Fondamenti dei Sistemi Operativi

Architettura di un SO

Sommario

- Progettazione dei sistemi operativi.
- Flessibilità dei sistemi operativi: Meccanismi e Politiche.
- Sviluppo dei sistemi operativi.
- Configurazione (Generazione) del SO.
- Il bootstrap.
- Tipi di architetture di un SO.
  - Architettura monolitica
  - Architettura a livelli (o stratificata): UNIX
  - Architettura a macchina virtuale
  - Architettura a microkernel (*client-server*)
- Modello generale di SO con architettura stratificata.
  - CPU & process management
  - Gestione della memoria centrale
  - Gestione della memoria secondaria: File Management
  - Gestione dei dispositivi e dell'I/O
- Interprete dei comandi.

Progettazione dei 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, real-time, distribuito…).

A livello più basso, gli obiettivi progettuali possono essere divisi in:

Obiettivi-Utente: il sistema operativo dovrebbe essere comodo da usare, facile da apprendere ed utilizzare, affidabile, sicuro e veloce.

Obiettivi-sistema: il sistema dovrebbe essere facile da progettare, implementare e manutenere; dovrebbe essere flessibile, affidabile, esente da errori ed efficiente.

Fornire le specifiche e progettare un O.S. è una operazione altamente creativa.

Per aumentare la flessibilità del kernel del SO: Meccanismi e Politiche

I kernel tradizionali (monolitici) sono poco flessibili.

Il 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, nel caso in cui le politiche debbano essere cambiate.

Ad esempio:

- ***assegnare*** l'esecuzione ad un processo è un ***meccanismo***;
- ***scegliere*** quale processo attivare è una ***politica***.

Le politiche sono soggette a variazione da caso a caso, i meccanismi devono mantenersi auspicabilmente inalterati.

Meccanismi e Politiche

Gli O.S. a microkernel, come vedremo, 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.

Estremizzazione:
il ***kernel fornisce solo i meccanismi***, mentre le ***politiche vengono implementate in user space***.

Microsoft Windows, invece, 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.

Sviluppo dei sistemi operativi

- Tradizionalmente scritti in linguaggio assembler, i sistemi operativi oggi sono generalmente scritti in linguaggi di alto livello.
- Un codice scritto in un linguaggio di alto livello:
  - può essere scritto e modificato 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 di alto livello.
- Linux e Windows (da XP in poi) sono per lo più scritti in C con integrazioni in assembler principalmente relative ai meccanismi di context switching ed ai driver dei dispositivi.

Configurazione (Generazione) del SO

- 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 attraverso l'interazione con l'utente oppure attraverso una fase di test diretto sull'HW. Vengono ricavate le seguenti 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 – collegati insieme – costituiscono il sistema operativo oggetto.
  - In tal modo solo i moduli strettamente necessari sono inclusi nell'O.S. installato.

All'accensione del calcolatore: il bootstrap

Il programma iniziale (***bootstrap program***), memorizzato nella ROM, inizializza il sistema (registri, controllori e memoria).

Poi legge da una posizione fissa del disco (*boot block*) un frammento di codice che viene portato in memoria.

Tale codice conosce l'indirizzo di disco (Master Boot Record o MBR) da cui caricare il resto del programma di avvio (che può prevedere di consentire di scegliere il SO da avviare).

Viene quindi iniziato il caricamento del *kernel* (nucleo) del SO, a cui segue la partenza del primo processo (*initiator*), il quale rimane in attesa del verificarsi di un evento.

Un evento è usualmente segnalato da un ***interrupt*** provocato dall'hardware o dal software.

