# Capitolo 3: struttura dei sistemi operativi

- Componenti del sistema operativo.
- Servizi del sistema operativo.
- Chiamate di sistema.
- Programmi di sistema.
- Struttura del sistema.
- Macchine virtuali.
- Progettazione e implementazione di un sistema operativo.
- Generazione del sistema operativo.

# Componenti del sistema operativo

- Gestione dei processi.
- Gestione della memoria centrale.
- Gestione dei file.
- Gestione dell'input/output.
- Il sistema di protezione.
- Gestione delle unità di memorizzazione secondarie.
- Reti informatiche.
- L'interprete dei comandi.

# Gestione dei processi

- Un *processo* è un programma in esecuzione.
  - Per compiere le proprie attività un processo ha bisogno di specifiche risorse, tra cui il tempo della CPU, la memoria, i file e i dispositivi di I/O.
- In relazione alla gestione del processo il sistema operativo deve:
  - Generare e cancellare processi.
  - Sospendere e riattivare processi.
  - Fornire meccanismi per:
    - la sincronizzazione dei processi,
    - la comunicazione tra processi.

# Gestione della memoria centrale

- La m*emoria centrale* è un grande vettore di parole o byte, ciascuno dei quali ha un proprio indirizzo.
  - è un deposito di dati velocemente accessibili e condivisi dalla CPU e dai dispositivi di I/O.
- La *memoria centrale* è un dispositivo di storage volatile. Perde il suo contenuto in caso di arresto del sistema.
- Il sistema operativo è responsabile di una serie di attività legate alla gestione della memoria:
  - tenere traccia di quali parti della memoria sono correntemente in uso e da chi sono usate;
  - decidere quali processi devono essere caricati in memoria quando lo spazio diventa disponibile;
  - allocare e deallocare lo spazio di memoria in base alle richieste delle applicazioni.

# Gestione dei file

- Un *file* è un sistema di informazioni correlate definite dal loro creatore.
  - Normalmente i file rappresentano programmi (sia in forma sorgente sia in forma oggetto) e dati.
- Il sistema operativo è responsabile delle seguenti attività legate alla gestione dei file:
  - Creare e cancellare i file.
  - Creare e cancellare le directory.
  - Supportare le primitive relative alla manipolazione di file e directory.
  - Copiare i file su *unità di memorizzazione secondaria.*
  - Salvare i file su *supporti di memorizzazione permanenti* (cioè non volatili).

# Gestione dell'input/output

- Il sistema di I/O consiste in:
  - Una componente per la gestione della memoria che include i vari buffer e la cache
  - Una componente per la gestione del processo di stampa.
  - Un'interfaccia generale del driver.
  - Un driver per ciascun dispositivo hardware specifici.

# Sistema di protezione

- La *protezione* è l'insieme dei meccanismi per il controllo dell'accesso alle risorse da parte dei processi o degli utenti del computer.
- Il sistema di protezione deve:
  - distinguere tra uso autorizzato o non autorizzato da parte dell'utente;
  - specificare i controlli che devono essere effettuati;
  - fornire un metodo per imporre i criteri stabiliti.

# Gestione della memoria di massa

- Poichè la memoria centrale (*unità di memorizzazione primaria*) è volatile e poco capiente per contenere tutti i dati e i programmi in modo permanente, un calcolatore deve essere dotato di un'*unità di memorizzazione secondaria* in cui salvare i dati della memoria centrale.
- La maggior parte dei computer usa i dischi come dispositivi di memorizzazione, sia per i programmi che per i dati.
- Il sistema operativo è responsabile delle seguenti attività legate alla gestione dei dischi:
  - Gestire lo spazio libero.
  - Allocare le unità disco.
  - Occuparsi della schedulazione dei dischi.

# Sistemi distribuiti

- Un sistema *distribuito* può essere inteso come insieme di processori che non condividono la memoria o il clock.
  - Ogni processore lavora su una propria memoria locale.
- I processori sono collegati attraverso una generica rete di comunicazione.
- La comunicazione si realizza rispettando uno specifico paradigma che raccoglie le regole di interazione (*protocollo).*
- Un sistema distribuito fornisce all'utente l'accesso alle varie risorse ovunque distribuite.
- L'accesso ad una risorsa condivisa incrementa:
  - la velocità di calcolo,
  - la disponibilità,
  - l'affidabilità.

# Interprete dei comandi

- Molti ordini sono impartiti al sistema operativo mediante *istruzioni di controllo* le quali permettono di:
  - Creare e gestire i processi.
  - Gestire le chiamate di I/O.
  - Gestire i sistemi di memorizzazione secondaria.
  - Gestire la memorizzazione centrale.
  - Accedere al file-system.
  - Accedere alle protezioni.
  - Accedere alla rete.
- Il programma che riceve ed interpreta le istruzioni di controllo è chiamato in diversi modi:
  - *Interprete delle istruzioni di controllo*
  - *Shell (in UNIX)*
  - La sua funzione è quella di recepire ed eseguire una sequenza di comandi.

