La memoria paginata è una tecnica di gestione della memoria centrale usata da molti SO che permette la multiprogrammazione e di poter utilizzare porzioni di memoria non contigue per un processo. Al programmatore ciò è trasparente, e vede la memoria logica come un enorme array che parte dall'indirizzo 0 fino al limite dello spazio di indirizzamento. Permette inoltre di non tenere l'intero processo in memoria, ma di scambiare pagine tra memoria centrale e area di swap quando necessario, riducendo l'uso di memoria centrale. Si separa quindi la visione dello spazio logico dei processi, potenzialmente grande quanto lo spazio di indirizzamento da quello fisico, grande quanto la memoria installata.
La memoria fisica è suddivisa in frame di dimensione fissa, e quella logica in pagine di dimensioni pari a quelle dei frame, in cui sono allocate.
La dimensione dei frame dipende dall'HW. Per semplicità è tipicamente una potenza di 2.
Gli indirizzi logici generati dalla CPU sono formati da numero di pagina (p) e offset nella pagina (d). p è usato come indice nella tabella delle pagine, una per processo, che contiene le traduzioni pagina-frame. Combinando numero di frame e offset si ottiene l'indirizzo fisico richiesto.
È il SO a decidere dove allocare le pagine, quindi deve mantenere una tabella dei frame per sapere quali sono vuoti e quali no.
Con la paginazione non si ha frammentazione esterna, poiché i frame sono di dimensione fissa; tuttavia c'è una minima frammentazione interna, poiché non tutti i programmi sono grandi un multiplo della dimensione della pagina, quindi parte dell'ultimo frame può essere inutilizzato. Riducendo la dimensione dei frame si riduce questo problema, ma anche le prestazioni.
Le traduzioni di indirizzi logici in fisici sono svolte in HW dall'MMU (Memory Management Unit), programmata dal SO a ogni cambio di contesto. Con tabelle delle pagine piccole, queste possono essere caricate in registri ad alta velocità; ma spesso sono molto grandi, quindi il SO carica solo l'indirizzo. Ciò significa che ogni accesso in memoria ne richiede un altro per recuperare la traduzione. Per ridurre il calo di prestazioni si usa un TLB (Translation Lookaside Buffer), una cache che mantiene le traduzioni recenti, evitando la maggior parte dei doppi accessi. È una memoria associativa, quindi piccola, veloce e costosa. Ogni entry mantiene anche un identificatore dello spazio di indirizzamento (ASID) così che possano esservi cachate traduzioni per più processi.
Protezione e condivisione:
Nella tabella delle pagine ogni riga mantiene dei bit di protezione, che indicano se la pagina è in sola lettura, lettura/scrittura o sola esecuzione. Viene anche mantenuto un bit di validità/invalidità, per marcare le pagine caricate, e rilevare accessi illegali. I tentativi di accessi illegali generano una trap hardware che sarà gestita dal SO (page fault).
Alcuni sistemi memorizzano anche la lungehzza della tabella delle pagine di un processo, poiché raramente questi usano tutto lo spazio di indirizzamento. Ciò permette di risparmiare memoria.
Spesso dei processi possiedono parti di codice uguali (es. librerie), se non l'intero codice (es. più istanze di un editor). Per evitare di duplicarle, queste pagine possono essere condivise tra i processi. Con la paginazione basta mappare lo stesso frame nella tabella delle pagine di più processi. Si usa anche per la comunicazione con memoria condivisa. La condivisione del codice ha 2 limitazioni: il codice non dev'essere automodificante, e deve trovarsi nella stessa locazione logica in tutti i processi, altrimenti non potrà autoreferenziarsi.
Struttura della tabella delle pagine:
I sistemi recenti hanno uno spazio di indirizzamento molto grande, il che significa tabelle delle pagine grandi (es. 64 bit, pagine di 64k (2^16 byte) = 2^{48} righe), che vanno memorizzate efficientemente.
- paginazione gerarchica: la tabella delle pagine è a sua volta paginata, anche 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 di overflow contenenta 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 caricarne solo alcune pagine, e iniziarne l'esecuzione. Quando tenterà di accedere a una pagina non caricata, genererà un page fault, poiché la pagina 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.

La memoria virtuale permette di eseguire processi che non sono completamente in memoria centrale o più grandi della memoria fisica e di condividere memoria facilmente. Ciò è trasparente al programmatore, che vede la memoria come un array uniforme grande quanto lo spazio di indirizzamento. In presenza di memoria paginata, è implementata con la richiesta di paginazione. Un processo viene avviato caricando 0+ pagine in altrettanti frame; quando tenterà di accedere a una pagina non caricata, genererà un page fault poiché il bit di invalidità è settato, e il SO, dopo aver verificato che non sia realmente un accesso illegale, caricherà dal disco la pagina richiesta, inserendola in un frame, dopodichè riavvierà l'istruzione interrotta. Quest'attività è svolta dal paginatore, che carica le pagine solo se necessarie. Permette di caricare solo il minimo necessario, e funziona anche per file mappati in memoria. Il numero minimo di frame di un processo dipende dall'architettura: è il numero massimo di frame cui un'istruzione può accedere.
Prestazioni:
Il tempo di accesso effettivo dipende dal tasso di page fault: un accesso a una pagina caricata richiede pochi ns, ma in caso contrario servono decine di ms per gestire il page fault e l'accesso al disco; quindi dev'essere raro per non rallentare il sistema.