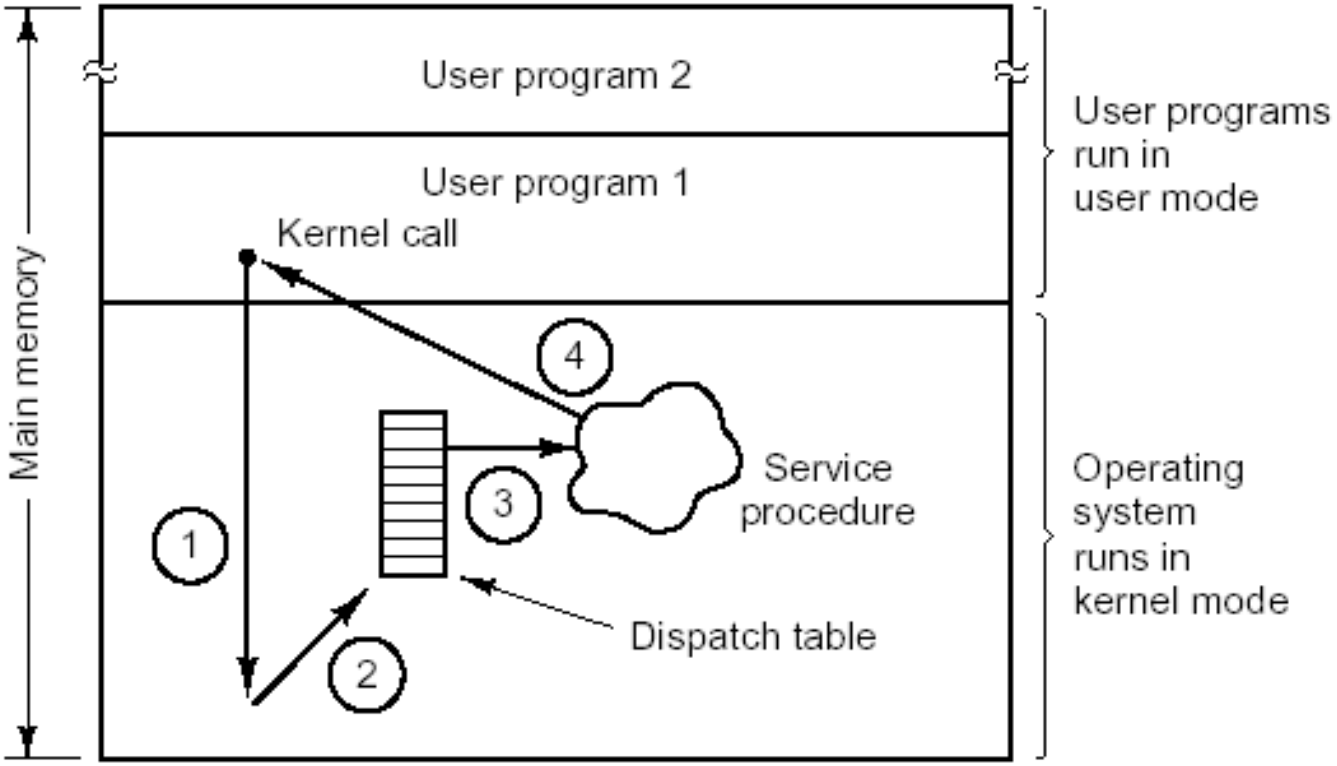
I moderni SO sono guidati dalle interruzioni (*interrupt driven*).

Tipi di architetture di un SO

L'architettura del S.O. può essere

- ✓ **monolitica**, quando il SO è composto da un unico modulo che serve le richieste dei programmi-utente una alla volta;
- ✓ **a macchina virtuale**, se esso offre a ciascun programma-utente visibilità di un particolare hardware;
- ✓ **client-server,** se esso prevede 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 esso è articolato in diversi moduli, ciascuno dei quali svolge specifiche funzioni, ed ogni modulo può servire le richieste di più programmi-utente.

Architettura monolitica di un O.S.