# Interprete dei comandi

- È la parte esterna del S.O., attraverso la quale gli utenti richiedono l'esecuzione dei programmi (siano essi programmi di utilità dello stesso S.O. o programmi applicativi) o accedono ai servizi del kernel.
- L'interfaccia può essere di due tipi:
  - **a caratteri:** shell o command interpreter è un vero e proprio ambiente di programmazione che permette all'utente di controllare l'esecuzione di comandi, sia in modo interattivo che in modo "batch" (mediante "script" di shell). Tipici S.O. con interfaccia a caratteri sono: DOS, UNIX, LINUX.
  - **grafica o iconica (GUI):** è un sistema d'interazione uomo-macchina basato su simboli (icone) che rappresentano le risorse su cui è possibile operare. Tipici S.O. con interfaccia iconica sono Microsoft Windows (in tutte le sue versioni), MAC OS, UNIX vers. OSF Motif, LINUX vers. OSF Motif.
- Una shell si basa sull'esecuzione di tre tipi di **comandi**:
  1. **oggetti eseguibili (o comandi esterni):** sono file contenenti programmi in formato eseguibile (p.es *sort* per la SHELL UNIX)
  2. **comandi built-in (o comandi interni):** sono comandi che realizzano funzioni semplici, eseguite direttamente dalla shell stessa (p.es *cd* per la SHELL UNIX)
  3. **script (o comandi batch):** sono "programmi" in linguaggio di shell
- Quando si digita un comando, la shell verifica se si tratta di un comando built-in. In caso positivo, lo esegue. Altrimenti la shell crea (fork) un nuovo processo che esegue il comando.
- Oltre all'esecuzione di comandi, la shell consente altre funzioni tipiche:
  - **ridirezione** dell'input e dell'output
  - **pipeline** di comandi
  - **filtering**

# Servizi del sistema operativo

- **Esecuzione di un programma** – il sistema deve potere caricare un programma in memoria e far funzionare tale programma.
- **Operazioni di I/O** – in generale gli utenti non possono controllare direttamente i dispositivi di I/O; è il sistema operativo che deve fornire i mezzi per compiere le operazioni di I/O.
- Manipolazione del file system – i programmi devono poter leggere, scrivere, creare e cancellare i file.
- **Comunicazioni** – la comunicazione può avvenire fra processi in esecuzione sullo stesso calcolatore o in esecuzione su calcolatori differenti collegati fra loro in rete. Le comunicazioni possono essere implementate attraverso una *memoria condivisa* o attraverso l'*invio di messaggi*.
- **Rilevamento degli errori** – il sistema deve assicurare un metodo efficiente per individuare errori a carico di CPU, hardware della memoria, dispositivi di I/O e programmi utente.

# Servizi addizionali

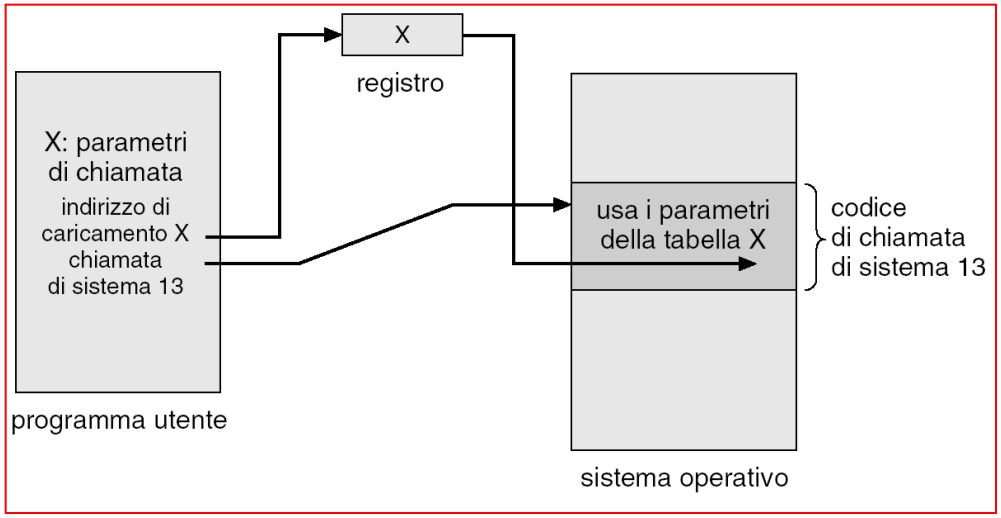
Esiste un altro gruppo di funzioni del sistema operativo che non servono a coadiuvare l'utente quanto piuttosto ad assicurare una gestione efficiente delle risorse della macchina.

- **Allocazione delle risorse** – quando ci sono più utenti o più processi contemporaneamente in funzione le risorse devono essere assegnate a ciascuno di loro in maniera opportuna.
- **Contabilità** – mantiene traccia di quali utenti usano quali risorse (specificazione di tipo e genere) per stilare statistiche d'uso e mantenerne aggiornata la contabilità.
- **Protezione e sicurezza** – implica la garanzia che tuttli gli accessi alle risorse del sistema siano controllati.

# Chiamate di sistema

- Le *chiamate di sistema* forniscono l'interfaccia tra un processo in esecuzione ed il sistema operativo.
  - Sono generalmente disponibili come istruzioni in linguaggio assembler.
  - I linguaggi sviluppati per sostituire il linguaggio assembler nella programmazione (soprattutto C, C++) permettono alle chiamate di sistema di avvenire direttamente.
    - In linguaggio Java (write once, run everywhere) si adoperano metodi *nativi*.
    - Per agevolare l'uso delle system call da parte dei programmatori, i compilatori mettono a disposizione una serie di funzioni di libreria (run-time support).
- Per passare i parametri tra un programma in esecuzione e il sistema operativo si usano tre metodi generali:
  - Passare i parametri nei *registri.*
  - Immagazzinare i parametri in una tabella in memoria, e l'indirizzo della tabella viene passato come parametro in un registro (LINUX).
  - I parametri sono posti (PUSH) in una memoria provvisoria detta *pila* (stack) dal programma e prelevati (POP) da essa dal sistema operativo.

