Sistemi operativi II

www.swappa.it & Gabriele Fioco

a.a. 2024-2025

Indice

1 Paginazione

Definizione La paginazione è una tecnica di gestione della memoria centrale indipendente dal programmatore che permette a un processo di utilizzare uno spazio degli indirizzi fisici non contiguo. Gli obbettivi della paginazione sono:

- caricare e scaricare piccole porzioni di memoria, aumentando la velocità di swapping;
- mantenere in memoria centrale solo le porzioni del processo che servono nel breve periodo, minimizzando l'occupazione di memoria;
- avere porzioni di memoria identica, efficientando la gestione.

Realizzazione e gestione La memoria fisica è suddivisa in blocchi di dimensione fissa detti frame, la memoria logica è suddivisa in blocchi di pari dimensione detti pagine. Quando un processo viene caricato, le sue pagine sono caricate in un qualsiasi frame. La dimensione dei frame dipende dall'hardware, tipicamente è una potenza di 2. Gli indirizzi logici generati dalla CPU sono formati da numero di pagina `p` e da un offset nella pagina `d`. `p` è usato come indice nella tabella delle pagine (una per processo), che contiene le corrispondenze da numero di pagina a frame. Sommando all'indirizzo del frame l'offset `d` si ottiene l'indirizzo fisico richiesto. Il SO deve decidere dove allocare le pagine, quindi mantenere una tabella dei frame che indica quali sono vuoti e quali no. La paginazione non ha frammentazione esterna, infatti i frame sono di dimensione fissa, vi è però frammentazione interna poiché non tutti i programmi sono grandi un multiplo della dimensione della pagina. Riducendo la dimensione dei frame si riduce la frammentazione interna, ma anche le prestazioni.

Supporto hardware La traduzione di indirizzi logici in fisici è svolta dall'hardware, in particolare dalla MMU (Memory Management Unit), configurata dal SO a ogni cambio di contesto. Se piccole, le tabelle delle pagine possono essere caricate in registri ad alta velocità, ma spesso sono molto grandi e quindi il SO carica solo l'indirizzo della tabella. Ciò significa che ogni accesso in memoria ne richiede due: uno per recuperare la traduzione, e uno per recuperare il dato stesso. Per ridurre il calo di prestazioni si usa un TLB (Translation Look-Aside Buffer), si tratta di una cache che mantiene le traduzioni recenti, evitando la maggior parte dei doppi accessi. È una memoria associativa, quindi piccola, veloce e costosa.

Ogni record mantiene un identificatore dello spazio di indirizzamento (ASID) così che possano esservi pià traduzioni per più processi.

Protezione Nella tabella delle pagine ogni riga mantiene dei bit di protezione, che indicano se la pagina è in sola lettura, sola esecuzione o lettura/scrittura. Viene anche mantenuto un bit di validità per indicare se la pagina appartiene allo spazio di indirizzamento logico del processo. I tentativi di accesso illegale generano una trap gestita dal SO. Alcuni sistemi memorizzano anche la lunghezza della tabella delle pagine di un processo per verificare che l'indirizzo logico sia in un range valido.

Condivisione di pagine La paginazione permette la condivisione di codice comune. Per evitare di duplicare le pagine, queste possono essere condivise tra processi, mappando lo stesso frame nella tabella delle pagine dei processi coinvolti. Si usa anche per la comunicazione con memoria condivisa. Il codice non dev'essere automodificante.

Struttura della tabella delle pagine I sistemi recenti hanno uno spazio di indirizzamento molto grande, quindi tabelle delle pagine molto grandi, che possono essere salvate efficientemente usando diverse tecniche:

- Paginazione gerarchica: la tabella delle pagine è a sua volta paginata più volte. Con 2 livelli si ha un indirizzo logico formato da `p1` (indice tabella esterna delle pagine), `p2` (offset nella tabella puntata da `p1`) e `d` (offset). Aumenta il numero di accessi necessari per la traduzione, riducendo le performance.
- Tabella delle pagine con hashing: si applica una funzione hash al numero di pagina logica richiesta, e si cerca il risultato in una tabella di hashing. L'entry puntata è una lista concatenata di elementi formati da traduzione pagina-frame e puntatore al successivo elemento della lista. Va scandita fino a trovare la traduzione cercata. Gli elementi di queste liste possono anche essere gruppi di traduzioni, invece che una sola, migliorando le prestazioni.
- Tabella delle pagine invertita: si tiene solo una tabella globale che tiene per ogni frame l'ASID cui appartiene, il numero di pagina logica in quello spazio e i bit di protezione. Questa soluzione permette di risparmiare memoria, ma complica la condivisione, e può servire una scansione di tutta la lista per trovare una traduzione (mitigabile con una tabella invertita con hashing, che però diminuisce le prestazioni).

