoria centrale ma si può trasferire temporaneamente in memoria ausiliare da cui riporta il memoria centrale nel momento in cui si deve riprenderne l'esecuzione. Quando un processo termina es. con lo scheduling del RR esso viene scaricato in memoria secondaria e caricato il nuovo processo e successivamente ricaricato. Questo procedimento si chiama avvicendamento dei processi in memoria o anche SWAPPING.

Il tempo in cui la CPU effettua le operazioni di un processo devono essere abbastanza lunghe per permetter ad un processo di effettuare molte operazioni prima che venga sostituito.



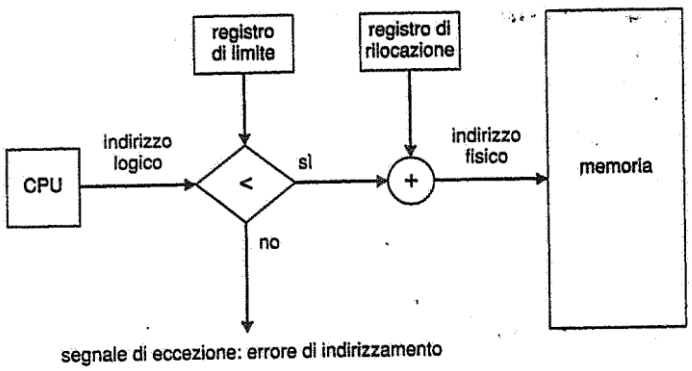
Una variante di questo criterio d'avvicendamento dei processi s'impiega per gli algoritmi di scheduling basati sulle priorità. Se si presenta un processo con priorità maggiore, il gestore della memoria può scaricare dalla memoria centrale il processore con priorità inferiore per far spazio all'esecuzione del processo con priorità maggiore. Quando il processo con priorità maggior termina, si può ricaricar in memoria quello con priorità minore e continuare la sua esecuzione. Normalmente un processo quando viene scaricato deve essere ricaricato nello stesso spazio di memoria occupato prima. Questa limitazione è dovuta al metodo di associazione di indirizzi. Se l'associazione degli indirizzi logici a quelli finisci si effettua nella fase di assemblaggio o caricamento, il processo non può essere caricato altrove mentre se avviene in fase d'esecuzione può essere riversato in uno spazio di memoria diverso.
L'avvicendamento dei processi richiede una memoria ausiliaria. Tale memoria ausiliaria deve essere abbastanza ampia da contenere le copie di tutte le immagini di memoria di tutti i processi utenti. Il sistema mantiene una coda dei processi pronti (ready queue) formati da tutti i processi pronti per l'esecuzione le cui immagini di memoria si trovano in memoria ausiliaria.
Per utilizzare al meglio la CPU è necessario che il tempo d'esecuzione di ciascun processo sia lungo rispetto al tempo d'avvicendamento. Occorre notare che il maggior tempo di avvicandamento è data dal tempo di trasferimento .
L'avvicendamento è soggetto ad altri vincoli. Per scaricare un processo è necessario essere certi che sia completamente inattivo soprattutto che non abbia operazioni di IO pendenti

3. ALLOCAZIONE CONTIGUA DELLA MEMORIA

La memoria centrale deve contener sia il SO che i vari processi che si voglio eseguire. Di solito si divide in 2 partizioni , una per il SO e l'altra per i processi utenti. Il SO si può collocare in memoria bassa che in quella alta ma normalmente viene collocata in memoria bassa vicino al vettore delle interruzioni.

Rilocazione e protezione della memoria

Tale protezione si può realizzare usando un registro di rilocazione che contiene il valore dell'indirizzo fisico minore, in registro limite contiene l'intervallo di indirizzi logici. Con i registri di rilocazione e limite, ogni indirizzo logico deve essere minore del contenuto del registro limite.



Quando lo scheduler della CPU seleziona un processo per l'esecuzione il dispatcher durante l'esecuzione del cambio di contesto carica il registro di rilocazione e il registro limite con i valori corretti. Poi che si confronta ogni indirizzo generato dalla CPU con i valori contenuti in questi registri, si possono proteggere i l SO i programmi e ai dati di altri utenti.
Lo schema con registro di rilocazione consente al SO di cambiar dinamicamente le proprie dimensioni. Tale flessibilità è utile in molte situazioni; il SO contiene codice e spazio di memoria per i driver dei dispositivi; se uno di questi non è comunemente usato è inutile tenere in memoria codice e dati poiché lo spazio occupata si potrebbe usare per altri scopi.

Allocazione della memoria

Uno dei metodi più semplici per l'allocazione della memoria consiste nel suddividere la stessa in partizioni di dimensione fissa. Ogni partizione deve contenere esattamente un processo quindi il grafo di multiprogrammazione è limitato dal numero di partizioni. Con il metodo delle partizioni multiple quando una partizione è libera può esser occupata da un processo presente nella coda d0ingresso; terminato il processo la partizione diviene nuovamente disponibile per un altro processo.
Nello schema a partizione fissa il So conserva una tabella in cui sono indicate le porzioni di memoria disponibili e quelle occupate. Inizialmente tutta la memoria è a disposizione dei processi utenti, si tratta di un grande blocco di memoria disponibile, un buco. Quando si carica un processo che necessita di memoria, occorre cercare un buco sufficientemente grande da contenerlo. Se ne esiste uno si assegna solo al parte di memoria necessaria, la rimanente va usata per soddisfare eventuali richieste successive. Quando entrano nel sistema vengono inse4riti in una coda d'ingresso. Quando a un processo si assegna dello spazio il processo stesso viene caricato in caricato in memoria e quindi competere con il controllo della CPU. Al termine rilascia la memoria che gli era stata assegnata e il sistema operativo può impiegarla per un altro processo presente nella coda d'ingresso.
Il SO può ordinare la coda d'ingresso secondo un algoritmo di scheduling. In generale è sempre presente un insieme di buchi di diverse dimensioni sparsi per la memoria. Quando si presenta un processo che necessita di memoria, il sistema cerca nel gruppo un buco di dimensioni sufficienti per contenerlo. Se è troppo grande in buco viene diviso in 2: uno che lo contiene e l'altra viene spostato nell'insieme dei buchi. Quando un processo termina si rilascia il blocco di memoria il quale viene inserito nell'insieme dei buchi liberi e se c'è un buco vicino questo viene accorpato in un unico buco più grande. Questa procedura è una particolare istanza del più grande problema dell'allocazione dinamica della memoria. I criteri più usati per scegliere un buco libero tra quelli disponibili sono i seguenti:

- **First-fit.** Si assegna il primo buco abbastanza grande. La ricerca può cominciare dall'inizio o dall'insieme dei buchi.
- **Best-fit.** Si assegna il più piccolo buco in grado di contener il processo. Si deve compiere la ricerca su tutta la lista
- **Worst-fit.** Si assegna il buco più grande e la ricerca avviene su tutta la lista.

Con l'uso delle simulazioni si è dimostrato che sia first-fit che best-fit sono migliori di worst-fit

Frammentazione

Entrambi i criteri first-fit e best-fit soffrono della frammentazione esterna: quando si caricano e si rimuovono i processi dalla memoria, si frammenta lo spazio libero della memoria in tante piccole parti. Si ha la frammentazione esterna se lo sazio di memoria totale è sufficiente per soddisfare una richiesta, la memoria è frammentata in tanti piccole parti. Questo problema può essere molto grave in quanto riduce le prestazioni del sistema. La scelta del primo buco abbastanza grande porta uno spreco di un terzo della memoria. La frammentazione interna consiste nella differenza tra lo spazio assegnata e quello richiesto.
Una soluzione al problema della frammentazione esterna è data dalla compattazione. Lo scopo è quello di riordinare il contenuto della memoria per riunire la memoria libera in un unico grosso

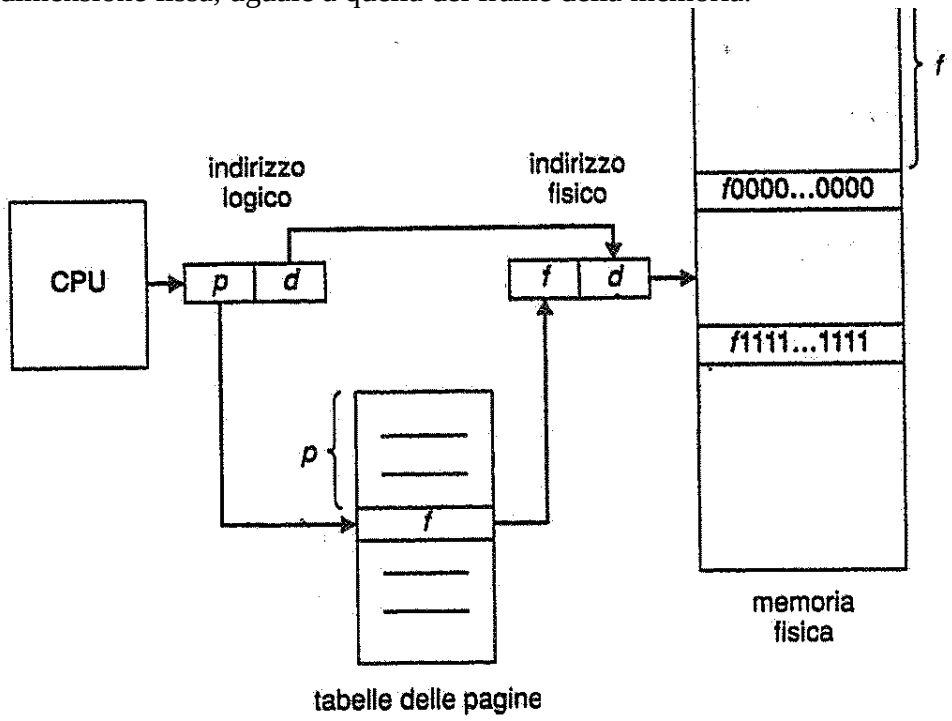
blocco. Il più semplice algoritmo di compattazione consiste nello spostare tutti i processi verso un'estremità della memoria mentre tutti i buchi vengono spostati nell'altra direzione.

4. PAGINAZIONE

La paginazione è un metodo di gestione della memoria che permette che lo sazio degli indirizzi fisici di un processo non sia contiguo. Elimina il gravoso problema della sistemazione di blocchi di memoria di diverse dimensioni in memoria ausiliaria. Il problema insorge perché quando alcuni frammenti di codice o dati residente in memoria centrale devono essere scaricati, si deve trovare lo spazio necessario in memoria ausiliaria. I problemi di frammentazione relativi alla memoria centrale valgono anche per la memoria secondaria con la differenza che in questo caso l'accesso è molto più lento e quindi è impossibile eseguire la compattazione.

Metodo di base

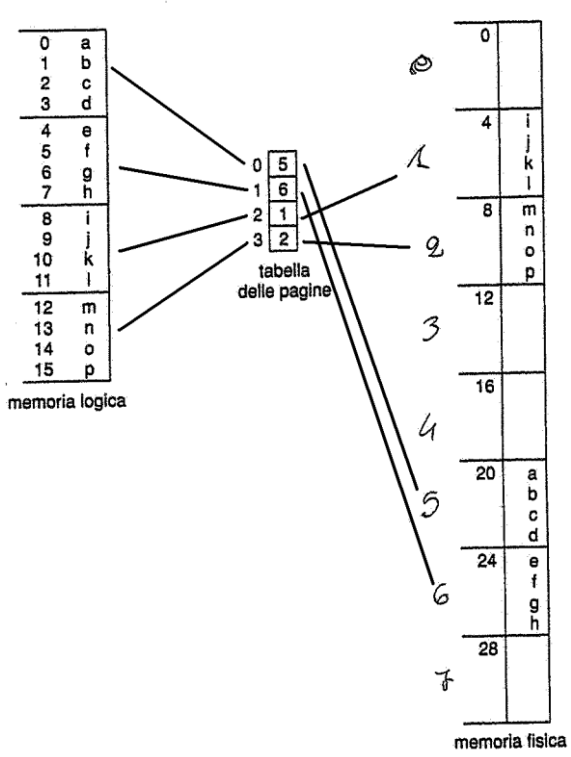
Il metodo di base per implementare la paginazione consiste nel suddividere la memoria fisica in blocchi di dimensione costante detti anche grame o pagine fisiche e nel suddividere la memoria logica in blocchi di pari dimensioni detti pagine. Quando si deve eseguire un processo si caricano le sue pagine nei grame disponibile prendendole dalla memoria ausiliaria divisa in blocchi di dimensione fissa, uguale a quella dei frame della memoria.



Ogni indirizzo generato dalla CPU è diviso in due parti: numero di pagina (p) e uno scostamento (d). il numero di pagina serve come indice per la tabella delle pagine contente l'indirizzo base in memoria fisica di ogni pagina. Questo indirizzo si combina con lo scostamento d che forma l'indirizzo fisico. La dimensione della pagina è definita dall'architettura e varia tra i 512 byte a 16 MB. La paginazione non è altro che una forma di rilocazione dinamica: ogni indirizzo logico l'architettura di paginazione fa corrispondere un indirizzo fisico. L'uso della tabella delle pagine è simile all'uso di una tabella di registri base uno per ciascun frame.

Con la paginazione si può evitare la frammentazione esterna: qualsiasi frame libero si può assegnare a un processo che ne abbia bisogno. Pero c'è il problema della frammentazione interna nel caso peggiore in cui si ha un processo che necessita di n pagine + 1 byte e si devono allocare n+1 pagine con una frammentazione interna media di mezza pagina per processo. Questo fa si che si convenga usare pagine di piccole dimensioni.

Quando si deve eseguire un processo si esamina la sua dimensione espressa in pagine. Se necessita di n pagine, devono essere disponibili almeno n frame.



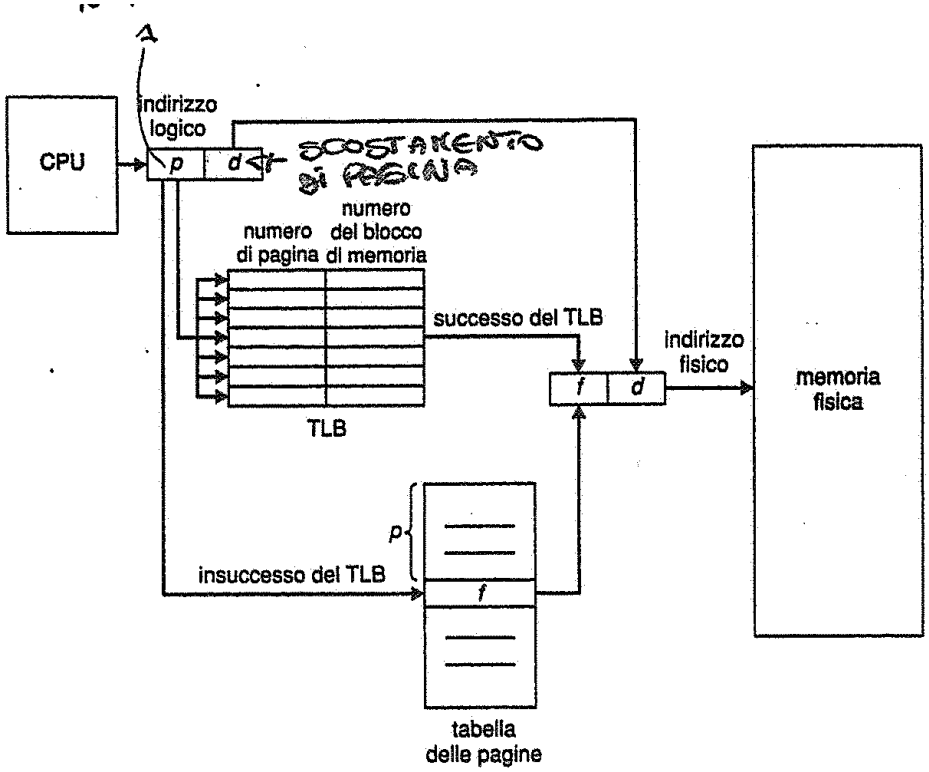
Un aspetto importante della paginazione è la netta distinzione tra la memoria vista dall'utente e l'effettiva memoria fisica: il programma utente vede la memoria come un unico spazio contiguo, contenente anche altri programmi. La differenza tra la memoria vista dall'utente e la memoria fisica è colmata dall'architettura di traduzione degli indirizzi.

Poiché il SO gestisce la memoria fisica, deve essere informato dei relativi particolari di allocazione: quali grame sono assegnati, quali sono disponibili, il loro numero totale e cosi via. In genere queste informazioni sono contenute in una struttura dati chiamata tabella dei frame contenenti un elemento per ogni frame, indicante se sia libero oppure assegnato e se è assegnato a quale pagina di quale processo. Il SO conserva una copia della tabella delle pagine per ciascun processo, cosi come conserva una copia dei valori contenuti nel contatore di programma e nei registri. Questa coppia si usa per tradurr gli indirizzi logici in fisici ogni volta che il SO deve associare esplicitamente un indirizzo fisico a uno logico. La paginazione quindi fa aumentare il cambio di contesto.

Architettura di paginazione

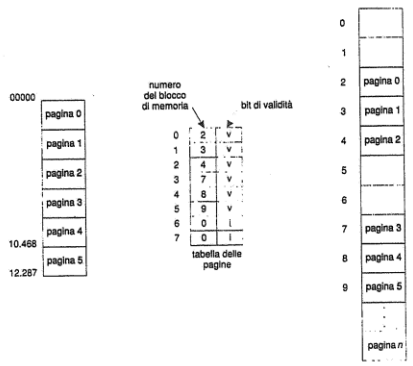
Ogni SO segue metodi propri per memorizzare le tabelle della pagine. Il PCB contiene insieme col valore di altri registi, come il registro delle istruzioni, un puntatore della tabella delle pagine. Per avviare un processo il dispatcher ricarica i registri utente i imposta i corretti valore della tabella delle pagine fisiche usando la tabella della pagine presente in memoria e alitava al processo. L'architettura d'ausilio alla tabella della pagine può essere realizzata in modi diversi, nei casi più semplice attraverso dei registri. L'uso dei registri è efficiente se la tabella stessa è ragionevolmente piccola max 256 elementi. La maggior parte dei calcolati usa comunque tabella di pagine molto grandi quindi non si possono utilizzare registri ma si utilizza un registro che punta alla tabella delle pagine mantenuta in memoria. Il cambio della pagina richiede solo l'aggiornamento del puntatore. Questo metodo presenta un problema connesso al tempo necessario di accesso a una locazione della memoria i. per accedere a tale locazione di memoria occorre tener presente del valore del registro PTBR aumentato del numero di pagina relativo a i. si ottiene il numero del frame che associa allo scostamento di pagina producendo cosi l'indirizzo desiderata. La soluzione tipica al problema riscontrato per il doppio accesso, consiste nell'impiego di una speciale piccola cache ad alta velocità in cui ogni suo elemento consiste di due parti: una chiave e un valore che corrisponde al frame. La TBL contiene una piccola parte degli elementi della tabella delle pagine; quando la CPU

genera un indirizzo logico, si presenta il suo numero di pagina alla TBL, se è presenta il corrispondente numero del grame è immediatamente disponibile e si usa per acceder alla memoria.
Se non è presente nella TBL è noto come insuccesso nella TBL si cerca la tabella della pagina in memoria che viene utilizzato il valore per l'accesso in memoria. Se la TBL è piena, il SO deve sostituirne uno e si sceglie l'elemento usato meno recentemente. Alcune TBL memorizzano gli identificatori dello spazio d'indirizzi (ASID)in ciascun elemento della TBL. Un'ASID identifica in modo univoco ciascun processo e si usa per fornire al processo la protezione del suo spazio di indirizzi.



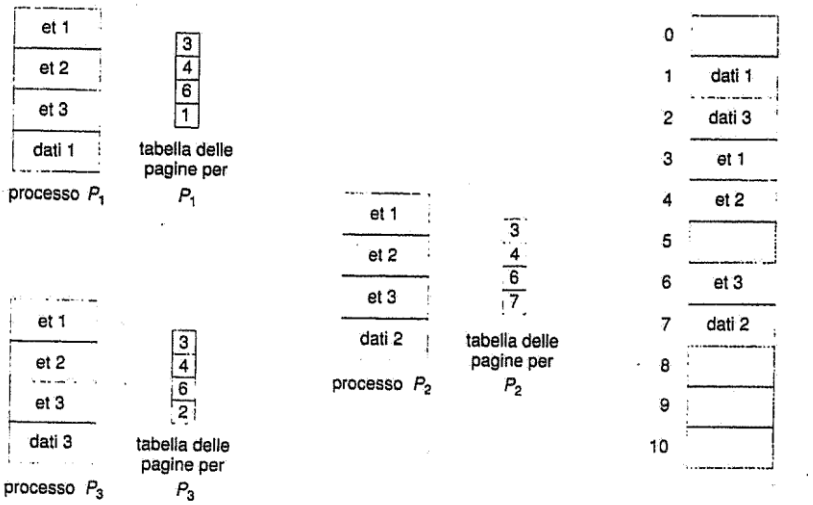
Protezione

In un ambiente paginato la protezione della memoria è assicurata dai bit di protezione associati a ogni frame. Normalmente tali bit si trovano nella tabella delle pagine. Un bit può determina se una pagina si può leggere e scrivere oppure solo leggere. Si può progettar un'architettura che fornisca la protezione di sola lettura, di sola scrittura o di sola esecuzione. In alternativa con bit di protezione distinti per ogni tipo d'accesso, si può ottenere una qualsiasi comunicazione di tali tipi d'accesso, i tentativi illegali causano l'invio di un segnale di eccezione al SO. Di solito si associa a ciascun elemento della tabella delle pagine un ulteriore bit, detto bit di validità. Tale bit se impostato a valido, indica che la pagina è nello spazio degli indirizzi logici del processo, altrimenti non lo è. Consente di riconoscere gli indirizzi illegali e di notificarne la presenza attraverso le eccezioni. Alcune architetture dispongono di registri, detti registri di lunghezza della tabella delle pagine per indicare le dimensioni della tabella.



