

Architettura dei sistemi operativi

Roberto De Virgilio

Sistemi operativi - 14 Novembre 2017

Struttura dei s.o.

- **Architettura di un sistema operativo**
 - descrive quali sono le varie componenti del s.o. e come queste sono collegate fra loro
 - i vari sistemi operativi sono molto diversi l'uno dall'altro nella loro architettura
- **Abbiamo già visto quali sono le componenti principali**
 - Gestione dei processi
 - Gestione memoria principale
 - Gestione memoria secondaria
 - Gestione file system
 - Gestione dei dispositivi di I/O
 - Protezione
 - Networking
 - Interprete dei comandi
- **Vediamo ora come sono collegati tra loro**

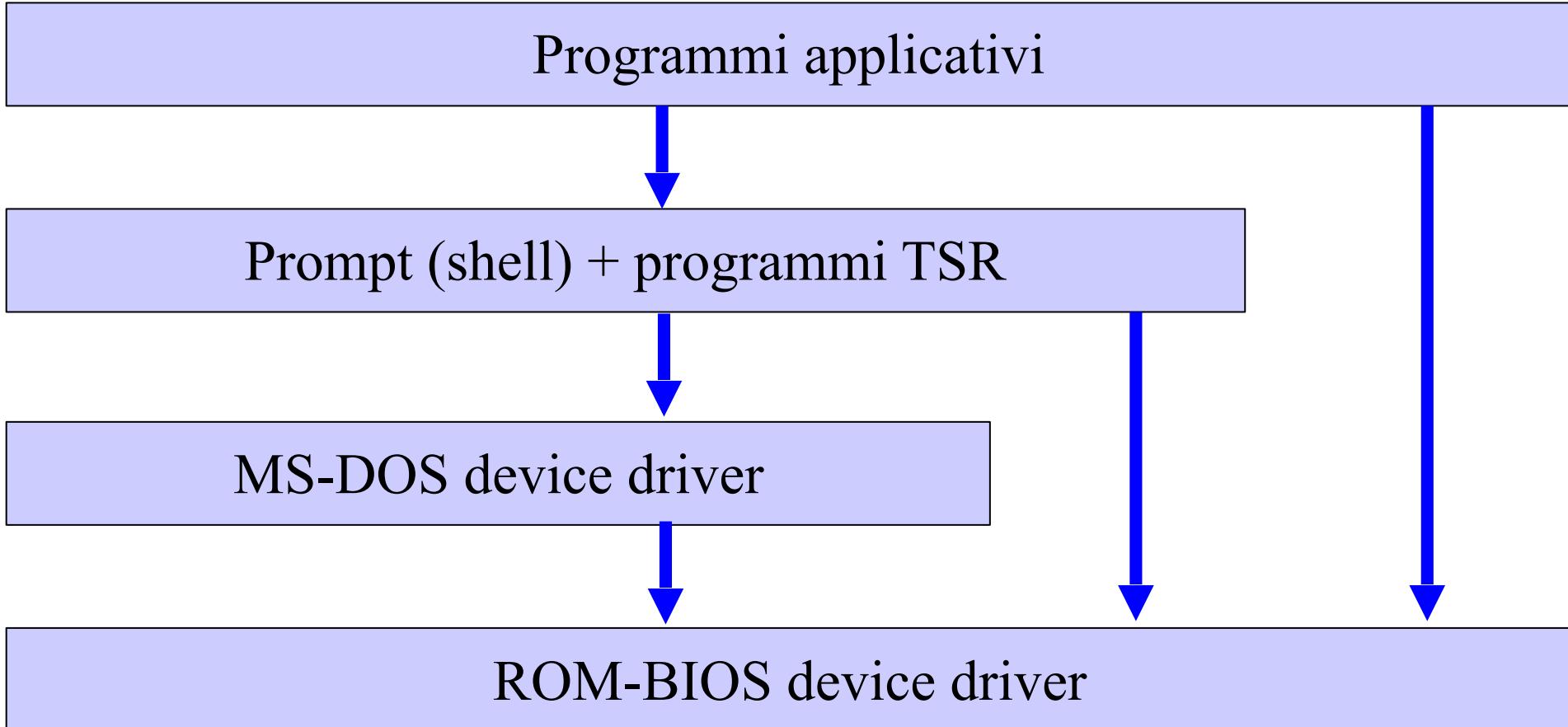
Struttura dei sistemi operativi

- **La progettazione di un s.o. deve tener conto di diverse caratteristiche**
 - efficienza
 - manutenibilità
 - espansibilità
 - modularità
- **Spesso, queste caratteristiche presentano un trade-off:**
 - sistemi molto efficienti sono poco modulari
 - sistemi molto modulari sono meno efficienti

Struttura dei sistemi operativi

- E' possibile suddividere i s.o. in due grandi famiglie, a seconda della loro struttura
 - sistemi con **struttura semplice**
 - sistemi con **struttura a strati**
- **Sistemi con struttura semplice (o senza struttura)**
 - in alcuni casi sono s.o. che non hanno una struttura progettata a priori;
 - possono essere descritti come una collezione di procedure, ognuna delle quali può richiamare altre procedure
 - tipicamente sono s.o **semplici** e **limitati** che hanno subito un'evoluzione al di là dello scopo originario

MS-DOS



MS-DOS

- ◆ **Commenti**

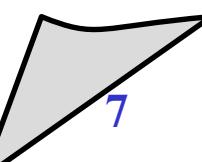
- ◆ le interfacce e i livelli di funzionalità non sono ben separati
 - ◆ le applicazioni possono accedere direttamente alle routine di base per fare I/O
- ◆ come conseguenza, un programma sbagliato (o “maligno”) può mandare in crash l'intero sistema

- ◆ **Motivazioni:**

i progettisti di MS-DOS erano legati all'hardware dell'epoca
8086, 8088, 80286 non avevano la modalità protetta (kernel)

UNIX

- **Anche UNIX è poco strutturato**
- **E' suddiviso in due parti**
 - kernel
 - programmi di sistema
- **Il kernel è delimitato**
 - in basso dall'hardware
 - in alto dal livello delle system call
- **Motivazioni**
 - anche Unix inizialmente fu limitato dalle limitazioni hardware...
 - ... ma ha un approccio comunque più strutturato