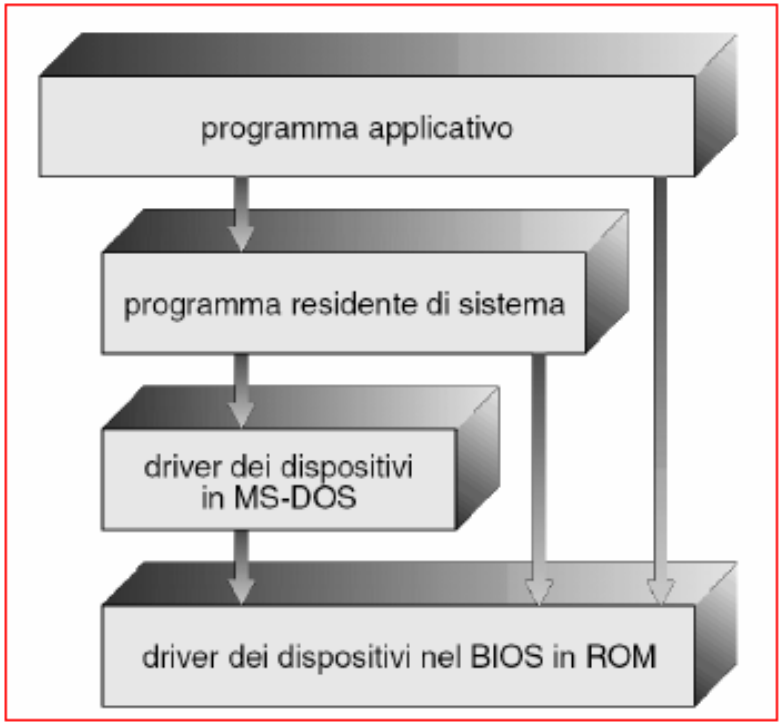


How a system call can be made:

(1) User program traps to the kernel.
(2) Operating system determines service number required.
(3) Operating system calls service procedure.
(4) Control is returned to user program.

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 e nasce per il processore Intel x8088 che non prevedeva meccanismi di protezione HW.



Architettura 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.

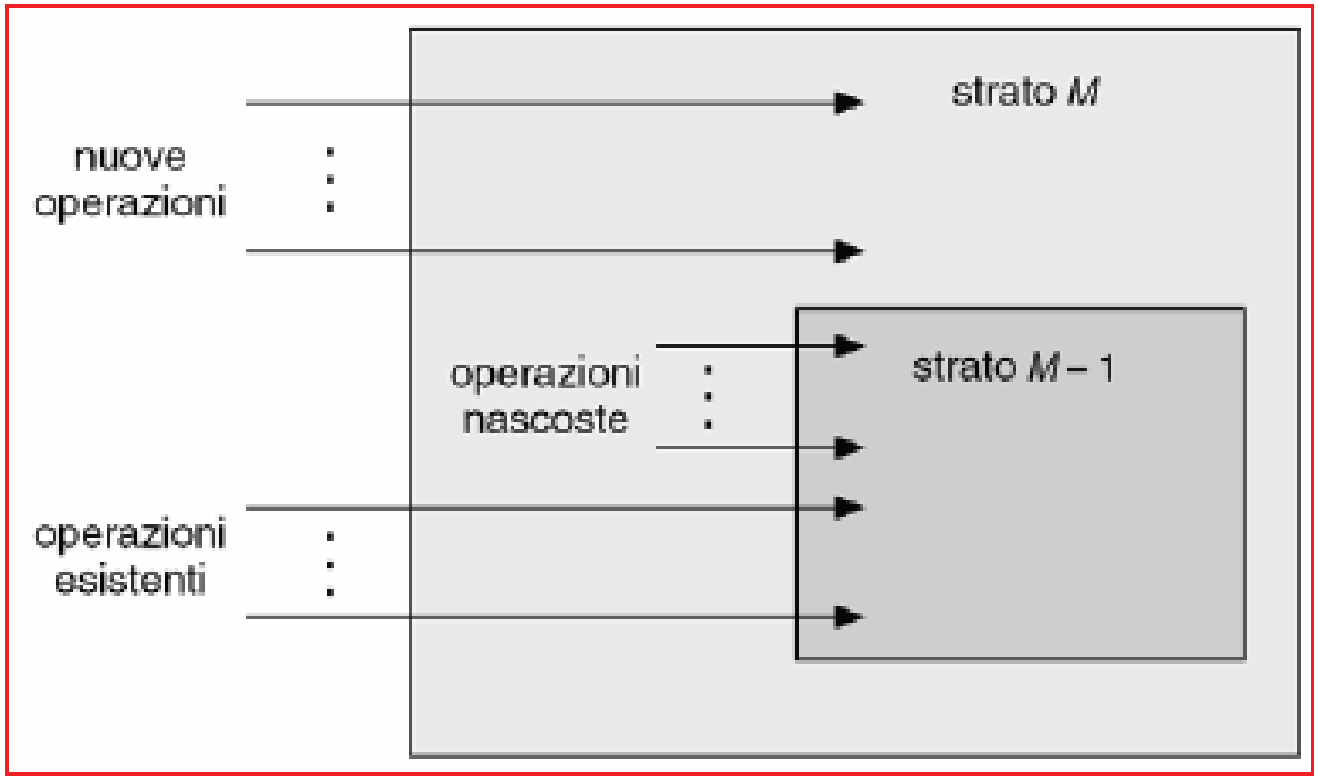
Uno strato del S.O. è un oggetto astratto costituito da un insieme di strutture dati e funzionalità atte ad intervenire su di esse.