Pagine condivise

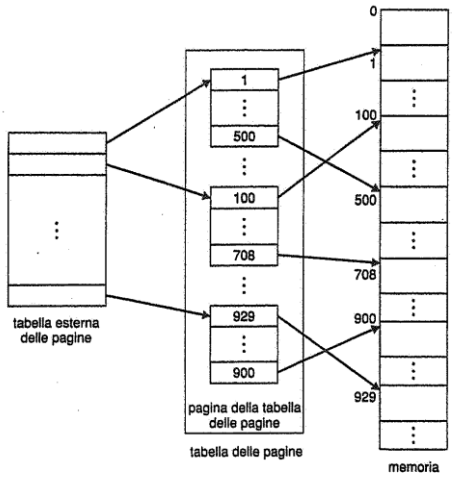
Un altro vantaggio della paginazione consiste nella possibilità di condividere codice comune.
Il codice rientrante, detto codice puro, è un codice non automodificante, non cambia durante l'esecuzione. Quindi due o più processi possono eseguire lo stesso codice nello stesso momento. Ciascun processo dispone di una propria copia dei registri e di una memoria dove conserva i dati necessari alla propria esecuzione.

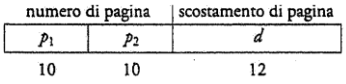


5. STRUTTURA DELLA TABELLA DELLE PAGINE

Paginazione gerarchica

La maggior parte dei moderni calcolatori dispone di uno spazio d'indirizzi logici molto grande(da 2^32 a 2^64 elementi). Chiaramente sarebbe meglio evitare di collocare la tabella delle pagine in modo contiguo in memoria centrale. Una semplice soluzione a questo problema consiste nel suddividere la tabella delle pagine in parti più piccole; questo risultato si può ottenere in molti modi. Un metodo consiste nell'adottare un algoritmo di paginazione a due livelli, in cui la tabella stessa è paginata.





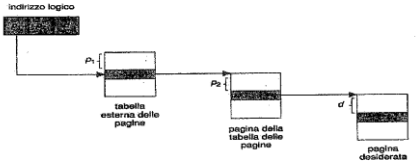


Tabella delle pagine di tipo hash

Un metodo di gestione molto comune degli spazi d'indirizzi relativi ad architetture oltre 32 bit consiste nell'impiego di una tabella delle pagine di tipo hash, in cui l'argomento della funzione hash è il numero della pagina virtuale.

Ciascun elemento è composto da 3 campi: il numero della pagina virtuale, l'indirizzo del frame corrispondente alla pagina virtuale, un puntatore al successivo elemento. Si applica la funzione hash al numero della pagina virtuale contenuto nell'indirizzo virtuale, identificando un elemento della tabella. Si confronta il numero di pagina con il primo campo degli elementi della lista e se coincidono si usa l'indirizzo relativo ala frame per generare l'indirizzo fisico. Ho soluzione efficace per indirizzi sparsi

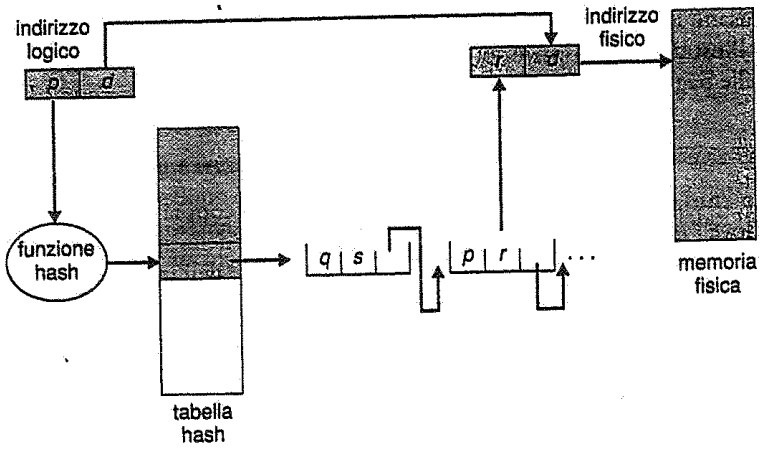
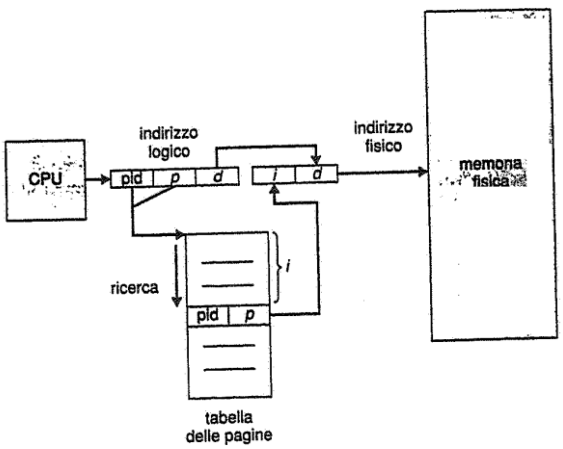


Tabella delle pagine invertita

Generalmente si associa una tabella delle pagine a ogni processo e tale tabella contiene un elemento per ogni pagina virtuale che il processo sta utilizzando , oppure un elemento per ogni indirizzo virtuale a prescindere dalla validità di quest'ultimo. Poiché la tabella è ordinata per indirizzi virtuali il sistema operativo può calcolare in che punto della tabella si trova l'elemento dell'indirizzo fisico associato e usare direttamente tale valore. Uno degli inconvenienti insiti in questo metodo è costituito dalla dimensione di ciascuna tabella delle pagine, che può contenere milioni di elementi e occupare grandi quantità di memoria fisica,, necessaria proprio per sapere com'è impiegata la rimanente memoria fisica. Per risolvere questo problema si può fare uso della tabella delle pagine invertita. Una tabella delle pagine invertita ha un elemento per ogni pagina reale o frame. Ciascun elemento è quindi costituito dell'indirizzo virtuale della pagina memorizzata in quella reale locazione di memoria con informazioni sul processo che possiede tale pagina. Ciascun indirizzo virtuale è una tripla del tipo seguente: <id-progesso,numero pagina, scostamento>. Viene ricercato tramite il pid il valore nella tabella della pagine se viene trovato lo scostamento più il valore il numero i.



Sebbene riduca la quantità di memoria necessaria per memorizzar ogni tabella delle pagine, aumenta pero il tempo di ricerca nella tabella. Poiché la tabella delle pagine invertita è ordinata per indirizzi fisici, mentre la ricerca si effettua per indirizzi virtuali, si deve effettuare la ricerca sull'intera tabella. Per limitare il problema si può usare una tabella hash che riduce la cerca a un solo o a pochi elementi della tabella delle pagine. Nei sistemi che adottano le tabelle delle pagine invertite l'implementazione è difficoltosa.

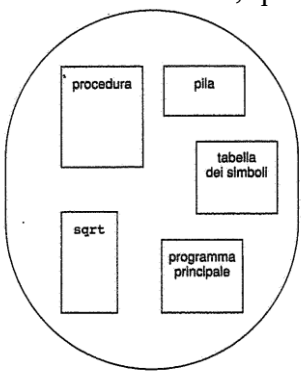
6. SEGMENTAZIONE

Un aspetto importante della gestione della memoria, inevitabile alla presenza della paginazione, è quello della separazione tra la visione della memoria dell'utente e l'effettiva memoria fisica. Lo spazio d'indirizzi tra la visione della memoria dell'utente e l'effettiva memoria fisica. Lo spazio d'indirizzi visto dall'utente non coincide con l'effettiva memoria fisica, ma lo si fa corrispondere alla memoria fisica.

Metodo di base

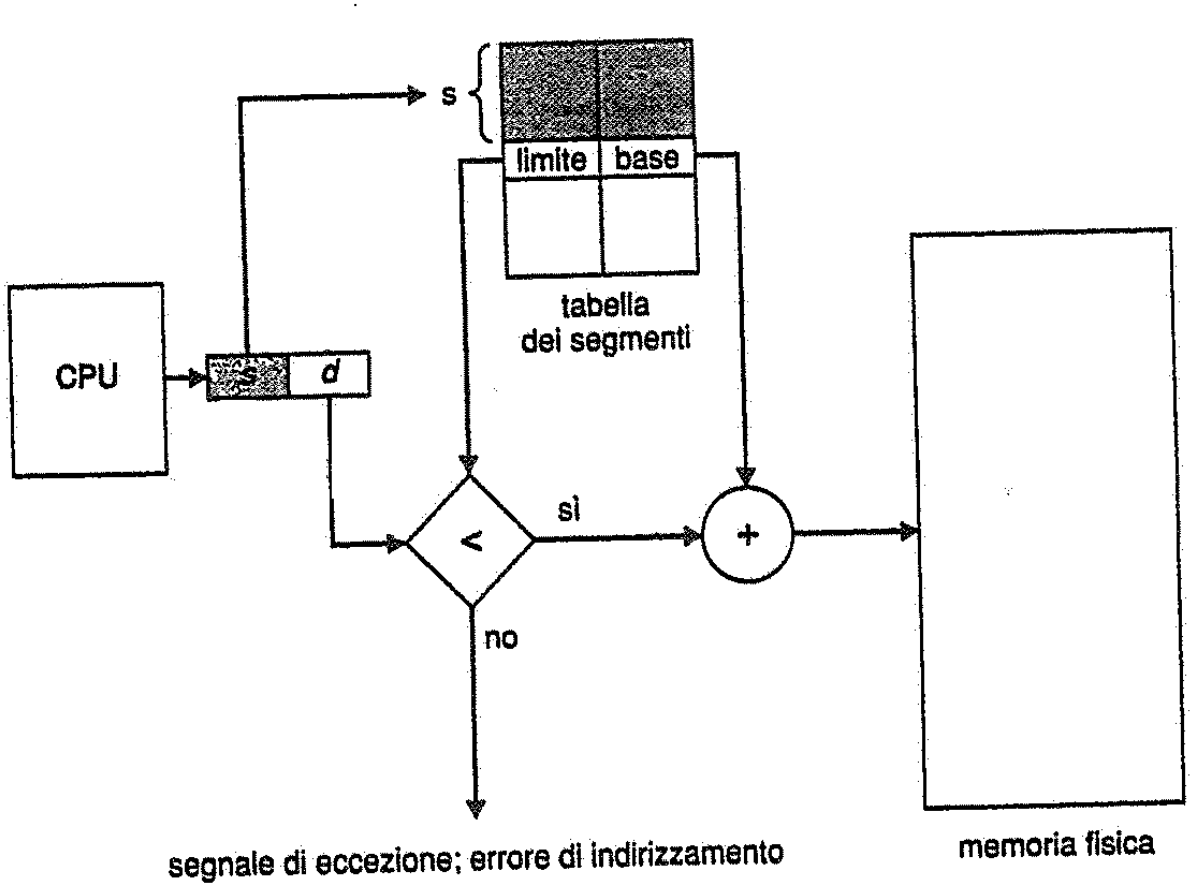
Ci si potrebbe chiedere se l'utente può considerare la memoria come un vettore lineare di byte, alcuni dei quali contengono istruzioni e altri dati. La tipica struttura di un programma con cui i programmatori hanno familiarità è costituita di una parte principale e di un gruppo di procedure funzioni o mugoli insieme con diverse strutture dati come tabella, matrici, pile, variabili e cosi via. Ciascuno di questi moduli o elementi di dati si identifica con un nome: tabella di simboli, funzione sqrt,…gli elementi che si trovano all'interno di un segmento sono identificato dal loro scostamento, misurato dall'inizio del segmento: la prima istruzione del programma, il settimo elemento della tabella dei simboli, la quinta istruzione e cosi via. La segmentazione è uno schema di gestione della memoria che consente di gestire questa rappresentazione della memoria dal punto di vista dell'utente. Uno spazio d'indirizzi logici è una raccolta di segmenti ciascuno dei quali ha un nome e una lunghezza.

Per semplicità i segmenti sono numerati e ogni riferimento si compie per mezzo di un numero anziché di un nome; quindi un indirizzo logico è una coppia < numero di segmento, scostamento>



Architettura di segmentazione

Sebbene l'utente possa far riferimenti agli oggetti del programma per mezzo di un indizio bidimensionale, la memoria fisica è in ogni caso una sequenza di byte unidimensionale. Per questo motivo occorre tradurre gli indirizzi bidimensionali definiti dall'utente negli indirizzi fisici unidimensionali. Questa operazione si come tramite una tabelle dei segmenti: ogni suo elemento p una coppia ordinata: la base del segmento e il limite. La base contiene l'indirizzo fisico iniziale della memoria dove il segmento risiede mentre il limite del segmento contiene la lunghezza del segmento. Un indirizzo logico è comprato da 2 parti: un numero di segmento s e un scostamento di segmento d. Il numero di segmento si usa come indice della tabella dei segmenti, lo scostamento d dell'indirizzo logico deve essere compreso tra 0 e il limite del segmento, altrimenti si invia un'eccezione. Se tale condizione è rispettata, si somma lo scostamento alla base del segmento per produrre l'indirizzo della memoria fisica dove si trova il byte desiderato.


s
limite
base
tabella
dei segmenti
CPU
s
d
<
sì
+
no
segnale di eccezione; errore di indirizzamento
memoria fisica

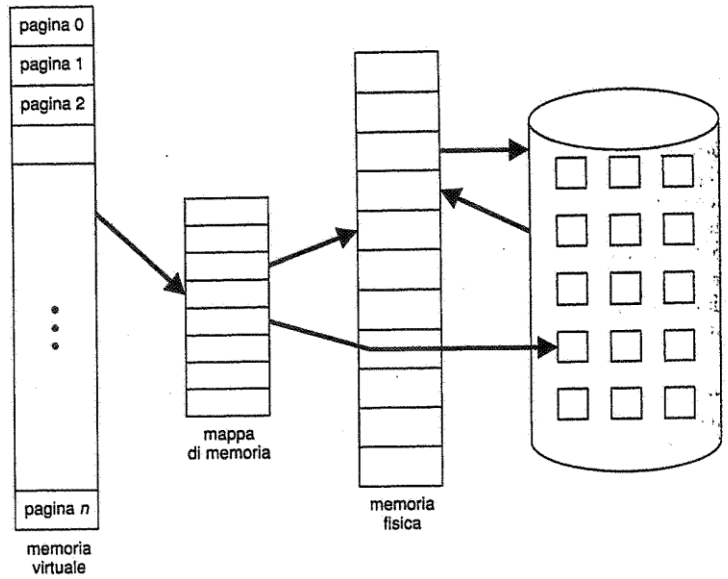
CAPITOLO 9: MEMORIA VIRTUALE

La memoria virtuale è una tecnica che permette di eseguire processi che possono anche non essere completamente contenuti in memoria. Il vantaggio principale offerto da questa tecnica è quello di permettere che i programmi siano più grandi della memoria fisica; inoltre la memoria virtuale astrae la memoria centrale in un vettore di memorizzazione molto grande e uniforme,separando la memoria logica, com'è vista dall'utente, da quella fisica. Questa tecnica libera i programmatori da quel che riguarda i limiti della memoria. La memoria virtuale è pero difficile da realizzare e s'è usata scorrettamente, può ridurre di molto le prestazioni del sistema.

1. INTRODUZIONE

Gli algoritmi di gestione della memoria sono necessari perché, per l'attivazione di un processo, le istruzioni da eseguire si devono trovare all'interno della memoria fisica. Il primo metodo per far fronte a tale requisito consiste nel collocare l'intero spazio d'indirizzi logici del processo relativo in memoria fisica. Il caricamento dinamico può aiutare ad attenuare gli effetti di tale limitazioni, ma richiede generalmente particolari precauzioni e un ulteriore impegno dei programmatori. La condizione che le istruzioni debbano esser nella memoria fisica sembra tanto necessaria quanto ragionevole, ma purtroppo riduce le dimensione dei programmi a valori strettamente correlati alle dimensioni della memoria fisica. Spesso i programmatori hanno codice per la gestione degli errori che vengono utilizzati molto di rado.

La memoria virtuale si fonda sulla separazione della memoria logica percepita dall'utente dalla memoria fisica. Questa separazione perfette di offrire ai programmatori una memoria virtuale molto ampia, anche se la memoria fisica disponibile è più piccola. La memoria virtuale facilita la programmazione, poiché il programmatore non deve preoccuparsi della quanta di memoria fisica disponibile o di quale codice si debba inserire nella sezioni sovrapponibili, ma si può concentrare solo sul problema da risolvere. L'espressione spazio degli indirizzi virtuali si riferisce alla collocazione dei processi in memoria dal punto di vista logico. Da tale punto di vista un processo inizia in corrispondenza di un certo indirizzo logico e si estende alla memoria contigua.



Si noti come allo heap sia lasciato di crescere verso l'alto nello spazio di memoria piche esso ospita la memoria allocata dinamicamente. Lo spazio vuoto e ben visibile che separo lo heap dalla pila è parte dello sazio degli indirizzi virtuali ma richiede pagine fisiche realmente esistenti colo nel caso che lo heap o la pila crescano. Qualora contenga buchi. Lo spazio degli indirizzi virtuali si definisce sparso.

Oltre a separare la memoria logica da quella fisica, la memoria virtuale offre il vantaggio di condividere i file e la memoria, mediante la condivisione delle pagine. Le librerie di sistema sono

condivisibili da diversi processi associando l'oggetto condiviso a uno spazio degli indirizzi virtuali, procedimento detto mappatura. Benché ciascun processo vede le librerie condivise come parte del proprio spazio degli indirizzi virtuali, le pagine che ospitano effettivamente le librerie nella memoria fisica sono in condivisone tra tutti i processi.
La memoria virtuale permette a un processo di creare una regione di memoria condivisibile da un altro processo.

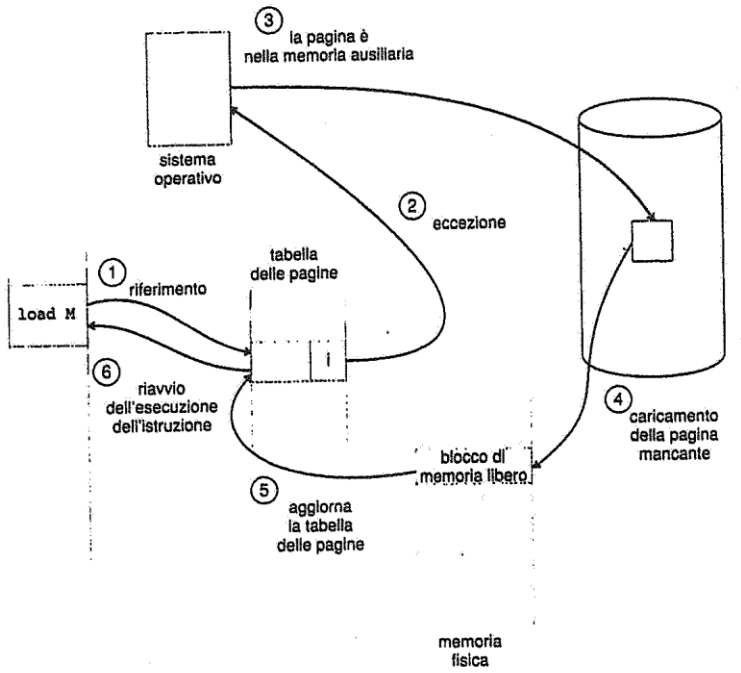
2. PAGINAZIONE SU RICHIESTA

Si consideri il caricamento in memoria di un eseguibile residente su disco. Una possibilità è quella di caricare l'intero programma nella memoria fisica al momento dell'esecuzione. Il problema però è che all'inizio non è detto che serva avere tutto il programma in memoria: se il codice fornisce all'avvio una lista di opzioni all'utente, è inutile caricare il codice per l'esecuzione di tutte le opzioni previste senza tener conto di quella effettivamente scelta dall'utente. Una strategia alternativa consiste nel caricare le pagine nel momento in cui servono raalmente; si tratta di una tecnica, detta paginazione su richiesta, comunemente adottata dai sistemi con mormora virtuale. Secondo questo schema, le pagine sono caricate in memoria solo usando richieste durante l'esecuzione del programma: ne consegue che le pagine che non vengono mai utilizzate non vengono mai caricate. I processi risiedono in memoria secondaria. Per eseguirli bisogna caricarli in memoria, non si carica in memoria mai una pagina che non sia necessaria. Nell'ambito della paginazione su richiesta, il modulo del So che si occupa della sostituzione della pagine si chiama paginature.

Concetti fondamentali

Quando un processo sta per esser caricato in memoria, il paginature ipotizza quali pagine saranno usate prima che il processo sia nuovamente caricato dalla memoria. Anziché caricare il memoria tutto il processo, il paginature trasferisce in memoria solo el pagine che ritiene necessarie. In questo modo è possibile evita il trasferimento in memoria dai pagine che no sono effettivamente usate riducendo il tempi d'avvicendamento e la quantità di memoria fisica richiesta.
È necessario che l'architettura permetti di distinguere le pagine presenti in memoria tra quelle non. A tal fine è utilizzabile il bit di validità. Occorre notare che indicare una pagina non valida non sortisce alcun effetto se il processo non tenta mai di accedervi. Se il processo cerca di accedere a una pagina non presente in memoria, causa un'eccezione di pagina mancante. L'architettura di paginazione traducendo l'indirizzo attraverso la tabella delle pagine, nota che il bit è non valido e invia al SO un segnale il quale provvede a caricarla in memoria.

- Si controlla una tabella interna per questo processo, di solito è conservata nel PCB
- Se il riferimento era non valido si termina il processo. Se era valido ma la pagina non è in memoria se ne effettua l'inserimento
- Si individua un frame libero dove collocare la pagina
- Si programma un'operazione sui dischi per trasferire la pagina nel frame libero
- quando la lettura è completata, si modifica la tabella interna e la tabella delle pagine
- Si riavvia l'istruzione interrotta dal segnale di eccezione.



È addirittura possibile avviare un processo senza pagine in memoria e caricarle man man che vengono richieste.
L'architettura deve fornire meccanismi per:

- **Tabella delle pagine.** Questa tabella ha la capacita di contrassegnare un elemento come non valido attraverso un bit di validità
- **Memoria secondaria.** Questa memoria conserva le pagine non presenti in memoria centrale. La sezione del disco usata per questo scopo richiama area d'avvicendamento.

Il punto cruciale della paginazione su richiesta è che deve essere possibile continuare l'esecuzione nel punto in cui si interrompe per caricare una pagina mancante e riavviare il processo esattamente nel punto in cui è stato interrotto.
Il sistema di paginazione si colloca tra la CPU e la memoria di un calcolatore e deve essere completamente trasparenza al processo utente.