Processi parzialmente in memoria Non sempre è necessario tenere l'intero processo in memoria centrale per l'esecuzione. All'avvio si possono caricare solo alcune pagine e iniziare l'esecuzione. Quando si tenterà di accedere a una pagina non caricata, si genererà un page fault, poiché la pagina di richiesta ha il bit di invalidità settato. Il SO, dopo aver verificato che non sia realmente un accesso illegale, recupererà la pagina richiesta dal disco, e riprenderà l'esecuzione del processo. La pagina caricata è messa in un frame libero oppure sostituita a una pagina del processo, o di un altro processo, che sarà scaricata in area di swap, e potrà essere ricaricata in seguito. Essendo le pagine di dimensione fissa, quest'area è semplice da gestire. Permette di tenere in memoria centrale solo le pagine necessarie nell'immediato futuro.

2 Segmentazione

Definizione La segmentazione è una tecnica di gestione della memoria centrale che permette a un processo di:

- utilizzare uno spazio degli indirizzi fisici non contiguo;
- tipizzare le porzioni di indirizzamento logico;
- supportare la divisione della memoria dal punto di vista degli utenti;
- non tenere l'intero processo in memoria, ma solo i segmenti usati nell'immediato futuro, e scambiarli tra memoria centrale e area di swap quando necessario. Separa quindi lo spazio di indirizzamento logico, potenzialmente grande quanto lo spazio di indirizzamento della macchina, da quello fisico, grande quanto la memoria fisica.

Metodi di base La memoria fisica è divisa in segmenti fisici (frame) di dimensione variabile, ognuno identificato da un nome, solitamente un numero. La memoria logica di un processo è divisa in segmenti logici (segmenti), caricati in frame di ugual dimensione. I segmenti necessari nell'immediato futuro

vengono caricati in un frame, gli altri nell'area di swap. Il compilatore costruisce i segmenti a seconda del programma, le librerie dinamiche sono assegnate ad altri segmenti, il loader associa un numero ai segmenti.

Realizzazione Il sistema operativo gestisce caricamento e scaricamento dei segmenti tra memoria e area di swap. La tabella dei segmenti contiene, per ogni segmento, base e limite. Gli indirizzi logici sono costituiti dal numero di segmento e dall'offset all'interno del segmento. Il numero è usato come indice nella tabella dei segmenti, l'offset viene sommato alla base, ottenuta dalla tabella, risultando così nell'indirizzo fisico. La Memory Management Unit (MMU) fornisce il supporto hardware, traducendo l'indirizzo logico in fisico. La MMU contiene la tabella dei segmenti o il suo riferimento, in quest'ultimo caso servono due accessi a memoria per recuperare il dato richiesto: per evitare il calo di prestazioni si usa una cache che mantiene le traduzioni recenti, detta Transation Lookaside Buffer (TLB). Ogni record del TLB mantiene anche un identificatore dello spazio di indirizzamento (ASID), così che possa memorizzare traduzioni di più processi.

Protezione e condivisione Un processo può accedere solo ai suoi segmenti dunque la protezione dagli accessi di altri processi è implicita. Ogni segmento contiene una porzione diversa del programma, i suoi elementi sono usati allo stesso modo, di conseguenza si associano dei bit di protezione a ogni segmento che lo definiscono in sola lettura, lettura/scrittura e sola esecuzione. La condivisione di codice o dati è semplice e avviene mappando lo stesso frame nelle tabelle dei segmenti di più processi. La condivisione è utile per non avere più copie del codice dello stesso programma e per condividere librerie tra processi. Per condividre codice vi sono due vincoli: il codice non deve essere automodificante; il segmento condiviso deve avere lo stesso numero di logico nei vari processi. Non ci sono vincoli per la condivisione di dat, che può essere usata anche per comunicazione con memoria condivisa.

Frammentazione Quando si inserisce un nuovo segmento in memoria questo va inserito in un frame abbastanza grande per contenerlo. Dopo un po' di tempo è probabile che ci siano frame piccoli sparsi per la memoria, che congiunti potrebbero ospitare qualche segmento. Questo problema si definisce frammentazione esterna. Una soluzione è compattare periodicamente la memoria creando frame più grandi e riallocare quelli già caricati.

Segmentazione con paginazione Paginazione e segmentazione hanno entrambi vantaggi e svantaggi. È possibile combinare i vantaggi di entrambi. Si divide la memoria fisica in frame di dimensione fissa, evitando frammentazione esterna. Si divide la memoria logica in segmenti tipizzati di dimensioni diverse; a loro volta i segmenti sono divisi in pagine, ognuna di stessa dimensione, pari a quella dei frame. Una pagina di un segmento è caricata in un frame. Gli indirizzi logici sono formati da numero di segmento, pagina nel segmento, offset nella pagina; quelli fisici da numero di frame e offset nel frame. La traduzione è svolta dall'MMU, gestita dal SO.