La chiamata fork() che crea una copia del processo che la chiama. Spesso però il nuovo processo chiama subito exec(), che lo sostituisce con un altro programma; quindi è bene assegnare a entrambi lo stesso spazio di indirizzamento, e se uno modifica una pagina, sarà copiata nel suo spazio logico, risparmiando memoria e velocizzando la chiamata (copy on write). I SO tengono un pool di pagine libere per tali richieste, il cui contenuto è azzerato prima di assegnarle ai processi. Molte pagine non sono modificabili, il che permette altro risparmio.
La chiamata vfork(), funziona analogamente, ma senza copy on write.
Le applicazioni in grado di gestire la memoria autonomamente hanno prestazioni scarse con la memoria virtuale. Alcuni SO lo permettono, e forniscono aree di disco da usare con chiamate di I/O grezzo.
Processi realtime possono chiedere al SO di tenere residenti alcune pagine per evitare ritardi.
Sostituzione della pagina:
Quando un processo richiede una nuova pagina, potrebbero non esserci frame liberi in cui caricarla; inoltre per la località dei processi, pagine usate tempo fa non saranno riusate presto; quindi al caricamento di una nuova pagina, si può sostituirla a una già presente trasferendola in area di swap (se modificata), assegnando nuovi frame solo se necessario. La pagina ideale da sostituire è quella che non sarà usata per più tempo, ma è impossibile saperlo a priori. Per ridurre il tempo prima che il processo riprenda, il SO può tenere dei frame liberi in cui caricare la pagina richiesta, dandola al processo mentre quella da scaricare è in coda di I/O. Inoltre quando il paginatore è a riposo si possono scrivere le pagine modificate su disco per evitare di farlo in seguito. Esistono vari algoritmi per selezionare una vittima: Il più semplice è la sostituzione FIFO: viene selezionata la pagina più vecchia. Si basa sull'idea che le pagine caricate tempo fa non saranno riusate. Ciò però non è sempre vero, ad esempio per le librerie, quindi le prestazioni sono scarse. Soffre inoltre dell'anomalia di Belady: aumentando i frame al processo, il tasso di page fault può aumentare.
Algoritmo migliore è LRU (least-recently used): si sotitusce la pagina che non si usa da più tempo. È molto buono, ma richiede grande supporto HW per non rallentare il sistema. Possibili implementazioni sono: inserire un timestamp per ogni entry della tabella delle pagine, sostituendo quella con TS più basso; oppure uno stack che ad ogni accesso ponga in cima la pagina acceduta, sostituendo quella alla base. Non soffrono dell'anomaia di Belady. Per ridurre il supporto HW, esistono varie approssimazioni dell'LRU: si può porre un bit di riferimento ad ogni entry della tabella delle pagine, settato dall'HW e periodicamente azzerato dal SO. Ciò consente di sapere quali pagine sono state usate dall'ultimo azzeramento. Questa tecnica può essere estesa con più bit di riferimento: si tiene un byte per ogni pagina, il cui MSB è il bit di riferimento. Periodicamente il SO shifta a destra il contenuto del byte. La pagina con il valore più basso è quella da sostituire. Altra approssimazione è l'algoritmo della seconda possibilità: le pagine sono scandite in ordine FIFO: se quella selezionata ha bit di riferimento 0 è la vittima; altrimenti lo pone a 0, dandole la seconda possibilità e continuando con le successive circolarmente. Successive scansioni partono dall'ultimo elemento scandito. Se tutti i bit di riferimento sono a 1 degenera in FIFO. Esiste anche una versione migliorata, che controlla bit di riferimento e di modifica: la pagina migliore da sostituire è quella che li ha entrambi a 0, poiché riduce le chiamate I/O.

Esistono algoritmi basati sul conteggio: ogni pagina ha un contatore di riferimenti. Si può scegliere la pagina usata meno di frequente (le pagine attive hanno valori alti), facendo decadere il contatore periodicamente per evitare che una pagina usata pesantemente e mai riusata non venga scaricata; oppure si può sostituire quella usata più di frequente (le pagine recenti hanno valori bassi).
La selezione può riguardare solo i frame del processo, oppure essere globale.
Algoritmi di allocazione:
Per spartire m frame su n processi, il metodo più semplice è l'allocazione omogenea: ogni processo ottiene m/n frame. È equo, ma alcuni processi usano poca memoria sprecando il resto. Altro metodo è l'allocazione proporzionale: i frame sono divisi in modo direttamente proporzionale a dimensione/priorità, ecc.
Trashing:
Se i frame allocati a un processo sono insufficienti a contenere la sua località (le pagine usate in quella fase di elaborazione), questi passerà più tempo a paginare che ad elaborare: è il trashing, e si risolve spostando dei processi in area di swap per assegnare i loro frame ad altri. In caso di sostituzione globale, il trashing può propagarsi tra i processi. Non è possibile prevedere quanti frame servono a un processo, ma esistono varie approssimazioni: il modello working set definisce la località di un processo come le pagine accedute negli ultimi N (finestra del WS) accessi. Queste pagine compongono il WS. Se il totale delle pagine dei WS è maggiore del numero di frame, il sistema è in trashing. Ciò è utile anche alla prepaginazione: quando si riprende un processo sospeso, si carica in memoria il suo WS, così da ridurre i page fault all'avvio. È un metodo rozzo per controllare il trashing. La strategia della frequenza di page fault permette di prevenirlo: l'obiettivo è controllare il tasso di page fault di un processo: se sale sopra una certa soglia, gli si assegna un nuovo frame; se scende sotto una certa soglia, ne perde uno. Se non ci ne sono frame liberi, allora il processo va sospeso per evitare il trashing.

Il file system fornisce supporto per la memorizzazione e l'accesso di file e programmi ed è l'aspetto più visibile all'utente del SO.