# Passaggio dei parametri come blocco




# Tipologie di chiamate di sistema

- Controllo del processo.
- Gestione dei file.
- Gestione del dispositivo.
- Gestione delle informazioni.
- Comunicazioni.

# Programmi di sistema

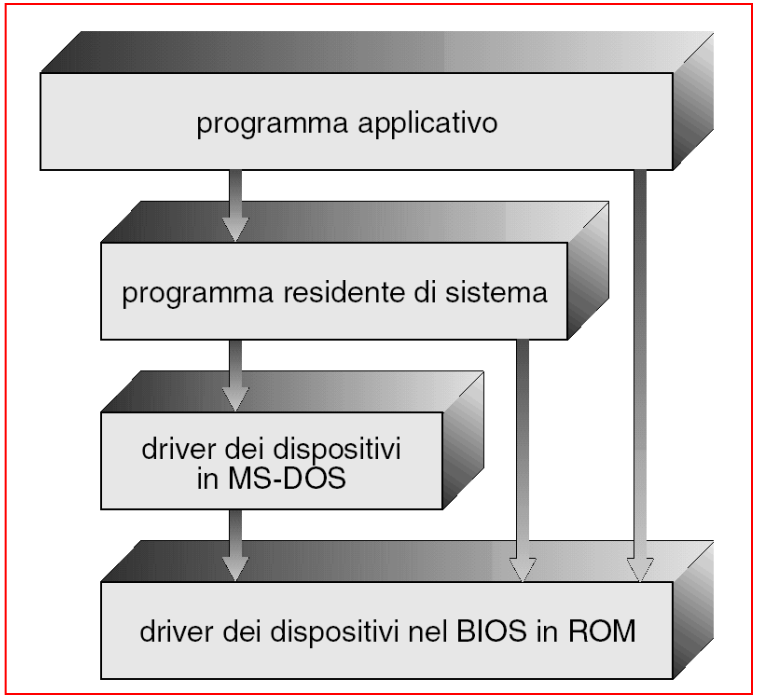
- I programmi di sistema forniscono un ambiente opportuno per lo sviluppo e l'esecuzione dei programmi. Possono essere distinti in:
  - Gestione dei file (gestione di file e directory).
  - Informazioni di stato (dati relativi all'uso della macchina formattate ed inviate in output).
  - Modifica dei file (manipolazione di file di testo).
  - Supporto ai linguaggi di programmazione (compilatori, assemblatori, interpreti per i linguaggi di programmazione più comuni).
  - Caricamento ed esecuzione dei programmi (linker, loader, debugger per linguaggi di alto livello e/o linguaggio macchina).
  - Comunicazioni (process networking, mail editor, messenger, remote desktop, file transfer).
  - Programmi applicativi (web browser, utility di sistema).
- La visione del sistema operativo offerta alla maggior parte degli utenti è definita dai programmi di sistema piuttosto che dalle effettive chiamate di sistema.

# Architettura di un S.O.

- Una tipica suddivisione delle architetture dei S.O. è la seguente:
  - **monolitiche**, quando esse sono composte da un unico modulo che serve le richieste dei programmi-utente singolarmente;
  - **a macchina virtuale**, se esse offrono a ciascun programma-utente visibilità di un particolare hardware;
  - **client-server,** se esse prevedono un nucleo minimo di funzioni comuni (*microkernel*) a tutte le stazioni di un sistema distribuito, a cui alcune stazioni (*server*) aggiungono funzioni specifiche per offrire servizi ad altre stazioni (*client*);
  - **a livelli**, se esse sono articolate in diversi moduli, ciascuno dei quali svolge specifiche funzioni, ed ogni modulo può servire le richieste di più programmi-utente.