3 Memoria virtuale

Definizione La memoria virtuale è una tecnica che permette: l'esecuzione di processi non completamente in memoria; l'astrazione della memoria in un unico grande array uniforme, separando la visione logica della memoria degli utenti dalla memoria fisica. Di conseguenza, i programmi possono essere più grandi della memoria fisica. Permette inoltre la condivisione di file e codice e la crezione efficiente di processi.

Richiesta di paginazione Una tecnica molto usata è la richiesta di paginazione. Consiste nel caricare in memoria solamente le pagine richieste durante l'esecuzione. A differenza della paginazione, il bit di validità nella tabella delle pagine non indica soltanto se la pagina è legale, ma anche se è caricata in memoria. Se un processo tenta di accedere a una pagina non caricata genera una page fault, quindi il SO interrompe l'esecuzione, trasferisce la pagina dallo spazio di swap alla memoria e riprende dall'istruzione che ha causato la page fault. Nel caso di richiesta di paginazione pura allora il processo ha inizialmente zero pagine in memoria, quindi potrebbe generare page fault potenzialmente ad ogni istruzione. Sappiamo però che i programmi tengono a fare riferimenti locali, quindi ad avere poche page fault.

Copy-on-write La tecnica di copy-on-write permette a un processo figlio di condividere, inizialmente, le pagine col padre. Quando uno dei due processi modifica una pagina, questa viene copiata. Utile perchè dopo una chiamata fork() spesso però il nuovo processo chiama subito exec(), che lo sostituisce con un

altro programma. Alcuni SO forniscono una chiamata alternativa vfork(), analoga alla fork() ma senza copy-on-write. funziona analogamente, ma senza copy on write.

Sostituzione della pagina Se un processo richiede una nuova pagina ma non vi sono frame liberi, se ne cerca uno non usato e lo si sostituisce. Si osserva allora che, in caso di sostituzione, sono necessari due trasferimenti. Si può risolvere questo overhead con un bit, detto bit di modifica, associato ad ogni pagina e settato dall'hardware ogni qualvolta che la pagina è stata modificata. Se il bit di modifica non è settato la pagina scelta come vittima viene sovrascritta senza ricaricarla in area di swap. Vi sono varie politiche per selezionare una vittima, idealmente si vuole quello con la più bassa frequenza di page fault. La scelta della vittima può essere solo tra i frame del processo oppure globale.

- Sostituzione ottima: è la politica ottima, sostituisce la pagina che non verrà usata per più tempo. Un algoritmo però non esiste.
- Sostituzione FIFO: seleziona la pagina caricata meno recentemente. Ha prestazioni molto scarse perchè potrebbe togliere dalla memoria pagine costantemente utilizzate. Inoltre, si osserva che allocare più frame a un processo ne aumenta i page fault (Anomalia di Belady).
- Sostituzione LRU: sostituisce la pagina non usata da più tempo. Approsimazzione della sostituzione ottima. Richiede molto supporto hardware, vi sono diverse implementazioni: inserire un timestamp (TS) per ogni entry della tabella delle pagine, sostituendo quella con TS più basso; usare uno stack che ad ogni accesso pone in cima la pagina acceduta, sostituendo quella alla base.
- Sostituzione LRU approssimata: per ridurre il supporto HW, esistono varie approssimazioni dell'LRU. La maggior parte dell'HW fornisce un bit di riferimento per ogni pagina, settato a 1 quando la pagina viene referenziata. Il SO azzerra periodicamente i bit di riferimento, ed è quindi in grado di sapere quali pagine sono state usate. La pagina da sostituire è quella con il bit a 0. Si può estendere la quantità di bit per pagina tenendo la storia più recente: periodicamente il SO shifta verso il bit meno significativo. Interpretando i bit di ogni pagina come un intero senza segno, la pagina con il valore più basso è quella da sostituire. L'algoritmo della seconda possibilità scandisce le pagine in ordine FIFO, se quella selezionata ha bit di riferimento 0 è la vittima, altrimenti lo pone a 0 e passa a quella successiva. Successive scansioni partono dall'ultimo elemento scandito. Una versione migliorata controlla sia il bit di riferimento che il bit di di modifica: la pagina migliore da sostituire è quella che li ha entrambi a 0, poiché riduce le chiamate I/O.
- Sostituzioni basate sul conteggio: ogni pagina ha un contatore di riferimenti. Si può: sostituire la pagina usata meno frequentemente, basandosi sull'idea che pagine molto usate sono molto referenziate. Il contatore deve periodicamente diminuire per evitare che una pagina precedentemente molto usata e mai riusata non venga scaricata; sostituire quella usata più frequentemente, basandosi sull'idea che pagine poco usate sono appena state messe in memoria e quindi devono ancora essere usate.