Un file è un insieme di informazioni, memorizzato in memoria secondaria, identificato univocamente da un nome/path. Tipicamente è visto come un insieme di bit, la cui interpretazione spetta a uno specifico programma/utente. Può contenere testo, un eseguibile, …
Ha diversi attributi memorizzati nel descrittore: nome (non sempre presente nel descripttore), ID univico, tipo, dimensione, locazione (puntatore/i ai blocchi) , permessi, data ultimo accesso, ID proprietario, …
Operazioni su file: (chiamate di sistema)
-creazione: cerca lo spazio per memorizzare il file e crea un nuovo descrittore per quel file
-scrittura: specificando nome file e dati da scrivere. Mantiene un puntatore alla posizione della prossima scrittura
-lettura: specificando nome file e blocco di memoria in cui inserire i dati letti. Mantiene un puntatore alla posizione della prossima lettura
-seek: setta uno dei 2 puntatori
-cancellazione: cancella descrittore del file ed eventuali riferimenti e rilascia lo spazio
-troncamento: cancella il contenuto del file ma non il descrittore
-rinomina, accodamento, …
Per non cercare ad ogni chiamata il descrittore del file, in molti SO vanno aperti con la chiamata open, che aggiunge un elemento alla tabella dei file aperti contenente le informazioni sui file aperti (ID, posizione, processo, …), mantenute fino alla chiusura con la chiamata close. I processi non specificano il nome del file, ma l'indice nella tabella. SO più complessi usano 2 tabelle: una generale e una per processo (con riferimenti a quella generale). In questo modo più processi possono usare un file conteporaneamente. Memorizzano puntatori nel file (processo), contatore aperture del file (generale, varia ad ogni open/close, se arriva a 0 si rimuove l'elemento), posizione del file su disco (generale, presa dal descrittore), e permessi (processo).
Si possono bloccare file o parti di file con blocco condiviso (blocco in lettura, più processi possono accedere contemporaneamente) o esclusivo (blocco in scrittura, un solo processo può accedere; ottenibile solo se non ci sono altri blocchi attivi). Il lock può essere obbligatorio (va acquisito prima di poter usare il file) o consigliato (sta al processo decidere se acquisirlo, può accedere comunque).
Un file può essere di vari tipi. Tipicamente lo si include nel nome come estensione: un codice che ne indica il tipo. Non è obbligatorio: è l'applicazione ad interpretare i dati.
Un file può essere strutturato in 3 modi:
-nessuna: sequenza di byte
-semplice: linee di lunghezza fissa o variabile
-complessa: documenti formattati e simili
L'accesso a un file può avvenire in modo sequenziale (in ordine, un record dopo l'altro), diretto (si specifica la posizione all'interno del file a cui si vuole accedere; utile per file di blocchi a lunghezza fissa) o indicizzato (nel file c'è un indice con puntatori a diversi blocchi; utile per grandi file).
I FS possono essere vasti, quindi vanno organizzati al meglio: i dischi sono divisi in partizioni (volumi), strutture a basso livello in cui sono memorizzati file e directory. Una partizione può essere una parte di disco o occupare più dischi. L'utente non sa l'allocazione fisica dei dati, ma solo quella logica.
Una directory (cartella) è un file speciale con una tabella che associa nome di file e puntatore a descrittore (o il descrittore stesso, a seconda delle implementazioni). Permette di organizzare file. Ogni partizione ha una directory radice.
Le operazioni sulla struttura di directory sono:
-ricerca di un file
-creazione/cancellazione di file/directory
-listing di una directory
-rinomina (può anche spostare cambiando il path)
-attraversamento del file system
La struttura logica della directory può essere:
-singolo livello: tutti i file si trovano in un'unica directory. È semplice ma 2 file non possono avere lo stesso nome, quindi con molti file ci sono problemi, e non permette di organizzare i file

-2 livelli: si può far si che ogni utente abbia la propria directory (UFD), contenene i suoi file. La radice contiene i puntatori alle varie UFD. Non permette la condivisione di file
-Albero: la radice contiene puntatori a file e directory, e ogni sottodirectory può puntare ad altri file/ directory. È la struttura più comune. Ogni file è identificato da un path univoco che parte dalla radice e attraversa le directory fino ad arrivare al file.
-Grafo aciclico: come sopra ma consente di mettere un riferimento allo stesso file in più directory, rendendolo visibile a più utenti (quindi più path per raggiungerlo). Per ogni file si mantiene un contatore di riferimenti; quando arriva a 0 può essere rimosso dal disco. Si implementa con dei link o duplicando i descrittori, ma in questo caso bisogna mantenere la coerenza.
-Grafo generale: come sopra, ma permette i cicli, quindi è più semplice da realizzare. L'unico problema riguarda la rimozione effettiva dei file: se un file si autoreferenzia non arriverà mai a 0 riferimenti. Periodicamente si scansiona tutto il file system listando tutti i file raggiungibili ed eliminando gli altri (costoso)
Per rendere disponibile un FS ai processi, questo va montato. Si specifica nome del device e punto di mount, dai cui sarà accessibile. Il SO controlla che il non sia corrotto.
In un SO multiutente, i file sono condivisibili, ed è necessario memorizzare permessi di file/directory dei vari utenti. La protezione è tipicamente basata sull'identificazione dell'utente: ogni file/directory ha una ACL (Access Control List), con i permessi dei vari utenti. È flessibile, ma le ACL sono lunghe e di difficile gestione. Si possono raggruppare gli utenti in 3 categorie: proprietario, gruppo e altri, ognuno con i suoi permessi. Il proprietario stabilisce i permessi, il gruppo sono utenti con cui condividere il file.
Si può anche assegnare una password al file.