Prestazioni della paginazione su richiesta

La paginazione su richiesta può aver un effetto rilevante sulle prestazioni di un calcolatore. Il motivo si può comprendere calcolando il tempo d'accesso effettivo per una memoria con paginazione su richiesta. Per calcolare il temo d'accesso effettivo occorre conosce il tempo necessario alla gestione di un'assenza di pagina. Alla presenza di un'assenza di pagina si esegue la seguente sequenza:

- segnale d'eccezione al sistema operativo
- Salvataggio dei registri utente
- Verifica che l'interruzione sai dovuta ad una mancanza di pagina
- Controlla della correttezza del riferimento alla pagina e determinazione nella locazione della pagina nel disco.
- Lettura dal disco e trasferimento nel frame libero
- Durante l'attesa, allocazione della CPU a un altro processo utente
- Ricezione di un'interruzione dal controllore del disco
- Salvataggio dei registri e dello stato dell'altro processo utente
- Verifica della provenienza dell'interruzione dal disco
- Aggiornamento della tabella delle pagine e di altre tabelle per segnalare che la pagina richiesta è attualmente presente in memoria
- Attesa che la CPU sia nuovamente assegnata a questo processo
- Recupero dei registri utente.

Per mantenere a un livello ragionevole il rallentamento dovuto alla paginazione, si può permetter un'assenza di pagine ogni 399.990 accessi. Quindi è fondamentale tenere bassa la frequenza delle assenze. Dato che l'IO all'area di avvicendamento è più rapido rispetto al file system, il SO copia tutta l'immagine di un file nell'area di avvicendamento all'avvio ed esegue li la paginazione su richiesta.

3. **COPIATURA SU SCRITTURA**

Nella fork un processo figlio è un duplicato del padre. per ovviare la copiatura si può utilizzare la tecnica detta copiatura si scrittura il cui funzionamento si fonda sulla condivisione iniziale delle pagine da parte dei processi genitori e dei processi figli. Le pagine condivise si contrassegnano come pagine da copiare su scrittura significa che se un processo genitore o giglio scrive su una pagina condivisa, il sistema deve creare una copia di tale pagina. È chiaro che adoperando la tecnica della copiatura su scrittura, si copiano soltanto le pagine modificate se uno dei due processi mentre le altre sono condivisibili tra i processi. Quando è necessaria effettuare la tecnica della copiatura su scrittura, è importante capire da dove si attingeva la pagina libera necessaria. Molti sistemi forniscono insiemi di pagine libere. L'allocazione di queste pagine avviene di solito secondo una tecnica nota come azzeramene su richiesta prima dell'allocazione si riempiono di zeri le pagine cancellando il contenuto. La vfork offre la soluzione copiatura su scrittura.

4. **SOSTITUZIONE DELLE PAGINE**

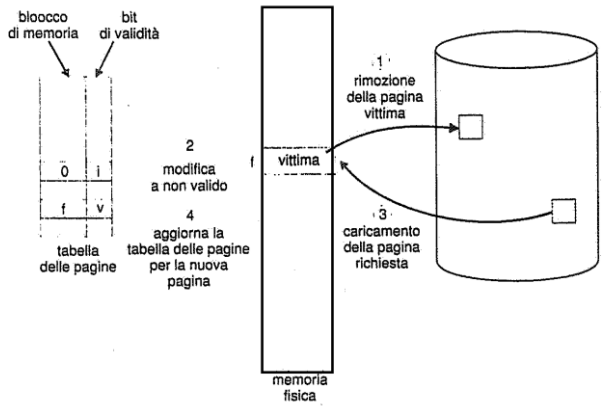
Se un processo di 10 pagine se ne impiega effettivamente solo la meta, la paginazione su richiesta fa risparmiare l?IO necessarie per caricare le 5 pagine che sono necessarie. Aumentando il grado di multiprogrammazione si soprassegna la memoria. Eseguendo 6 processi, ciascuno dei quali è formato da 10 pagine di cui solo 5 sono utilizzate, s'incrementerebbe l'utilizzo e la produttività della CPU e si risparmierebbero 10 frame. Se improvvisamente tutti hanno bisogno di tute le 10 pagine, sarebbero necessari 60 frame mentre ne sono disponibili solo 40. quindi bisogna decidere quanta memoria assegnare all'IO e quanta alle pagine dei programmi è un problema rilevante. Se durante l'esecuzione di un processo utente si verifica un'assenza di pagina, il So determina la locazione del disco in cui risiede la pagina desiderata, ma poi scopre che la lista dei frame libera è vuota: la memoria è tutta in uso. Questo problema è noto come sovrallocazione.
Il SP può scegliere di terminare il processo oppure scaricare l'intero processo liberando suoi frame e riducendo il livello di multiprogrammazione.

Sostituzione di pagina

La sostituzione delle pagine segue il seguente criterio: se nessun frame è libero si può liberarne uno attualmente inutilizzato. È possibile liberalo scrivendo il suo contenuto nell'area d'avvicendamento e modificando la tabella delle pagine per indicare che la pagina non si trova più in memoria. Il frame liberato si può utilizzare per contenere la pagina che ha causato l'eccezione. Quindi:

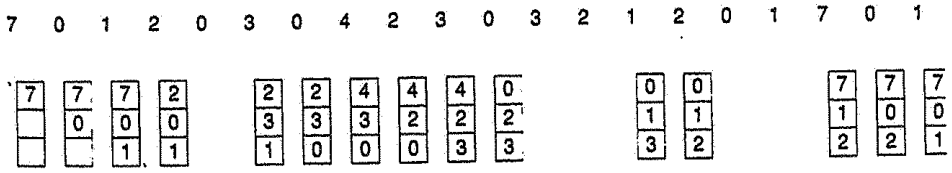
- S'individua la locazione nel disco
- Si cerca un frame libero (se esiste lo si usa altrimenti si libera scrivendo la pagina vittima nel disco)
- Si scrive la pagina richiesta nel frame appena liberato
- Si riavvia il processo utente

Dato che sono necessari 2 trasferimenti se non esiste nessun frame libero, questo sovraccarico si può ridurre usando il bit di modifica. L'architettura deve disporre di un bit che imposta automaticamente ogni volta che nella pagina si scrive una parola. Quando si sceglie una pagina da sostituire si verifica il suo bit di modifica, se è inattivo la si può cancellare perché esiste una copia sul disco altrimenti bisogna essere prima copiata e poi sostituita.
Per realizzare la paginazione su richiesta, è necessario risolvere 2 problemi principali: occorre sviluppare un algoritmo di allocazione dei frame e uno per la sostituzione delle pagine. Esistono molti algoritmi per la sostituzione delle pagine probabilmente ogni SO ne ha uno suo.



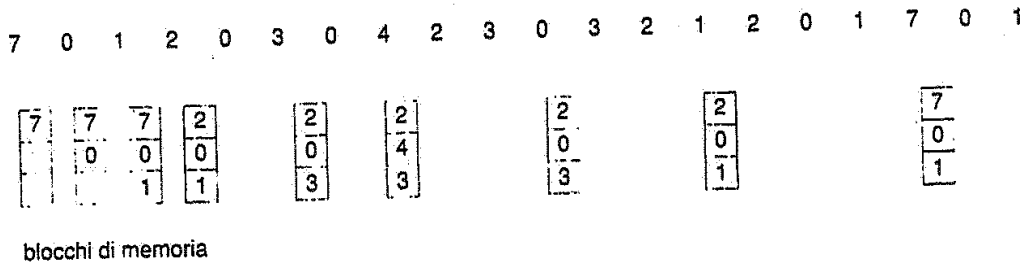
Sostituzione delle pagine secondo l'ordine di arrivo FIFO

l'algoritmo più semplice è quello FIFO. Associa ad ogni pagine l'istante in cui è arrivata e se si deve sostituire un pagina si sceglie quella che sta da più tempo in memoria. Le sue prestazioni non sono sempre buone dato che se nel momento in cui si individua una pagina da sostituire e questa a breve serve di nuovo, questo comporta un ricaricamento della pagina aumentando la frequenza di pagina mancante



Sostituzione ottimale delle pagine

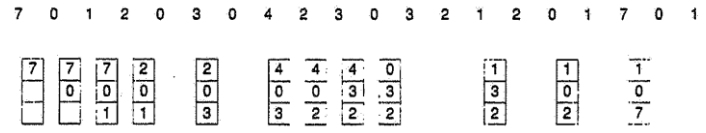
L'algoritmo ottimale è quello che fra tutti gli algoritmi presenta la minima frequenza di assenze di pagine e non presenta mai l'anomalia di Belady (la frequenza delle assenze di pagine può aumentare con l'aumentare del numero dei frame assegnati). L'algoritmo è semplice si sostituisce la pagina che no si userà per il periodo di tempo più a lungo. Sfortunatamente questo algoritmo è difficile da realizzare perché richiede la conoscenza futura delle successione dei riferimenti.



Sostituzione delle pagine usate meno recentemente LRU

La distinzione fondamentale tra quello ottimale e il FIFO consiste nel fatto che il FIFO impiega l'istante in cui una pagina è stata caricata in memoria, mente l'algoritmo ottimale impiega l'istante in cui la pagina è usata. Usando come approssimazione di futuro vicino un passato recente, si sostituisce la pagina che non è stata usata per il periodo più lungo noto come LRU.

Si associa a ogni pagina l'istante e in cui è stata usata per l'ultima volta. Quando si deve sostituire una pagina LRU sceglie quella che non è stata usata per il periodo più lungo. Questa strategia costituisce l'algoritmo ottimale di sostituzione delle pagine con ricerca all'indietro ne tempo. È considerato un algoritmo valido



Il problema principale consiste nel determinare un ordine per i frame definito secondo il momento dell'ultimo usa. Si possono realizzare le seguenti soluzioni:

- **Contatore.** Nel caso più semplice 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 alla pagina si copia il contenuto del registro contatore nel campo del momento d'uso nella tabella relativa a quella specifica pagina. Occorre ricercare nella tabella delle pagine quella usata meno recentemente e aggiornare il campo.
- **Pila.** Un altro metodo è usare una lista doppiamente concatenata che rappresenta il numero delle pagine. Ogni volta che si fa un riferimento a una pagina, la si estrae dalla pila e la so colloca in cima a quest'ultima. In questo modo in cima alla pila si torva sempre la pagina usata per ultima.

Gli algoritmi a pila non presentano l'anomalia di Belady.

Sostituzione della pagine per approssimazione a LRU

Sono pochi i sistemi che utilizzano LRU. Molti sistemi offrono aiuto fornendo un bit di riferimento. Il bit di riferimento a una pagina è impostato automaticamente dall'architettura del sistema ogni volta che si fa riferimento alla pagina. Inizialmente azzera tutu i bit. Quando si inizia l'esecuzione di un processo utente, l'architettura del sistema imposta a 1 il bit associato o a ciascuna pagina. Dopo qualche tempo è possibile stabilire quali pagine sono stata usate semplicemente esamino i bit di riferimento

- **Algoritmo con bit supplementari di riferimento.** Ulteriori informazioni si possono ottenere registrando i bit di riferimento a intervalli regolari. È possibile conservare in una tabella in memoria una serie di bit per ogni pagina. A intervalli regolari il timer invia un segnale. Il SO shifta a destra di un bit. Se ad esempio i registri sono di 8 bit, si tiene conto di 8 intervalli. Se il registro vale 11111111 significa che è stato sempre utilizzato, se 11000100 e 01110111 significa che il primo è stato utilizzato più di frequente. La pagina quindi è associata ad un numero senza segno. Nel caos in cui i bit dei registri sono a lunghezza zero e si lascia solo il bit di riferimento, si utilizza l'algoritmo a seconda chance
- **Algoritmo con seconda chance.** l'algoritmo di base è un algoritmo di sostituzione di tipo FIFO. Dopo aver selezionato una pagina si controlla il bit. Se vale zero si sostituisce, se vale 1 si porta a zero e si continua la scansione. La coda viene scandita in modo circolare. Se una pagina è usata abbastanza spesso, in modo che il suo bit di riferimento valga sempre 1, questa non viene mai sostituita. Una volta trovata la pagina vittima si sostituisce e si s'inserisce la nuova pagina. Nel caso peggiore in cui tutti i bit valgono 1, si deve scorrere tutta la lista per riportare i valori a 0.
- **Algoritmo con seconda chance migliorato.** Si può considerare il bit di modifica con quello di riferimento come una coppia:
  (0,0) ne recentemente usato ne modificato- migliore pagina da sostituire
  (0,1) non usato recentemente ma modificato – prima di essere sostituita deve essere scritta in memoria secondaria
  (1,0) usato recentemente ma non modificato – probabilmente presto sarà di nuovo usata
  (1,1) usata e modificata
  Ogni pagina rientra in una di queste 4 classi. Si usa la scansione circolare ma anziché controllare se la pagina ha il bit di riferimento impostato a 1 e si esamina l'appartenenza a una classe e si seleziona la 1 pagina che si trova nella classe minima non vuoto. La coda deve essere scandita più volte prima di rimuovere una coda.

Sostituzione delle pagine basata su conteggio

Esistono altri algoritmi basati sul conteggio dei riferimenti fatti a ciascuna pagina:

- **Algoritmo di sostituzione delle pagine meno frequentemente usate.** Richiede che si sostituisca la pagina con il conteggio più basso. La ragione di questa scelta è che una pagina usata attivamente ha un conteggio alto. Questo è sconveniente quando la pagina viene

utilizzato molto nella fase iniziale e successivamente non usata, quindi ha un conteggio alto e non può essere sostituita

- **Algoritmo di sostituzione delle pagine più frequentemente usate.** È basato sul fatto che probabilmente la pagina con il contatore più basso stata appena inserita e non è stata ancora usata.

Algoritmi con memorizzazione transitoria delle pagine

Oltre agli algoritmi si possono usare anche altre procedure: generalmente i sistemi usano un gruppo di frame liberi. Quando si verifica un'assenza di pagina si scegli il frame vittima ma prima che si scriva in memoria secondaria si trasferisce la pagina richiesta in un frame libero del gruppo. Questo permette la rapida esecuzione senza che si attendi la scrittura nella memoria secondaria. Ogniqualvolta il dispositivo di paginazione è inattivo, scegli una pagina modificata, la si scrive nel disco e si riempie il suo bit di modifica. Questo permette di recuperare il tempo dell'inattività e di cancellarla qualora è scelta come vittima.

Applicazione e sostituzione della pagina

In alcuni casi le applicazioni che accedono ai dati tramite la memoria virtuale del sistema operativo non conseguono prestazione i miglior di quelle che il sistema, senza impiegare alcun buffer, potrebbe offrire. Alcuni programmi come i database utilizzano spazio di dischi a basso livello in modo da velocizzare alcune operazioni che non vengono effettuate senza file system.

CAPITOLO 10: INTERFACCIA DEL FILE SYSTEM

Per la maggior parte degli utenti il file system è l'aspetto più visibile di un sistema operativo. Il file system consiste di due distinte pari; un insieme di file e una struttura della directory

1. CONCETTO DI FILE

I calcolatori possono memorizzare le informazioni su diversi supporti come dischi, nastri magnetici e dischi ottici. Il SO associa i file a dispositivi fisici, di solito non volatili. Un file è un insieme di informazioni correlate e registrate in memoria secondaria. I file possono essere numerici, alfabetici, ecc. un file ha una struttura definita secondo il tipo: un file di testo è organizzato come una sequenza di caratteri, un file sorgente da una sequenza di procedure.

Attributi dei file

Per comodità degli utenti, ogni file ha un nome che si usa come riferimento. Alcuni sistemi nella composizione dei nomi, distinguono le lettere maiuscole dalle minuscole

Un file ha altri attributi che possono variare secondo il SO che tipicamente sono;

- **Nome.** Il nome simbolico del file è l'unica informazione in forma umana
- **Identificatore.** Si tratta di un'etichetta unica che identifica il file all'interno del file system
- **Tipo.** Questa informazione è necessaria ai sistemi che gestiscono tipi di file diversi
- **Locazione.** Si tratta del puntatore al dispositivo e alla locazione del file
- **Dimensione.** Si tratta della dimensione corrente del file
- **Protezione.** Le informazioni di controllo sull'accesso al file
- **Ora, data e identificatore utente.** Possono essere relative alla creazione, modifica. Sono utili al fine della protezione.

Operazioni sui file

Un file è un tipo di dato astratto. Per definire adeguatamente un file è necessario considerare le operazioni che si possono eseguire su di esso. Il SO deve offrire operazioni per creare, scrivere, leggere, spostare, ecc.

- **Creazione di un file.** Per creare un file è necessario compiere 2 passaggi: si deve trovare lo spazio per il file nel file system e si deve creare un nuovo elemento nella directory in cui registrare il nome del file.
- **Scrittura nel file.** È indispensabile una chiamata che verifichi il nome e le informazioni che si vogliono scrivere
- **Lettura di un file.** Per leggere da un file è necessaria una chiamata di sistema che specifichi il nome del file e la posizione in memoria dove collocare il successivo blocca del file.
- **Riposizionamento.** Si ricerca l'elemento appropriato nella directory e si assegna un nuovo valore al puntatore alla posizione corrente nel file.
- **Cancellazione di un file.** Si cerca l'elemento nella directory associata e si rilascio lo spazio allocato e si elimina l'elemento della directory.
- **Troncamento del file.** Si potrebbe voler cancellare il contenuti di un file ma mantenere i suoi attribuiti, si azzera cosi il suo contenuto.

Queste 6 operazioni di base comprendono l'insieme minimo delle operazioni richieste per i file. Quando si devono effettuare più operazioni su un file, anziché aprire e chiudere lo stesso file, attraverso la chiamata di sistema open si memorizza in una piccola tabella detta tabella dei file aperti il quale nelle operazioni successive provvede ad individuarlo in tale tabella. La realizzazione della open e close è più complicata nei sistemi multiutente quando più utenti possono accedere allo stesso file. Quindi il SO introduce due tabella, uno per ciascun processo e uno per il sistema. Ciascun elemento associato ad ogni processo punta a una tabella del sistema.

A ciascun file aperto sono associate le seguenti informazioni:

- **Puntatore al file.** Indica la posizione corrente nel file dove è possibile effettuare read e write

- **Contatore dei file aperti.** Si tiene conto dei file aperti e non appena un file viene chiuso da tutti i processi, viene tolto dalla tabella
- **Posizione nel disco del file.** La maggior parte delle operazioni richiede al sistema di modificare i dati contenuti nel file. L'informazione per localizzare il file nel disco è mantenuta in memoria.
- **Diritti di accesso.** Ciascun file può essere aperto solo da coloro che hanno i permessi nel farlo.

Alcuni SO permettono ad un file di proteggersi da un accesso concorrente attraverso i lock. Un lock condiviso funziona simile al lock x lettura e un lock esclusivo simile a quello per scrittura.

Tipi di file

Nella progettazione di un file system, ma che dell'intero SO si deve sempre considerata la possibilità che questo ricerca e gestisca i tipi di file. La maggior parte dei SO alla fine del nome del file associa un'estensione costituita da un punto e da una serie di caratteri che identificano i tipi di file.

2. METODI D'ACCESSO

I file memorizza informazioni al momento dell'uso è necessario acceder a queste informazioni e trasferirle in memoria.

Accesso sequenziale

Il più semplice metodo d'accesso è l'accesso sequenziale. Le informazioni del file si elaborano ordinatamente un record dopo l'altro; questo metodo d'accesso è di gran lunga il più comune e usato dagli editor di testo. Le più comuni operazioni solo letture e scrittura. La lettura fa avanzare il puntatore e la scrittura avviene in coda al file.

Accessi diretto

Un altro metodo è l'accesso diretto. Un file è formato da elementi logici record di lunghezza fissa; cioè consente ai programmi di leggere e scrivere rapidamente dagli elementi senza un ordine particolare. Il metodo ad accesso diretto è molto utile quando è necessario accedere immediatamente a grandi quantità di informazioni. Spesso le basi di dati sono di quello tipo. Per il metodo ad accesso diretto si devono modificare le operazioni. Quindi si hanno n dove n è il numero del blocco al posto di read next e write n al posto di write next. Il numero di blocco è un blocco relativo dall'inizio del file a partire dal blocco 0.

3. STRUTTURA DELLA DIRECTORY

Finora si è analizzato il concetto di file system. In realtà il numero di file può variare da zero a molti. In file system di un calcolatore può essere molto ampio.

Struttura della memoria di massa

Un disco può essere interamente usato per un file system. Talvolta per utilizzare altre parti in maniera differente, preferibile concentrare diversi file system su un solo disco o assegnare alcune parti di un disco al file system, altri ad altre. Il disco può essere suddiviso in partizione e in ogni partizione è possibile installare un file system. Chiameremo volume ogni blocco di disco che contiene un file system ogni volume deve avere in se le informazioni sui file presenti nel sistema. Tali informazioni risiedono in una directory del dispositivo o d'indice del volume.

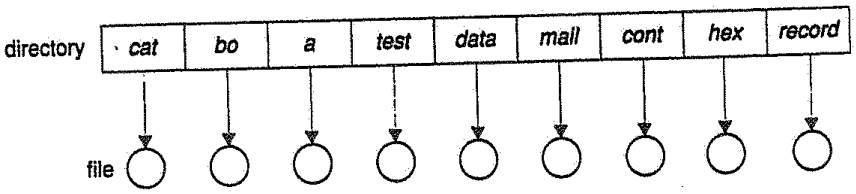
Generalità sulle directory

La directory si può considerare come una tabella di simboliche traduce i nomi dei file negli elementi in essa contenuti. Da questo punti di vista si capisce che la stessa directory si può organizzare in molti modi diversi. Deve essere possibile inserire nuovi elementi cancellarne di esistenti cercare un elemento.

- **Ricerca di un file.** Deve esserci la possibilità di scorrere una directory per individuare l'elemento associato a un particolare file
- **Creazione di un file.** Deve essere possibile creare nuovi file e aggiungerli alla directory
- **Cancellazione di un file.** Quando un file non serve è più deve essere possibile rimuoverli.