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).

La principale difficoltà di approccio è nell'esatta definizione dei vari livelli.

Architettura stratificata

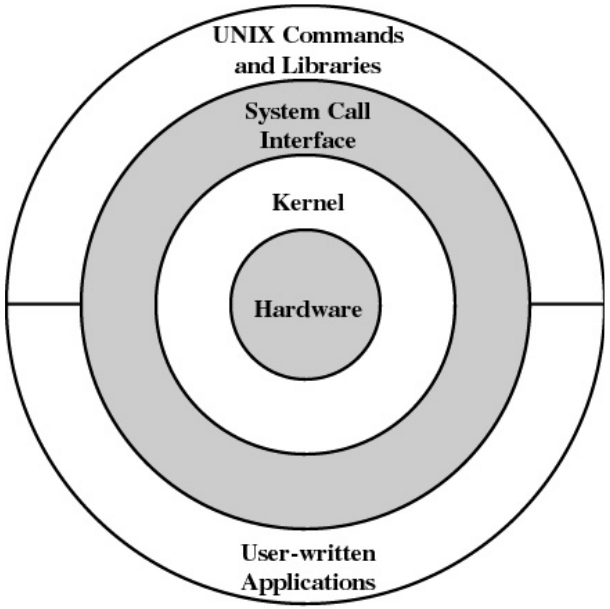


Struttura di un generico strato

Ciascuno strato nasconde i suoi dettagli implementativi agli strati superiori che esclusivamente ne conoscono e adoperano le funzioni.

L'architettura a strati di UNIX

- UNIX: limitato dalle funzionalità hardware, l'originario sistema operativo aveva una struttura elementare. Consisteva in:
  - Programmi di sistema
  - Nucleo(*kernel*)
    - Consiste in tutto ciò che 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.



(Gli utenti)		
Shell e comandi Compilatori e interpreti Librerie di sistema		
Interfaccia della chiamata di sistema al kernel		
Gestione dei segnali dei terminali Driver dei terminali a caratteri	File system Sistema di gestione dello swapping Driver del disco e dei nastri	Schedulazione della CPU Rimpiazzamento di pagine Richiesta di paginazione Memoria virtuale
Interfaccia del kernel verso l'hardware		
Controllori dei terminali Terminali	Controllori dei dispositivi Dischi e nastri	Controllori della memoria Memoria fisica