In LAN i file possono essere condivisi trasferendoli con FTP, HTTP, …, con o senza autenticazione, oppure montando dei FS di rete, tipicamente una sottodirectory di un FS su un server. Una macchina può essere client e server contemporaneamente. I dettagli variano con l'implementazione. Si possono fornire accessi diversi ai vari utenti.
Modifiche a file condivisi possono essere subito visibili, oppure solo dopo la close. Una sessione sono tutte le operazione eseguite tra la open e la close. Alcuni SO non consentono la modifica di file condivisi.

Il FS è tipicamente implementato a strati: al livello più basso ci sono i dischi (accesso casuale, a blocchi, mantengono dati senza alimentazione), poi il controllo I/O (driver), il FS di base (usa i driver per recuperare blocchi), il modulo di organizzazione dei file (gestisce allocazione e spazio libero e traduce indirizzi logici in fisici) e il FS logico, che gestisce struttura di directory, protezione e descrittori (FD con gli attributi di file/ directory) (metadati). Se il SO supporta più FS, alcuni livelli possono essere condivisi.
Servono diverse strutture dati: ogni partizione possiede un blocco di controllo del boot, tipicamente il primo settore, con le informazioni su come avviare il SO in quella partizione (se c'è); un blocco di controllo della partizione, con informazioni su dimensione e numero di blocchi, informazioni sui blocchi liberi, vari dati che variano col FS, FD e directory. Il SO mantiene una tabella delle partizioni, con informazioni sui FS montati; alcuni descrittori di directory acceduti di recente, e le tabelle dei file aperti. Una partizione può anche non contenere un FS (raw disk).
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 (tabella o lista concatenata). È semplice, ma una ricerca nella directory richiede una scansione lineare. Si migliora ordinandola o con una cache, ma l'implementazione è più complessa. La cancellazione di un elemento si può fare con un flag, sostituendo l'elemento da cancellare con l'ultimo elemento, … Si può usare una tabella di hash: si ricava la posizione nella tabella con una funzione hash sul nome, velocizzando la ricerca. Il problema sono le collisioni, mitigabili ingrandendo la tabella o con tabelle di hash in cui ogni elemento punta a una lista di overflow concatenata, così da poter inserire tutti gli elementi.
Allocazione dei file:
-Allocazione contigua: ogni file occupa un certo numero di blocchi contigui su disco. Si memorizza per ogni file indirizzo di inizio e lunghezza in blocchi. Accessi sequenziali e diretti sono rapidi, ma si crea frammentazione esterna, e se i file aumentano di dimensione può essere necessario spostarli (costoso). Una soluzione sono gli extent: si trova un altro buco in cui inserire altri blocchi e si memorizzano le posizioni dei frammenti. Ottime prestazioni, ma vanno evitate eccessive frammentazioni.
-Allocazione collegata: si memorizza per ogni file puntatore a blocco iniziale e finale, e in ogni blocco c'è il puntatore al successivo. Ciò permette ai blocchi di trovarsi ovunque sul disco e rapidi accessi sequenziali. Quelli diretti sono lenti (tutti i puntatori devono essere scanditi), i puntatori occupano nell'insieme molto spazio, e un puntatore corrotto causa la perdita del file. Lo "spreco" di spazio per i puntatori si riduce, al prezzo di frammentazione interna, raggruppando più blocchi in un cluster, e inserendo un puntatore per cluster. Una variante comune è la FAT (File Allocation Table): per ogni file si memorizza il blocco iniziale, e in una locazione speciale del disco si mantiene una tabella (FAT) che per ogni blocco del disco contiene o il blocco successivo o fine file o 0 se è vuoto. Ciò supera molti limiti appena visti, ma la FAT va tenuta in memoria per avere prestazioni.
-Allocazione indicizzata: per ogni file si mantiene, tipicamente nel FD, un array di puntatori, ognuno dei quali punta un blocco del file. Supera tutti i problemi delle altre soluzioni, ma è critica la dimensione del blocco indice: troppo piccolo non permette file grandi, troppo grande spreca spazio per file piccoli. Una soluzione è una lista di blocchi indice, oppure un indice multilivello (ogni puntatore punta a un altro blocco indice, fino ad arrivare ai blocchi), oppure una 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:
-Bitmap: si memorizza un bit per blocco: 1=libero, 0=occupato. Rende semplice trovare il primo blocco libero, o n blocchi liberi consecutivi, ma per avere prestazioni va tenuta in memoria centrale, e periodicamente aggiornata su disco e può essere grande.
-Lista collegata: ogni blocco libero contiene un puntatore al successivo, e si memorizza il puntatore al primo. Rende semplice trovare il primo blocco libero, ma non n blocchi consecutivi.
-raggruppamento: nel primo blocco libero si memorizzano gli indirizzi di n blocchi, di cui n-1 sono liberi, e l'ultimo contiene altri n puntatori, e così via. Permette di trovare i blocchi liberi rapidamente.
-Conteggio: spesso i blocchi liberi sono contigui. Si tiene una lista con indirizzo blocco libero e quanti lo seguono. La lista dei blocchi liberi è quindi breve.
In generale, le prestazioni si migliorano con le cache: una sul disco tiene le tracce che ritiene saranno usate presto, e in memoria centrale il SO tiene i blocchi che ritiene saranno usati presto. Questa cache può essere

gestita con tecniche di memoria virtuale, migliorando le prestazioni (memoria virtuale unificata). 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.