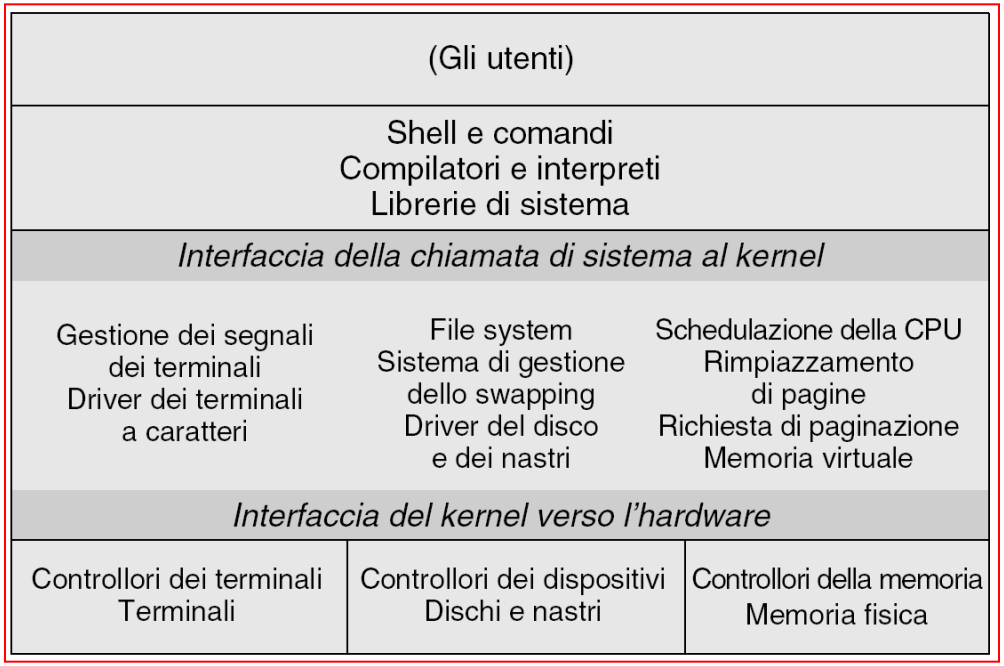
# Struttura di MS-DOS

- MS-DOS – scritto per fornire la massima funzionalità nel minor spazio.
  - Non è diviso in moduli.
  - Benchè MS-DOS possieda una certa struttura, le sue interfacce ed i livelli di funzionalità non sono ben separati.
  - È vulnerabile a programmi errati o maliziosi anche se nasce per il processore Intel x8088 che non prevedeva meccanismi di protezione HW



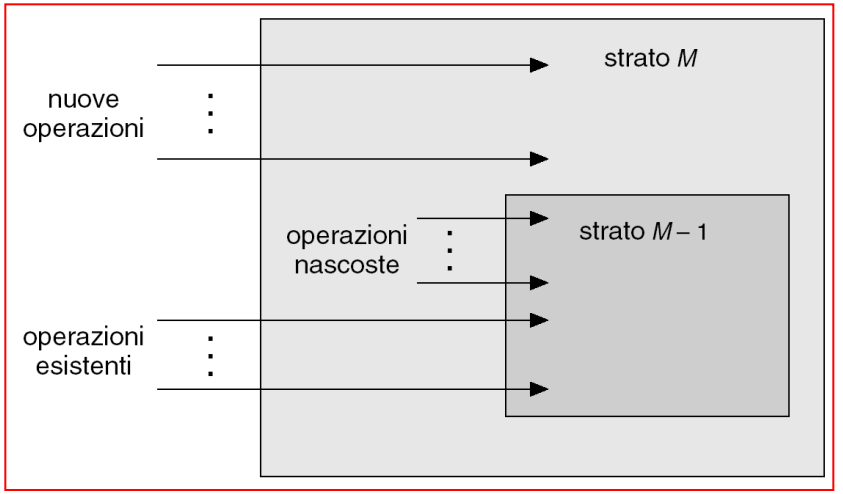
# Struttura di UNIX

- UNIX, limitato dalle funzionalità hardware, l'originario sistema operativo UNIX aveva una stuttura elementare. Consisteva in:
  - Programmi di sistema
  - Nucleo (*kernel*)
    - Consiste in qualsiasi parte si trovi sotto l'interfaccia di chiamata di sistema e sopra l'hardware fisico.
    - Le chiamate di sistema definiscono l'interfaccia per i programmi applicativi (API)
    - Un'enorme quantità di funzionalità combinate in un solo livello.
    - Per migliorare il controllo sulla macchina è auspicabile una strutturazione maggiormente flessibile di tipo modulare.



# La struttura stratificata

- Il sistema operativo è diviso in un certo numero di strati (livelli, o *layer*), ognuno costruito sulla sommità dello strato inferiore. Lo strato più basso (il livello 0) è l'hardware; quello più elevato (il livello N) è l'interfaccia utente.
- Uno strato del S.O. è un oggetto astratto costituito da un insieme di strutture dati e funzionalità atte ad intervenire su di esse.
- Con la *modularità*, gli strati sono selezionati in modo che ogni utente usi solo le funzioni (operazioni) ed i servizi degli strati di livello più basso.
- La modularità agevola l'individuazione di errori nel progetto e implementazione di un S.O., ma necessita di una attenta progettazione (ogni strato può adoperare solo funzionalità di livello inferiore).
- Ciascuno strato nasconde i suoi dettagli implementativi agli strati superiori che esclusivamente ne conoscono e adoperano le funzioni.
- La principale difficoltà di approccio è nell'esatta definizione dei vari livelli.