UNIX

Utente

Shell e comandi; compilatori e interpreti
Librerie di sistema

Interfaccia system call

Gestione terminali
sistemi I/O caratteri
driver di terminale

File system
Meccanismo di swapping
driver dischi, nastri

Cpu scheduling
page replacement
virtual memory

Interfaccia hardware

Controllori di terminale
Terminali

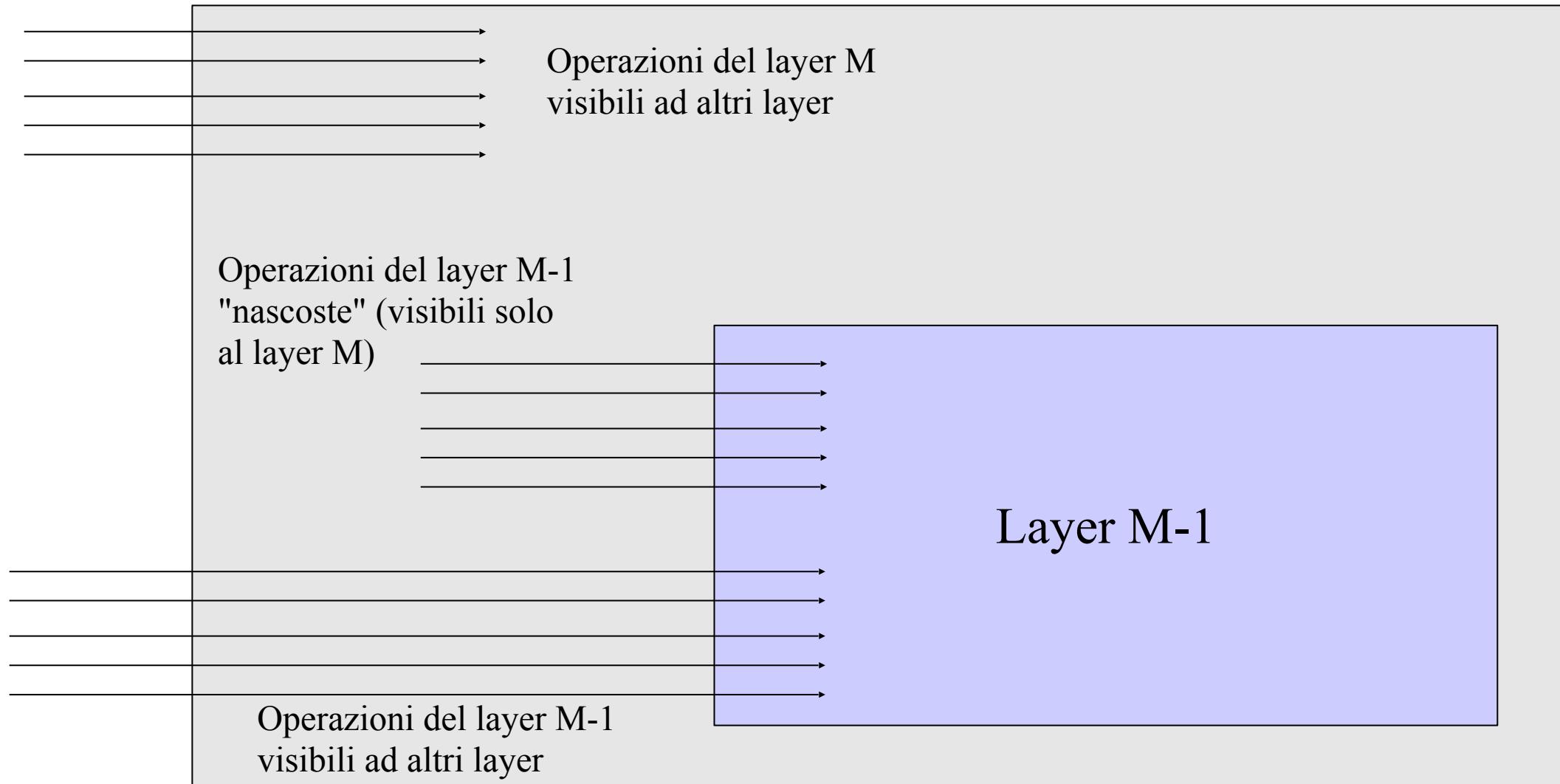
Controllori mem. second.
Dischi e nastri

MMU
Memoria

Sistemi con struttura a strati

- Il s.o. è strutturato tramite un insieme di strati (layer)
- Ogni strato
 - è basato sugli strati inferiori
 - offre servizi agli strati superiori
- Motivazioni
 - il vantaggio principale è la modularità
 - encapsulation e data hiding
 - abstract data types
 - vengono semplificate le fasi di implementazione, debugging ristrutturazione del sistema