Montaggio:
Il montaggio permette ai processi di accedere a un FS: si specifica il device e il punto di mount, da cui sarà accessibile. Il SO controlla la coerenza del FS. Il FS si trova in stato incoerente se salta l'alimentazione e dei dati che dovrebbero trovarsi su disco ma erano in cache vanno persi. Il controllo della coerenza corregge questi errori, ma non è sempre possibile (es. se si perde un blocco indice). Si può far si che le scritture siano sincrone, ma diminuiscono le prestazioni. Alcuni FS tengono un log: il montaggio controlla se c'erano operazioni in corso annullandole per evitare danni.
E' comune che un SO interagisca con più FS diversi (locali diversi, di rete, …). Ciò rende le chiamate molto complesse. Il Virtual FS lo rende trasparente, e le chiamate interagiscono solo con esso. Ogni file/cartella nel sistema ha un proprio vnode, con ID unico. A livello più basso il VFS controlla i vari FS.

I dischi sono la principale memoria secondaria. Un disco è visto come un grande array di blocchi logici, la più piccola unità trasferibile in un'operazione, mappato nei settori in modo sequenziale, consentendo una semplice traduzione in indirizzo fisico (cylinder, head, sector).
I dischi hanno un tempo di seek (movimento testina su traccia) e uno di rotazione (movimento testina su settore). Schedulare le richieste in coda permette di ridurre il tempo di seek e aumentare la larghezza di banda. Esistono vari algoritmi:
-FCFS: l'ordine in cui arrivano. Semplice ed equo ma poco performante
-SSTF (Shortest Seek Time First): si serve per prima la richiesta che fa spostare meno la testina. Riduce i tempi di accesso, ma è possibile la starvation e non è ottimale
-SCAN: la testina si muove da un estremo all'altro del disco servendo le richieste che incontra, andata e ritorno. Se una richiesta arriva e sta per essere raggiunta dalla testina, sarà servita presto, altrimenti deve aspettare il ritorno. Riduce il numero di movimenti della testina, quindi l'usura. La variante C-SCAN serve le richieste in una sola direzione, garantendo tempi d'accesso più uniformi
-LOOK/C-LOOK: come sopra ma la testina si sposta solo fino agli estremi delle richieste da servire
La scelta va fatta in base al carico di lavoro (es. SSTF con pesante carico può incorrere in starvation), l'allocazione dei file (contigua o indicizzata) e alla posizione di directory e blocchi indice (vicini ai file o no). In genere si usano SSTF o LOOK. La schedulazione può essere fatta in HW, che può migliorare anche il tempo di rotazione, ma il SO può avere bisogno di dare priorità ad alcune operazioni, come la paginazione.
Amministrazione del disco:
Prima di poter essere usato, il disco va diviso in settori che il controller possa leggere/scrivere. È la formattazione fisica. Un settore è formato da intestazione, zona dati e terminatore, e contengono numero di settore e i dati di correzione errori. Il disco può essere partizionato e le partizioni formattate logicamente con un file system o usate in modo raw.
I settori si possono danneggiare, per la delicatezza del disco. Ciò causa perdita dei dati (se occupati). Si possono isolare marcandoli come danneggiati nella FAT, oppure si possono riservare dei settori di scorta da usare in caso alcuni si danneggino; oppure si può slittare di un settore i dati dopo il settore guasto.
Avvio:
All'avvio del computer, un programma in ROM carica dal disco di avvio il MBR (Master Boot Record), contenente anche un programma che carica un secondo loader dalla partizione del SO. Questo loader può caricare il kernel che continuerà l'avvio, o un loader più complesso. Sono scritti dal SO in settori fissi per un recupero semplice.
Swap:
L'area di swap contiene pagine/segmenti/immagini di processi parzialmente eseguiti scaricati dalla memoria centrale. Sono dati che presto saranno acceduti, quindi va gestita velocemente per non rallentare il sistema. Si può realizzare con un file gestito attraverso il file system. È semplice da implementare, ma soffre di frammentazione esterna e accessi lenti. Si può creare una partizione usata in modo raw dal gestore dello spazio di swap. È veloce, ma c'è frammentazione interna. Alcuni SO le implementano entrambe.
RAID:
Le configurazioni RAID permettono di vedere più dischi fisici come uno logico con particolari prestazioni o affidabilità. Le prestazioni sono migliorate con lo striping, la divisione di singoli byte o blocchi su più dischi (es. 8 dischi, per ogni byte, 1 bit per disco, oppure i byte di ogni blocco distribuiti in modo che il byte i vada sul disco i%8 +1). Permette alta velocità di trasferimento ma un guasto fa perdere tutti i dati. L'affidabilità è migliorata con la ridondanza, che può essere mirroring (copia dei dati su più dischi), o codici correttori d'errore (es. 4 dischi + 1 di parità). Le scritture aumentano, diminuendo le prestazioni, ma in un'unità di tempo possono essere eseguite più letture, e un guasto non causa perdita di dati.
Esistono vari livelli di RAID: i più comuni sono RAID 0 (striping a livello di blocco), RAID 1 (mirroring) e le combinazioni 0+1 e 1+0, per affidabilità e prestazioni. La scelta va fatta in base a esigenze e budget.
Collegamento dei dischi:
Un computer può accedere ai dischi tramite le porte di I/O (IDE, SATA, …), ossia per collegamento all'host, oppure per collegamento via rete (NAS). Nel primo caso i comandi sono richieste di blocchi logici mandati direttamente ai device; nel secondo sono chiamate di procedura remota mandate al NAS. Consente a computer in LAN di condividere dischi.

Memoria terziaria:
Le memorie terziarie hanno un basso costo per GB, alta affidabilità e velocità piuttosto bassa (tipicamente qualche MB/s). Sono utili per backup e per memorizzare enormi quantità di dati. Tipicamente sono supporti rimovibili inseriti in un lettore.