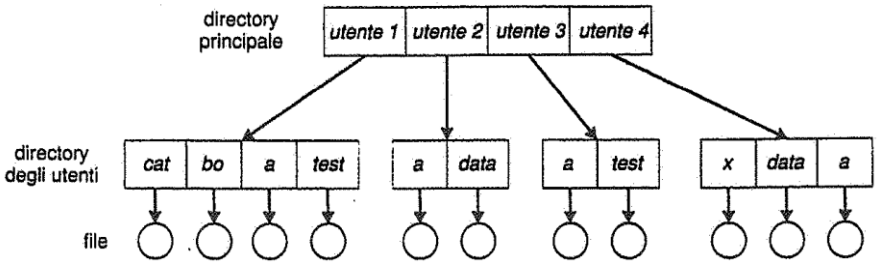
- **Elencazione di una directory.** Deve esistere la possibilità di elencare tutti i file di una directory e il contenuto degli elementi della directory associati ai rispettivi file nell'0elecono
- **Ridenominazione di un file.** Piche un nome di un file rappresenta per i suoi utenti il contenuto del file, questo nome deve poter essere cambiato
- **Attraversamento del file system.** Si potrebbe voler accedere a ogni directory e a ciascun file contenuto di una directory. Per motivi di affidabilità è opportuno salvare in contenuto e la struttura dell'intero file system a intervalli regolari

Directory a livello singolo



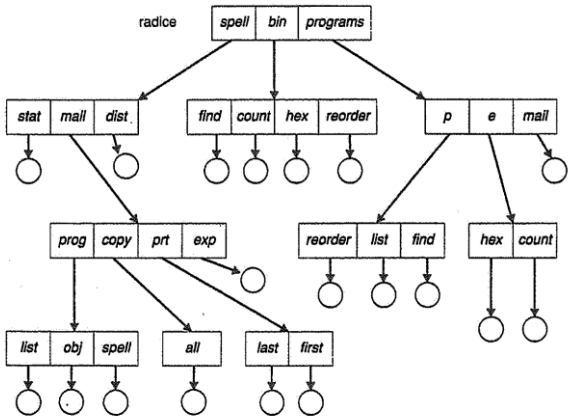
La struttura più semplice di una directory è quella a livello singolo. Tutti i file sono contenuti nella stessa directory. Una directory a livello singolo presenta pero limiti notevoli che si manifestano all'aumentare del numero dei file in essa contenuti oppure se il sistema è usato da più utenti. Poiché si trovano tutti nella stessa directory, i file devono accedere nomi unici; se due utenti volessero creare un file prova non lo possono fare.

Directory a due livelli



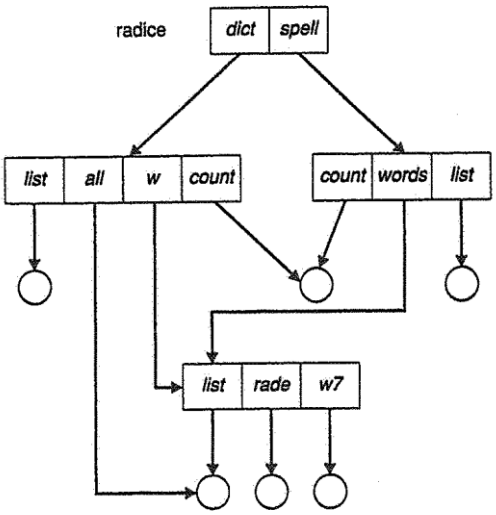
La directory a livello singolo crea confusine sui nomi dei file tra diversi utente. La struttura a due livelli presenta per ogni utente una propria directory utente. Tutte le directory utente hanno una struttura simile ma in ciascuna sono elencate solo i file del proprietario. Quando comincia l'elaborazione di un lavoro dell'utente, si fa una ricerca della nella directory principale del sistema. Quando un utente fa un riferimento a un file particolar, il sistema operativo esegue la ricerca solo nella directory di quell'utente. La struttura della directory a due livelli presenta ancora dei problemi. In effetti questa struttura isola un utente dagli altri. Questo isolamento può essere un vantaggio quando gli utenti sono completamente indipendenti ma uno svantaggio quando gli utenti vogliono cooperare e accederà a file di altri utenti .un directory a 2 livelli è visto come un albero di altezza 2. il nome utente e un nome del file definiscono il path. Un altro problema si riferisce al fatto delle chiamate di sistema e i file di sistema che per non sprecare spazio nel copiare ogni file di sistema in ciascun utente, si definisce una speciale directory utente contente i file di sistemi. Ogni volta che si indica un file da caricare, il sistema operativo lo cerca innanzitutto nella director utente locale e se lo trova, lo usa. Se non lo trova il sistema cerca automaticamente nella speciale directory utente contenente i file di sistema. La sequenza delle directory in cui è stata fatta la ricerca avviata dal riferimento a un file è detta percorso di ricerca

Directory con struttura ad albero



La corrispondenza struttura tra directory a 2 livelli e albero a 2 livelli permette di generalizzare facilmente il concetto. Il file system dell'MS-DOS è strutturato ad albero. L'albero ha una dir radice e ogni file del sistema ha un unico nome di percorso. Una dir contiene un insieme di file o sottodir. Le dir sono semplicemente file trattati in modo diverso. Normalmente ogni utente dispone di una dir corrente. Quando si fa riferimento a u file, si esegue una ricerca nella dir corrente. Se il file non si trova in tale dir, l'utente deve specificare un altro path. La dir corrente iniziale di un utente è stabilita all'avvio del lavoro d'elaborazione. I nomi di percorso possono essere di 2 tipi: percorsi assoluti o relativi. Il primo comincia dalla radice dell'albero, il secondo parte dalla dir corrente. Una decisione importante è il modo in cui effettuare la cancellazione di una dir. Se una dir è vuota la si può eliminare, se è piena o si cancella automaticamente tutto il suo contenuto oppure si devono cancellare manualmente ciascun file o sottodir contenuti e solo quando è vuota si può cancellare.

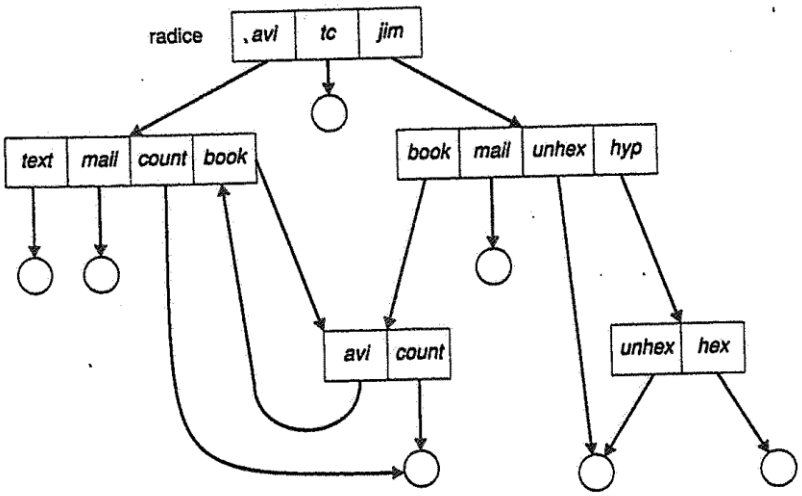
Directory con struttura a grafo aciclico



Si pensi al lavoro di un progetto di 2 programmatori che devono lavorare sullo stesso file, quindi la sottodir comune deve essere condivisa. Nel sistema esiste quindi una dir condivisa o un file condiviso. La struttura ad albero no ammette la condivisione. Un grafo aciclico invece permette alle dir di avere sottodir e file condivisi. Lo stesso file o la stessa sottodir possono essere in due dir diverse. La condivisione non è una copia di un file ma ne esiste uno solo effettivo, perciò tutte le modifiche sono immediatamente visibile. Nel sistema unix, per la condivisione prevede la creazione di un nuovo elemento di dir, chiamato collegamento. Un collegamento è un puntatore a un altro file o dir. Quando si fa riferimento a un file si compie la ricerca nella dir, l'elemento cercato è contrassegnato come collegamento e riporta il nome di percorso del file o della dir reale. Un altro comune metodo per la realizzazione dei file condivisi prevede semplicemente la duplicazione di tutte le informazioni relative ai file in entrambe le dir di condivisione, facendo

sorgere allora il problema della coerenza. Un altro problema è la cancellazione poiché è necessario stabile in quali casi è possibili allocare e utilizzare lo spazio allocato in condivisione. In un sistema con collegamenti simbolici la soluzione è semplice. Il cancellamento di un collegamento non influisce sul file originale mentre se si cancella il file il collegamento non viene rimosso ma è a cura dell'utente rimuoverlo. Il sistema UNIX usa per i collegamenti non simbolici un contatore di ogni collegamento di un file. Quando il contatore vale 0, allora è possibile rimuoverlo.

Directory con struttura a grafo generale



Un serio problema connesso all'uso di una struttura a grafo aciclico consiste nell'assicurare che non vi siano cicli. Quando si aggiungono collegamenti a una struttura ad albero, tale struttura si trasforma in una semplice struttura a grafo.
Il vantaggio principale della struttura a grafo aciclico è data dalla semplicità degli algoritmi necessari per attraversarlo e per determinare quando no ci siano più riferimenti a un file. È preferibile evitare un duplice attraversamento di sezioni condivise idi un grafo ciclico perché costituirebbe una perdita di tempo. Se si permette che esistano cicli, è preferibili evitare la duplice ricerca per evitare cicli infiniti di ricerca. Se esistono cicli il contatore dei link non è zero quindi la cancellazione non potrebbe mai avvenire in UNIX. La difficoltà dei grafi è proprio quella di evitare i cicli quando si aggiungono nuovi collegamenti.

4. MONTAGGIO DI UN FILE SYSTEM

Così come si deve aprire un file per poterlo usare, per essere reso accessibile ai processi di un sistema, un file system deve essere montato. La procedura di montaggio è molto semplice: si fornisce al SO il nome del dispositivo e la sua locazione nella struttura di file e dir alla quale agganciare il file system il passo successivo consiste nella verifica da parte del sistema operativo della validità del file system contenuto nel dispositivo. La verifica si compite chiedendo al driver del dispositivo di leggere la directory di dispositivo e controllando che tale dire abbia il formato previsto. infine il SO annota nella sua struttura della directory che un certo file system è montato a punto di montaggio specificato. Questo schema permette al SO di traversare la sua struttura della dir passando da un file system a un altro secondo le necessita. Un sistema Mac ogni volta che rileva per la prima volta un disco, cerca un file system. Se ne trova uno esegue aromaticamente il montaggio del file alla radice, aggiungendo un'icona sulla scrivania. Micorsoft associa una lettera per ogni volume. Il percorso quindi è lettera:\perocrso\file. Le versioni più recenti di Win è possibile monta r un file system in qualsiasi punto dell'albero.

5. CONDIVISIONE DEI FILE

Una volta che più utenti possono condividere file, l'obiettivo diventa estendere la condivisione a più file system. Infine ci possono essere diverse interpretazione delle azioni conflittuali su file condivisi

Utenti multipli

Se un SO permette l'uso del sistema da parte di più utenti, diventano particolarmente rilevanti i problemi relativi alla condivisione dei file, alla loro identificazione tramite nome e alla loro protezione. Il sistema può permettere a ogni utente di accedere ai file egli altri utente oppure chiedere i permessi di accesso. Per realizzare i meccanismi di condivisione e protezione, il sistema deve memorizzare e gestire gli attributi di dir e file. La maggior parte dei sistema hanno adotta i concetti di proprietario, gruppo e altri. Il proprietario è l'utente che può cambiare gli attributi di un file, concedere l'accesso. L'attributo gruppo invece si definisce al sottoinsieme di utenti autorizzati a condividere l'accesso. Gli identificatori del gruppo e del proprietario sono memorizzati insieme con alti attributi del file.

File system remoti

I metodi con i quali i file si condividono in una rete sono cambiati molto, seguendo l'evoluzione della tecnologia. Un primo metodo consiste nel trasferimento dei file tramite i programmi come l'ftp.

- **Metodo client-server.** I file system remoti permettono il montaggio di uno o più file system di uno o più calcolatori remoti in un calcolatore locale. Il calcolatore contenenti i file si chiamano server, mentre quelli che ricevono sono chiamati client. La relazione tra cliente e server è piuttosto comune tra i calcolatori in rete. Il server in genere specifica e file disponibili su di un volume. Tra le soluzioni per l'autenticazione è quella dell'invio comune di chiavi cifrate. Nel caso di UNIX e del suo file system di rete NFS, l'autenticazione avviene tramite le informazioni di connessione relative al client. I protocolli NFS permettono relazioni da molti a molti, cioè + server possono fornire dati a + client. Una volta montato il file system remoto, l richieste delle operazioni su file sono inviate al server usando il protocollo DFS. Per il montaggio può applicare la stessa semantica dei file system locali. Nella richiesta di apertura di un file, il client invia il nome utente e la richiesta che verrà esaminata dal server per gestire i permessi.
- **Sistemi informativi distribuiti**. Per semplificare la gestione dei servizi client-server i sistemi informativi distribuiti sono stati concepiti per forre un accesso unificato alle informazioni necessarie per il calcolo remoto. Il sistema dei nomi di dominio DNS fornisce le traduzioni dai nomi dei calcolatori agli indirizzi di rete per il web. Altri sistemi informativi distribuiti forniscono uno spazio identificato da nomeutente/pwd/identificatoreutente/idenitficatoregruppo/ per un servizio distribuito. Nel caso delle reti microsoft CIFS le informazioni di rete si usano insieme con gli elementi di autenticazione dell'utente per creare un nome utente di rete che il server usa per decidere se premetter o negare l'accesso. L'industria si sta orientando verso il protocollo LDAP come meccanismo sicura per la comunicazione distribuita. Se il protocolla avrà successo, un'organizzazione potrà usare una singola dir LDAP per memorizzare le informazioni su tutti gli utenti e le risorse di tutti i calcolatori.
- **Malfunzionamenti**. I file system locali possono presentare malfunzionamenti per varie cause: problemi dei dischi che li contengono, alterazione dei dati relativi alle strutture delle dire o a informazioni necessarie alla gestione dei dischi, malfunzionamenti dei controllori dei dischi. Molte cause portano al crollo dei sistema, all'emissione di una condizione d'errore e alla necessita di un intervento umano per risolver il problema. L'uso di file system remoti implica maggior possibilità di malfunzionamenti. Nel caso delle reti si possono verificare interruzioni del collegamento tra due calcolatori, dovute a malfunzionamenti o a improprie configurazioni. Nel caso di un crollo del server o a un malfunzionamento della rete si porta all'inaccessibilità del file system remoto. Invece di comportarsi come la perdita di un file system locale, deve provvedere a terminare tutte le operazioni sul server e attendere che il la connessione ritorni. Per mantener questo tipo di recupero dal malfunzionamento è necessario tenere delle informazioni sullo stato dei client e dei server.

Semantica della coerenza

La semantica della coerenza è un importante criterio per la valutazione di qualsiasi file system che consenta la condivisone dei file. In particolare questa semantica deve specificare quando le modifiche ai dati appartate da un utente possono essere osservate da alti ruteni. È direttamene correlati agli algoritmi di sincronizzazione. Tuttavia a causa delle lunghe latenze e delle basse velocità tra trasferimento dei dischi e delle reti, i complessi algoritmi di solito no si impiegano per l'IO su file.

- **Semantica unix.** Le scritture in un file aperto da parte di un utente sono immediatamente visibili ad altri utenti che hanno aperto contemporaneamente lo stesso file.
- **Semantica delle sessioni.** Le scritture in un file aperto da un utente sono visibili immediatamente ad altri utenti ma una volta chiuso le modifiche sono visibili solo agli utenti che apriranno il file successivamente.
- **Semantica dei file condivisi immutabili.** Un altro metodo p quello dei file condivisi immutabile: una volta che un file è stato dichiarato condiviso, non può essere modificato.

6. **PROTEZIONE**

La salvaguardia delle informazioni contenute in un sistema di calcolo da i danni fisici e da accessi impropri è fondamentale. Generalmente l'affidabilità è assicurata da più copie dei file. Molti calcolatori hanno programmi di sistema che programmano a intervalli regolari la creazione di copie che potrebbero essere cancellati per sbaglio o a causa di danneggiamenti dei dischi.

Tipi d'accesso

La necessita di proteggere i file deriva direttamene dalla possibilità di accedervi. I sistemi che non permettono l'accesso a file di altri utenti non richiedono protezione. Ciò che serve è un accesso controllato. Si possono controllare:

- lettura
- scrittura
- esecuzione
- aggiunta
- cancellazione
- elencazione

sono stai proposti molti meccanismi di protezione e come al solito ogni meccanismo ha vantaggi e svantaggi.

Controllo degli accessi

Il problema della protezione comune è il controllo degli accessi. Lo schema generale per realizzare tale lista è quello dell'ACL a ogni file e dir ossia la lista degli utenti e delle operazioni che si possono fare. Questo ha il vantaggio di permettere complessi metodi d'accesso ma ha lo svantaggio della costruzione di una lista per ogni file e il compito è noioso.

Per risolvere questo problema può essere introdotto una versione diversa della lista dove vengono specificati il proprietario, il gruppo e l'universo. Affinché questo schema funzioni correttamente è necessario uno stretto controllo dei permessi e delle liste di controllo.

Altri metodi di protezione.

Un altro metodo consiste nell'associare una parola chiave, password, a ciascun file. Tener un buon livello di sicurezza delle password consiste nell'avere password complesse e che cambino ogni tot di tempo. Questo presenta un grande svantaggio che è quello di memorizzare numerose password.

CAPITOLO 11: REALIZZAZIONE DEL FILE SYSTEM

Il file system fornisce il meccanismo per la memorizzazione e l'accesso al contenuto dei file, compresi dati e programmi. Il file system risiede permanentemente nella memoria secondaria, progettata per ottenere in modo permanente grandi quantità di dati

1. STRUTTURA DEL FILE SYSTEM

I dischi costituiscono la maggior parte della memoria secondaria in cui si conserva il file system hanno due caratteristiche importanti: si possono scrivere localmente e si può accedere direttamente a qualsiasi blocco. Anziché trasferire un byte alla volta, per migliorare l'efficienza dell'IO vengono trasferiti interi blocchi. Per fornirne un efficiente e conveniente accesso ad disco, il So fa uno di uno o più file system. Un file system presenta due problemi di progettazione piuttosto diversi. Il primo riguarda la definizione dell'aspetto dei file system agli occhi utente. Un file system di solito è composto da molti livelli distinti.

Il livello più basso è il controllo dell'I/O costituito dai driver dei dispositivi e dai gestori di segnali. Un driver di dispositivi può concepire come un traduttore di comandi. Il file system di base deve soltanto inviare dei generici comandi appropriato driver di dispositivo per leggere e scrivere blocchi fisici nel disco. Ogni blocco si identifica col suo indirizzo numerico nel disco. Il modulo di organizzazione dei file è a conoscenza dei file e dei loro blocchi logici e cosi come dei blocchi fisici dei dischi. Conoscendo il tipo di allocazione dei file usato e la locazione dei file, può tradurre gli indirizzi dei blocchi negli indirizzi dei blocchi fisici. Infine il file system logico gestisce i metadati; si tratta di tute le strutture del file system eccetto gli effettivi dati.
Nei file system stratificati la duplicazione di codice è ridotta al minimo. Il controllo dell'I/O e talvolta il codice di base del file system possono essere comuni a numerosi file system che pi gestiscono il file system logico e i moduli per l'organizzazione dei file.

2. REALIZZAZIONE DEL FILE SYSTEM

Introduzione

Per realizzare un file system si usano parecchie strutture dati, sia nei dischi sia in memoria. Queste strutture variano secondo il SO e il file system. Nei dischi il file system teine informazioni su come eseguire l'avviamento di un SO memorizzato nei dischi stessi, il numero totale di blocchi, il numero e la locazione dei blocchi liberi. Molte di loro sono analizzate in modo particolare. Fra le strutture dati troviamo:

- **il blocco di controllo dell'avviamento.** Il boot control block contiene le informazioni al sistema per l'avviamento di un SO da quel volume. Se un disco non contiene un So quel blocco è vuoto.
- **Blocchi di controllo dei volumi.** Ciascuno di loro contiene i dettagli relativo al volume come il numero e la dimensione dei blocchi

- **Le strutture delle directory.** Usate per organizzare i file. Nel caso di unix in nomi dei file e i numeri sono memorizzati nell'i-node in win ossia con il file system NTFS nella tabella principale dei file (master file table)
- **I blocchi di controllo dei file. FCB** Contenenti molti dettagli dei file compresi i permessi

Le informazioni tenute in memoria servono sia per la gestione del file system sia per migliorare le prestazione attraverso l'uso d cache. Id dati si caricano al momento del montaggio e si eliminano allo smontaggio. Le strutture contenenti queste informazioni comprendono: la tabella di montaggio interna alla memoria che contiene informazioni relative a ciascun volume montato; la struttura delle directory tenute in memoria contenente le informazioni relative a tute le dir a cui i processi hanno avuto accesso di recente; la tabella generale dei file aperti; la tabella dei file aperti per ciascuno processo.
Le applicazioni per creare un nuovo file, eseguono una chiamata al file system logico, il quale conosce il formato della struttura della dir. Il sistema carica quindi la dir. Esso crea e alloca un nuovo FCB. Alcuni SO, compreso unix, trattano le dir esattamente come i file, distinguendoli con un campo per il tipo, mentre il SO win NT dispone di chiamate di sistema appropriato per le dir.
Una volta che il file è stato trovato, si copia l'FCB nella tabella generale di file aperti. Questa tabella non contiene solo l'FCB ma tiene conto anche del numero di processi che utilizza il file. Il nome dato all'elemento della tabella è detto FD per unix e file handle per win.