Sistemi con struttura a strati



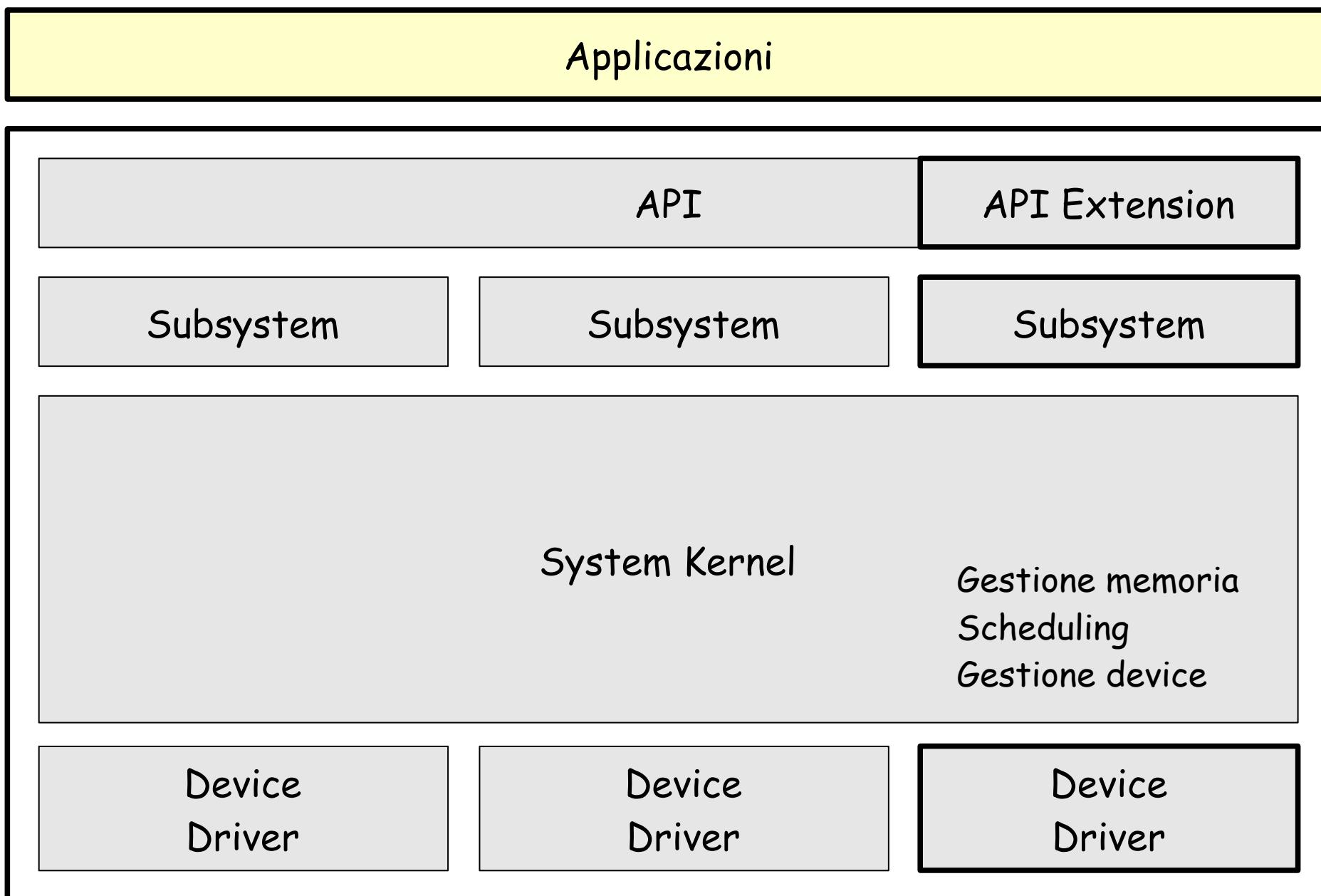
Esempi

- The O.S. (Dijkstra)
 - THE multiprogramming system
- 5) Programmi utente
- 4) Gestione I/O
- 3) Console device/driver
- 2) Memory management
- 1) CPU Scheduling
- 0) Hardware
- Venus OS (Barbara H. Liskov)
 - Experimental multiprogramming system
- 6) Programmi utente
- 5) Device driver e scheduler
- 4) Memoria virtuale
- 3) Canali di I/O
- 2) CPU Scheduling
- 1) Interprete di istruzioni
- 0) Hardware

Sistemi con struttura a strati

- **Problemi dei sistemi con struttura a strati**
 - *tendono a essere meno efficienti*
 - ogni strato tende ad aggiungere **overhead**
 - *occorre studiare accuratamente la struttura dei layer*
 - le funzionalità previste al layer N devono essere implementate utilizzando esclusivamente i servizi dei livelli inferiori
 - in alcuni casi, questa limitazione può essere difficile da superare
 - esempio: meccanismi di swapping di memoria
 - Win 9x: swap area è un file in memoria
 - Linux: swap area ha una partizione dedicata
- **Risultato:**
 - i moderni sistemi con struttura a strati tendono ad avere meno strati

OS/2



Organizzazione del kernel

- **Esistono 4 categorie di Kernel**
 - **Kernel Monolitici**
 - Un aggregato unico (e ricco) di procedure di gestione mutuamente coordinate e astrazioni dell'HW
 - **Micro Kernel**
 - Semplici astrazioni dell'HW gestite e coordinate da un kernel minimale, basate su un paradigma client/server, e primitive di **message passing**
 - **Kernel Ibridi**
 - Simili a Micro Kernel, ma hanno componenti eseguite in **kernel space** per questioni di maggiore efficienza
 - **ExoKernel**
 - Non forniscono livelli di astrazione dell'HW, ma forniscono librerie che mettono a contatto diretto le applicazioni con l'HW

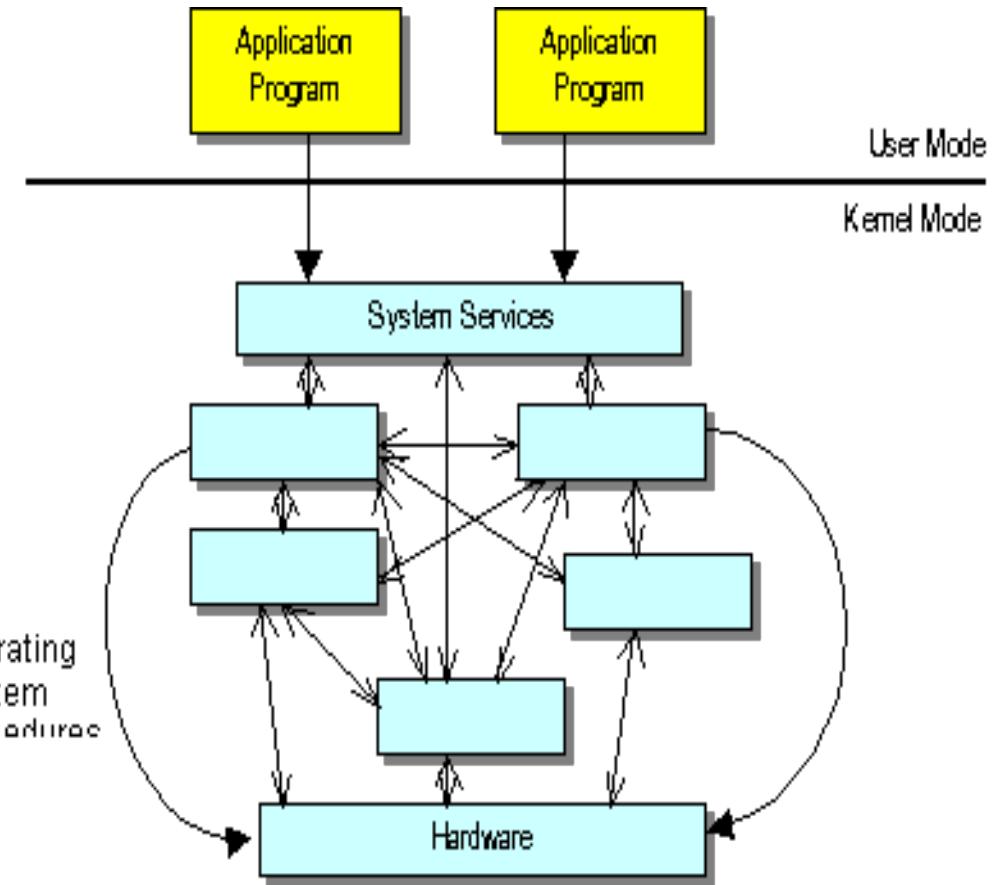
Organizzazione del kernel

- **Kernel Monolitici**

- Un insieme completo e unico di procedure mutuamente correlate e coordinate

- **System calls**

Implementano servizi forniti dal kernel, tipicamente realizzati in moduli eseguiti in kernel mode



Esiste modularità, anche se l'integrazione del codice, e il fatto che tutti i moduli sono eseguiti nello stesso spazio, è tale da rendere tutto l'insieme un corpo unico in esecuzione