Esistono vari tipi di dischi rimovibili: alcuni scrivibili una volta, altri riscrivibili. Sonodischi magnetici (floppy), dischi ottici (cd/dvd) e dischi magneto-ottici. Spesso il SO li gestisce come fossero fissi: vi si può creare un file system o usarli in modo raw, più processi possono usarli, in modo diretto o sequenziale.
I nastri magnetici sono più capienti e costano meno. L'accesso è solo sequenziale, poiché un accesso diretto significherebbe (ri)avvolgere continuamente il nastro. Tipicamente sono usati per backup. Il SO permette alle applicazioni di usarli in modo raw, solo in modo esclusivo. Tipicamente quindi un nastro può essere letto solo dall'applicazione che l'ha scritto. Teoricamente è possibile implementare un file system su nastro, ma è piuttosto complesso.
Un problema di gestione della memoria terziaria è la codifica se le cartucce possono essere usate su diverse macchine. Altro problema è la nomina dei file: il SO non può sapere su quale cartutuccia si trova il file richiesto. Una soluzione è di avere un ID unico per cartuccia, e inserirlo nel nome del file, ma è scomodo per l'utente. Tipicamente il problema è lasciato alle applicazioni.
Memorizzazione gerarchica:
Un braccio meccanico può essere usato dal computer per scambiare le cartucce da un archivio al lettore. Può essere usato per salvataggi (quando la cartuccia è piena, ne inserisce un'altra), o per estendere il FS oltre la memoria secondaria: i file piccoli e usati spesso sono tenuti su disco per uso attivo; quelli grandi e vecchi su memorie terziarie: se servono sono trasferiti automaticamente su disco.

Il sottosistema di I/O ha il compito di gestire le periferiche efficientemente e permettere alle applicazioni di usarle tramite chiamate di sistema, nascondendo i dettagli dell'HW sottostante.
L'implementazione è stratificata: al livello più basso si trova l'HW e ciò che riguarda la comunicazione; appena sopra si trovano i device driver, la cui funzione è nascondere le differenze tra device dello stesso tipo (es. vari modelli di dischi). Sono scritti dal produttore del device: da un lato interagiscono col controller del device, dall'altro col SO con un'interfaccia standard, che varia col SO. Infine si trova il resto del sottosistema di I/O, che esegue operazioni che non dipendono dal dispositivo. Il kernel mantiene informazioni sull'uso dei device in strutture simili alla tabella dei file aperti.
I device differiscono per:
-Trasferimento a caratteri (1 byte/operazione) o a blocchi di dimensione fissa (n byte/operazione)
-Accesso sequenziale (byte trasferiti in ordine prefissato) oppure diretto (in ordine qualsiasi)
-Trasferimento sincrono (blocca il processo finchè non finisce) o asincrono (il processo continua parallelamente)
-Condivisibile (utilizzabile da più processi contemporaneamente) o dedicata (un solo processo)
-Velocità e latenza
-Lettura e scrittura, sola lettura, sola scrittura
Alle applicazioni molte differenze sono nascoste dal SO, che raggruppa le periferiche in poche categorie convenzionali. Le chiamate di sistema variano col SO, ma tipicamente riguardano il blocco di periferiche, i trasferimenti, i file memory mapped, orologi/timer, funzioni grafiche e sonore. Molti SO permettono alle applicazioni di interagire direttamente col driver se necessario (escape).
Dispositivi a blocchi:
L'interfaccia di questi device si occupa di dischi e altre periferiche a blocchi. Questi device devono comprendere i comandi read, write, e seek se ad accesso diretto. Le applicazioni spesso vi interagiscono attraverso il file system, ma il SO e alcune applicazioni possono usare chiamate di I/O grezzo.
L'accesso a file memory mapped è spesso stratificato sopra i driver di device a blocchi, e usa tecniche di memoria virtuale.
Dispositivi a caratteri:
L'interfaccia di questi device 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:
Molti SO forniscono un'interfaccia diversa da quella dei dischi per le reti, poiché sono molto diverse. Tipicamente sono i socket, che permettono a un'applicazione di aprire una connessione a un host in attesa, o di attendere connessioni, e poi comunicare con chiamate simili a read e write su un flusso full duplex.
Orologi e timer:
Delle chiamate permettono di ottenere l'ora, l'uptime del sistema o di far eseguire un'operazione a un certo tempo (es. interrupt). L'HW è un temporizzatore programmabile. Poiché più processi e il SO possono usarlo, questi tiene una lista di tutti gli eventi e programma continuamente il timer in ordine cronologico.
I/O bloccante e non bloccante:
Le chiamate bloccanti bloccano il processo finchè non sono completate, poi lo rimettono nello stato di pronto. Vanno bene ad esempio i nastri. Quelle non bloccanti invece non bloccano il processo, e sono utili per le interfacce grafiche. È bene distinguere tra non bloccante e asincrona: nel primo caso il processo riceve solo i dati già pronti; nel secondo l'operazione è eseguita parallelamente al processo, che viene poi notificato.
Il sottosistema di I/O fornisce:
-Schedulazione: per alcune periferiche (es. dischi), è utile schedulare la coda di richieste per ridurre i tempi di attesa e migliorare le prestazioni globali.
-Buffering: un buffer è una zona di memoria temporanea con i dati in transito da device a device o da processo a device. Ha 3 funzioni: ovviare alla differenza di velocità tra 2 device (es. da modem a disco: quando il buffer è pieno viene scritto con una sola operazione); adattatore tra periferiche con blocchi di dimensione diversa; supportare la semantica della copia, ossia 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 copie di dati, per evitare costosi accessi. Differisce dal buffer in quanto questi contiene l'unica copia del dato. A volte le 2 funzioni si sovrappongono: ad esempio il buffer del disco può essere usato anche come cache.
