

Sistemi Operativi

Modulo 3: Architettura dei sistemi operativi

Renzo Davoli
Alberto Montresor

Copyright © 2002-2009 Renzo Davoli, Alberto Montresor

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license can be found at: <http://www.gnu.org/licenses/fdl.html#TOC1>

Sommario



- ◆ **Servizi di un sistema operativo**
- ◆ **System calls**
- ◆ **Sistemi con struttura semplice**
- ◆ **Sistemi con struttura a strati**
- ◆ **Microkernel**
- ◆ **Macchine virtuali**
- ◆ **Progettazione di un sistema operativo**

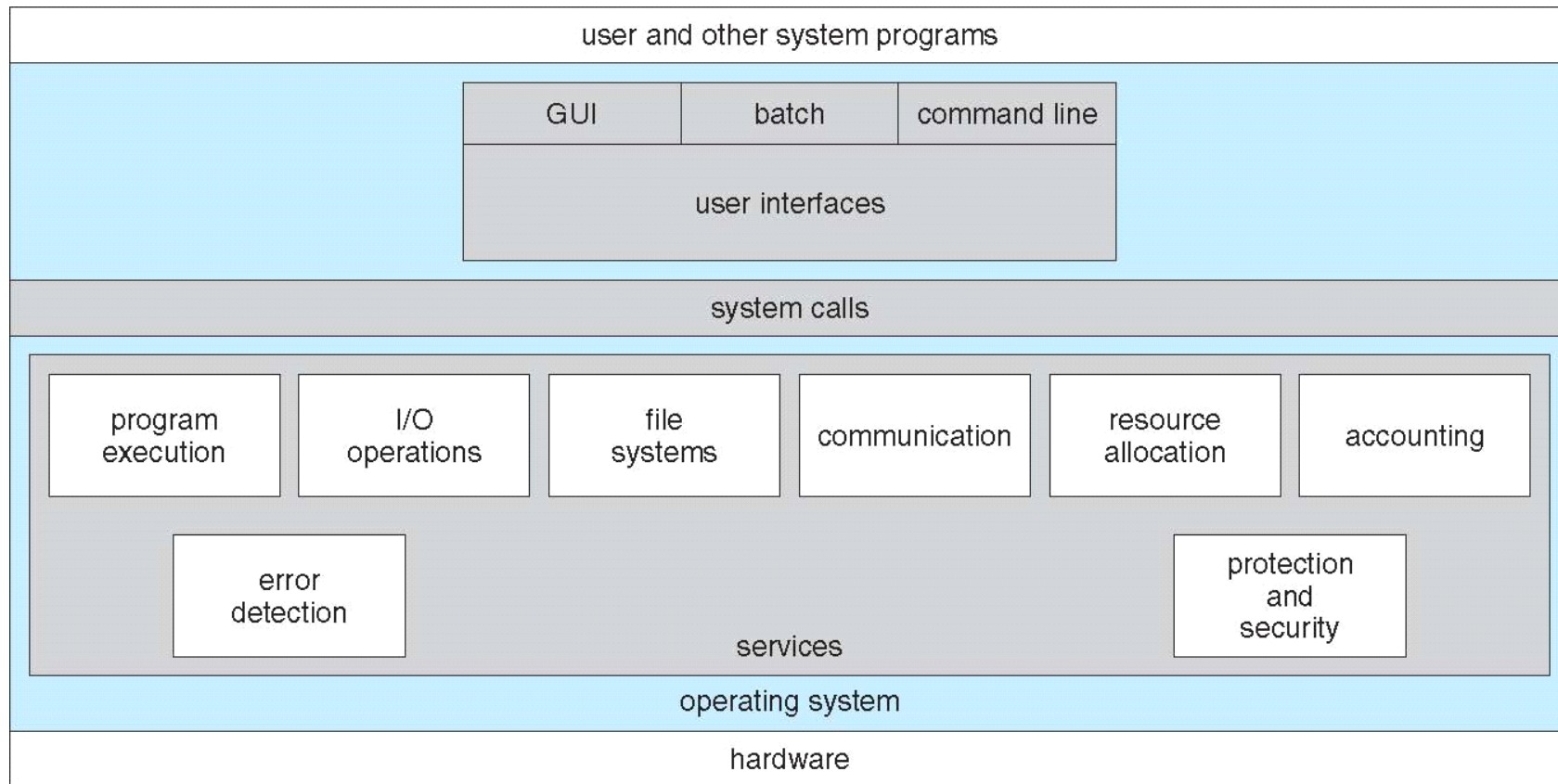
Servizi dei ss.oo.