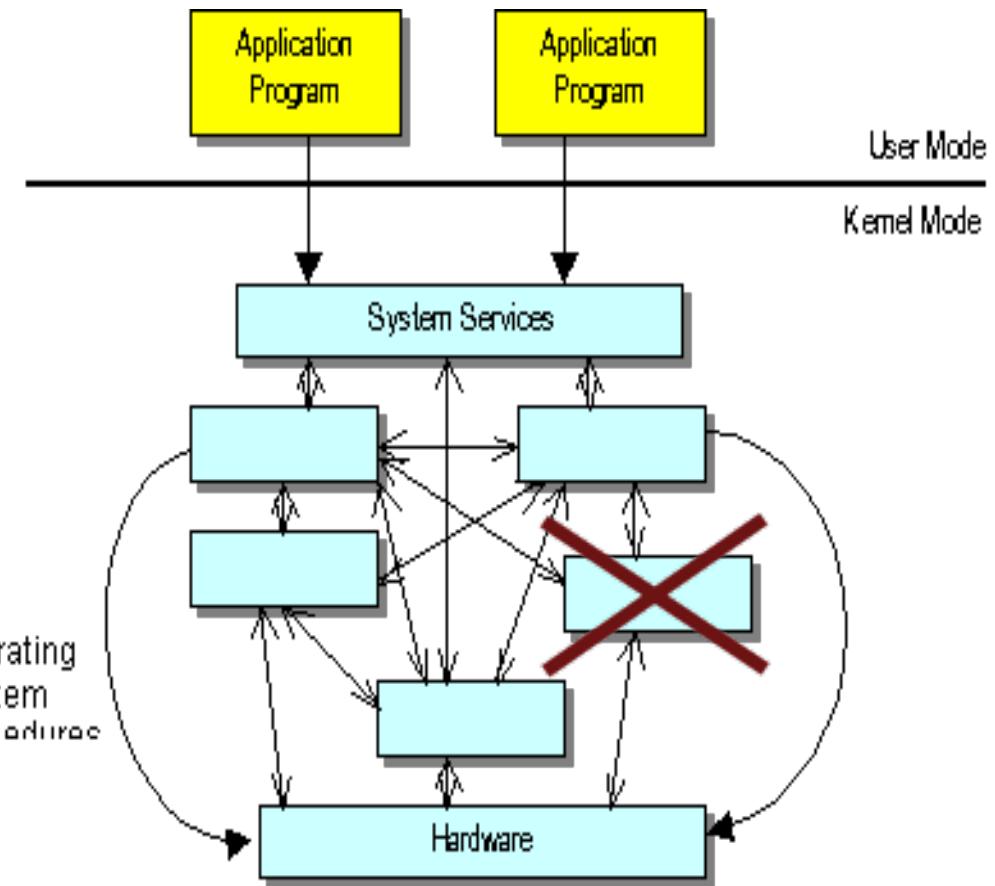
Organizzazione del kernel

- **Kernel Monolitici**

- Un insieme completo e unico di procedure mutuamente correlate e coordinate

- **System calls**

Implementano servizi forniti dal kernel, tipicamente realizzati in moduli eseguiti in kernel mode



Esiste modularità, anche se l'integrazione del codice, e il fatto che tutti i moduli sono eseguiti nello stesso spazio, è tale da rendere tutto l'insieme un corpo unico in esecuzione

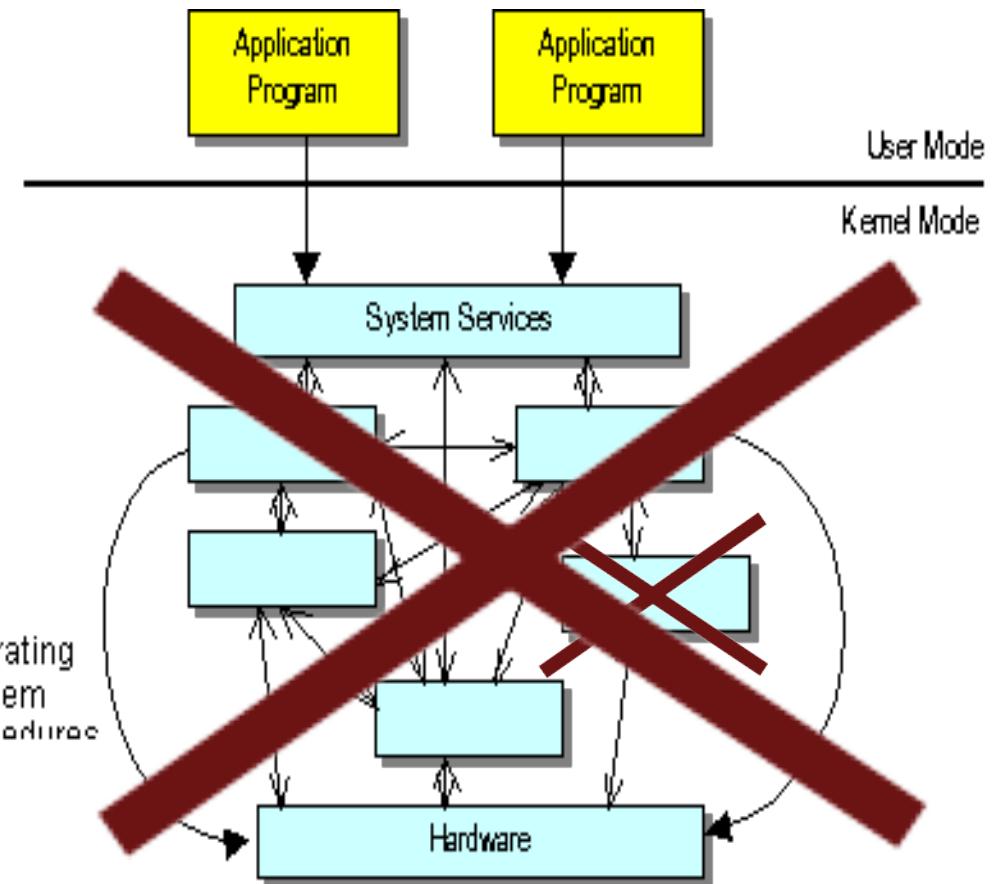
Organizzazione del kernel

- **Kernel Monolitici**

- Un insieme completo e unico di procedure mutuamente correlate e coordinate

- **System calls**

Implementano servizi forniti dal kernel, tipicamente realizzati in moduli eseguiti in kernel mode



Esiste modularità, anche se l'integrazione del codice, e il fatto che tutti i moduli sono eseguiti nello stesso spazio, è tale da rendere tutto l'insieme un corpo unico in esecuzione