-Spooling e prenotazione dei device: alcuni device non possono essere usati da più processi contemporaneamente. Lo spooler conserva i dati per questi device in buffer (su disco), e li invia in ordine. Alcuni SO permettono ai processi di prenotare l'accesso esclusivo a un device (possibili deadlock).
-Gestione errori: possono verificarsi guasti meccanici, elettronici, disturbi, … Il SO fa fronte agli errori temporanei, ma in caso di guasti permanenti deve informare il processo che la richiesta è fallita, tipicamente ritornando un codice d'errore dalla chiamata.
La trasformazione delle richieste di I/O in operazioni HW avviene in questo modo: Il processo effettua una chiamata di I/O; il SO ne controlla la correttezza e se è consentita, quindi controlla se può soddisfare subito la richiesta (es. richiesto un blocco già in cache), e in questo caso il controllo torna subito al processo, con gli eventuali dati richiesti; altrimenti manda la richiesta al driver del device, ed eventualmente blocca il processo; il driver elabora la richiesta, se necessario alloca buffer in memoria del SO e da comandi al controller, che esegue l'operazione e avverte il SO con un interrupt; il driver viene avvertito, e segnala al sottosistema di I/O che la richiesta è completata; 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, il numero di copie dei dati, 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 I/O ed elaborazioni contemporaneamente ed evitare di sovraccaricare un componente coinvolto (CPU, memoria, bus, device), poiché rallenterebbe l'intero sistema.

Per protezione si intende la sicurezza delle risorse (oggetti) fisiche (HW) e logiche (file, strutture dati, …) contro accessi non autorizzati di utenti, processi, …
Si distinguono politiche (regole) e meccanismi (strumenti che le applicano), per maggior flessibilità. La protezione è fornita da un meccanismo che controlla l'accesso alle risorse, che deve fornire i mezzi per stabilre le regole e applicarle. Riceve le richieste dei processi e decide se autorizzarle.
Ogni risorsa ha un identificativo e un insieme di operazioni. I processi dovrebbero poter accedere solo alle risorse minime necessarie alla loro computazione, così da limitare i danni in caso di errori. Ogni processo opera in un dominio di protezione, che definisce le risorse a cui può accedere e le operazioni lecite (diritti d'accesso). I domini non sono necessariamente disgiunti. L'associazione tra processo e dominio può essere statica oppure dinamica. Nel primo caso il processo è fisso in un dominio, il che significa che in vari momenti può avere più autorizzazioni del necessario, non rispettando il principio di minima conoscenza (a meno che si possano cambiare i diritti d'accesso); nel secondo caso può cambiare dominio (se ne ha il diritto). È più difficile da implementare, ma permette di applicare meglio il principio di minima conoscenza.
Un dominio può essere realizzato in vari modi. Può essere un utente (si cambia dominio quando si cambia utente), un processo (cambio ad ogni cambio di contesto), o anche una procedura. Lo switch 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.
Matrice d'accesso:
Fornisce un meccanismo per specificare le politiche. Ha come righe i domini e come colonne le risorse. Nella cella i,j sono presenti le operazioni che un processo del dominio i può fare sulla risorsa j. I domini possono essere visti anche come risorse, con operazione switch (se nella cella i,j è presente switch, allora un processo del dominio i può passare al dominio j).
La modifica del contenuto della matrice è controllata da 3 operazioni:
-Copia: copia l'autorizzazione che un dominio ha su un oggetto a un altro dominio, con o senza diritto di propagarlo. Copiando e cancellando si possono trasferire autorizzazioni
-Proprietà: consente a un processo di cambiare i diritti che altri domini su un oggetto di proprietà del suo dominio
-Controllo: consente a un processo di cambiare i diritti di un dominio di cui il suo dominio ha il controllo
I diritti copia e proprietà permettono di evitare il propagarsi dei diritti di accesso, ma non garantiscono il contenimento delle informazioni, che è considerato un problema irrisolvibile.
La matrice è molto grande e sparsa, quindi va memorizzata efficientemente:
-Tabella globale: è l'implementazione può semplice. Memorizza una lista di terne <dominio, oggetto, diritti>. Un processo in un dominio i è autorizzato a eseguire un'operazione m su un oggetto j solo se esiste una terna <i,j,d> con m ϵ d.
Questa lista è molto grande, e non permette di raggruppare oggetti o domini, che ne faciliterebbe la gestione.
-Access Control List (ACL): per ogni oggetto si memorizza una lista di coppie <dominio, diritti>. Si possono anche definire dei diritti predefiniti. Questo sistema risponde ai bisogni degli utenti, e memorizza informazioni globali, ma è inefficiente su grandi sistemi, poiché le ACL sono scandite spesso.
-Capability List (CL): per ogni dominio si memorizza una lista di coppie <oggetto, diritti> dette capability. Questo sistema rende semplice l'accesso a informazioni sui processi, e memorizza informazioni locali, ma la revoca è poco efficiente.
Queste strutture non sono direttamente accessibili ai processi, che potrebbero alterarle, ma solo dal SO.
-Meccanismo lock-key: compromesso tra ACL e CL. Ogni oggetto ha una lista di stringhe di bit uniche dette lock, e ogni dominio ne ha un'altra di key. Un processo in un dominio può accedere a un oggetto solo se una sua key apre uno dei lock della risorsa.
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 (tutti), temporanea o permanente.