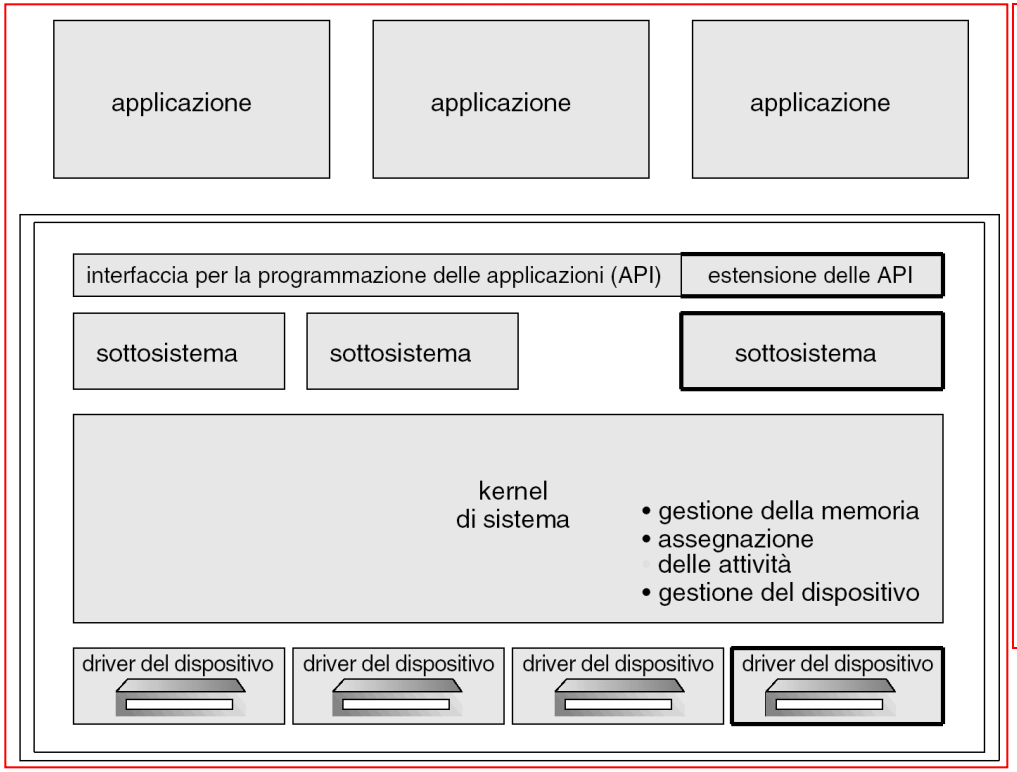



**Struttura di un generico strato**

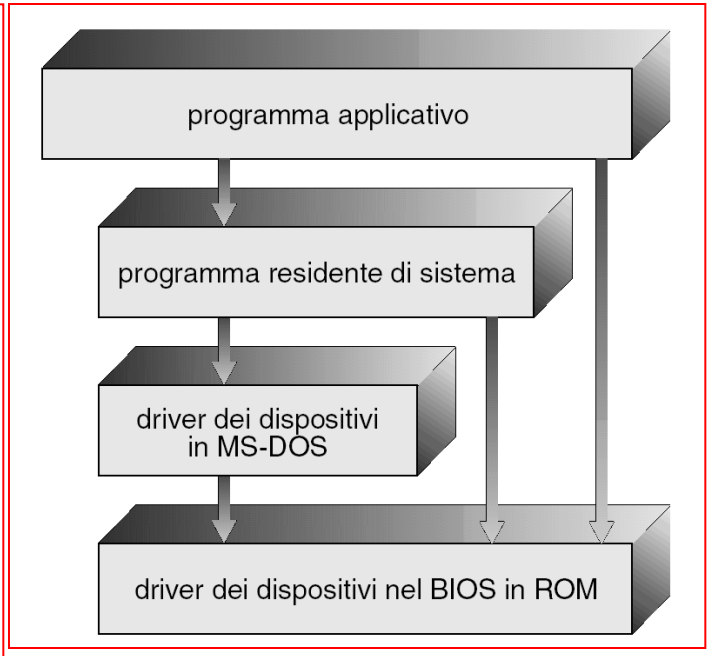
# La struttura stratificata

- Le implementazioni a strati tendono ad essere meno efficienti di quelle di genere diverso, ogni strato aggiunge sovraccarico alle system call:
  - Es. I/O => Memory manager => CPU scheduler => HW
- Maggiore efficienza si ottiene progettando pochi strati con più funzionalità