Organizzazione del kernel

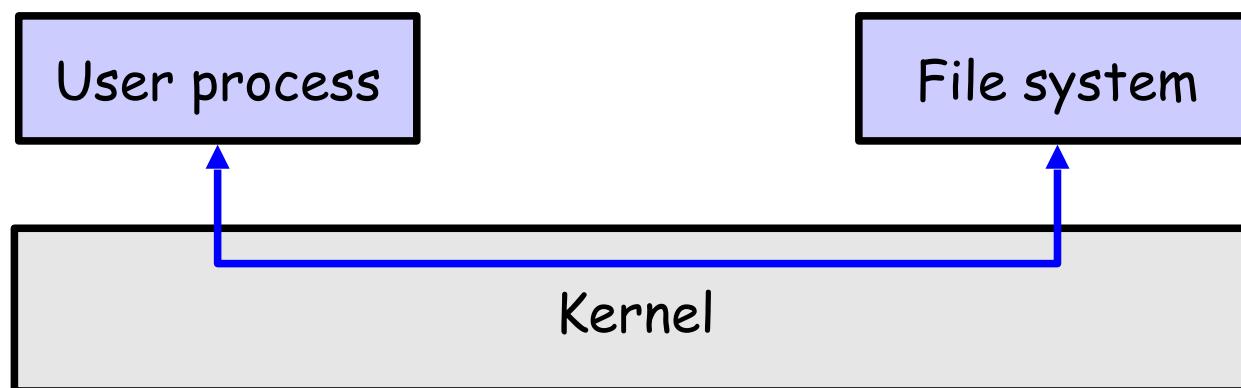
- **Kernel Monolitici**
 - Efficienza
 - L'alto grado di coordinamento e integrazione delle routine permette di raggiungere ottimi livelli di efficienza
 - Modularità
 - I più recenti kernel monolitici (Es. LINUX) permettono di effettuare il caricamento (load) di moduli eseguibili a runtime
 - Possibile estendere le potenzialità del kernel, solo su richiesta
- **Esempi di Kernel monolitici: LINUX, FreeBSD UNIX**

Microkernel o sistemi client/server

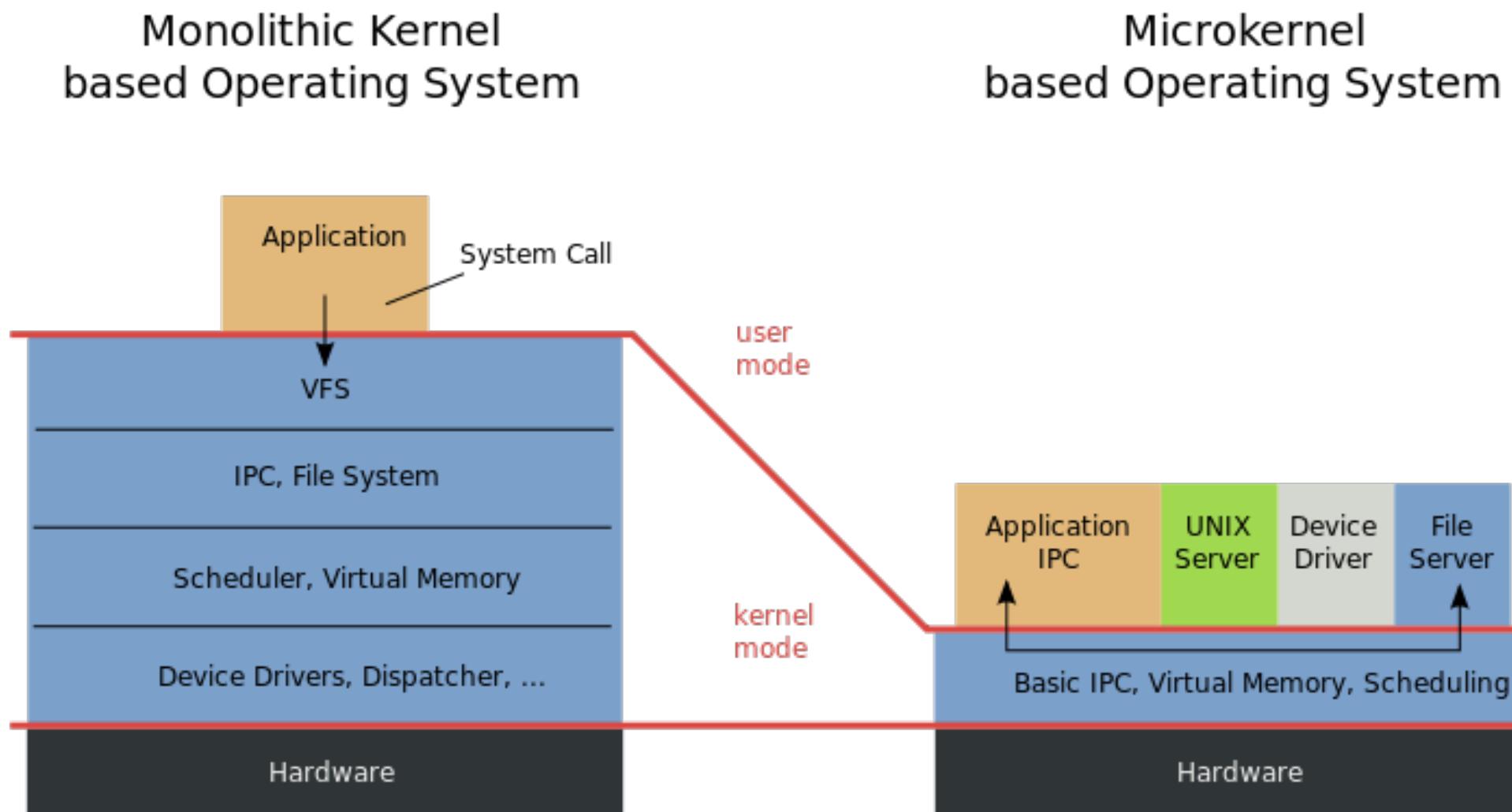
- **Problema**
 - nonostante la struttura a strati, i kernel continuano a crescere in complessità
- **Idea**
 - rimuovere dal kernel tutte le parti non essenziali e implementarle come processi a livello utente
- **Esempio**
 - per accedere ad un file, un processo interagisce con il processo gestore del file system
- **Esempio di sistemi operativi basati su microkernel:**
 - AIX, BeOS, L4, Mach, Minix, MorphOS, QNX, RadiOS, VSTa

Microkernel o sistemi client/server

- **Quali funzionalità deve offrire un microkernel?**
 - funzionalità minime di gestione dei processi e della memoria
 - *meccanismi di comunicazione* per permettere ai processi clienti di chiedere servizi ai processi serventi
- **La comunicazione è basata su *message passing***
 - il microkernel si occupa di smistare i messaggi fra i vari processi



Microkernel o sistemi client/server



Microkernel o sistemi client/server

- System call di un s.o. basato su microkernel
 - **send**
 - **receive**
- Tramite queste due system call, è possibile implementare l'API standard di gran parte dei sistemi operativi

```
int open(char* file, ...)  
{  
    msg = < OPEN, file, ... >;  
    send(msg, file-server);  
    fd = receive(file-server);  
    return fd;  
}
```

Microkernel o sistemi client/server

- **Vantaggi**
 - *il kernel risultante è molto semplice e facile da realizzare*
 - *il kernel è più espandibile e modificabile*
 - per aggiungere un servizio: si aggiunge un processo a livello utente, senza dover ricompilare il kernel
 - per modificare un servizio: si riscrive solo il codice del servizio stesso
 - *il s.o. è più facilmente portabile ad altre architetture*
 - una volta portato il kernel, molti dei servizi (ad es. il file system)
 - ◆ possono essere semplicemente ricompilati
 - *il s.o. è più robusto*

se per esempio il processo che si occupa di un servizio cade, il resto del sistema può continuare ad eseguire