Partizioni e montaggio

Un disco si può configurare in vari modi, secondo il SO che lo gestisce. Si può suddivider in più partizioni, oppure un volume può comprendere più partizioni su molteplici dischi.
Ciascuna partizione è priva di struttura logica se non contiene alcun file system ed è detto raw disk. Alcuni SO quali UNIX impiega una partizione priva di struttura per l'area di avvicendamento. Un disco privo di stortura logica può anche contenere informazioni necessarie per sistemi RAID di gestione dei dischi. Le informazioni relative all'avviamento del sistema si possono registrare in un'apposita partizione, che anche il questo caso ha un proprio formato perché nella fase di avviamento il sistema non ha ancora caricato i driver di dispositivo del file system quindi non può interpretano il formato. Nei pc si può configurare una installazione di più SO e durante la fase di boot l'utente sceglie quale SO avviare. Nella fase di caricamento il SO esegue il montaggio della partizione radice che contiene il kernel dei sistema operativo e in alcuni casi altri file di sistema. Secondo il sistema operativo, il montaggio degli altri volumi avviene automaticamente in questa fase oppure si può compiere successivamente in modo esplicito. Durante l'operazione di montaggio, il sistema verifica la coerenza della partizione e una eventuale correzione. Infine annota nella struttura della tabella di montaggio che un file system è stato montato.

File system virtuali

un metodo ovvio ma non ottimale per realizzare più tipi di file system è scrivere procedure di gestione di file e directory separate per ciascun tipo di file system. Al contrario la maggior parte dei SO impiega tecniche orientate agli oggetti per semplificare e organizzare in maniera modulare la soluzione. Gli utenti possono accedere a file contenuti in più file system nei dischi locali oppure tramite la rete. Per isolare le funzioni i base delle chiamate di sistema dai dettagli di realizzazione si adoperano apposite strutture dati. In questo modo la realizzazione del file system si articola in tre strati principali. Il primo strato è l'interfaccia del file system, basata sulle chiamate di sistema open,.., e sui descrittori di file. il secondo strato si chiama strato del file syste$_m$ virtuale *VFS* e svolge due funzioni importanti.
Separa le operazioni generiche del file system dalla loro realizzazione definendo un'interfaccia VFS uniforme.
Permette la rappresentazione univoca di un file su tutta la rete. Il VFS è basato su una struttura di rappresentazione dei file detta vnode che contiene un indicatore numerico unico per tutta la rete per ciascun file.
Quindi, il VFS distingue i file locali da quelli remoti, e distingue i file locali secondo i relativi tipi di file system.

I quattro tipi più importanti di oggetti definiti in questo sistema sono:
l'oggetto inode, che rappresenta il singolo file;
l'oggetto file, che rappresenta un file aperto;
l'oggetto superblock, che rappresenta un intero file system;
l'oggetto dentry, che rappresenta il singolo elemento della directory.
Per ognuno di questi tipi, VFS specifica un insieme di operazioni da implementare. Ogni implementazione dell'oggetto file per uno specifico tipo di file deve implementare tutte le funzioni specificate nella definizione dell'oggetto file.

3. REALIZZAZIONE DELLE DIRECTORY

La selezione degli algoritmi di allocazione e degli algoritmi di gestione delle dir ha un grande effetto sull'efficienza, le prestazioni e l'affidabilità del file system.

Lista lineare

Il più semplice metodo di realizzazione di una dir è basato sull'uso si una lista lineare contenente i nomi dei file con puntatori ai blocchi di dati. Questo metodo *è* di facile programmazione, ma la sua esecuzione *è* onerosa in termini di tempo. Per creare un nuovo file occorre prima esaminare la directory per essere sicuri che non esista gia un file con Io stesso nome, quindi aggiungere un nuovo elemento alla fine della dir. Per cancellare un file occorre cercare nella dir il file con quel nome, quindi rilasciare lo spazio che gli era assegnano. Esistono vari metodi per riutilizzare un elemento della directory: si può contrassegnare l'elemento come un usato oppure può essere aggiunto a una lista di elementi di dir liberi una terza possibilità prevede la copiatura dell'ultimo elemento della dir in una locazione liberata e a diminuzione della lunghezza della directory. Il vero svantaggio dato da una lista lineare di elementi di directory è dato dalla ricerca lineare di un file. Le informazioni sulla diretto vengono usate frequentemente, e gli utenti avvertirebbero una gestione lenta e accesso a tali informazioni. In effetti, molti SO impiegano una cache per memorizzare le informazioni sulla dir usata più recentemente. Una lista ordinata permette una ricerca binaria e riduce il tempo medio di ricerca. Un vantaggio della lista ordinata *è* che consente di produrre l'elemento, ordinato del contenuto della directory senza una fase d'ordinamento separata.

Tabella hash

Un'altra struttura dati che si usa per realizzare le directory *è* la tabella hash. In questo metodo una lista lineare contiene gli elementi di directory, ma si usa anche una struttura dati hash. La tabella hash riceve un valore calcolato usando come operando il nome del file e riportai puntatore al nome del file nella lista lineare. Attraverso questa struttura dati si può diminuire nettamente del tempo di ricerca nella dir. L'inserimento e la cancellazione sono abbastanza semplici anche se occorre prender provvedimenti per evitare collisioni. Per eliminare la collisione invece che un singolo valore, può essere concatenata a ciascun elemento una lista.

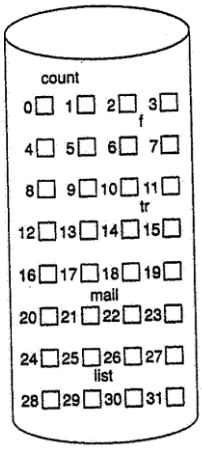
4. METODI DI ALLOCAZIONE

La natura ad accesso diretto dei dischi permette una certa flessibilità nella realizzazione dei file. Molti file si memorizzano nello stesso disco. Il problema principale consiste dunque nell'allocare lo spazio per questi file in modo che lo spazio nel disco sia usato efficientemente e l'accesso ai file sia rapido. Esistono 3 metodi principali per l'allocazione della spazio di un disco: può essere contigua, concatenata o indicizzata.

Allocazione contigua

Per usare il metodi di allocazione contigua ogni file deve occupare un insieme di blocchi contigui del disco. Gli indirizzi del disco definiscono un ordinamento lineare nel disco stesso. Con questo ordinamento l'accesso al blocco b+1 dopo il blocco b non richiede normalmente alcuno spostamento della testina. Quindi il numero dei posizionamenti richiesti per accedere a file il cui spazio è allocato il modo contiguo è trascurabile, cosi come è trascurabile il tempo di ricerca.

Accedere ad un file in cui la sua allocazione è contigua, è facile. Quando si usa un accesso sequenziale, il file system memorizza l'indirizzo dell'ultimo blocco a cui è stato fatto riferimenti e se è necessario legge il blocco successivo. L'allocazione contigua presenta pero alcuni problemi una difficoltà riguarda l'individuazione dello spazio per un nuovo file. La realizzazione del sistema di gestione dello spazio libero determina il modo in cui tale compito viene eseguito. Si può usare ogni sistema di gestione anche se alcuni sono più lenti di altri
Per l'allocazione contigua si possono usare gli stessi criteri dell'allocazione dinamica della memoria: il problema generale infatti è quello di soddisfare una richiesta di dimensione n data una lista di buchi liberi. I più comuni criteri di ricerca del buco libero sono first fit e best fit. Questi algoritmi soffrono della frammentazione esterna. Una soluzione detta deframmentazione è quella di copiare i file system in un altro supporto e successivamente riscriverlo in modo compatto ma questo sprecava molto temo. Alcuni sistemi dovevano effettuarlo offline, i nuovi lo fanno ondine. Un altro problema è da determinazione della quantità di spazio necessaria. Un file può avere poco spazio e quindi si possono avere 2 soluzioni: o nel momento della creazione si da molto spazio oppure nel momento che questo cresce si copia in un buco libero più grande e quello precedente si libera. Per ridurre al minimo questo problema alcuni SO fanno uso di una versione modificata: inizialmente si assegna una porzione di spazio contiguo e se questa non p abbastanza grande si aggiunge un'altra porzione di spazio.

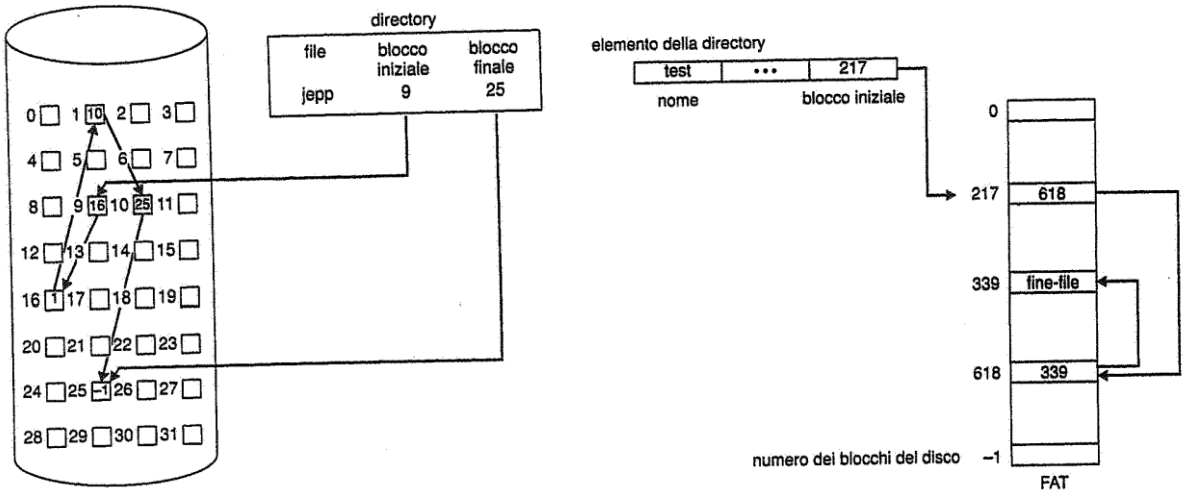


directory

file	blocco iniziale	lunghezza
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Allocazione concatenata

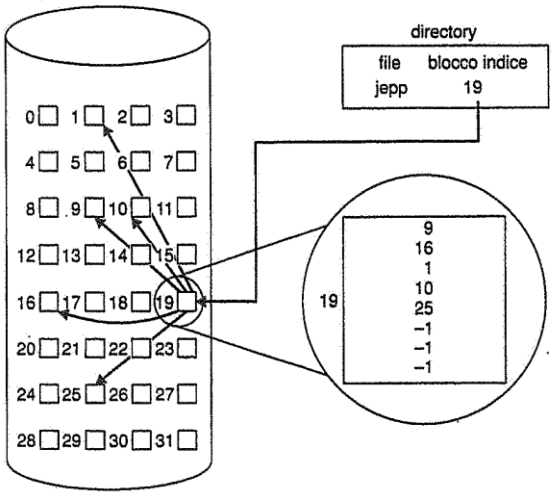
L'allocazione concatenata risolve tutti i problemi sorti dall'allocazione contigua. Con questo tipo di allocazione ogni file è composto da una lista concatenata di blocchi del disco i quali possono essere sparsi ovunque. Per creare un nuovo file si crea semplicemente un nuovo elemento della dir. Un'operazione di scrittura determina la ricerca di un blocco libero, la scrittura di tale blocco e la concatenazione di tale blocco. Per leggere un file basta semplicemente scorrere tutto il file. Non è necessario dichiarare a priori la grandezza del file. Gli svantaggi di questo tipo di allocazione è che può essere usata efficientemente solo per i file ad accesso sequenziale in quanto quelli diretti non è possibile in modo veloce sapere il blocco i. un altro svantaggio è lo spazio richiesto per i puntatori. Per risolvere questo problema è possibile riunire un numero di blocchi detti cluster ed effettuare il puntamento solo di cluster a cluster. Questo metodo migliora la produttività del disco. Un altro problema riguarda l'affidabilità. Poiché sono tenuti molti puntatori, se per errore uno di questi venisse cancellato, non si potrebbe accedere a tutto il file. Una variante importante del metodo di allocazione concatenata consiste nell'uso della tabella di allocazione dei file FAT. Per contenere tale tabella si riserva una sezione del disco all'inizio di ciascun volume. La FAT ha un elemento per ogni blocco del disco ed è indicizzata dal numero di blocco, si usa essenzialmente come una lista concatenata. Lo schema di allocazione basato sulla FAT se non si usa una cache può causare un significativo numero di posizionamenti della testina. La testina deve spostarsi all'inizio per leggere la FAT e poi leggere il blocco.



Allocazione indicizzata

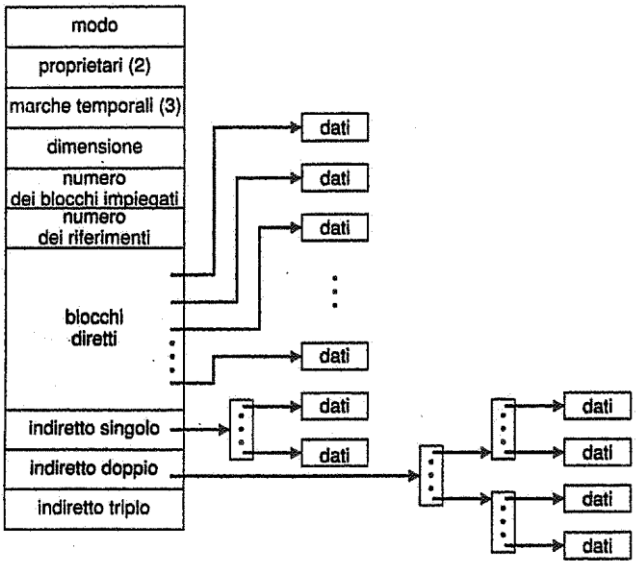
L'allocazione concatenata risolve il problema della frammentazione esterna e quello della dichiarazione della dimensione dei file. Tuttavia in mancanza di una FAT l'allocazione concatenata non è in grado di sostenere un efficiente accesso diretto. L'allocazione indicizzata risolve questo problema raggruppando tutti i puntatori in una sola locazione. Ogni file ha il proprio blocco indice: si tratta di un array d'indirizzi di blocchi del disco.

Una volta creato il file, tutti i puntatori del blocco indice sono impostati a nil. Non appena si scrive il blocco i si aggiorna il puntatore i. l'allocazione indicizzata consente l'accesso diretto senza soffrire di frammentazione esterna.



Ogni file deve avere un blocco indice. Ci sono diversi schemi di realizzazione dei blocchi indici:

- **Schema concatenato.** Un blocco indice è formato normalmente di uno solo blocco di disco perciò ciascun blocco indice può essere letto e scritto esattamente con un'operazione. Per permettere la presenza di lunghi file è possibile collegare tra loro parecchi blocchi indice.
- **Indice a più livelli.** Una variante della rappresentazione concatenata consiste nell'impiego di un blocco indice di primo livello che punta a un insieme di blocchi indice di secondo livello che loro volta puntano ai blocchi dei file.
- **Schema combinato.** Un'altra possibilità è la soluzione adottata da UNIX che consiste nell'avere i primi 15 puntatori del indice dell'inode. I primi 12 puntano a blocchi su disco, gli altri 3 puntano a blocchi indiretti che sono rispettivamente singolo, doppio e triplo.



Prestazioni

I metodi d'allocazione presentati hanno diversi livelli di efficienza di memorizzazione e differenti tempi d'accesso ai blocchi di dati. Prima di sceglier un metodo di allocazione è necessario determinare il modo in cui si usano i sistemi: un sistema con una prevalenza di accessi sequenziali farà uso in un metodo differente da quello con prevalenza di accessi diretti.

Per qualsiasi tipo di accesso, l'allocazione contigua richiede un solo accesso per ottener un blocco. Poiché è facile tenere l'indirizzo iniziale de file in memoria, si può calcolare immediatamente l'indirizzo del disco dell'i-esimo blocco. Con l'allocazione concatenata si può tenere in memoria anche l'indirizzo del blocco successivo e leggerlo direttamente. Questo metodo è valido per l'accesso sequenziale mentre per quel che riguarda l'accesso diretto un accesso all'i-simo blocco può richiedere i letture del disco. Da tutto ciò segue che alcuni sistemi gestiscono i file ad accesso diretto usando l'allocazione contigua e i file ad accesso sequenziale tramite l'allocazione concatenata. L'allocazione indicizzata è più complessa. Se il blocco indice è gia in memoria l'accesso può essere diretto. Tuttavia per tenere il blocco indice in memoria occorre una quantità di spazio notevole. Le prestazioni di tale allocazione dipenono dalla struttura dell'indice, dalla dimensione del file e dalla posizione del blocco. Alcuni SO combinano le allocazioni in base alle proprie esigenze.

5. GESTIONE DELLO SPAZIO LIBERO

Poiché la quantità di spazio dei dischi è limitata, è necessario utilizzare lo spazio lasciato dai file cancellati per scrivere nuovi file. Per tener traccia dello spazio libero in un disco, il sistema conserva una lista dello spazio libero. Per creare file occorre cercare nella lista dello spazio la quantità di spazio necessaria e assegnarla al nuovo file

Vettore di bit

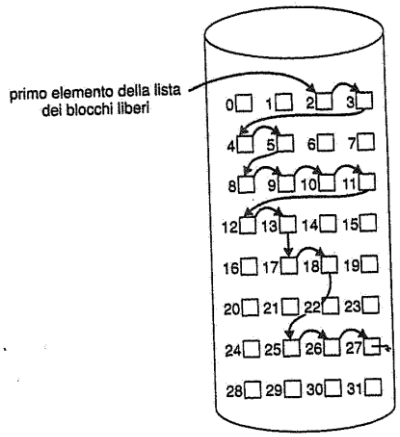
Spesso la lista dello spazio libero si realizza come una mappa di bit o vettore di bit. ogni blocco è rappresentato da un bit: se il blocco è libero il bit è 1, altrimenti e 0. i vantaggi principali che derivano da questo metodo sono la sua relativa semplice ed efficienza nel trovare il primo blocco libero e n blocchi liberi consecutivi nel disco.

Sfortunatamente i vettori di bit sono efficienti solo se tutto il vettore è mantenuto in memoria centrale e viene di tanto in tanto scritto in memoria secondaria allo scopo di consentire eventuali operazioni di ripristino.

Lista concatenata

Un altro metodo di gestione degli spazi liberi consiste nel collegarli tutti, tenere un puntatore al primo di questi in una speciale locazione del disco e caricalo in memoria. Questo primo blocco contiene un puntatore al successivo blocco libero e cosi via. Questo schema non è tuttavia

efficiente: per attraversare la lista occorre leggere ogni blocco e l'operazione richiede un notevole tempo di IO. Fortunatamente l'attraversamento della lista dello spazio libero non è un'operazione frequente. Il metodo oche fa uso della fat include il conteggio dei blocchi liberi nella struttura dati per l'allocazione.



Raggruppamento

Una possibile modifica del metodo della lista dello spazio libero prevede la memorizzazione degli indirizzi in n blocchi liberi nel primo di questi. I primi n-1 sono effettivamente liberi, l'ultimo contiene l'indirizzo ai successivi n blocchi liberi. L'importanza è data dalla possibilista di trovare rapidamente gli indirizzi di un gran numero di blocchi liberi.

Conteggio

Generalmente più blocchi contigui si possono allocare o liberare contemporaneamente, soprattutto quando lo spazio viene allocato usando l'algoritmo di allocazione. Anziché tener una lista di n indirizzi liberi, è sufficiente tenere l'indirizzo del primo e il numero di blocchi liberi.

6. EFFICIENZA E PRESTAZIONI

I dischi tendono di solito a essere il principale collo di bottiglia per le prestazioni in una sistema

Efficienza

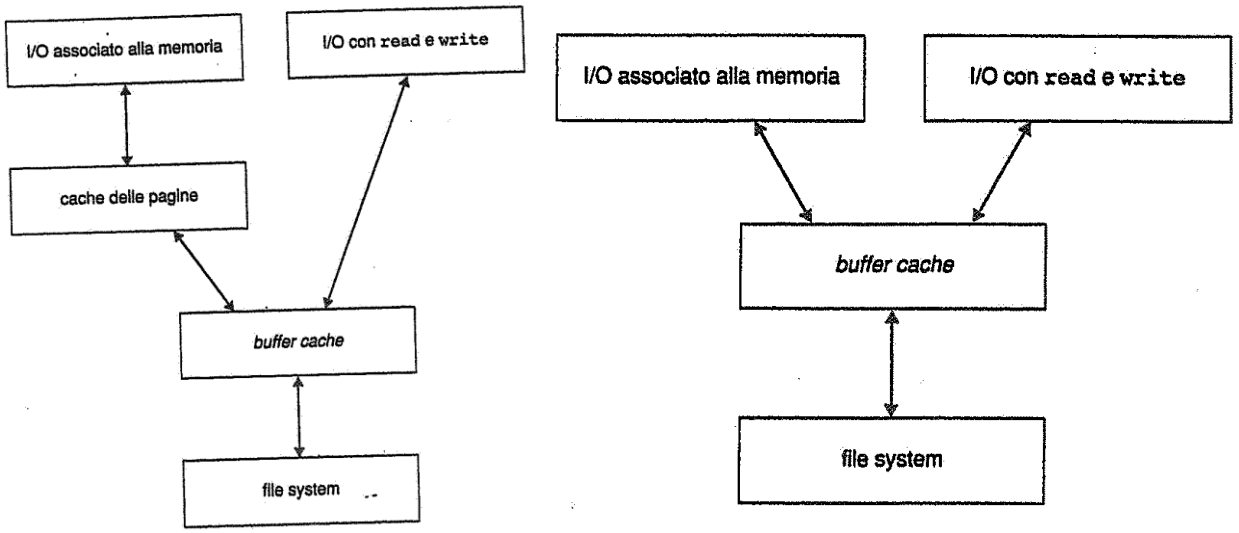
L'uso efficiente di un disco dipende fortemente dagli algoritmi usati per l'allocazione del disco e la gestione delle dir. Si devono tenere in considerazione anche il tipo di dati normalmente contenuti in un elemento. Di solito si memorizza la data dell'ultima scrittura, altri dell'ultimo accesso aggiornando ogni volta un campo della dir. Questa modifica richiede la lettura nella memoria del blocco, la modifica e la riscrittura.

Una delle difficoltà nella scelta della dimensione dei puntatori o di qualsiasi altra dimensione di allocazione fissa all'interno di un SO è la pianificazione degli effetti provocati dal cambiamento della tecnologia. Con la crescita della capacita dei dischi, i dischi più grandi si dovevano suddividere in partizioni di 32 MB fino a quando non è stato modificato il file system. Il modo della gestione delle strutture e l'allocazione dinamica. Naturalmente gli algoritmo che manipolano queste tabelle sono ora più complessi e il SO è un può più lento dovendo allocare e rilasciare dinamicamente gli elementi.