Quando una pagina è richiesta si può prevedere che qualche pagina successiva verrà richiesta.

Allocazione dei frame Maggiore è il numero di frame, minore è la quantità di page fault. Per spartire m frame su n processi vi sono due metodi:

- Allocazione omogenea: ogni processo ottiene m/n frame. È la più semplice. Alcuni processi potrebbero usare poca memoria sprecando il resto.
- Allocazione proporzionale: i frame sono divisi in modo direttamente proporzionale alla dimensione o alla priorità.

Si osserva che con sostituzione globale delle pagine il numero di frame allocato a un processo può cambiare.

Trashing Se un processo ha meno frame del numero di pagine attive, allora passa molto tempo a sostituire. Questo fenomeno è detto trashing. È causato dalla multiprogrammazione e dell'algoritmo di schedulazione a lungo termine che, in caso di basso utilizzo della CPU, aumenta a sua volta la multiprogrammazione. La sostituzione globale propaga il trashing tra i processi; la sostituzione locale può limitarlo. Si può prevenire col modello di località: si alloca a un processo la quantità di frame che sta effettivamente usando. Il modello working-set (WS) approssima la località di un processo come le pagine a cui ha acceduto negli ultimi N riferimenti. Il SO monitora il WS di ogni processo e alloca abbastanza frame per contenerlo. Se ci sono abbastanza frame, si può inizializzare un altro processo. Se non vi sono abbastanza frame, si sospende un processo. Un'altra approsimazione è la frequenza

dei page fault: si controlla il tasso di page fault di un processo e, se sale sopra una certa soglia, gli si assegna un nuovo frame; se scende sotto una certa soglia, ne perde uno. Se non ci sono frame liberi, allora il processo va sospeso.

Prestazioni Tecniche per migliorare le prestazioni:

- Scegliere un'adeguata dimensione delle pagine.
- Usare la tabella invertita delle pagine.
- Prepeginazione: caricare le pagine in memoria centrale in anticipo.
- Aumentare la dimensione del TLB.
- Aumentare la località del programma.
- Dare la possibilità ai processi real time di lasciare delle pagine residenti in memoria.
- Lasciare le pagine per I/O in memoria o nello spazio del SO o nello spazio del processo come residenti.

4 Gestione delle periferiche

Software di gestione delle periferiche Il sistema operativo astrae i dettagli di ogni dispositivo raggruppandoli in alcune tipologie. Le differenze sono encapsulate in moduli kernel chiamati driver: internamente sono sviluppati per il dispositivo specifico e rendono trasparente le differenze tra dispositivi dello stesso tipo, esternamente forniscono un'interfaccia uniforme e rendono trasparante le differenze tra i diversi tipi di dispositivo. Vi sono quindi tre strati: l'hardware, i driver, e il sottosistema di gestione I/O.

Caratteristiche delle periferiche Le periferiche si distinguono per alcune caratteristiche:

- Metodo di trasferimento dei dati: a carattere o a blocchi.
- Metodo di accesso: sequenziale in ordine fissato o diretto in qualsiasi ordine.
- Schedulazione del trasferimento: sincrono (bloccante) con altri aspetti del sistema o asincrono (non bloccante).
- Condivisione: dedicato o condivisibile.
- Velocità del dispositivo: data da latenza, tempo di ricerca, tempo di trasferimento e ritardo tra le operazioni.
- Direzione I/O: solo lettura, solo scrittura o lettura e scrittura.

Alcuni sistemi operativi hanno una back door per passare comandi arbitrari a un dispositivo.

Interfaccia dei dispositivi :

- Dispositivi a blocchi: l'interfaccia include comandi i comandi read, write e, se ad accesso diretto, seek. Sebbene le applicazioni generalmente interagiscano con questi dispositivi tramite il file system, è possibile accedere direttamente ai dispositivi tramite chiamate di I/O. L'accesso tramite file memory-mapped è un'ulteriore astrazione che sfrutta la memoria virtuale, e si stratifica sopra i driver di dispositivi a blocchi per migliorare l'efficienza dell'accesso ai dati.
- Dispositivi a caratteri: l'interfaccia di questi consente di trasferire singoli byte. Questi device devono comprendere i comandi get e put. È uno stile utile per device che producono dati in momenti inaspettati, come tastiere e modem, e va bene anche per stampanti, schede audio, ... Al di sopra possono essere implementati vari servizi, ad esempio buffering per leggere intere linee. -
- Periferiche di rete: l'interfaccia è rappresentata dai socket, che consentono a un'applicazione di stabilire una connessione con un host in ascolto o di attendere connessioni in ingresso. Successivamente, è possibile comunicare attraverso operazioni simili a quelle di lettura e scrittura su un flusso full duplex.
- Orologi e timer: vi sono chiamate che consentono di ottenere l'ora, l'uptime del sistema o di programmare l'esecuzione di operazioni in un momento specifico, come ad esempio gli interrupt. L'hardware coinvolto è un temporizzatore programmabile. Poiché sia il sistema operativo che più processi possono utilizzarlo, il sistema operativo mantiene una lista di tutti gli eventi e programma il timer in modo continuo seguendo l'ordine cronologico.