Microkernel o sistemi client/server

- **Vantaggi**
 - *sicurezza*
 - è possibile assegnare al microkernel e ai processi di sistema livelli di sicurezza diversi
 - *adattabilità del modello ai sistemi distribuiti*
 - la comunicazione può avvenire tra processi nello stesso sistema o tra macchine differenti
- **Svantaggi**
 - *maggiori inefficienze*
 - dovuta all'**overhead** determinato dalla comunicazione mediata tramite kernel del sistema operativo
 - parzialmente superata con i sistemi operativi più recenti

Confronto tra kernel monolitici e microkernel

- ♦ **Monolitico**

- Considerato obsoleto nel 1990...
- È meno complesso gestire il codice di controllo in un'unica area di indirizzamento (kernel space)
- È più semplice realizzare la sua progettazione (corretta)

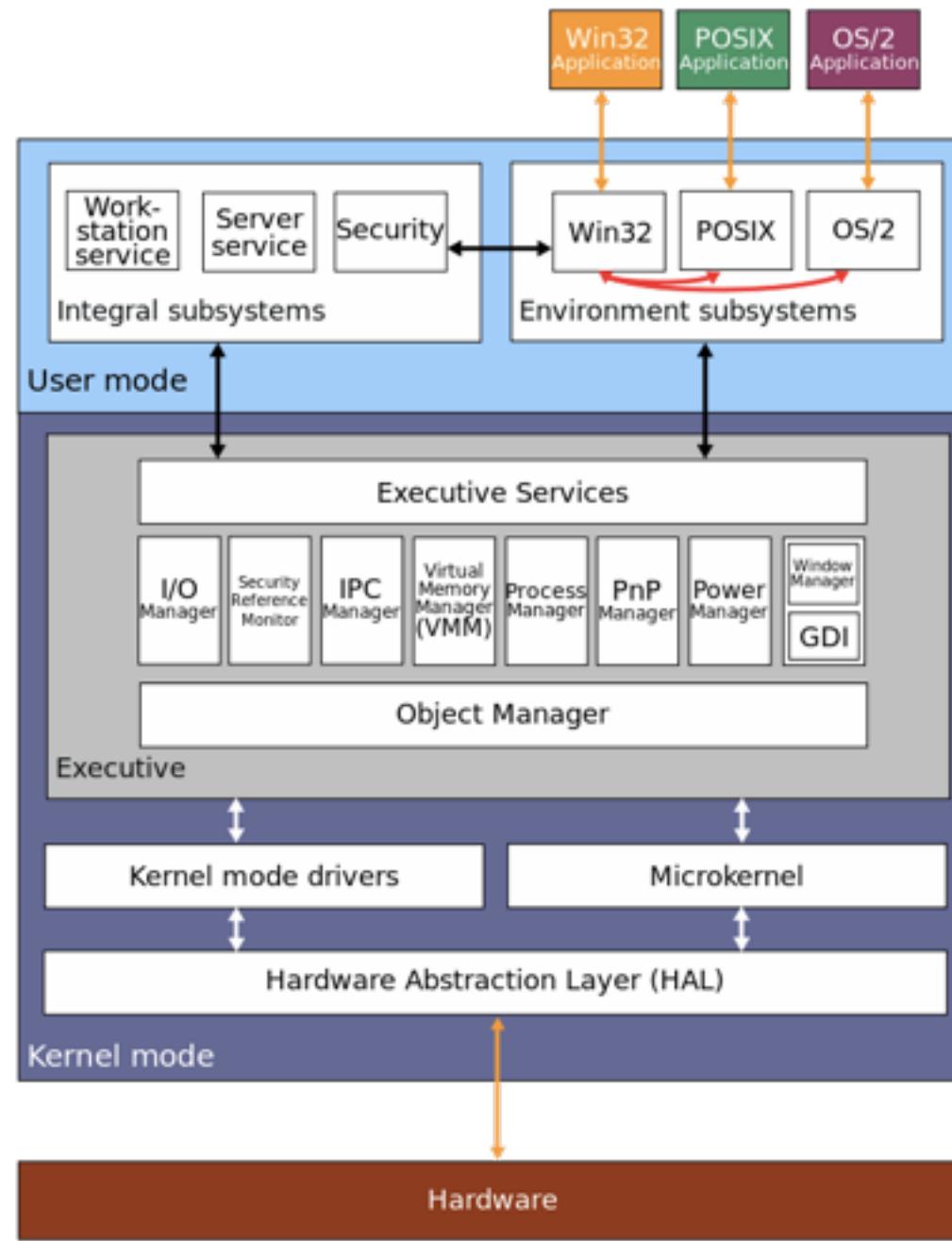
- ♦ **Micro Kernel**

- Più usato in contesti dove non si ammettono failure
- Es. QNX usato per braccio robot Space shuttle