L'architettura a macchina virtuale 1/2

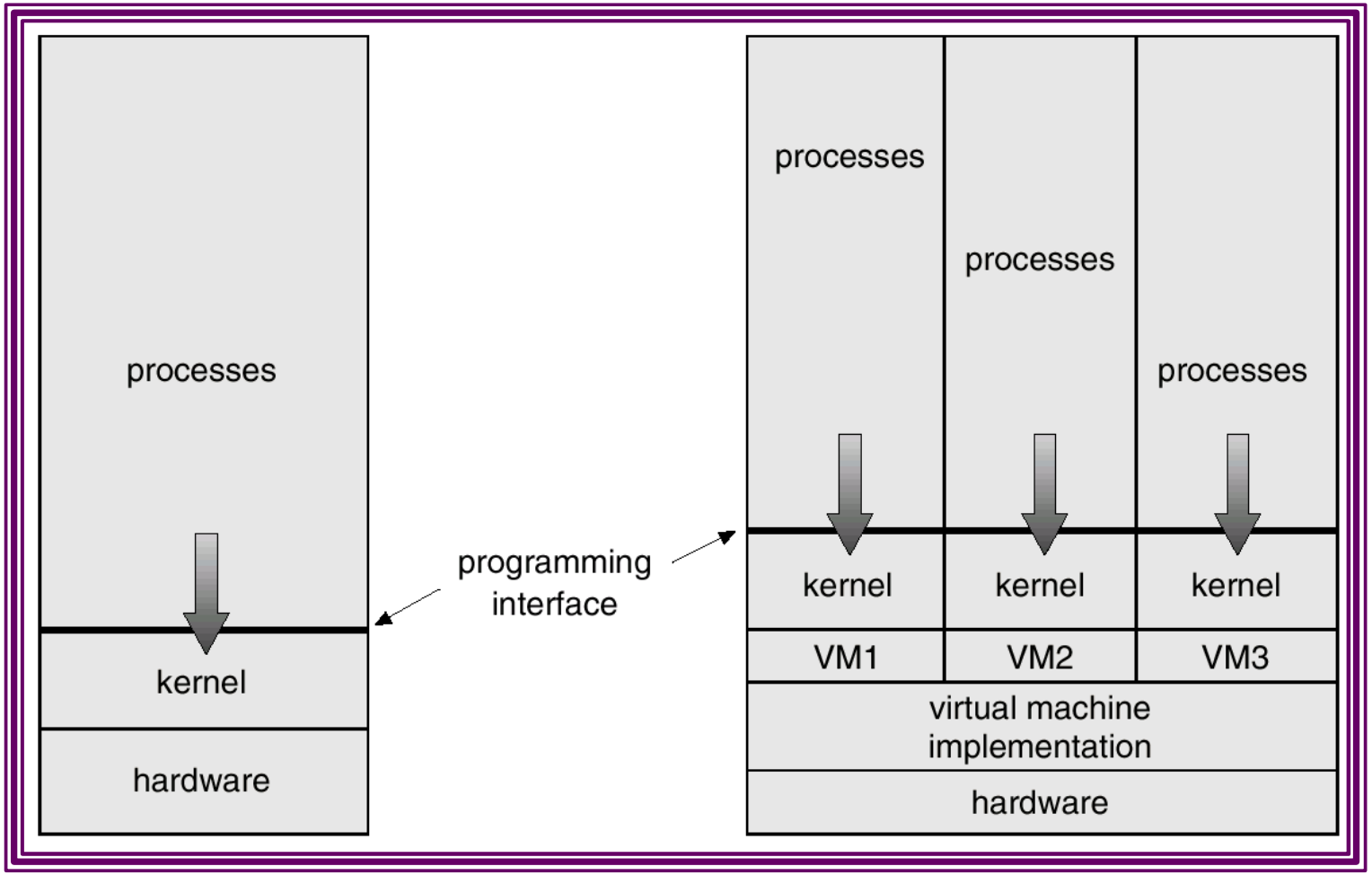
L'approccio a strati trova il suo logico sbocco nel concetto di *macchina virtuale*. Questa tratta l'hardware e il kernel del sistema operativo come un complesso di hardware puro.

Una macchina virtuale fornisce un'interfaccia identica al puro hardware 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.

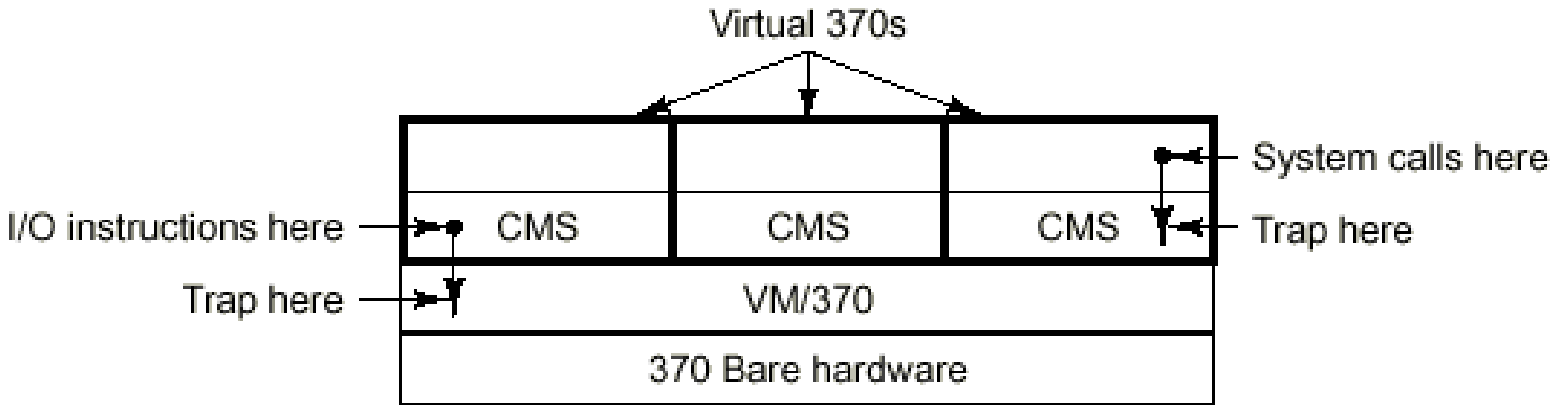
Quando un processo effettua una *system call*, questa viene tradotta in una *trap* al kernel della particolare VM_i, come se la trap debba essere eseguita sulla macchina reale. Le istruzioni della *trap* vengono poi tradotte in istruzioni di I/O verso il kernel operante effettivamente (*virtual machine implementation*) sull'hardware sottostante.