I/O bloccante e non bloccante, sicrono e asincrono Le chiamate bloccanti sospendono l'esecuzione del processo fino al completamento dell'operazione; le chiamate non bloccanti non interrompono

l'esecuzione del processo, ma restituiscono i dati attualmente disponibili. Le chiamate asincrone eseguono l'operazione di I/O parallelamente all'esecuzione del processo, a cui poi viene notificato il completamento.

Servizi del sottosistema I/O: Il sottosistema I/O del kernel fornisce diversi servizi:

- Schedulazione delle richieste: il sistema operativo mantiene e gestisce la coda delle richieste, una per ogni periferica. L'obbiettivo è ottimizzare l'uso globale delle periferiche e ridurre i tempi d'attesa. Solitamente il sistema operativo mantiene una tabella sullo stato d'uso per ogni dispositivo.
- Buffering: un buffer è una regione di memoria che contiene i dati mentre vengono trasferiti tra due dispositivi o tra un dispositivo e un'applicazione. I buffer servono a: adattere la differenza di velocità tra 2 dispositivi; adattare periferiche con blocchi di dimensione diversa; supportare la semantica della copia, ovvero evitare che un processo alteri i dati che deve mandare a un device mentre procede con la sua richiesta in coda: si copiano i dati in un buffer, e questo sarà scritto sul device.
- Caching: una cache è una memoria veloce che conserva la copia dei dati letti da una periferica per il riuso veloce. L'obbiettivo è evitare accessi lunghi, rileggendo dati non cambiati dall'ultima lettura. La differenza col buffer è che il buffer potrebbe contenere l'unica copia esistente del dato, mentre la cache mantiene una copia in una regione di memoria più veloce.
- Locking: alcuni dispositivi vanno usati in mutua esclusione, quindi il sistema operativo deve fornire meccanismi di blocco per garantire l'uso in mutua esclusione.
- Spooling e prenotazione dei device: alcuni device non possono essere usati da più processi contemporaneamente. Lo spool è un buffer che conserva i dati per i device che non possono essere usati da più processi contemporanemanete e li invia in ordine. Alcuni SO permettono ai processi di prenotare l'accesso esclusivo a un device, vi è possibilità di deadlock.
- Gestione errori: possono verificarsi guasti (meccanici, elettronici, disturbi, etc...). Il SO fa fronte agli errori temporanei, ma in caso di guasti permanenti deve informare il processo che la richiesta è fallita, solitamente ritornando un codice d'errore dalla chiamata a sistema.

Strutture dati kernel Il kernel deve tenere informazioni sullo stato d'uso dei dispositivi di I/O. Vi sono diverse strutture dati, come la tabella dei file aperti.

Realizzazione di una richiesta I/O La trasformazione delle richieste di I/O in operazioni HW segue una serie di passi. Quando il processo effettua una chiamata I/O il SO controlla se è corretta e se è consentita, quindi controlla se può essere soddisfatta immediatamente. Se è così esegue la richiesta e restituisce il controllo al processo, con gli eventuali dati richiesti; altrimenti manda la richiesta al driver del dispositivo e blocca il processo. Il driver elabora la richiesta, se necessario alloca un buffer in memoria e manda comandi al controller. Il controller esegue l'operazione e, quando completata, avverte il driver con un interrupt. Se necessario, i dati sono caricati nel buffer del driver. Il driver segnala al sottosistema di I/O che la richiesta è completata, quindi eventuali dati sono trasferiti al processo e se necessario lo si rimette in stato di pronto.

Prestazioni L'I/O è un fattore critico per le prestazioni. Per migliorarle bisogna:

- ridurre i cambi di contesto;
- ridurre il numero di copie dei dati;
- ridurre la frequenza degli interrupt usando grandi trasferimenti, controller intelligenti e polling;
- aumentare la concorrenza con controller DMA intelligenti;
- spostare le primitive di gestione nell'HW così da eseguire le operazioni concorrentemente ai controller e ai bus;
- bilanciare le performance di CPU, memoria, bus e dispositivi, un sovrccarico in uno di questi rende inattivi gli altri.