Prestazioni

Dopo aver scelto gli algoritmi fondamentali del file system le prestazioni possono essere migliorate in diversi modi. Alcuni controllori di unita a disco contengono una quantica di memoria locale sufficiente grande da memorizzare un'intera traccia del disco alla volta. Si legge la traccia e il controllore trasferisce quindi al SO tutte le richieste di settore. Quando i blocchi sono trasferiti dal controllore del disco alla memoria centrale, il sistema operativo ha la possibilità di inserirli in una propria cache nella memoria centrale. Alcuni sistemi riservano una sezione separata della memoria centrale come cache del disco, altri impiegano una cache delle pagine per i file. Questo metodo è noto come memoria virtuale unificata. Alcune versioni di linux e unix prevedono la cosiddetta buffer cache unificata. Serve per risolvere i problemi derivati dalla double caching ossia nell'accesso ad un file sia tramite la mappatura dei file sia tramite l'ordinaria chiamata di sistema.

Con una buffer cache unificata sia l'associazione alla memoria che le chiamate di sistema usano la stessa cache.



In generale l'algoritmo LRU è ragionevole per sostituire blocchi e le pagine. Le scritture sincrone avvengono nell'ordine in cui le riceve il sottosistema per la gestione del disco. Nella maggior parte dei casi si usano scritture asincrone. Nelle scritture asincrone si memorizzano i dati nella cache e si restituisce immediatamente il controllo alla procedura. gli accessi frequenti si potrebbero invece ottimizzare con tecniche note come rilascio indietro e lettura anticipato. Il rilascio indietro rimuove una pagina dalla memoria di transito non appena si verifica una lettura di pagina successiva; le pagine precedenti non saranno usate. La lettura anticipata invece si leggono e si mettono nella cache la pagina richiesta e parecchie successive poiché si pensi che serviranno presto.
La cache delle pagine, il file system e i driver del disco interagisco in modi interessanti. Quando i dati vengono scritti su un file del disco, le pagine sono memorizzate nella cache che qui funge da buffer mentre il driver del disco ordina la propria coda di dati in uscita in base all'indirizzo sul disco. Per grandi quantità di dati la scrittura è più veloce della lettura se avviene tramite file system

7. **RIPRISTINO**

Poiché i file e le dir sono mantenuti sia in memoria centrale sia nei dischi, è necessario aver cura di assicurare che il verificarsi di un malfunzionamenti nel sistema non comporti la perdita di dati.

Verifica della coerenza

Una parte delle informazioni contenute nella dir è mantenuta in memoria centrale o in cache allo scopo di accelerarne gli accessi. Queste informazioni sono generalmente più aggiornate delle corrispondenti informazioni presenti in memoria secondaria piche la scrittura nei dischi dei dati contenuti nella cache non si verifica necessariamente nell'istante in cui occorrono le modifiche. Si consideri il crollo del sistema, il contenuto della cache e del buffer sono persi. Questo evento può lasciare il file system in uno stato di incoerenza. Il verificatore della coerenza è un programma di sistema che confronta i dati delle dir con quelli contenuti nei blocchi dei dischi tendono di correggere ogni incoerenza. Unix usa la cache per la lettura mentre la scrittura avviene in modo sincrono.

Copie di riserva e recupero dei dati

Piche possono verificarsi malfunzionamenti e perdite dei dati anche nei dischi magnetici, c'è bisogno che i dati presenti nei dischi vengono salvati altrove in modo che un eventuale perdita posso essere rapidamente risolta. Ciò avviene tramite le cosiddette copie di riserva o comunemente conosciute come backup.

8. **FILE SYSTEM CON REGISTRAZIONE DELLE MODIFICHE**

Spesso si adottano algoritmi e tecnologie anche nel caso in cui sono stati progettati per altre situazioni. È il caso degli algoritmi per il ripristino sviluppati nell'area dei sistemi di gestione delle basi dati. Questi algoritmi sono stati applicati con successo al problema della verifica della

coerenza. Si ricordi che le strutture dati del file system che risiedono nei dischi, come le strutture delle dir, i puntatori a blocchi possono diventerà incoerenti nel caso di un crollo del sistema. Il metodo che consente l'incoerenza delle strutture per poi correggere gli errori in una fase di ripristino presenta diversi problemi tra questi è che l'incoerenza potrebbe essere irrisolvibile. La soluzione a questo problema consiste nell'applicare agli aggiornamenti dei metadati relativi al file system metodi di ripristino basati sulla registrazione delle modifiche. Fondamentalmente tutte le modifiche vengono annotate in un file log. Ogni insieme di operazioni che esegue uno specifico compito si chiama transazione e non appena le modifiche sono riportate nel file di registrazione, le operazioni si considerano portate a termine. Se si verifica un crollo del sistema si verifica il contenuto del log. Le transazioni presenti non sono mai state ultimate nel file system anche se il SO le definisce portate a termine.

CAPITOLO 12: MEMORIA SECONDARIA E TERZIARIA

1. STRUTTURA DEI DISPOSITIVI DI MEMORIZZAZIONE

Dischi magnetici

I dischi magnetici sono il mezzo fondamentale di memoria secondaria dei moderni sistemi di calcolo. Concettualmente i dischi sono relativamente semplice: i piatti hanno una forma piana e una superficie di material magnetico dove le informazione vengono registrate magneticamente. Le testine di lettura e scrittura sono sospesa su ciascuna superficie d'ogni piatto e sono attaccate al braccio del disco che le muove in blocco. La superficie di un piatto è divisa logicamente in tracce circolari a loro volta suddivise in settore. L'insieme delle tracce corrispondenti a un braccio formano un cilindro. In un'unita a disco possono esservi migliaia di cilindri. Quando un disco è in funzione un motore lo fa ruotare ad alta velocità. L'efficienza è caratterizzato dal tempo di trasferimento, il tempo di posizionamento e il tempo di ricerca. Poiché le testine di un disco sono sospese su un cuscino d'aria sottilissimo, esiste il pericolo che la testina urti la superficie in tal caso si parla di crollo della testina e causa la rottura irreversibile del disco. Un disco può essere rimovibile ciò permette che diversi dischi siano montanti secondo le necessita. I dischetti sono dischi magnetici dell'rodine di 1.44 MB dove la tesina viene poggiata sul disco per leggere e scrivere e per questo ruota più lentamente. I dischi possono essere collegati attraverso de fili detti bus e possono essere EIDE, SATA, PATA, SCSI, USB. Il trasferimento dei dati in un bus è eseguito da speciali unita di elaborazione dette controllori. Gli adattatori o controllori di macchina sono i controllori posti all'estremità relativa al calcolatore del bus.

Nastri magnetici

I nastri magnetici sono stati i primi supporti di memorizzazione secondaria. Pur avendo una veloce trasferimento, il tempo di posizionamento è troppo elevato. Gli usi principali adesso sono quelli della creazione di backup.

2. STRUTTURA DEI DISCHI

I moderni dischi sono considerati come grandi array monodimensionale di blocchi logici, dove un blocco logico è la minima unita di trasferimento. La dimensione di un blocco logico è di solito di 512 byte sebbene alcuni dischi si possono formattare a basso livello allo scopo di ottener una diversa dimensione dei blocchi logici. L'array monodimensionale di blocchi logici corrisponde in modo sequenziale ai settori del disco. Sfruttando questa corrispondenza almeno in teoria sarebbe possibile trasformare gli indirizzi logici in indirizzi fisici costituiti da numero di cilindro, traccia, settore. In pratica vi sono 2 motivi che rendono difficile quest'operazione in primo luogo la maggior parte dei dischi contiene settori difettosi, in secondo luogo il numero di settori per traccia in certe unita a disco no è costante. Nei supporti che impiegano velocità lineare costante la densità di bit per traccia è uniforme. Più è lontana dal centro del disco, tanto maggiore è la lunghezza della traccia. L'unita aumenta la sua velocità di rotazione man mano che le testine vanno verso l'esterno (cd e dvd). Mente quando la rotazione rimane la stessa la densità di bit decresce dalla tracce interne alle tracce più esterne e mantener e costante il flusso dei dati. Questo metodo è detto velocità angolare costante.

3. CONNESSIONE DEI DISCHI

I calcolatori accedono alla memoria secondaria in due modi: nei sistemi di piccole dimensioni il modo più comune è tramite le porte di IO, oppure in modo remoto per mezzo di un file system distribuito.

Memoria secondaria connessa alla macchina

Alla memoria secondaria connessa alla macchina si accede tramite le porte locali di IO. I comuni pc usano IDE o ATA. Le stazioni di lavoro a fibra ottica e SCSI. L'architettura SCSI è un'architettura a bus il cui supporto fisico è di solito un cavo piato con un gran numero di conduttori. Consente di

avere sul bus fino a 16 dischi. L'FC è un'architettura seriale ad alta velocità che può funzionare sia su fibra che su cavo e ci si aspetta grande potenzialità da questa connessione.

Memoria secondaria connessa alla rete

Un dispositivo di memoria secondaria connessa alla rete è un sistema di memoria special al quale si accede in modo remoto per mezzo di una rete id trasmissione di dati. I client accedono alla memoria connessa alla rete tramite un'interfaccia RPC come NFS per unix o CIFS per win.

La memoria secondaria connessa alla rete fornisce un modo semplice per condividere spazio di memorizzazione a tutti i calcolatori di una lan con la stessa facilita di gestione dei nome degli accessi caratteristica della memoria secondaria locale. Tuttavia un sistema del genere ha prestazioni inferiori rispetto a memorie secondarie locali.

4. SCHEDULING DEL DISCO

Una delle responsabilità del sistema operativo è quella di fare un uso efficiente delle risorse fisiche, nel caso delle unita a disco far fronte a questa responsabilità significa garantire tempo d'accesso contenuti e ampiezze di banda elevate. Il tempo d'accesso si può scindere in due componenti: tempo di ricerca cioè il tempo necessario finche il braccio dell'unita a disco sposti

Le testine fino al cilindro, latenza di rotazione cioè il tempo aggiuntivo necessario perché il disco ruoti finche il settore desiderato si trovi sotto la testina.

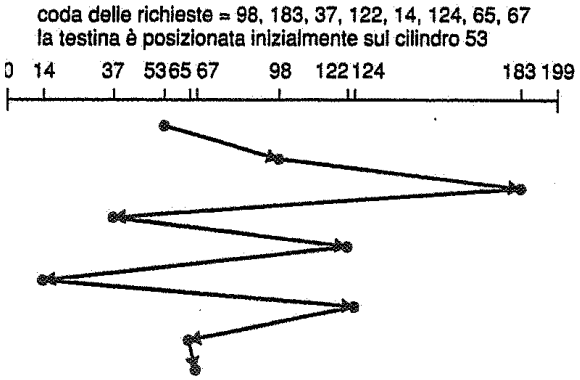
Ogni volta che devono compiere operazioni di IO con un'unita a disco, un processo impartisce al sistema operativo una chiamata di sistema. La richiesta contiene diverse informazioni:

- Se l'operazione sia di immissione o emissione dati
- L'indirizzo del disco
- L'indirizzo di memoria al quale eseguire il trasferimento
- Il numero di byte da trasferire.

Se l'unita a disco desiderata e il controllore sono disponibili, la richiesta si può immediatamente soddisfare; altrimenti le nuove richiese si aggiungono alla coda di richieste inevase relativa a quell'unita. La coda relativa a un'unita a disco in un sistema con multiprogrammazione può spesso essere piuttosto lunga, sicché il SO sceglie quale fra le richieste inevase conviene scrivere prima.

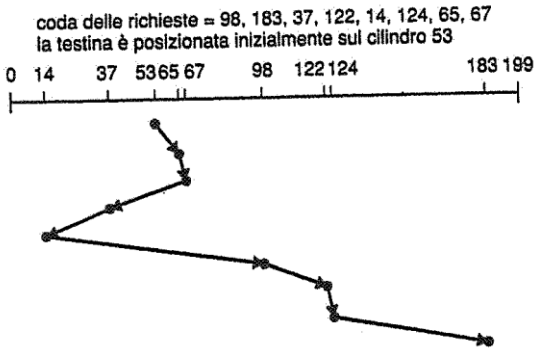
Scheduling in ordine d'arrivo FCFS

La forma più semplice è quello di esaudire le richieste in base all'ordine di arrivo.



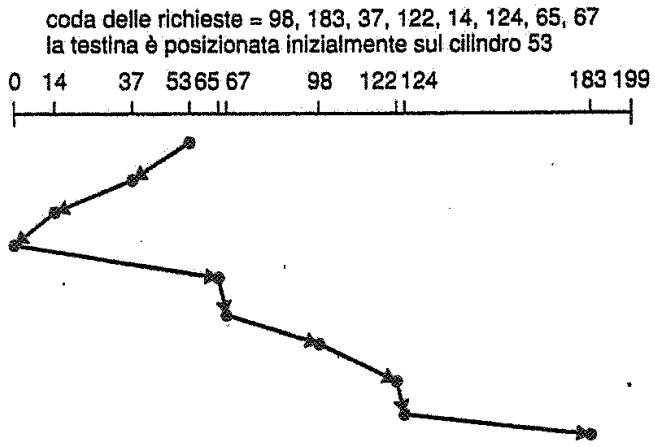
Scheduling per brevità SSTF

Sembra ragionevole servire le richieste più vicine alla posizione corrente prima di spostarla in un'area più lontana. Riscegli la richiesta che da minimo tempo di spostamento. Riduce di un terzo il percorso del FCFS ma nonostante ciò ha dei svantaggi che è quello della starvation e poi non è ottimale.



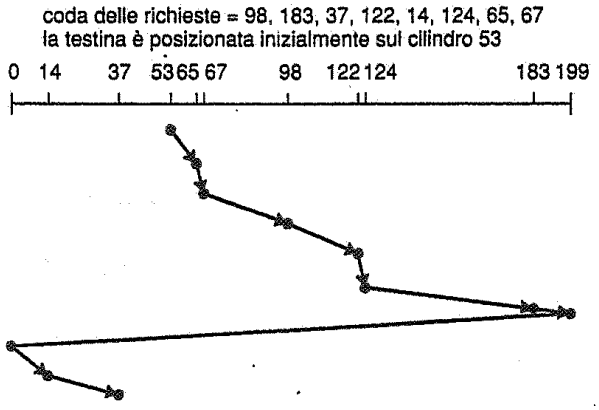
Scheduling per scansione S-SCAN

Secondo l'algoritmo S-SCAN il braccio dell'unita a disco parte da un estremo del disco e si sposta verso l'atra estremità risolvendo le richieste che incontra durante il percorso. Se una nuova richiesta arriva e la testina ha superato da poco tale posizione, essa deve attendere che la testina arriva all'altra estremità e inverte la sua marcia. è conosciuto anche come algoritmo dell'ascensore. Si noti che poche sono le richieste immediatamente vicine durante il cambio di marcia.



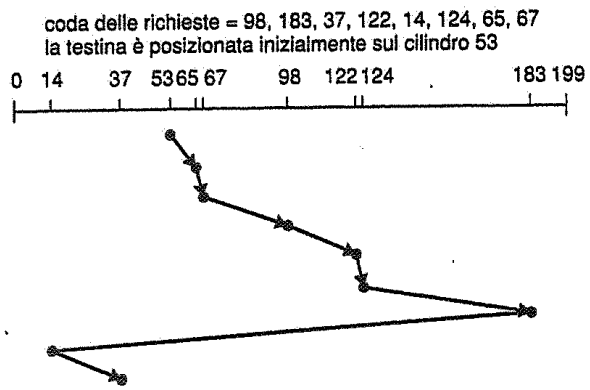
Scheduling per scansione circolare C-SCAN

L'algoritmo C-SCAN è una variante dell'S-SCAN e la variazione è che non inverte il senso di marcia bensi inizia dall'inizio del disco stesso vedendo il disco come una lista circolare



Scheduling look

Secondo la descrizione appena fatta, sia l'algoritmo SCAN che C-SCAN spostano il braccio dell'unita attraverso tutta l'ampiezza del disco; all'atto pratico nessuno dei sue algoritmi è codificato in questo modo: più comunemente il braccio si sposta solo finche ci sono altre richieste da servire in quella direzione dopo di che cambia immediatamente direzione senza giungere all'stremo del disco.



Scelta di un algoritmo di scheduling

Giacche esistono tanti diversi algoritmi bisogna individuare quale sia il migliore. Per qualunque algoritmo di scheduling, le prestazioni dipendono comunque in larga misura dal numero e dal tipo di richiesta. Anche la posizione delle dir e dei blocchi indici è importante piche ogni file deve essere aperto per essere usato e visto che l'apertura di un file richiede una ricerca attraverso la struttura della dir, vi saranno frequenti accessi alla directory. A causa dei diversi problemi lo scheduling del disco dovrebbe costituire un modulo a se stante del SO cosi come da poter essere sostituto da un altro algoritmo qualora ciò non fosse necessario. I produttori di unita a disco hanno collaborato alla limitazione di questi problemi incorporando tali algoritmi all'interno dei controllori contenuti nelle unita a disco facendo si che in automatico vengono scelti gli algoritmi più appropriati.

5. GESTIONE DELL'UNITA' A DISCO

Il SO è anche responsabile di molti altri aspetti della gestione delle unita a disco.

Formattazione del disco

Un disco magnetico nuovo è tabula rasa: un insieme di uno o più piatti privi di file system. Per prima cosa deve essere diviso in settori e questa fase è detta formattazione fisica o di basso livello. Si riempio il disco di una struttura dati per ogni settore consistente di una intestazione e una coda. L'intestazione e la coda contengono informazioni usate dal controllo re del disco ad esempio il numero del settore e un codice per la correzione degli errori EEC. Quando si effettua un'ordinaria operazione si aggiorna il valore EEC secondo il contenuto dell'area e i dati del settore. Se risulta una discrepanza tra EEC e i dati letti nel settore significa che il settore potrebbe essere difettoso. La formattazione fisica è effettuata in larga parte da i costruttori in modo che testino anche il disco e controllino in numero dei settori danneggiati. Per usare il disco il SO deve creare uno o più partizioni e in ognuna creare un file system.

blocco d'avviamento

affinché un calcolatore possa entrare in funzione ad esempio quando viene accesa o riavviata è necessario che esegua un programma iniziale; i solito questo programma d'avviamento iniziale è piuttosto semplice. Esso inizializza in SO in tutti i suoi aspetti, dai registri della CPU all'avvio del SO. Per far ciò il programma d'avviamento trova il kernel del SO lo carica nella memoria e salta a un indirizzo iniziale per avviar l'esecuzione del SO. Per la maggior parte dei calcolatori il blocco d'avviamento viene memorizzato nella ROM. Poiché la ROM non è aggiornabile, nella ROM viene memorizzato un caricatore d'avviamento il quale va a leggere all'inizio della partizione che contiene il SO e quest'ultimo permette l'avvio. Questo sistema colloca nel primo settore del disco chiamato MBR il codice di avviamento e una tabella che indica le varie partizioni e un flag che indica in quale partizione è presente il SO.

Blocchi difettosi

Le unita a disco sono strutturalmente portate ai malfunzionamenti perché sono costituiti da parti mobile a bassa tolleranza. A volte può verificarsi un guasto irreparabile e l'unita a disco deve essere sostituita. Più frequentemente invece uno o più settori diventa difettoso. Essi sono trattati a differenza del disco e secondo il controllore.

Nel DOS bisognava chiamare una funzione del SO che permetteva di correggere o prendere soluzioni nel caso in cui uno di questi settori diventasse malfunzionante. Unita a disco più sofisticate come dischi SCI hanno strategie di recupero dei blocchi difettosi. Il controllore tiene una lista dei blocchi malfunzionante dell'unita a disco che è inizializzata durante la formattazione fisica da parte del produttore e aggiornata man mano che si verificano nuovi malfunzionamenti. La formattazione mette anche a disposizione dei settori di riserva non visibili al SO nel caso in cui uno diventi difettoso viene rimpiazzato da uno di questi settori. La maggior parte di questi dischi mette a disposizione interi cilindri per questa operazione.

6. GESTIONE DELL'AREA D'AVVICENDAMENTO

L'avvicendamento è stato introdotto dove abbiamo trattato lo spostamento di interi processi tra disco e memoria centrale. In quel contesto l'avvicendamento interviene quando l'ammontare della memoria fisica si abbassa fino al punto di raggiungere la sogli critica e i processi passano dalla memoria all'area di avvicendamento per liberare memoria. Nella pratica pochissimi SO realizzano l'avvicendamento nel modo descritto: essi infatti, combinano l'avvicendamento con tecniche di memoria virtuale per coinvolgere nell'operazione solo alcune pagine e on necessariamente interi processi. Tant'è che alcuni sistemi considerano avvicendamento e paginazione termini intercambiabile.

La gestione dell'area d'avvicendamento è un altro compito di basso libello del sistema operativo la memoria virtuale usa lo spazio dei dischi come esenzione della membra centrale: poiché l'accesso alle unita a disco è molto più lento dell'accesso alla memoria centrale, l'uso di un'area d'avvicendamento riduce notevolmente le prestazioni del stima. L'obiettivo principale nella progettazione e realizzazione di un'area di avvicendamento è di fornire la migliore produttività per il system della memoria virtuale.

Uso dell'area d'avvicendamento

L'area d'avvicendamento è usata in modi diversi da sistemi operativi diversi in funzione degli algoritmi di gestione della memoria applicati. I sistemi che adottano l'avvicendamento dei processi nella moria possono usare l'area di avvicendamento per mantener l'intera immagine del processo inclusi si segmenti dei dati e del codice; i sistemi a paginazione invece possono semplicemente memorizzarvi pagine non contenute nella memoria centrale. Lo spazio richiesto dall'area d'avvicendamento per un sistema può quindi variare secondo la quanto di memoria fisica la memoria virtuale.

Un sistema che esaurisca l'area d'avvicendamento potrebbe essere costretto a terminare forzatamente i processi o ad arrestarsi completamente.