L'architettura a macchina virtuale 2/2



Pro&Contro dell'architettura VM

- Il concetto di macchina virtuale comporta la completa protezione delle risorse di sistema poichè ogni macchina virtuale è isolata da tutte le altre.
- L'isolamento, d'altra parte, non permette la condivisione diretta delle risorse.
- Una macchina virtuale è uno strumento perfetto per la ricerca e lo sviluppo di sistemi operativi. Lo sviluppo del sistema avviene sulla macchina virtuale anziché su di una macchina fisica, senza interrompere la normale operatività del sistema.



The structure of VM/370 with CMS.

Architettura a microkernel

- Il *kernel è ridotto all'osso*, *fornisce soltanto i meccanismi*:
  Sposta il maggior numero di funzionalità dal kernel allo *spazio utente* (programmi di sistema e programmi utente): ad esempio, tutte le politiche di gestione del file system, dello scheduling, della memoria sono implementate come processi.
- I kernel si riducono in dimensioni e funzionalità. Generalmente essi implementano solo:
  - Un *meccanismo di comunicazione* tra processi
  - Una *minima gestione* della memoria e dei processi
  - Gestione dell'hardware di basso livello (*driver*)
- La comunicazione avviene con la tecnica dello scambio di messaggi.

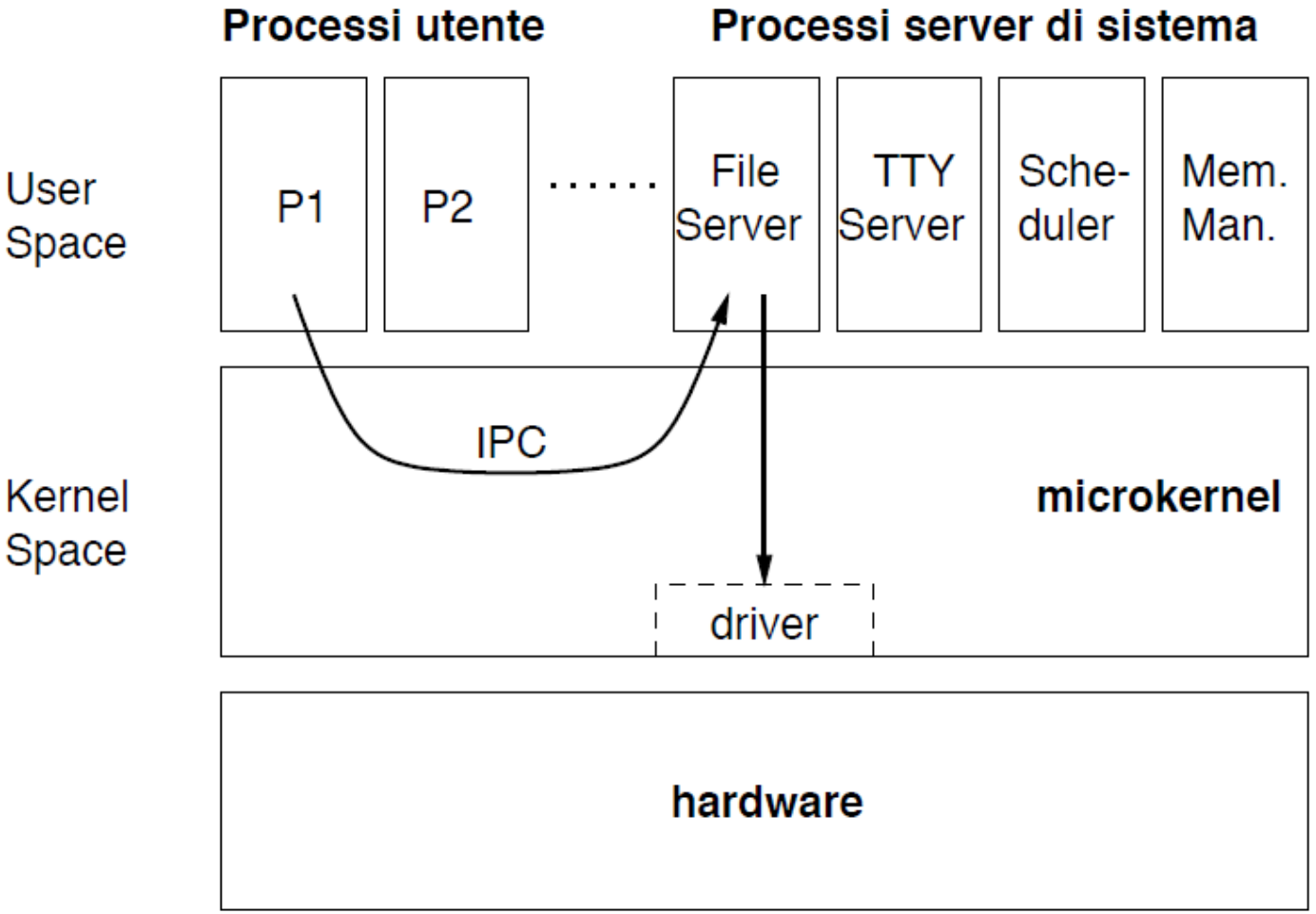
Benefici:

- Facilità di estendere il sistema operativo microkernel (è sufficiente aggiungerli allo spazio utente senza modificare il microkernel).
- Migliore portabilità del S.O.. Meno codice viene eseguito in kernel mode.
- Maggiore affidabilità (se un servizio viene a mancare, il complesso del S.O. rimane inalterato). Immediata scalabilità in ambiente di rete.

Difetti:

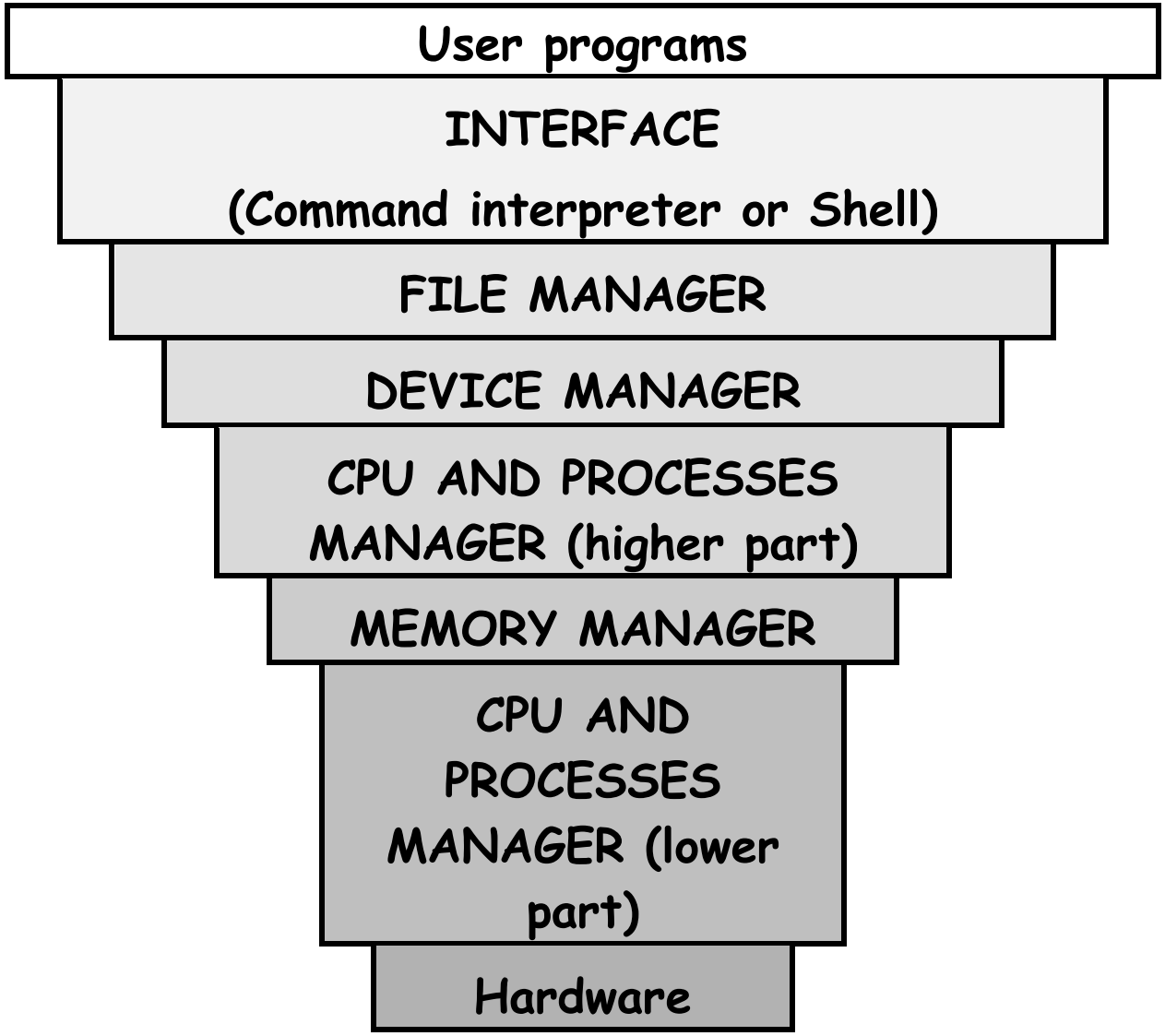
- I microkernel possono soffrire di un calo di prestazioni dovuto ad un aumento di sovraccarico di funzioni di sistema.