5 File System

Definizione Il SO astrae dalle proprietà fisiche dei dispositivi di memorizzazione per definire un'unità logica di memorizzazione detta file. Un file è un insieme di informazioni, è memorizzato in memoria secondaria ed è identificato univocamente da un nome. È visto come un insieme di bit la cui interpretazione spetta a un utente/programma. Il file system (FS) è l'aspetto più visibile del SO, fornisce il supporto per la

memorizzazione e l'accesso a file e programmi. La realizzazione del file system pone due sfide: definire l'interfaccia utente; implementare gli algoritmi e le strutture dati per mappare i file logici nei dispositivi fisici di memorizzazione.

Struttura Il FS è composto da diversi livelli:

- Gestione I/O: composto dai driver dei dispositivi e dai gestori degli interrupt per trasferire tra memoria e dischi.
- File system di base: da i comandi ai driver dei dispositivi per leggere e scrivere i blocchi.
- Modulo di organizzazione dei file: conosce i file e i loro blocchi logici. Si occupa dello spazio libero.
- File system logico: si occupa dei metadati, che definiscono la struttura dei file. Si occupa anche della struttura dei direttori e della protezione.

Strutture dati Servono diverse strutture dati. Su disco vi sono:

- blocco di controllo del boot per ogni partizione: contiene le informazioni necessarie per avviare il SO nella data partizione;
- blocco di controllo della partizione: contiene le informazioni sulla partizione come il numero di blocchi, la dimensione dei bloccchi, il numero di blocchi liberi, etc..;
- struttura per i direttori per organizzare i file;
- i file control block (FCB) che contengono i dettagli sui file come i permessi, la dimensione, i blocchi di dati, le date e il proprietario.

In memoria vi sono:

- tabella delle partizioni montate;
- una cache sulle directory aperte recentemente;
- la tabella dei file aperti del sistema;
- la tabella dei file aperti per processo (contiene i puntatori alla precedente);
- i buffer dei FCB mentre sono letti o scritti.

Operazioni sui file Per creare un file un processo chiamata il file system logico, questi alloca un nuovo FCB quindi aggiorna la directory. Quando un processo vuole usare un file lo apre, allora il sistema controlla se è già aperto da altri processi e se così fosse crea un puntatore alla tabella dei file aperti del sistema; altrimenti cerca il file nella struttura delle directory. Quando il file è stato trovato, il suo FCB è copiato nella tabella dei file aperti del sistema, quindi viene aggiunto un puntatore al record di questa tabella nella tabella del processo. La chiamata open() ritorna un puntatore al record nella tabella dei file aperti. Tutte le operazioni avvengono su questo puntatore. Quando un file viene chiuso il record nella tabella dei processi è rimosso e il contatore nella tabella dei file aperti del sistema è decrementato.

Realizzazione delle directory Le directory sono dei file speciali: una tabella con nome file e puntatore al FD (o FD stesso a seconda dell'implementazione). Possono essere realizzate con una lista concatenata di nomi di file con puntatore al FCB. È semplice, ma l'accesso richiede una scansione lineare. Si migliora ordinandola o con una cache. In alternativa si può usare una tabella di hash: si ricava la posizione nella tabella con una funzione hash sul nome, velocizzando la ricerca. Il problema maggiore è che la dimensione è fissa e dipende dalla funzione di hash. Se ci sono collisioni si può usare una lista di collisione.

Allocazione dei file Bisogna allocare i file in modo tale che lo spazio sia usato efficientemente e che l'accesso sia veloce.

- Allocazione contigua: ogni file occupa blocchi contigui. Per ogni file si memorizzano l'indirizzo di inizio e la lunghezza. Gli accessi sequenziali e diretti sono rapidi. Si crea frammentazione esterna. Se i file aumentano di dimensione può essere necessario spostarli. Una soluzione è allocare una certa porizione di blocchi. Quando questi non bastano, si alloca un'altra porzione, detta estensione. Vi è frammentazione interna ed esterna.
- Allocazione collegata: ogni file è una lista concatenata di blocchi. Si memorizza il puntatore al blocco iniziale e finale. I blocchi possono trovarsi ovunque sul disco. Gli accessi sequenziali sono rapidi, quelli diritti sono lenti. I puntatori occupano molto spazio. Un puntatore corrotto causa la perdita del file. Si possono raggruppare più blocchi in cluster riducendo lo spazio per i puntatori, ma causando

frammentazione interna. Un'alternativa è la FAT (File Allocation Table): per ogni file si memorizza il blocco iniziale, e in una locazione del disco si mantiene una tabella che per ogni blocco del disco contiene o il blocco successivo del file o un carattere fine file o 0 se è vuoto. La tabella va tenuta in memoria per avere buone prestazioni. Il vantaggio è che l'accesso diretto è più veloce.