Collocazione dell'are d'avvicendamento

Le possibili collocazioni per un'area d'avvicendamento sono due: all'interno del normale file system o in una partizione del disco a se stante. Se l'area d'avvicendamento è semplicemente un grande file all'interno de file system, si possono usare le ordinarie funzioni dei file system pere crearla, assegnargli un nome e allocare spazio per essa. Questo criterio sebbene sia semplice da realizzare è inefficiente: l'attraversamento della struttura delle dir e l'uso delle strutture dati per l'allocazione dello spazio nei dischi richiede tempo. Le prestazioni si possono migliorare impiegando la memoria fisica come cache per le informazioni relative alla posizione dei blocchi e anche usando strutture speciali per l'allocazione in blocchi fisicamente contigui del file d'avvicendamento, ma il costo dovuto all'attraversamento del file system e delle sue strutture dati permane. In alternativa all'area di avvicendamento si può creare un'apposta partizione del disco non formattata e si velocizza perché no si deve attraversare il file system. Si crea frammentazione interna ma dato che questa area ha vita breve, è un prezzo che si può pagare.

7. STRUTTURE RAID

L'evoluzione tecnologica ha reso le unita a disco progressivamente più piccole e meno costose. Oggi è possibile equipaggiare sistemi con più dischi senza spendere somme esorbitanti. Attraverso la configurazione raid è possibile da un lato velocizzare l'accesso e da un altro effettua copie di dischi qualora uno potrebbe rompersi.

Miglioramenti dell'affidabilità tramite ridondanza

La possibilità che un disco si rompa si guasti è molto alta molto più alta della possibilità che uno specifico disco isolato presenti un guasto. La soluzione dell'affidabilità sta nell'introdurre una ridondanza e quindi in caso di guasto basta sostituire il disco. Il metodo più semplice è quello del mirroring ogni disco logico consiste di due fisici e ogni scrittura si effettua in entrambi id dischi. Ciò è sufficiente a risolvere i problemi dipesi da un guasto relativo alla rottura di un disco no pero nel caso di un disastro naturale.

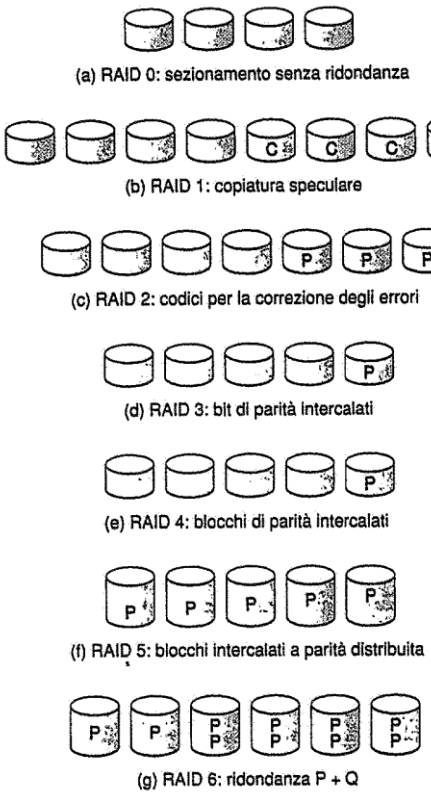
Nel caso in cui si hanno interruzioni di corrente e una scrittura no n andasse a buon fine, si deve avere il modo per non far corrompere il file system. una soluzione prevede la scrittura in uno solo dei dischi e solo successivamente nell'altro in modo da avere nella copia sempre scritture portate a termine.

Miglioramento delle prestazioni tramite il parallelismo

L'access in parallelo di più dischi può portare grossi vantaggi. Con la copiatura speculare dei dischi la frequenza con la quale si possono gestire le richieste di lettura raddoppia poiché ciascuna richiesta si può inviare indifferentemente a uno dei due dischi. Per migliorare la capacita di trasferimento i dati vengono distribuiti in sezione su più dischi. Questa forma nota come sezionamento dei dati, consiste nel distribuire i bit di ciascun byte su pi dischi facendo si che la quantità di dati letti o scritti sai 8 volte superiore. Questo pero porta ad un grave svantaggio, nel caso in cui uno dei dischi si rompa, tutti i dati sono irrecuperabile.

Livelli RAID

La tecnica copiatura speculare offre un'alta affidabilità ma è costosa, la tecnica del sezionamento offre un'alta capacita di trasferimento ma non migliora l'affidabilità.



- **Raid 0.** il livello 0 si riferisce a batteria dei dischi on sezionamento a livello di blocchi ma senza ridondanza
- **Raid 1.** il livello 1 si riferisce alla copiatura speculare
- **Raid 2.** noto anche come organizzazione dei codice per la correzione degli errori ECC. da molto tempo i dischi usano tecniche di riconoscimento degli errori. In un sistema di questo tipo ogni byte di memoria ha associato un bit di parità che indica con valore 1 sono in numero pari o dispari.
- **Raid 3.** con il livello 3 o organizzazione con bit di parità intercalati, si migliora il raid 2 usando un unico bit di parità per individuare gli errori. Se uno dei settori è danneggiato si conosce esattamente di quale settore si tratta e calcolando la parità si conosce il suo valore. È migliore rispetto al 2 quindi il 2 non viene proprio usato. È migliore dell'1 per 2 motivi: 1 perchè si usa un solo disco per la parità anziché uno per ogni disco, 2 e che essendo la lettura e scrittura su byte distribuiti la velocità aumenta di n volte.
- **Raid 4.** noto come organizzazione con blocchi di parità intercalati, s'impiega il sezionamento a livello dei blocchi come nel raid 0 e inoltre si tiene un blocco di parità in un disco separato. Se un dei dischi si guasta, il blocco di parità serve per recuperare il contenuto.
- **Raid 5.** noto come organizzazione con blocchi intercalati a parità distribuita, differisce dal 4 per il fatto che invece di memorizzare i dati in n dischi e la parità in un disco separati, i dati e le informazioni di parità sono distribuite tra gli n+1 dischi. Ogni blocco memorizza la parità e gli altri i dati. È il più comune sistema raid.
- **Raid 6.** l livello 6 noto anche come schema di ridondanza P+Q è molto esimie al5 5 ma memorizza ulteriori informazioni ridondanti per potere gestire guasti contemporanei di più dischi. Invece della parità si usano codici di read solomon.
- **Raid 0+1.** il livello 0+1 consiste in una combinazione del raid 0 e raid 1. il livello 1 fornisce le prestazioni mentre l'1 l'affidabilità. Se si guasta un singolo disco, 'intera sezione di dati diventa inaccessibile, lasciando disponibile solo l'altra sezione
- **Raid 1+0.** si fa prima la copiatura e poi si sezionano i dati. Con un guasto nel raid 1+0 il singolo disco diventa inaccessibile ma il suo duplicato è ancora disponibile, come tutti gli altri dischi.

Il raid può essere implementato sia a livello software che hardware. Un'altra caratteristica del raid è la previsione di dischi di scorta che possono sostituire immediatamente dischi rotti.

8. STRUTTURE PER LA MEMORIZZAZIONE TERZIARIA

Dispositivi per la memorizzazione terziaria

La memoria terziaria consiste di mezzi rimovibili di cui dischetti, cd rom e dvd
I dischi rimuovili sono costituiti da dischi ottici cd e dvd che possono essere riscrivibile oppure a sola lettura. La scrittura avviene attraverso un laser che magnetizza un settore.

Nastri

Costituite da una bobina che gira intorno a ma il tempo di riavvolgimento è troppo lento.

Tecnologie future

Memorizzazione oleografica, adopera la luce laser per memorizzare ologrammi su mezzi speciali.

CAPITOLO 13: SISTEMI DI I/O

I due compiti principali di un calcolatore sono l'I/O e l'elaborazione. Spesso il compito principale è costituito dall'I/O mentre l'elaborazione è semplicemente accessoria.
Il ruolo di un sistema operativo nell'I/O di un calcolatore è quello di gestire e controllare le operazioni e i dispositivi di I/ O.

6. INDRODUZIONE

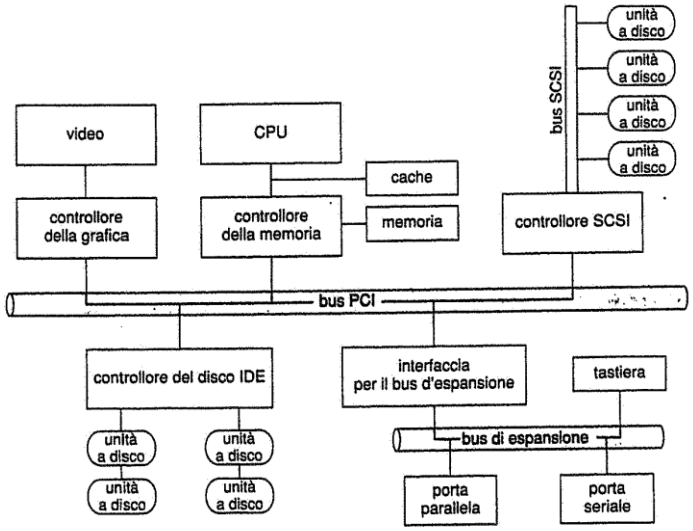
Il controllo dei dispositivi connessi a un calcolatore è una delle questioni più importanti che riguardano i progettisti di sistemi operativi. Poiché i dispositivi di I/O sono così largamente diversi per funzioni e velocità ad esempio un mouse, un disco e un jukebox di CD-ROM, e altrettanto diversi devono essere i metodi di controllo. Tali metodi costruiscono il *sottosistema di I/O* del kernel.
D'altra parte, però, si assiste a una crescente varietà di dispositivi di I/O; alcuni di loro sono tanto diversi dai dispositivi precedenti dal rendere molto difficile il compito di integrarli nei calcolatori e nei sistemi operativi esistenti.

7. ARCHITETTURE E DISPOSITIVI DI I/O

I calcolatori fanno funzionare un gran numero di tipi di dispositivi: la maggior parte rientra nella categoria dei dispositivi di memorizzazione secondaria e terziaria , dispositivi di trasmissione.
Un dispositivo comunica con un sistema di calcolo inviando segnali attraverso un cavo o attraverso l'etere e comunica con il calcolatore tramite un punto di connessione (porta), ad esempio una porta seriale. Se uno o più dispositivi usano in comune un insieme di fili. La connessione e detta bus. Un bus è un insieme di fili e un protocollo rigorosamente definito che specifica l'insieme dei messaggi che si possono inviare attraverso i fili. In termini elettronici, i messaggi si inviano tramite configurazioni di livelli di tensione elettrica applicate ai fili con una definita scansione temporale.
I bus sono ampiamente usati nell'architettura dei calcolatori.
Un calcolatore è un insieme di componenti elettronici che può far funzionare un porta, un bus o un dispositivo. Un controllore di porta seriale è un semplice controllore di dispositivo; di tratta di un singolo circuito integrato nel calcolatore che controlla i segnali presenti nei fili della porta seriale.
Alcuni dispositivi sono dotati di propri controllori incorporati.
L'unità d'elaborazione dà comandi e fornisce dati al controllore per portare a termine trasferimenti di I/O tramite uno o più registri per dati e segnali di controllo. La comunicazione con il controllore avviene attraverso la lettura e la scrittura, da parte dell'unità d'elaborazione, di configurazioni di bit in questi registri. Un modo in cui questa comunicazione può avvenire è tramite l'uso di speciali istruzioni di I/O che specificano il trasferimento di un byte o una parola a un indirizzo di porta I/O. l'istruzione di I/O attiva le linee di bus per selezionare il giusto dispositivo e trasferire bit dentro o fuori dal registro di dispositivo.
Certi sistemi usano le tecniche. I PC ad esempio usano istruzioni di I/O per controllare alcuni dispositivi e l'I/O mappato in memoria per controllare altri. il controllore della grafica ha alcune porte di I/O per le operazioni di controllo di base, ma dispone di un'ampia regione lappata in memoria, detta memoria grafica, che serve a mantenere i contenuti dello schermo. Il processo scrive sullo schermo inserendo i dati nella regione lappata in memoria; il controllore genera l'immagine dello schermo sulla base del contenuto di questa regione di memoria.
Una porta di I/O consiste in genere in quattro registri: status, control, data-in e data-out.

- La CPU legge il registro data-in per ricevere dati.
- La CPU scrive nel registro data-out per emettere dati.
- Il registro status contiene alcuni bit che possono essere letti e indicano lo stato della porta; ad esempio indicano se è stata portata a termine l'esecuzione del comando corrente

Il registro control può essere scritto per attivare un comando o per cambiare il modo di funzionamento del dispositivo. Ad esempio, un certo bit nel registro control della porta seriale determina il tipo di comunicazione tra *half-duplex e full-duplex*
La tipica dimensione dei registri di dati varia tra 1 e 4 byte. Certi controllori hanno circuiti integrati FIFO che possono contenere parecchi byte per l'immissione e l'emissione dei dati, in modo da espandere la capacità del controllore oltre la dimensione del registro di dati.



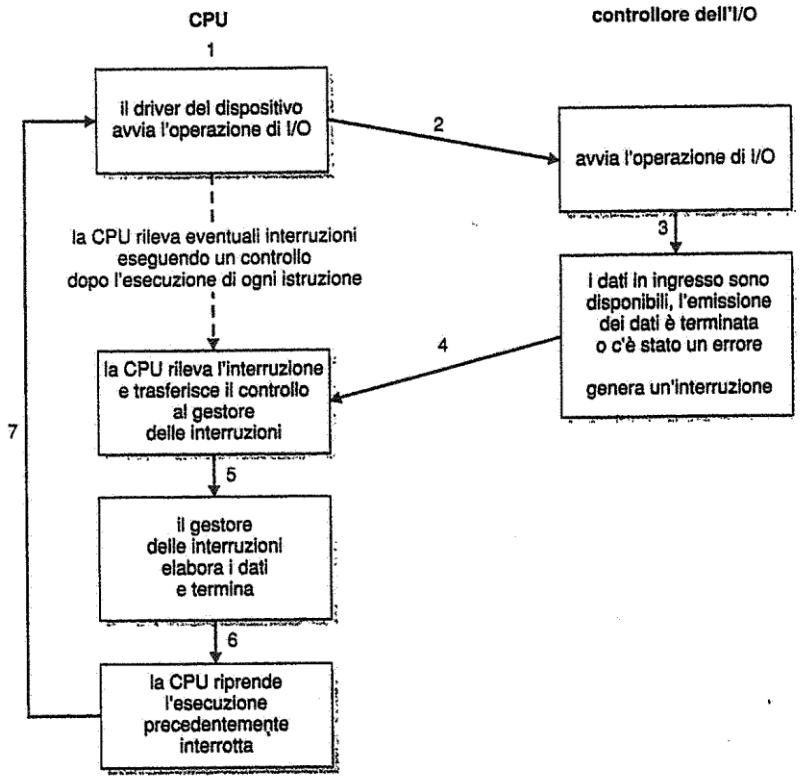
Interrogazione ciclica

Il protocollo completo per l'interazione fra la CPU e un controllore può essere intricato, ma la fondamentale nozione di negoziazione *(handshaking)* è semplice, ed è illustra con un esempio.
Il controllore specifica il suo stato per mezzo del bit busy del registro status; pone a 1 il bit busy quando è impegnato in un'operazione, e lo pone a 0 quando è pronto a eseguire il comando successo. La CPU comunica le sue richieste tramite il bit comand-ready nel registro command; pone questo bit a 1 quando il controllore deve eseguire un comando.

1. La CPU legge ripetutamente il bit busy finché non valga 0.
2. La CPU pone a 1 il bit write del registro dei comandi e scrive un byte nel registro data-out
3. La CPU pone a 1 il bit command-ready .
4. Quando il controllore si accorge che il bit command-ready è posto a 1, pone a 1 il bit busy
5. Il controllore legge il registro dei comandi e trova il comando write; legge il registro data-out per ottenere il byte da scrivere, e compie l'operazione di scrittura nel dispositivo
6. Il controllore pone a 0 il bit command-ready, pone a 0 il bit error nel registro status per indicare che l'operazione di I/O ha avuto esito positivo, e pone a 0 il bit busy per indicare che l'operazione è terminata.

La sequenza appena descritta si ripete per ogni byte.
Durante l'esecuzione del passo 1, la CPU è in attesa attiva (*busy-waiting)* o in interrogazione ciclica (polling): itera la lettura del registro status finché il bit busy assume il valore 0. Se il controllore e il dispositivo sono veloci, questo metodo è ragionevole, ma se l'attesa rischia di prolungarsi, sarebbe probabilmente meglio se la CPU si dedicasse a un'altra operazione.
Quando, ad esempio, i dati affluiscono in una porta seriale o della tastiera. Il piccolo buffer del controllore diverrà presto pieno, e se la CPU attende troppo a lungo prima di riprendere la lettura dei byte, si prenderanno informazioni.
In molte architetture di calcolatori sono sufficienti tre cicli di istruzioni di CPU per in-terrogare ciclicamente un dispositivo: read, lettura di un registro del dispositivo; logical-and, configurazione

logica usata per estrarre il valore di un bit di stato, e branch, salto a un altro punto del codice se l'argomento è diverso da zero.
Anziché richiede alla CPU di eseguire un'interrogazione ciclica, può essere più efficiente far si che il controllore comunichi alla CPU che il dispositivo è pronto. Il meccanismo dall'architettura che permette tale comunicazione si chiama interruzione della CPU o, più brevemente, interruzione *(interrupt).*



Interruzioni

La CPU ha un contatto, detto linea di richiesta dell'interruzione, del quale la CPU controlla lo stato dopo l'esecuzione di ogni istruzione. Quando rileva il segnale di un controllore nella linea di richiesta dell'interruzione, la CPU salva lo stato corrente e salta alla routine di gestione dell' interruzione *(interrupt-handler routine),* che si trova e un indirizzo prefissato di memoria. Questa procedura determina le cause dell'interruzione, porta a termine l'elaborazione necessaria ad esegue un'istruzione return from interrupt per far sì che la CPU ritorni nello stato in cui si trovava prima della sua interruzione. Il controllore del dispositivo *genera* un segnale d'interruzione della CPU lungo la linea di richiesta delle interruzioni, che la CPU *rileva* e *recapita* al gestore delle interruzioni, che a sua *evade* il compito corrispondente servendo il dispositivo.
Nei sistemi operativi moderni necessarie capacità di gestione delle interruzioni più raffinate.

1. Si deve poter posporre la gestione dell'interruzione durante le fasi critiche dell'elaborazione.
2. Si deve disporre di un meccanismo efficiente per passare il controllo all'appropriato gestore delle interruzioni, senza dover esaminare ciclicamente tutti i dispositivi *(polling)* per determinare quale abbia generato l'interruzione.
3. Si deve disporre di più livelli d'interruzione, di modo che il sistema possa distinguere le interruzioni ad altra priorità da quelle a priorità inferiore, servendo le richieste con la celerità appropriata del caso.

In un calcolatore moderno queste tre caratteristiche sono fornite dalla CPU e dal controllore delle interruzioni .

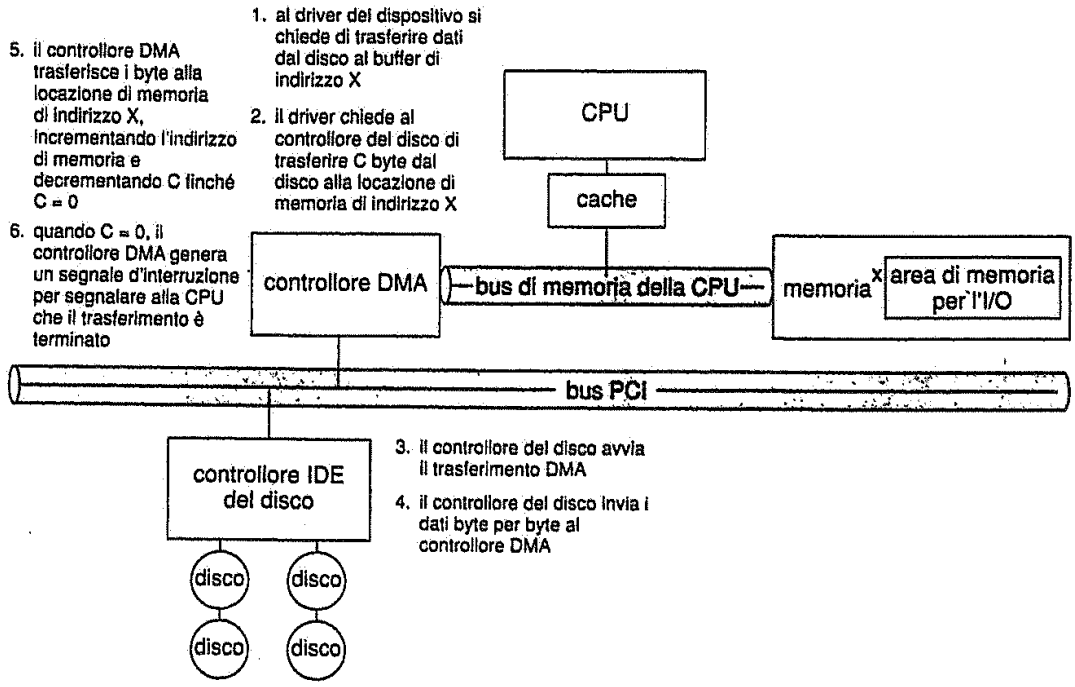
La maggior parte delle CPU ha due linee di richiesta delle interruzioni. Una è quella delle interruzioni non mascherabili, riservata a eventi quali gli errori di memoria irrecuperabili. La secondo linea è quella delle interruzioni mascherabili: può essere disattivata dalla CPU prima dell'esecuzione di una sequenza critica di istruzioni che non deve essere interrotta.
Il meccanismo delle interruzioni accetta un indirizzo.
Nella maggior parte delle architetture questo indirizzo è uno scostamento relativo a una tabella detta vettore delle interruzioni, contenente gli indirizzi di memoria gli specifici gestori delle interruzioni.
In pratica, tuttavia, i calcolatori hanno più dispositivi (e quindi, più gestori delle interruzioni) che elementi nel servirsi di una tecnica detta concatenamento delle interruzioni (*interrupt chaining)*, in cui ogni elemento del vettore delle interruzioni punta alla testa di una lista di gestori nella lista delle interruzioni. Quando si verifica un'interruzione, si chiamano uno alla volta i gestori nella lista corrispondente finché non se ne trova uno che può soddisfare la richiesta .
E descritto il vettore delle interruzioni della CPU Intel Pentium.
Il meccanismo delle interruzioni realizza anche un sistema di livelli di proprietà delle interruzioni. Esso permette alla CPU di differire la gestione delle interruzioni di bassa priorità senza mascherare tutte le interruzioni, e permette a un'interruzione di priorità alta di sospendere l'esecuzione della procedura di servizio di un 'interruzione di priorità bassa. Un sistema operativo moderno interagisce con il meccanismo delle interruzioni in vari modi. All'accensione della machina esamina i bus per determinare quali dispositivi siano presenti, e installa gli indirizzi dei corrispondenti gestori delle interruzioni nel vettore delle interruzioni. Durante l'I/O, i vari controllori di dispositivi generano i segnali d'interruzione della CPU quando sono pronti per un servizio.
Il meccanismo delle interruzioni si usa anche per gestire un'ampia gamma di eccezioni, come la divisione per zero, l'accesso a indirizzi di memoria protetti o inesistenti o il tentativo di eseguire un'istruzione privilegiata in modalità utente.
Un altro esempio è dato dall'esecuzione delle chiamate di sistema. Solitamente i programmi sfruttano routine di libreria per eseguire chiamate di sistema. La routine controlla i parametri passati dall'applicazione, li assembla in una struttura dati appropriata da passare al kernel, e infine esegue una particolare istruzione detta interruzione software o trap. Quando la chiamata di sistema esegue l'istruzione di eccezione, l'architettura delle interruzioni memorizza le informazioni riguardanti lo stato cui era giunta l'esecuzione del codice utente, passa al modo supervisore e recapita l'interruzione alla procedura del kernel che realizza il servizio richiesto. Le interruzioni si possono inoltre usare per gestire il controllo del flusso all'interno del kernel. Si consideri ad esempio l'elaborazione richiesta per completare una lettura da un disco. Un passo necessario è quello i copiare dati dalla regione di memoria usata dal kernel al buffer dell'utente. Questa azione richiede tempo, ma non è urgente e non dovrebbe bloccare la gestione delle interruzioni con priorità più alta. Un altro passo è quello di avviare l'evasione desse successive richieste di IO relative a quell'unita a disco. Questo passo ha priorità più alta: se le unta a disco si devono usare in modo efficiente, è necessario avviare l'evasione della successiva richiesta di IO non appena la precedente sia stata soddisfatta.
Un'architettura del kernel basata su thread è adatta alla realizzazione di più livelli di priorità delle interruzioni e a dare la precedenza alla gestione delle interruzioni rispetta alle elaborazioni in sottofondo delle procedure del kernel e delle applicazioni. Riassumendo i segnali d'interruzione sono usati diffusamente dai SO moderni per gestire eventi asincroni e per eseguire in modalità supervisore le procedure del kernel. Per far si che i compiti più urgenti siano portati a termine per primi, i calcolatori moderni usano un sistema di priorità delle interruzioni. I controllori dei dispositivi, gli errore i e le chiamate di sistema generano segnali d'interruzione al fine di innescare l'esecuzione di procedure del kernel.