Microkernel: Funzionamento di base



IPC → Inter Process Communication facility

Modello generale di SO con architettura a strati



Modello generale di SO con architettura a strati

L'Interfaccia comprende il Job Scheduler.

I livelli tra l'Interfaccia e l'Hardware costituiscono il **Nucleo** o **kernel**.

I moduli di un livello possono utilizzare moduli dei livelli sottostanti mediante chiamate al Supervisore o ***primitive*** sincrone (***trap*** o ***interrupt interni)***.

Ogni livello garantisce la gestione (*management*) di una risorsa e comprende tutti i moduli che contribuiscono a svolgere tale compito (*resource manager*).

I *resource manager* sono:

- il gestore dei ***file***
- il gestore dei ***dispositivi***
- il gestore della ***CPU*** e dei ***processi***
- il gestore della ***memoria centrale***

Il Manager della CPU e dei processi (o Process Scheduler) è suddiviso in due parti:

- la parte alta comprende i moduli per la ***creazione/distruzione di processi*** e per la ***comunicazione tra processi***;
- la parte bassa comprende i moduli per la ***sincronizzazione di processi***.

CPU & process management

Il SO è responsabile delle seguenti attività connesse con la gestione della CPU:

- tener traccia di quale processo stia correntemente usando la CPU, quali siano i processi in attesa di poterla utilizzare e quali, infine, siano in attesa di altri eventi (in genere termine di operazioni di I/O o disponibilità di risorse);
- decidere a quale processo dare priorità tra quelli in attesa di poterla usare;
- allocare e disallocare la CPU secondo la priorità dei processi.

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,
  - evitare, prevenire e risolvere situazioni di **stallo** (deadlock)

Gestione della memoria centrale

- La *memoria 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 della memoria secondaria

- 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.

File Management

- 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 dei dispositivi e dell'I/O

Il sistema di I/O consiste in:

- Una componente per la gestione dei vari buffer e della cache.
- Una componente per la gestione del processo di stampa.
- Un'interfaccia generale tra dispositivo e driver.
- Un driver per ciascun dispositivo hardware specifico (memorie secondarie, printer, tablet, ecc.).

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.

Interprete dei comandi

Molti ordini sono impartiti al SO mediante *istruzioni di controllo* (***interfaccia***) che 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 (*batch* o *script*).