**Struttura a strati di OS/2**




**Struttura di MS-DOS**

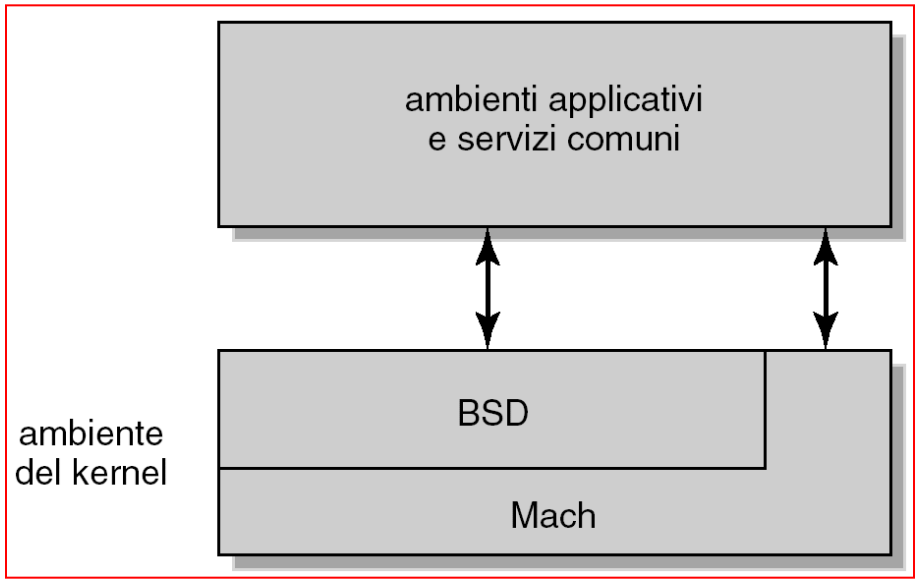


# Struttura a microkernel

- Sposta il maggior numero di funzionalità possibile dal kernel allo *spazio utente* (programmi di sistema e programmi utente).
- La prima implementazione nasce con *Mach* messo a punto alla CMU negli anni ottanta.
- I kernel si riducono in dimensioni e funzionalità. Generalmente essi implementano solo:
  - Gestione dei processi
  - Gestione della memoria
  - Funzioni di comunicazione
- La comunicazione avviene con la tecnica dello scambio di messaggi.
- Benefici:
  - Facilità di estendere il sistema operativo microkernel (è sufficiente aggiungerli allo spazio utente senza modificare il microkernel).
  - Migliore portabilità del S.O..
  - Maggiore affidabilità (se un servizio viene a mancare il complesso del S.O. rimane inalterato).
  - Maggiore sicurezza (meno codice viene eseguito in kernel mode).
- Difetti:
  - I microkernel possono soffrire di un calo di prestazioni dovuto ad un aumento di sovraccarico di funzioni di sistema.

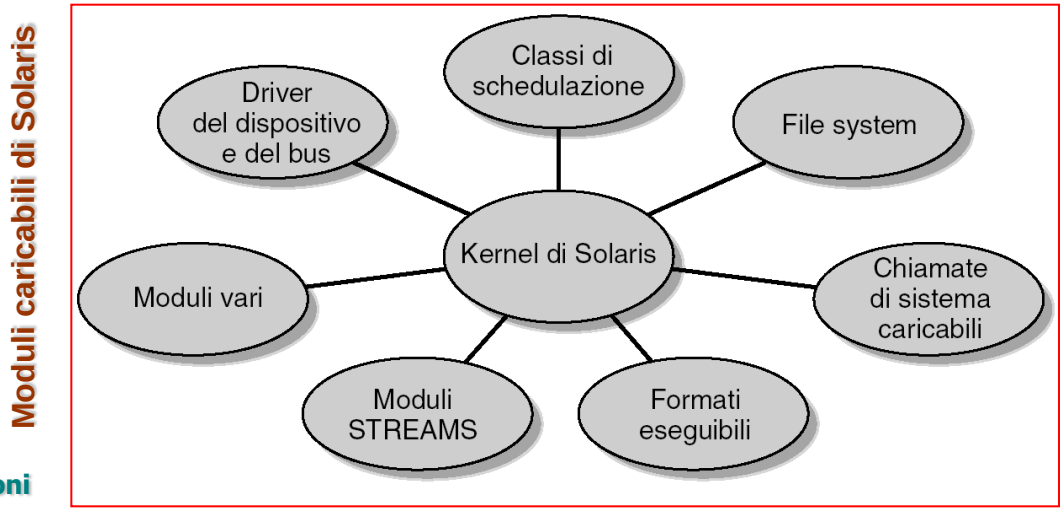
# Struttura ibrida

- Consiste in una suddivisione layerizzata in cui uno degli strati è un microkernel.
- Apple Macintosh OSX ne è un esempio. La struttura prevede la presenza di:
  - Kernel environment (Mach + BSD + supporto driver I/O + estensioni del kernel)
    - Microkernel Mach (gestione memoria, supporto RPC e IPC, schedulazione dei thread)
    - BSD (interfaccia a riga di comando, supporto per la gestione dei file e la rete, implementazione delle API POSIX)
  - Application environment
  - System services
  - Applicazioni e servizi comuni posso far uso diretto sia delle procedure Mach che di quelle BSD



# Struttura modulare

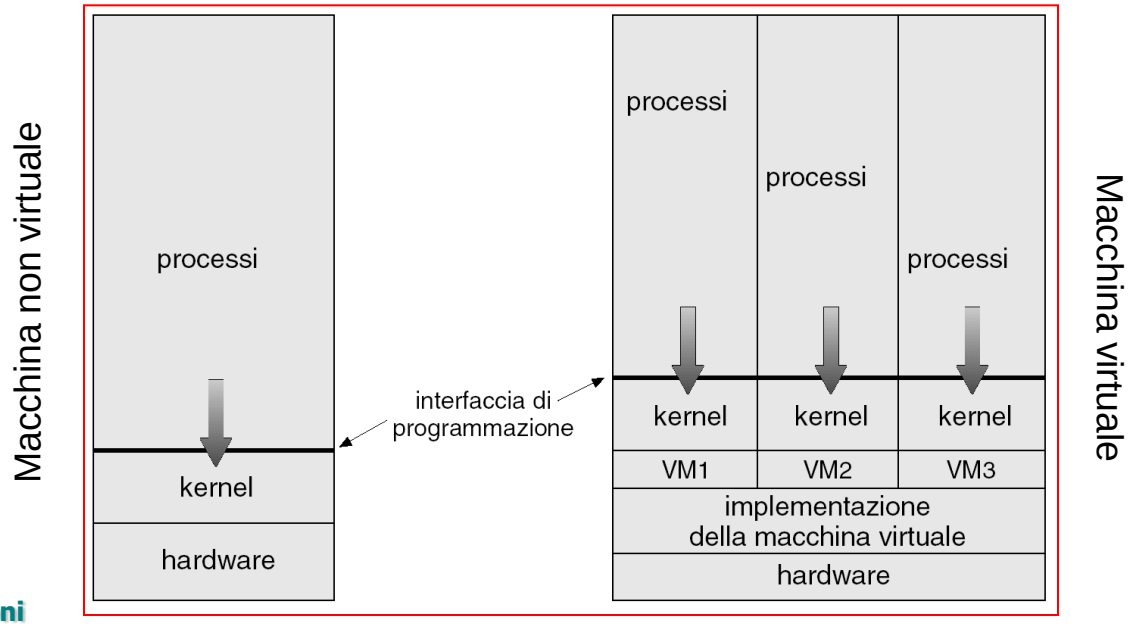
- Coinvolge tecniche di OOP per creare kernel modulari.
- In una struttura modulare il kernel possiede una serie di componenti di base collegandosi dinamicamente ai moduli addizionali in fase di boot o in caso del verificarsi di eventi specifici durante l'esecuzione
  - Si utilizzano moduli caricabili dinamicamente
    - È la tecnica di progettazione alla base delle moderne implementazioni di UNIX come Linux, MAC OS X o Solaris
    - Tutti i driver dei dispositivi vengono caricati come moduli dinamici
  - Analogamente alla struttura a layer ogni modulo può richiamare il kernel e per il suo tramite anche altri moduli
  - Analogamente alla struttura a mcrokernel il modulo primario ha solo funzioni basilari
    - La comunicazione diretta migliora l'efficienza