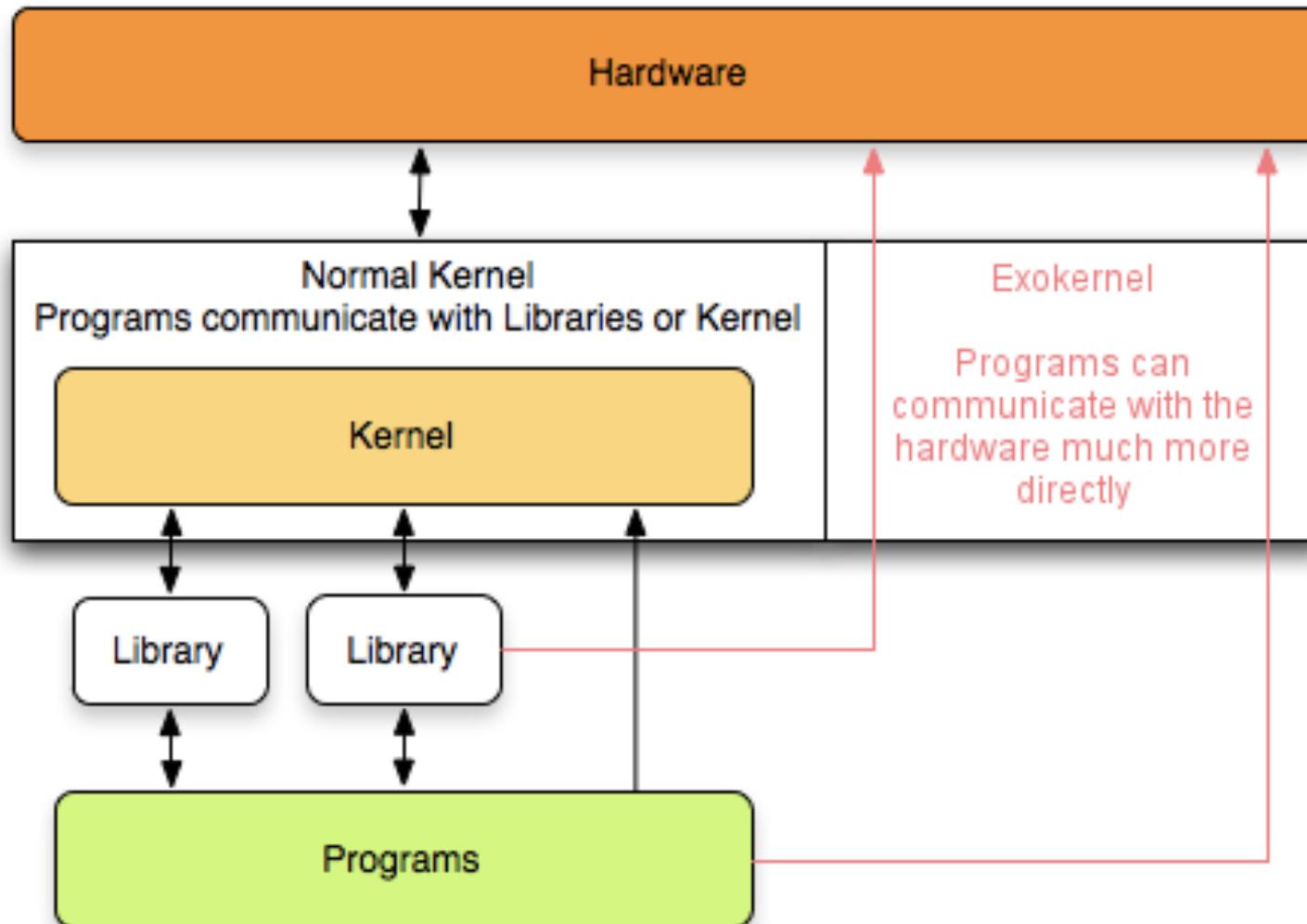
Kernel Ibridi

- **Kernel Ibridi (Micro kernel modificati)**
 - Si tratta di micro kernels che mantengono una parte di codice in “kernel space” per ragioni di maggiore efficienza di esecuzione
 - ...e adottano message passing tra i moduli in **user space**
- **Es. Microsoft Windows NT kernel**

Windows NT 4.0 / 2000



ExoKernel (kernel di sistemi operativi a struttura verticale)



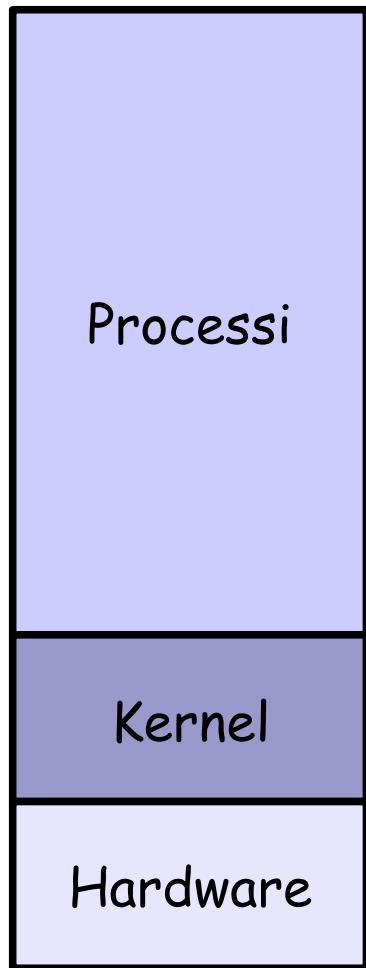
ExoKernel (kernel di sistemi operativi a struttura verticale)

- **Approccio radicalmente modificato per implementare O.S.**
- **Motivazioni**
 - Il progettista dell'applicazione ha tutti gli elementi di controllo per decisioni riguardo alle prestazioni dell'HW
 - Dispone di Libreria di interfacce connesse all'ExoKernel
 - Es. User vuole allocare area di memoria X o settore disco Y
- **Limiti**
 - Tipicamente non vanno oltre l'implementazione dei servizi di protezione e multiplazione delle risorse
 - Non forniscono astrazione concreta del sistema HW