- Allocazione indicizzata: per ogni file si mantiene un array, detto blocco indice, di puntatori a un blocco del file. L'i-esimo elemento dell'array contiene l'i-esimo blocco del file. L'accesso diretto è veloce e non vi è frammentazione. La dimensione del blocco indice è critica: se è troppo piccolo non permette file grandi. Vi sono diverse soluzioni: la lista concatenata di blocchi indice; un indice multilivello, ogni puntatore punta a un altro blocco indice, fino ad arrivare ai blocchi; soluzione mista. La scelta dell'algoritmo va fatta in base al tipo di accessi, la loro frequenza, ... Si possono anche combinare.

Gestione dello spazio libero Il SO mantiene una lista degli spazi liberi. Quando si crea un file, si cerca spazio nella lista e lo si alloca; quando si cancella, si aggiunge il suo spazio alla lista. La lista si può implementarein vari modi.

- Bitmap: si memorizza un bit per blocco: se è 1 è libero, se è 0 è occupato. Rende semplice trovare un blocco libero o blocchi liberi consecutivi. Per avere buone prestazioni la bitmap va tenuta in memoria centrale e periodicamente aggiornata su disco, quindi va bene solo per sistemi con poca memoria.
- Lista collegata: ogni blocco libero contiene un puntatore al successivo, e si memorizza il puntatore al primo. È semplice trovare un solo blocco libero, ma non blocchi liberi consecutivi in quanto richiedere molte operazioni I/O.
- Raggruppamento: alternativa alla lista collegata. Il primo blocco libero memorizza gli indirizzi di n blocchi liberi: i primi $n-1$ sono liberi, l'ultimo contiene altri n indirizzi, e così via. Permette di trovare blocchi liberi consecutivi rapidamente.
- Conteggio: spesso i blocchi liberi sono contigui. Quindi si può tenere l'indirizzo del primo blocco libero e i prossimi n blocchi liberi. La lista dei blocchi diventa breve.

Prestazioni Le prestazioni si migliorano con un buffer cache che tiene i blocchi assumendo siano presto riusati. Questa cache può essere gestita con la memoria virtuale (page cache) per salvare i dati come pagine piuttosto che come blocchi del FS. Si può evitare la doppia copia di dati

Si può anche evitare la doppia copia di dati (cache e memoria virtuale processi) in caso di file mappati in memoria mappando pagine della page cache nella memoria virtuale dei processi (buffer cache unificata). La tecnica di sostituzione migliore è LRU. Per letture sequenziali, possono essere utili read-ahead (mette dei blocchi successivi in cache per risparimiare accessi) e free-behind (cancella una pagina dalla cache quando è richiesta la successiva). È utile anche schedulare gli accessi al disco. Alcuni SO permettono di creare dischi virtuali in memoria centrale, gestiti con un FS, piccoli e performanti.

Manutenzione Se un'operazione comporta dei cambiamenti nella struttura dati del SO, ma un crash interrompe i cambiamenti o questi restano in cache, allora il FS si trova in stato non coerente. Si può far si che le scritture siano sincrone, ma diminuiscono le prestazioni. Un'altra alternativa è eseguire un controllore della consistenza, che rileva e ripara le inconsistenze nei metadati. Infine, alcuni FS tengono un log delle modifiche dei metadati in un file. Un insieme di operazioni che esegue un compito è detto transazione. Tutte le modifiche vengono salvate nel log e si considerano completate, poi vengono effettivamente concluse e cancellate dal log. Se si verifica un'interruzione, all'avvio del sistema si controlla se vi sono operazioni da completare nel log e si concludono le operazioni in corso. Se è fallita la transazione allora bisogna annullare i cambiamenti applicati al file system.

6 Protezione

Definizione e obbiettivi La protezione consiste nel proteggere le risorse fisiche e informative da accessi non autorizzati. Bisogna definire:

- regole: specificano chi e come può utilizzare una risorsa;
- meccanismi: strumenti che applicano le regole.

Dominio di protezione Ogni oggetto è caratterizzata da un identificativo e da un insieme di operazioni. Secondo il principio di minima conoscenza i processi devono accedere solo alle risorse minime necessarie alla loro computazione. A questo scopo, ogni processo opera in un dominio di protezione, che è definito dall'insieme degli oggetti e dei diritti di accesso (operazioni lecite) su tali oggetti. I domini di protezione non sono necessariamente disgiunti. L'associazione di un dominio a un processo può essere: statica, il processo è fisso in un dominio; dinamica, può, se ne ha il diritto, cambiare dominio. L'associazione dinamica è più difficile da implementare, ma permette di applicare meglio il principio di minima conoscenza. Il cambio di dominio non cambia la struttura del dominio né i permessi, per modificarli bisogna averne l'autorità. Tipicamente un processo è assegnato a un dominio dal SO, i cui permessi sono decisi dagli utenti/admin. Un dominio può essere realizzato in vari modi: ogni utente è un dominio, si cambia dominio quando cambia l'utente; un processo è un dominio, il dominio cambia ad ogni cambio di contesto; una procedura è un dominio, il dominio cambia quando viene chiamata una nuova procedura.