# Macchine virtuali

- L'approccio a strati trova il suo logico sbocco nel concetto di *macchina virtuale*. Questa tratta l'hardware e il sistema operativo kernel come un complesso di hardware puro.
- Una macchina virtuale fornisce un'interfaccia identica al puro harware sottostante senza altre funzionalità aggiuntive (file system, system call).
- Il sistema operativo crea l'"illusione" che un processo abbia un proprio processore ed una propria memoria (virtuale) adoperando la **schedulazione della CPU** e le tecniche di **memoria virtuale**.

**Modelli di sistema**



# Macchine virtuali

- La macchina “fisica” ripartisce le proprie risorse a beneficio di una o più macchine virtuali.
  - La schedulazione della CPU consente di emulare la presenza di un processore dedicato per ciascuna VM in modalità multiuser.
  - Lo spooling con l’appoggio del file system può fornire lettori di schede e stampanti in linea di tipo virtuale.
  - Un normale terminale time-sharing fornisce la funzione di console all’operatore di una macchina virtuale.
- Il maggiore sforzo implementativo si ha con i dischi. Il sistema ripartisce lo spazio fisico in più spazi logici.
- Il software di una macchina virtuale implementa macchine virtuali multiple su di una singola macchina fisica.
- Il software di una macchina virtuale non tiene conto dei tool supportati dall’utente.

# Macchine virtuali: (s)vantaggi

- Il concetto di macchina virtuale è difficile da implementare perchè è necessario molto lavoro per creare un esatto duplicato della macchina fisica.
- La macchina fisica ha una modalità utente e una di controllo, l'O.S. sulla macchina virtuale supporta la modalità di controllo ma la macchina virtuale può solo funzionare in modalità utente.
  - Deve essere implementata una virtualizzazione della modalità utente e una della modalità controllo.
  - Entrambe le modalità dovranno funzionare in modalità utente sulla macchina fisica.
  - Le azioni che causano un trasferimento di modalità sulla macchina fisica devono ripercuotersi in un trasferimento dalla modalità utente virtuale alla modalità monitor virtuale sulla VM.
- Il trasferimento di modalità avviene nel modo seguente:
  - Modalità utente virtuale -> modalità monitor virtuale (es. SystemCall).
  - Il monitor virtuale cambia il contenuto del PC e dell'IR della VM.
  - La VM eseguirà ora istruzioni privilegiate ma adoperando la modalità utente sulla macchina fisica, quindi il monitor virtuale viene bloccato sulla macchina fisica.
  - In ogni caso l'istruzione privilegiata viene eseguita sulla VM usando (se necessario) meccanismi e dispositivi virtualizzati.
  - Lo stato della VM è stato modificato come se l'istruzione fosse stata eseguita in modalità monitor su una macchina reale.
  - Ovviamente i tempi di esecuzione possono essere molto differenti.

# Macchine virtuali: (s)vantaggi

- Il concetto di macchina virtuale comporta la completa protezione delle risorse di sistema poichè ogni macchina virtuale è isolata da tutte le altre macchine virtuali.
- L'isolamento, tuttavia, non permette la condivisione diretta delle risorse. Si possono ottenere forme di condivisione:
  - Via minidisk (condivisione di file su un volume fisico secondo meccanismi implementati via software).
  - Rete di VM (ogni VM trasmette informazioni attraverso una rete di comunicazioni virtuale modellata via software).
- Una macchina virtuale è uno strumento perfetto per la ricerca e lo sviluppo di sistemi operativi. Lo sviluppo del sistema avviene sulla macchina virtuale anzichè su di una macchina fisica e così le normali operazioni di sistema raramente fanno sì che si debba interrompere lo sviluppo del sistema.