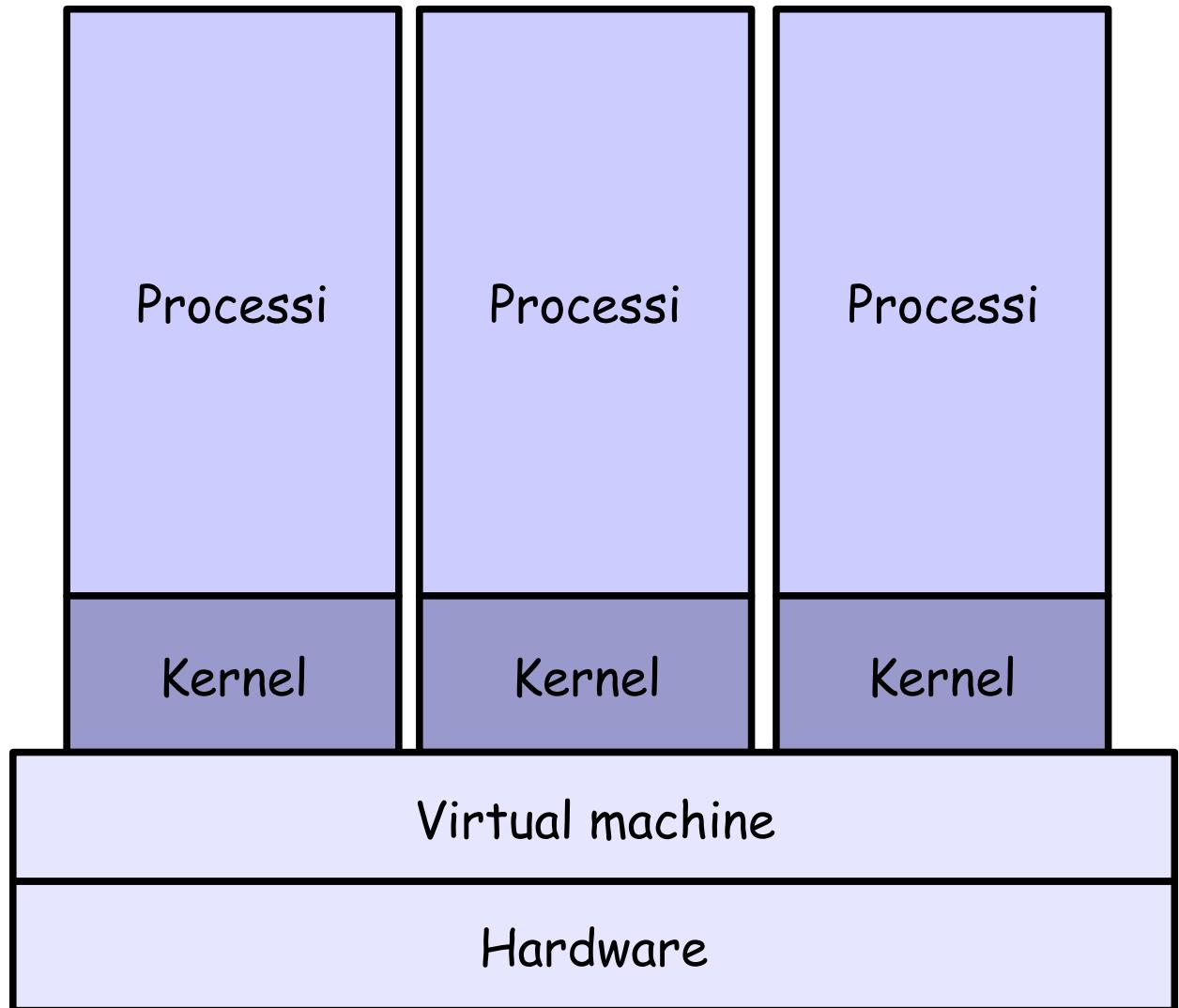
Macchine virtuali

- **E' un approccio diverso al multitasking**
 - invece di creare l'illusione di molteplici processi che posseggono la propria CPU e la propria memoria...
 - si crea l'astrazione di un macchina virtuale
- **Le macchine virtuali**
 - emulano il funzionamento dell'hardware
 - è possibile eseguire qualsiasi sistema operativo sopra di esse

Macchine virtuali



Senza VM



Con VM

Macchine virtuali

- **Vantaggi**
 - consentono di far coesistere s.o. differenti
 - esempio: sperimentare con la prossima release di s.o.
 - possono fare funzionare s.o. monotask in un sistema multitask e "sicuro"
 - esempio: MS-DOS in Windows NT
 - possono essere emulate architetture hardware differenti
 - (Intel o Motorola CISC su PowerPC)
- **Svantaggio**
 - soluzione inefficiente
 - difficile condividere risorse
- **Esempi storici: IBM VM**

Java

- **Gli eseguibili Java (detti bytecode) viene eseguito dalla Java virtual machine**
- **Questa macchina viene emulata in quasi tutte le architetture reali**
- **Vantaggi**
 - il codice è altamente portabile e relativamente veloce
 - (molto più di un codice interpretato)

debugging facilitato

controlli di sicurezza sul codice eseguibile

Progettazione di un sistema operativo

- **Definizione del problema**
 - definire gli obiettivi del sistema che si vuole realizzare
 - definire i "constraint" entro cui si opera
- **La progettazione sarà influenzata:**
 - al livello più basso, dal sistema hardware con cui si va ad operare
 - al livello più alto, dalle applicazioni che devono essere eseguite dal sistema operativo
- **A seconda di queste condizioni, il sistema sarà...**
 - batch, time-shared, single-user, multi-user, distribuito, general-purpose, real-time, etc....

Progettazione di un sistema operativo

- **Richieste dell'utente**
 - comodo da usare, facile da imparare, robusto, sicuro, veloce
- **Richieste degli sviluppatori**
 - facile da progettare, da mantenere e da aggiornare, veloce da implementare
- **Sono richieste vaghe...**
 - vanno esaminate con cura caso per caso
 - non vi è una risposta definitiva

Politiche e meccanismi

- **Separazione della politica dai meccanismi**
 - la politica decide cosa deve essere fatto
 - i meccanismi attuano la decisione
- **E' un concetto fondamentale di software engineering**
 - la componente che prende le decisioni "politiche" può essere completamente diversa da quella che implementa i meccanismi
 - rende possibile
 - cambiare la politica senza cambiare i meccanismi
 - cambiare i meccanismi senza cambiare la politica

Politiche e meccanismi

- **Nei sistemi a microkernel**
 - si implementano nel kernel i soli meccanismi, delegando la gestione della politica a processi fuori dal kernel
- **Esempio: MINIX**
 - il gestore della memoria è un processo esterno al kernel
 - decide la memoria da allocare ai processi ma non accede direttamente alla memoria del sistema
 - può accedere però alla propria memoria (è un processo
 - come tutti gli altri)

quando deve attuare delle operazioni per implementare la politica decisa
lo fa tramite chiamate specifiche al kernel (system task)

Politiche e meccanismi

- **Controesempio: MacOS <=9 (non Mac OS X)**
 - in questo sistema operativo, politica e meccanismi di gestione dell'interfaccia grafica sono stati inseriti nel kernel
 - lo scopo di questa scelta è di forzare un unico look'n'feel dell'interfaccia
- **Svantaggi:**
 - un bug nell'interfaccia grafica può mandare in crash l'intero sistema
- **Windows 9x non è differente...**

System generation: tailoring the O.S.

- **Portabilità**
 - lo stesso sistema operativo viene spesso proposto per architetture hardware differenti
 - è sempre possibile prevedere molteplici tipi di dispositivi periferici, e spesso anche diverse architetture di CPU e BUS
- **Occorre prevedere meccanismi per la generazione del S.O. specifico per l'architettura utilizzata**

System generation: parametri

- I parametri tipici per la generazione di un sistema operativo sono:
 - tipo di CPU utilizzata (o di CPU utilizzate)
 - quantità di memoria centrale
 - periferiche utilizzate
 - parametri numerici di vario tipo
 - numero utenti, processi, ampiezza dei buffer, tipo di processi

System generation

- I metodi che possono essere utilizzati sono:
 - rigenerazione del kernel con i nuovi parametri driver
 - UNIX e LINUX
 - prevedere la gestione di moduli aggiuntivi collegati durante il boot
 - extension MacOS
 - DLL Windows
 - moduli Linux



Current conditions at Pescara, Italy (IBP) 42.26N 014.12E 11M (IBP)
Last updated Feb 10, 2012 - 02:50 PM EST / 2012-02-10 1950 UTC
Temperature: 1 C
Relative Humidity: 80%
Wind: from the W (270 degrees) at 15 MPH (13 KT) gusting to 45 KPH
Weather: light snow grains
Sky conditions: overcast

Su	Mo	Tu	We	Th	Fr	Sa
feb 29	30	31	01	02	03	04
05	06	07	08	09	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
mar 26	27	28	29	01	02	03

Su	Mo	Tu	We	Th	Fr	Sa
mar 04	05	06	07	08	09	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

silvio@Stain ~ \$ cd Video
silvio@Stain:~/Video\$ movgrab http://vimeo.com/27998081

Formats available for this Movie: flv
Selected format: flv
Progress: 61.47% 15.4M of 25.1M 693.6K/s