Matrice d'accesso Il modello di protezione si astrae come una matrice, detta matrice d'accesso. La matrice supporta la protezione dinamica. Le righe sono i domini di protezione e le colonne le risorse. La cella (i, j) definisce i diritti d'accesso che il processo nel dominio D_i ha sulla risorsa O_j. Anche i domini sono considerati oggetti a cui è possibile dare il diritto di switch. Permettere cambiamenti controllati nella matrice necessita di altri tre meccanismi:

- Copia: copia l'autorizzazione che un dominio ha su un oggetto a un altro dominio, con o senza diritto di propagarlo.
- Proprietà: chi ha il diritto di proprietà su un oggetto può cambiare i diritti che altri domini hanno su quell'oggetto.
- Controllo: il diritto di controllo è applicabile solo agli oggetti di tipo dominio. Consente a chi ha il diritto di controllo su un dominio di cancellare i diritti d'accesso.

I diritti di copia e proprietà permettono di evitare il propagarsi dei diritti di accesso, ma non garantiscono il contenimento delle informazioni.

Alcuni SO permettono agli utenti di definire diritti aggiuntivi. La protezione può essere anche basata sul linguaggio di programmazione. Permette un controllo più granulare, ha meno sicurezza ma più flessibilità ed efficienza.

Implementazione della matrice d'accesso La matrice è grande e sparsa, quindi va memorizzata efficientemente. Vi sono diverse tecniche:

- Tabella globale: è l'implementazione può semplice. Memorizza una lista di terne <dominio, oggetto, diritti>. Un processo in un dominio D_i è autorizzato a eseguire un'operazione M su un oggetto O_i se e solo se esiste una terna $<D_i, O_i, R_i>$ con $M \in R_i$. Occupa molto spazio e non permette di raggruppare oggetti o domini.
- Liste di controllo degli accessi (Access Control List, ACL): per ogni oggetto si memorizza una lista di coppie <dominio, diritti>. Risponde ai bisogni degli utenti e memorizza informazioni globali ma è inefficiente su grandi sistemi poiché le ACL sono scandite spesso.
- Liste di capacità dei domini (Capability List, CL): per ogni dominio si memorizza una lista di coppie <oggetto, diritti>. Un oggetto è rappresentato dal suo nome fisico o dall'indirizzo, detto capacità. I domini non sono direttamente accessibili ai processi ma solo dal SO. Rende semplice l'accesso a informazioni sui processi e memorizza informazioni locali ma la revoca è poco efficiente.
- Meccanismo serratura-chiave: compromesso tra ACL e CL. Ogni oggetto ha una lista di stringhe di bit dette serrature e ogni dominio ha una lista di stringhe di bit dette chiavi. Un processo in un dominio può fare un'operazione su un oggetto se e solo se una sua chiave corrisponde con la serratura dell'oggetto per l'operazione indicata.

Revoca dei diritti su oggetti In un sistema di protezione dinamico, può essere necessario revocare dei permessi. La revoca può essere:

- immediata o ritardata;
- selettiva su un numero limitato di utenti o generale su tutti;
- parziale su un numero limitato di diritti o totale su tutti;

- temporanea o permanente.

Con le ACL la revoca è semplice, ma con le CL è più complessa poichè le informazioni sono sparse per il sistema. Quindi vi sono varie soluzioni:

- Riacquisizione: le capacità sono periodicamente cancellate dal dominio. Se un processo tenta di usare una capacità cancellata, tenterà di riacquisirla. Se è stata revocata gli sarà negata.
- Puntatori alle capacità: per ogni oggetto si tiene una lista di puntatori alle capacità ad esso associate. Per cancellare una capacità basta eliminare il puntatore. Comune ma costoso.
- Indirezione: le capacità non puntano direttamente agli oggetti ma a un record di una tabella globale, che a sua volta punta a un oggetto. La revoca cancella il puntatore nella tabella. Non consente la revoca selettiva.
- Chiavi: ogni oggetto ha associata una chiave principale. Quando si crea una capacità su un oggetto, gli viene associata la chiave principale. Quando si vuole usare l'abilitazione, si confronta la chiave con la chiave principale. La revoca cambia la chiave principale, invalidando le abilitazioni precedenti.