Accesso diretto alla memoria DMA

Quando un dispositivo compie trasferimenti di grandi quantità di dati, come nel caso di un'unita a disco, l'uso di una costosa CPU per il controllo dei bit di stato e per la scrittura di dati nel registro del controllore byte alla volta, detto IO programmato, sembra essere uno spreco. In molti calcolatori

si evita di sovraccaricare la CPU assegnando una parte di questi compiti a un'unita di elaborazione specializzata, della controllore dell'accesso diretto alla memoria DMA. Per dar avvio a un trasferimento DMA la CPU scrive in memoria un comando strutturato per il DMA. La CPU scrive l'indirizzo di questo comando nel controllore del DMA, e prosegue con la sua esecuzione. Il controllore DMA agisce quindi direttamente sul bus della memoria, presentano al bus gli indirizzi di memoria necessarie per eseguire il trasferimenti senza l'aiuto della CPU. Un semplice controllore DMA è un componente o ordinario dei PC e le schede di IO dette bus mastermind di un PC includono di solito componenti DMA ad alta velocità.

La procedura di negoziazione tra il controllore del DMA e il controllore del dispositivo si svolge grazie a una coppia di fili detti DMA request e DMA acknowledge. Il controllore del dispositivo manda un segnale sulla linea DMA request quando una parola di dati è disponibile per il trasferimento. Questo segnale fa si che il controllore DMA prenda possesso del bus di memoria, presenti l'indirizzo desiderato ai fili d'intirizzimento della memoria e mandi un segnale lungo la linea DMA acknowledge. Quando il controllore del dispositivo riceve questo segnale, trasferisce in memoria la parola di dati e rimuove il segnale dalla linea DMA request. Quando l'intero trasferimento termina, il controllore del DMA interrompe la CPU. Quando il controllore del DMA prende possesso del bus di memoria, la CPU è temporaneamente impossibilitata ad accedere alla memoria centrale, sebbene abbia accesso ai dati contenuti nella sua cache primaria e secondaria. Questo fenomeno noto come sottrazione di cicli, può rallentare la computazione della CPU; ciononostante l'assegnamento del lavoro di trasferimento di dati a un controllore DMA migliora e in generale le prestazioni complessive del sistema. In alcune architetture per realizzare la tecnica DMA si usano gli indirizzi della memoria fisica, mentre in altre s'impiega l'accesso diretto alla memoria virtuale, in questo caso si usano indirizzi virtuali che poi si traducono in indirizzi fisici.



8. INTERFACCIA DI I/O PER LE APPLICAZIONI

Si spiega come un'applicazione possa aprire n file residente in un disco senza sapere di che tipo di disco si tratti e come si possano aggiungere al calcolatore nuove unita a disco e latri dispositivi senza che si debba modificare il sistema operativo.

I metodi qui esposti coinvolgono l'astrazione, l'incapsulamento e la stratificazione dei programmi. In particolare si puo compiere un procedimento di astrazione rispetto ai dettagli delle differenze tra i dispositivi per l'IO identificandone alcuni tipi generali. A ognuno di questi tipi si accede per mezzo di un unico insieme di funzioni- un'interfaccia. Le differenze sono incapsulate in moduli del kernel detti driver dei dispositivi.
Lo scopo dello strato dei driver dei dispositivi è di nascondere al sottosistema di IO del kernel le differenze tra i controllori dei dispositivi in mood simile a quello con cui le chiamate di sistema di IO incapsulando il comportamento dei dispostivi in alcun classi generiche che nascondo le differenze alle applicazioni.
Sfortunatamente per i produttori di dispositivi, ogni tipo di sistema operativo ha le sue convenienze riguardanti l'interfaccia dei driver dei dispositivi.
I dispositivi possono differire in molti aspetti:

- **Trasferimento a flusso di caratteri o a blocchi.** Un dispositivo del primo tipo trasferisce dati un byte alla volta mentre uno del secondo tipo ne trasferisce un blocco alla volta.
- **Accesso sequenziale o diretto.** Un dispositivo del primo tipo trasferisce dati sentendo ordine prestabilito, mentre l'utente di un dispositivo ad accesso diretto può richiedere l'accesso a una qualunque delle possibili locazioni di memorizzazione
- **Dispositivi sincroni o asincroni.** Un dispositivo sincrono trasferisce dati con un tempo di risposta prevedibile, mentre un dispositivo asincrono ha tempi di risposta irregolari
- **Condivisibili o riservati.** Un dispositivo condivisibile può essere usato in modo concorrente da diversi processi mentre ciò è impossibile se un dispositivo è riservato.
- **Velocità di trasferimento.** Può variare da alcuni byte a alcuni gb
- **Lettura e scritture, solo lettura o solo scrittura.**

Per ciò che riguarda l'accesso delle applicazioni ai dispositivi, molte di queste differenze sono nascoste dal sistema operativo e i dispostivi sono raggruppati in poche classi convenzionali.

Dispositivi con trasferimento a blocchi o a caratteri

L'interfaccia per i dispostivi a blocchi sintetizza tutti gli aspetti necessari per accedere alle unita a disco e ad altri dispositivi basati sul trasferimento di blocchi di dati. Il SO e certe applicazioni particolari come quelle per la gestione della basi di dati possono trovare più convenirne trattare questi dispositivi come una semplice sequenza lineare di blocchi. In questo caso si parla di IO a basso livello. La tastiera è un esempio di dispositivo al quale si accede tramite un'interfaccia a flusso di caratteri.

Dispositivi di rete

Poiché i modi di intirizzimento e le prestazioni tipiche dell'IO di rete sono notevolmente differenti da quelli dell'IO dell'unita a disco, la maggior parte dei SO fornisce un'interfaccia per l'IO di rete diversa da quelle delle normali operazioni detta socket. Una volta creata una socket si possono usare le normali operazioni di IO.

Orologi e timer

La maggior parte dei calcolatori ha timer e orologi per l'invio di segnali, segnalare l'ora corrente e in tempo trascorso. Il dispositivo che misura la durata di un lasso di tempo e che può avviare un'operazione si chiama timer programmabile.

IO bloccante e non bloccante

Un altro aspetto delle chiamate di sistema è la scelta fra IO bloccante e non bloccante. Quando un'applicazione impiega una chiamata di sistema bloccante si sospende l'esecuzione delle applicazioni, che passa dalla coda dei processi pronti per l'esecuzione alla coda d'attesa. Quando la chiamata di sistema termina l'applicazione è posta nuovamente nella coda dei processi pronti in modo che possa riprendere l'esecuzione solo allora essa ricevere i valori riportati dalla chiamata di sistema.
Alcuni processi a livello utente necessitano di una forma di IO non bloccante. Un esempio è quello di un'interfaccia utente con cui s'interagisce col mouse e la tastiera mentre elabora dati e li mostra sullo schermo.

La differenza tra chiamata non bloccante e asincrona è che una read non bloccante restituisce immediatamente il controllo fornendo i dati che è stato possibile leggere. Una chiamata read asincrona richiede un trasferimento di cui il sistema garantisce il completamento ma solo in un momento successivo e non prevedibile.

9. SOTTOSISTEMA PER L'I/O DEL KERNEL

Il kernel fornisce molti servizi riguardanti l'IO e scheduling dell'IO gestione del buffer, della cache.

Scheduling di IO

Fare lo scheduling di un insieme di richieste di IO significa stabilirne un ordine d'esecuzione efficace. Lo scheduling può migliorare le prestazioni complessive del sistema , distribuire equamente gli accessi dei processi ai dispositivi dei processi ai dispositivi e ridurre il tempo d'attesa media per il completamento di un'operazione di IO. I progettisti di SO realizzano scheduling mantenendo una coda di richieste per ogni dispositivo. Quando un'applicazione richiede l'esecuzione di una chiamata di sistema di IO bloccante, si aggiunge la richiesta alla coda relativa al dispositivo appropriato. Lo scheduler dell'IO riorganizza l'ordine della coda per migliorare l'efficienza totale del sistema e il tempo medio d'attesa cui sono sottoposte le applicazioni. Il sistema operativo può anche tentare di essere equo in modo che nessuna applicazione ricerca un sevizio carente o può dare fiorita a quello richieste la cui corretta esecuzione potrebbe essere inficiata in un ritardo del servizio. I kernel che mettono a disposizione di IO asincrono devono essere in grado di tener traccia di più richieste di IO contemporaneamente. A questo fine alcuni sistemi annetto una tabella dello stato dei dispostivi alla coda dei processi in attesa. Gli elementi della tabella indicano il dispositivo e lo stato.

Gestione del buffer

Un buffer è un'area di memoria che contiene dati durante il trasferimento fra due dispositivi o tra un'applicazione e un dispositivo. Si ricorre al buffer di 3 ragioni: la prima è la necessita di gestire la differenza di velocità tra produttore e consumatore. Un modem ad esempio che è nettamente più lento di qualsiasi altro dispositivo ha bisogno di una doppia bufferizzazione altrimenti si renderebbe troppo critico il problema della sincronizzazione con il disco. Un secondo uso del buffer riguarda la gestione dei dispositivi che trasferiscono dati in blocchi di dimensione diversa. Il terzo modo in cui si può impiegare un buffer è per la realizzazione della semantica delle copie nell'ambito dell'IO delle applicazioni. La semantica delle copie garantisce che la versione dei dati scritta nel disco sia conforme a quella contenuta nel buffer al momento della chiamata di sistema indipendentemente da ogni successiva modifica.

Cache

Una cache è una regione di memoria veloce che serve per mantener copie di certi dati: l'accesso a queste copie è più rapido dell'accesso agli originali. Ad esempio le istruzioni di un processo correntemente in esecuzione sono memorizzate in un disco copiate nella memoria fisica e copiate ulteriormente nella cache primaria e secondaria della CPU. La differenza tra buffer e cache consiste nel fatto che il primo può contenere dati di cui non esiste altra copia, la seconda mantiene su un mezzo più efficiente una copia d'informazioni gia memorizzate.

Code ad uso esclusivo dei dispositivi

Una coda di file da stampare è un buffer contenente dati per un dispositivo che no può accettare flussi di dati intervallati. Sebbene una stampante possa servire una sola richiesta alla volta, diverse applicazioni devono potere richieder simultaneamente la stampa di dati, senza che questi si mischino. Il SO risolve questo problema filtrando tutti i dati per la stampante: i dati da stampare provenienti da ogni singola applicazione si registrano in uno specifico file in un disco. Quando un'applicazione termina di emettere dati da stampare, si aggiungetele file nella coda di stampa, quest'ultima viene copiata sulla stampante un file per volta. In questo caso il SO fornisce un'interfaccia di controllo che permette agli utenti e agli amministratori del sistema di esaminare la coda ed eliminare elementi della coda prima che siano stampati, sospendere una stampa e cosi via.

Gestione degli errori

Un SO che usi la protezione della memoria può proteggersi da molti tipo di errori dovuti ai dispositivi o alle applicazione cosicché il blocco completo del sistema non è l'ordinaria conseguenza di piccoli difetti tecnici. I SO sono spesso capaci di compensare efficacemente le conseguenza negative dovute a errori generati da cause contingenti.
Il sistema unix usa una variabile intera detta errno per codificare tutti gli errori.

Protezione dell'IO

Gli errori sono strettamente connessi alla tematica della protezione. Un processo utente che cerchi di impartire istruzioni IO illegali può disturbare il funzionamento normale di un sistema sia che lo faccia internazionalmente sia accidentalmente. Onde evitare che gli utenti impartiscano istruzioni di IO illegali, si definiscono come privilegiate tute le istruzioni relative all'IO. Ne consegue che gli utenti no potranno impartire in via diretta alcuna istruzione, ma dovranno farlo attraverso il SO. U programma utente per eseguire IO invoca una chiamata di sistema per chiedere al So di svolger una data operazione del suo interesse. Inoltre il sistema di protezione della memoria deve tutelare dall'accesso degli utenti tutti gli indirizzi mappati in memoria e gli indirizzi delle porte di IO. Il kernel tuttavia non può semplicemente negare qualunque tentativo di accesso da parte degli utenti: quasi tutti i videogiochi e i programmi di grafica usano l'accesso diretto al controllore della grafica mappato in memoria.

Strutture dati del kernel

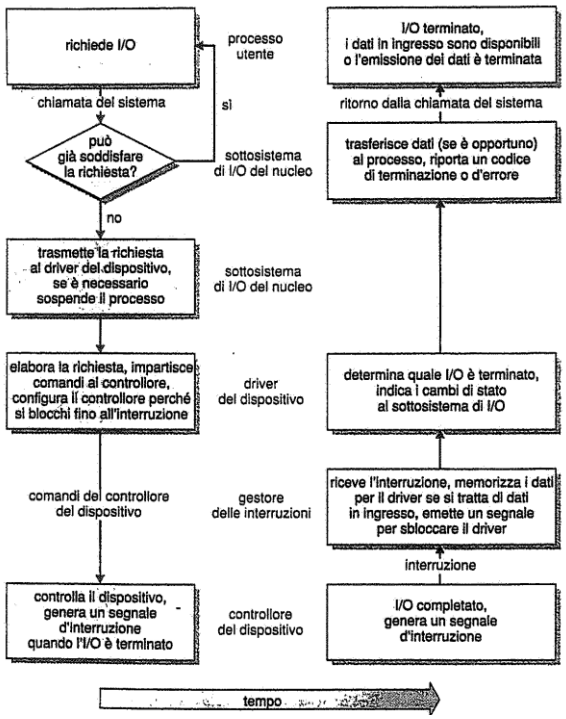
Il kernel ha bisogno di mantenere informazioni sullo stato dei componenti coinvolti nelle operazioni di IO e usa a questo fine diverse strutture dati interne. Il SO Unix, per mezzo del file system permette l'accesso a diversi oggetti: file degli utenti, dispostivi, spazio degli indirizzi. Quando il kernel ad esempio deve leggere un file utente ha bisogno di controllare la buffer cache prima di decidere l'effettiva esecuzione. Per leggere un disco tramite un'interfaccia a basso livello, il kernel deve accertarsi del fatto che la dimensione dell'insieme dei dati di cui è stato richiesto il trasferimento sia multiplo della dimensione dei settori del disco. Alcuni SO quali windows NT usano metodi orientati agli oggetti in misura più rilevante o sullo scambio di messaggi. Quando l'operazione è di emissione di dati il messaggio contiene i dati da scrivere quando è di emissione il messaggi contiene l'indirizzo del buffer che si usa per ricevere i dati.

10. TRASFORMAZIONE DELLE RICHIESTE DI IO IN OPERAZIONI DEI DISPOSITIVI

Si consideri ad esempio la lettura di un file da un'unita a disco. L'applicazione fa riferimento ai dati per mezzo del nome del file: è compito del file system fornire il modi di giungere attraverso la struttura delle directory alla regione del disco appropriata, cioè quella dove i dati del file sono fisicamente recenti. In UNIX il nome è associato a un elemento di una lista di oggetti detti numeri di inode. La prima parte del nome in DOS identifica il dispositivo e successivamente la collocazione. Unix utilizza una tabella di montaggio per associare i prefissi dei nomi di percorso ai corrispondenti a nomi di dispostivi. La seguente descrizione del tipico svolgimento di una richiesta di lettura bloccante indica che l'esecuzione di un'operazione di IO richiede una gran quanta di passi:

a) un processo impartisce una chiamata di sistema read bloccante relativa a un descrittore di file di un file precedentemente aperto.
b) il codice della chiamata di sistema all'interno del kernel controlla la correttezza dei parametri. Se sono presenti nel buffer cache, si passano i dati richiesi al processo chiamante e l'operazione è conclusa.
c) altrimenti è necessario eseguire un'operazione fisica di io cosi si rimuove il processo dalla coda dei processi pronti per l'esecuzione per inserirlo nella coda d'attesa relativa al dispositivo interessato. E si effettua lo scheduling della richiesta di OI. Infine il sottosistema per l'IO invia la richiesta al driver del dispositivo, secondo il sistema operativo ciò avviene tramite la chiamata di una proceduta o per mezzo dell'emissione di un messaggio interno al kernel.

d) il driver del dispositivo assegna un buffer nello spazio d'indirizzi del kernel che serve per ricevere i dati immessi ed esegue lo scheduling dell'IO. Infine il driver impartisce il comandi al controllore del dispositivo scrivendo nei suo registri.
e) il controllore aziona il dispositivo per compiere il trasferimento dei dati.
f) il driver può eseguire un'interrogazione ciclica o può avere predisposto un trasferimento DMA nella memoria del kernel.
g) tramite le interruzioni si attiva l'appropriato gestore delle interruzioni chiedono aver memorizzato i dati necessari, avverte con un segnale il driver del dispositivo e restituisce il controllo.
h) il driver riceve il segna , individua la richiesta di IO si accerta della riuscita o del fallimento dell'operazione e segnala al sottosistema per l'IO del kernel il completamento dell'operazione
i) il kernel trasferisce dati e/o codice di stato nello spazio degli indirizzi del processo chiamante
j) nel momento in cui è posto nella coda dei processi pronti per l'esecuzione il processo non è più bloccato: quando lo scheduler gli assegnerà la CPU esso riprenderà l'elaborazione. L'esecuzione della chiamata è terminata.



11. **PRESTAZIONI**

l'IO è uno tra i principali fattori che influiscono sulle prestazioni di un sistema: richiede un notevole impiego della CPU per l'esecuzione del codice dei driver e per uno scheduling equo ed efficiente dei processi quando essi sono bloccati o riavviati. Sebbene i calcolatori moderni siano capaci di gestire migliaia di interruzioni al secondo, la gestione delle interruzioni è un processo relativamente oneroso: ciascuna di loro fa si che il sistema cambi stato, esegua il gestore delle interruzioni e infine ripristini lo stato originario. Se il numero di cicli impiegato nell'attesa attiva non è eccessivo,l'IO programmato può essere più efficiente di quello basato sulle interruzioni. Per eliminare cambi di contesto implicati dal trasferimento di ciascun carattere dal demone del kernel, gli sviluppatori di solaris hanno implementato nuovamente il demone telnet tramite thread interni al kernel. Secondo stime della sun, queste migliorie hanno portato il massimo numero di connessioni contemporanee sostenibili da un grande server da qualche centinaio a qualche miglioro. Ad esempio i concentratori di terminali convogliano il traffico di informazioni proveniente da centinaio di terminali a un'unica porta di un grande calcolatore. I canali di IO sono unita di elaborazione specializzate presenti nei

mainframe e altri sistemi di altro profilo; il loro compito è sollevare la CPU di una parte del peso della gestione dell'IO. Per migliorare l'efficienza dell'IO si possono applicare diversi principi:

- Ridurre il numero dei cambi di contesto
- Ridurre il numero di copiature dei dati in memoria durante i trasferimenti fra dispositivi e applicazioni
- Ridurre la frequenza delle interruzioni
- Aumentare il tasso di concorrenza usando controllori DMA intelligenti o canali di IO per sollevare la CPU dalle semplice copiature di dati
- Realizzare le primitive direttamente tramite dispositivi fisici cosi da permettere che la loro esecuzione sia simultanea alle operazioni di bus e di CPU
- Equilibrare le prestazioni della CPU del sottosistema

Il driver inoltre esegue raffinati algoritmi di gestione degli errori e di recupero dati e più che hanno un'incidenza notevole sulle prestazioni complessive del sistema, svolge diverse funzioni di ottimizzazione delle presentazioni dell'unita a disco.