# Progettare sistemi operativi

- Il progetto di un O.S. al più alto livello sarà influenzato dalla scelta dell'HW e del tipo di sistema (batch, time-sharing, mono/multi user, realtime, distribuito…).
- A livello più basso, gli obiettivi progettuali possono essere divisi in:
  - **Obiettivi-Utente:** il sistema operativo dovrebbe essere comodo da usare, facile da imparare ed utilizzare, affidabile, sicuro e veloce.
    - Tali prerogative non possono influenzare gli obiettivi di sistema perchè non c'è accordo generale su come realizzarli.
  - **Obiettivi-sistema:** il sistema dovrebbe essere facile da progettare, implementare e manutenere; dovrebbe essere flessibile, affidabile, esente da errori ed efficiente.
    - I requisiti di sistema possono essere soggetti ad interpretazione e non risultare univoci.
- Fornire le specifiche e progettare un O.S. è una operazione altamente creativa. Nell'ambito dell'ingegneria del software, tuttavia, sono stati sviluppati le linee guida generali per un tale genere di attività.

# Meccanismi e politiche

- Principio della separazione della *politica* dal *meccanismo*.
- I meccanismi determinano *come* fare qualcosa, le politiche determinano *che cosa* sarà fatto.
- La separazione tra politica e meccanismo è un principio molto importante; questo permette la massima flessibilità.
  - Le politiche sono soggette a variazione da caso a caso, i meccanismi devono mantenersi auspicabilmente inalterati.
  - Gli O.S. a microkernel portano ai massimi livelli la separazione tra policy e meccanismo implementando un set di primitive di base pressoché indipendenti che permettono di aggiungere meccanismi e politiche più avanzati tramite moduli utente o moduli kernel.
  - Microsoft Windows è un esempio di meccanismo e policy altamente integrati per fornire all'utente una sensazione "globale". Tutte le applicazioni hanno interfacce simili perchè l'interfaccia stessa è parte del kernel.

# Implementare sistemi operativi

- Tradizionalmente scritti in linguaggio assembler, i sistemi operativi oggi sono generalmente scritti in linguaggi ad alto livello.
- Un codice scritto in un linguaggio ad alto livello:
  - può essere scritto più velocemente;
  - è più compatto;
  - è più leggibile e consente un più agevole debugging;
  - i miglioramenti nella tecnologia del compilatore portano ad un indiretto avanzamento del codice dell'intero O.S. mediante semplice ricompilazione.
- Un sistema operativo è molto più portabile se è scritto in un linguaggio ad alto livello.
- Linux e Windows XP sono perlopiù scritti in C con integrazioni in assembler principalmente relative ai meccanismi di context switching ed ai driver dei dispositivi.

# Implementare sistemi operativi

- Svantaggi dell'implementazione di un O.S. mediante linguaggi ad alto livello sono la ridotta velocità e le maggiori esigenze in termini di memoria.
- In ogni caso un compilatore produce sul codice di un O.S. scritto in un linguaggio ad alto livello:
  - ottimizzazioni più efficaci di quelle ottenibili da una collezione di piccole ed efficienti procedure in assembler;
  - una migliore gestione delle complesse dipendenze dei dati (indispensabile nei processori moderni che gestiscono una coda di acquisizione delle istruzioni molto profonda);
  - una migliore gestione di unità funzionali multiple.
- È più probabile ottenere miglioramenti delle performance come risultato di più efficienti algortimi e strutture dati piuttosto che di un perfezionamento di codice nativo.
- Anche in O.S. significativamente grandi il codice critico dal punto di vista prestazionale generalmente riguarda il memory manager ed il CPU scheduler. Identificato un bottleneck nelle performance si può dunque sostituire il relativo codice con procedure in assembler.
- La valutazione delle performance è ottenibile mediante tool esterni all'O.S. che processano un file di log generato dal sistema operativo stesso.
  - L'alternativa è nella visualizzazione in tempo reale delle prestazioni in forma statistica a carico del O.S. ma ciò può indurre a falsare lievemente le valutazioni.

# Generazione del sistema operativo

- I sistemi operativi sono progettati per funzionare su di una determinata classe di macchine; il sistema deve essere configurato per ogni sua collocazione specifica.
- Il programma SYSGEN recupera le informazioni riguardanti la configurazione specifica dell'hardware.
  - SYSGEN raccoglie le informazioni sulla specifica configurazione dell'HW con l'interazione con l'utente oppure attraverso una fase di test diretto sull'HW. Vengono ricavate le seguent informazioni:
    - CPU e relative opzioni di utilizzo.
    - Memoria (tipologia e dimensioni).
    - Periferiche (porta, interrupt number, tipo, modello e caratteristiche).
    - Opzioni dell'O.S. (numero e dimensioni dei buffer, algoritmi di schedulazione di CPU e accessi a disco, livello di multiprocessing).
- Le informazioni raccolte in fase preliminare vengono adoperate per la selezione di moduli da librerie precompilate che poi –collegàti insieme– costituiscono il sistema operativo oggetto.
  - In tal modo solo i moduli strettamente necessari sono inclusi nell'O.S. installato.

# Generazione del sistema operativo

- La selezione dei moduli può anche avvenire a runtime in fase di esecuzione piuttosto che di compilazione o linking.
  - In questo caso le dimensioni dell'O.S. sono notevolmente superiori ma risulta essere più efficace la capacità di adattamento alle modifiche HW.
- **Boot**: procedura di start-up di una macchina.
  - **Bootstrap program**: codice contenuto nella memoria a sola lettura (ROM) in grado di individuare il kernel, caricarlo nella memoria centrale e iniziarne l'esecuzione.
  - **Boot loader:** è un semplice caricatore di avvio generalmente installato nel MBR, il quale recupera dal disco il bootstrap program di un dato O.S. che a sua volta carica il kernel (caricamento a due passi).