Con le ACL la revoca è semplice, ma con le CL è più complessa in quanto le informazioni sono sparse per il sistema. Esistono varie soluzioni:

-Riaquisizione: le CL sono periodicamente svuotate. Se un processo tenta di usare una capability cancellata, tenterà di riacquisirla. Se è stata revocata gli sarà negata
-Puntatori alle capacità: per ogni oggetto si tiene una lista di puntatori a tutte le capability ad esso associate. Per cancellare una capability basta seguire il puntatore. Costoso ma efficace
-Indirezione: le capacità puntano a una cella di una tabella globale con tutte le capability, che a sua volta punta all'oggetto associato. La revoca cancella il valore in questa tabella. Non consente la revoca selettiva
-Chiavi: Ogni capability ha una key. Ogni oggetto possiede una master key, in base alla quale la capacità è ricercata e verificata. Alla creazione di una capability, le viene associata una key uguale alla master key in quel momento. La revoca cambia la master key. Non consente la revoca selettiva
Alcuni SO scaricano la responsabilità della protezione sul compilatore. Ciò permette maggiore flessibilità rispetto all'implementazione nel SO, ma meno sicurezza.

Un limite della paginazione è che non è possibile tipizzare le varie porzioni di memoria logica (codice, dati, …). Come la paginazione, la segmentazione è una tecnica di gestione della memoria centrale che consente la multiprogrammazione e di poter utilizzare porzioni di memoria non contigue per un processo, in modo trasparente al programmatore. Permette inoltre di 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.
La memoria fisica è divisa in segmenti fisici (frame) di dimensione variabile, ognuno identificato da un numero. La memoria logica di un processo è divisa in segmenti logici (segmenti), ognuno inserito in un frame di egual dimensione. I segmenti possiedono informazioni di tipo.
L'obiettivo è di dare consistenza logica alle porzioni di spazio di indirizzamento dei processi, consentendone la tipizzazione e una condivisione semplice.
Gli indirizzi logici prodotti dalla CPU sono costituiti da numero di segmento (s) e offset (d). s è usato come indice nella tabella dei segmenti, una per processo, contenente le traduzioni segmento-frame, col relativo indirizzo base del segmento e la sua lunghezza. Combinando indirizzo base e offset si ottiene l'indirizzo fisico richiesto. La traduzione è fatta in HW dall'MMU (Memory Management Unit), che ad ogni traduzione controlla anche dei bit di protezione nella tabella dei segmenti per impedire accessi a segmenti inesistenti, offset oltre il limite del segmento o altri accessi illegali, generando in quel caso una trap hardware gestita dal SO (segmentation fault).
L'MMU è programmata dal SO ad ogni cambio di contesto. Con tabelle piccole, queste possono essere caricate in registri ad alta velocità; ma spesso sono grandi, quindi il SO carica solo l'indirizzo in cui si trova la tabella dei segmenti. Per ogni accesso ne serve quindi un altro per recuperare la traduzione. Per evitare il calo di prestazioni si usa un TLB (Transation Lookaside Buffer), una cache che mantiene le traduzioni recenti. Ogni entry mantiene anche un identificatore dello spazio di indirizzamento (ASID), così che possa cachare traduzioni per più processi.
Protezione e condivisione:
Poiché ogni segmento contiene una porzione diversa del programma, definita dal programmatore, è probabile che i suoi elementi siano usati allo stesso modo; di conseguenza si associano dei bit di protezione a ogni segmento (es. sola esecuzione per segmento codice).
La condivisione di codice o dati è semplice: si mappa lo stesso frame nelle tabelle dei segmenti di più processi. La condivisione del codice è utile per non avere più copie del codice dello stesso programma (es. più istanze di un editor) e per condividere librerie tra processi. Gli unici vincoli sono che il codice non dev'essere automodificante, e che il segmento condiviso deve avere lo stesso numero di segmento logico nei vari processi, altrimenti non potrà autoreferenziarsi. Non ci sono vincoli per la condivisione di dati, che può essere usata anche per comunicazione con memoria condivisa.
Frammentazione:
Quando si inserisce un nuovo segmento in memoria si deve trovare un "buco" abbastanza grande per contenerlo. Esistono varie strategie: il più piccolo sufficiente (best fit), il primo sufficiente (first fit), il più grande (worst fit). Le prime 2 sono le più usate; tuttavia è improbabile che un buco sia grande esattamente quanto il segmento da inserire, di conseguenza il resto di quel buco resta inutilizzato. Dopo del tempo è probabile che ci siano molti piccoli buchi sparsi per la memoria, che riuniti potrebbero ospitare altri segmenti. È il problema della frammentazione esterna. Per ridurlo si può compattare periodicamente la memoria, ma è pesante. Non c'è frammentazione interna, dato che i segmenti sono di dimensione decisa in base alle necessità del programmatore.
Segmentazione con paginazione:
Paginazione e segmentazione hanno entrambi vantaggi e svantaggi. Questa tecnica permette di combinare i vantaggi di entrambi. Come nella paginazione, la memoria fisica è divisa in frame di dimensione fissa, che evita frammentazione esterna. Prende inoltre l'identificazione dei frame liberi, la scelta del frame in cui caricare una pagina e la gestione semplice dell'area di swap. Come nella segmentazione, la memoria logica è divisa in segmenti, a loro volta divisi in pagine inserite nei frame. Ciò consente di sfruttare i vantaggi della segmentazione in merito alla protezione, condivisione e tipizzazione. Gli indirizzi logici sono formati da una tripla numero di segmento (s), pagina nel segmento (p), offset nella pagina (d) e quelli fisici da numero di frame (f) e offset (d). La traduzione è svolta dall'MMU, gestita dal SO. Tutto ciò è trasparente al

programmatore.