- **Esecuzione di programmi** – capacità di caricare in memoria un programma ed eseguirlo.
- **Operazioni di I/O** – poiché i programmi non possono eseguire direttamente le operazioni di I/O, il sistema operativo deve fornire meccanismi per le operazioni di I/O.
- **Gestione del File system** – fornire ai programmi modalità per leggere, scrivere, creare e cancellare file.
- **Comunicazioni** – scambio di informazioni tra processi in esecuzione sullo stesso computer o su computer connessi in rete.
- **Scoperta di errori** – rilevamento di errori nella CPU, nella memoria, nell'I/O o nei programmi utente.

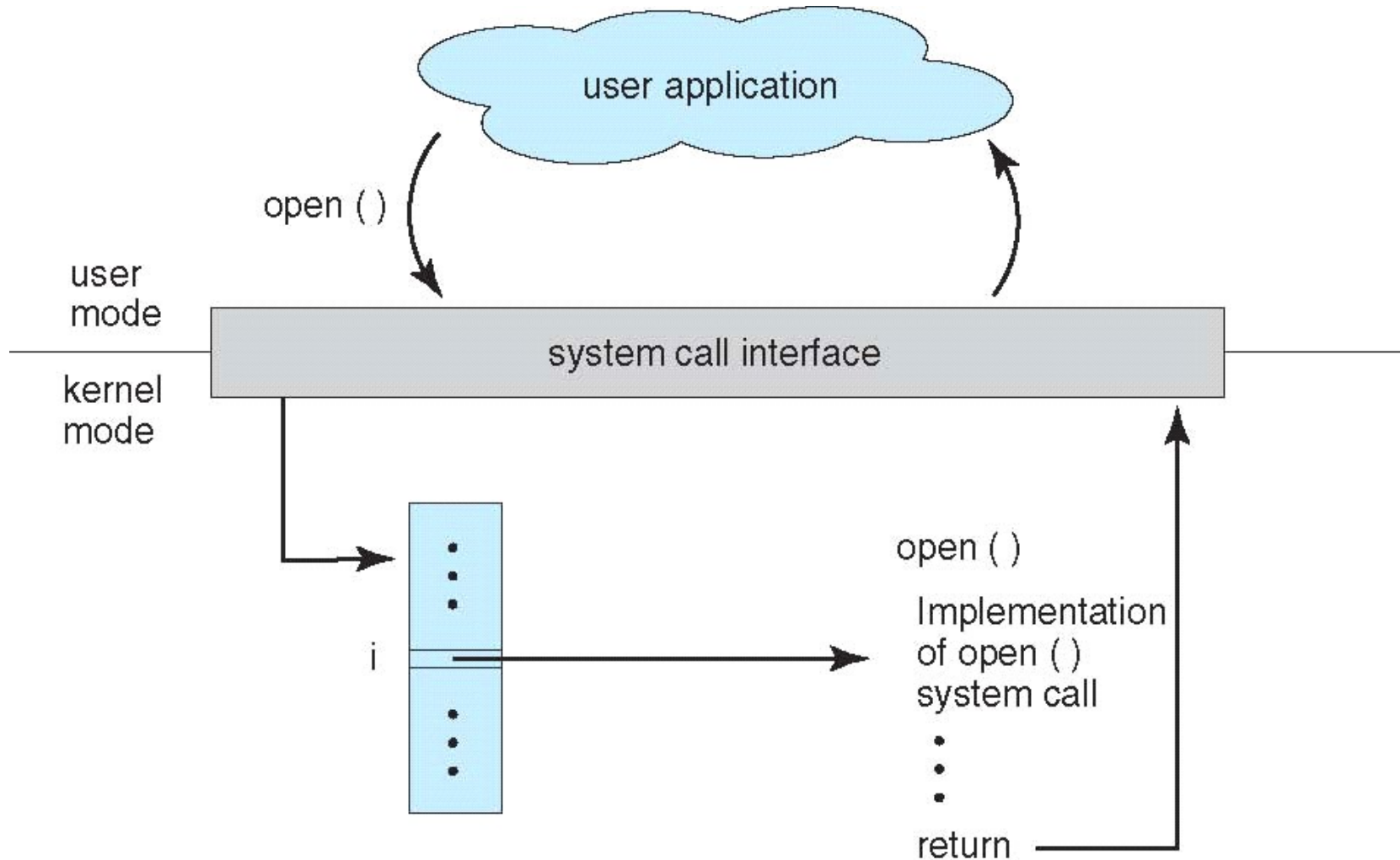
Servizi dei ss.oo.

- Esistono funzioni del sistema operativo che servono non tanto per l'esecuzione dei programmi utente, ma per assicurare l'esecuzione efficiente delle operazioni del sistema:
 - ↳ **Allocazione di risorse** – allocazione di risorse per i programmi eseguiti contemporaneamente.
 - ↳ **Accounting** – memorizzazione dell'uso delle risorse da parte dei vari utenti per il calcolo di costi e per la generazione di statistiche.
 - ↳ **Protezione** – controllo degli accessi a tutte le risorse del sistema.

Servizi dei ss.oo.



System calls (chiamate di sistema)

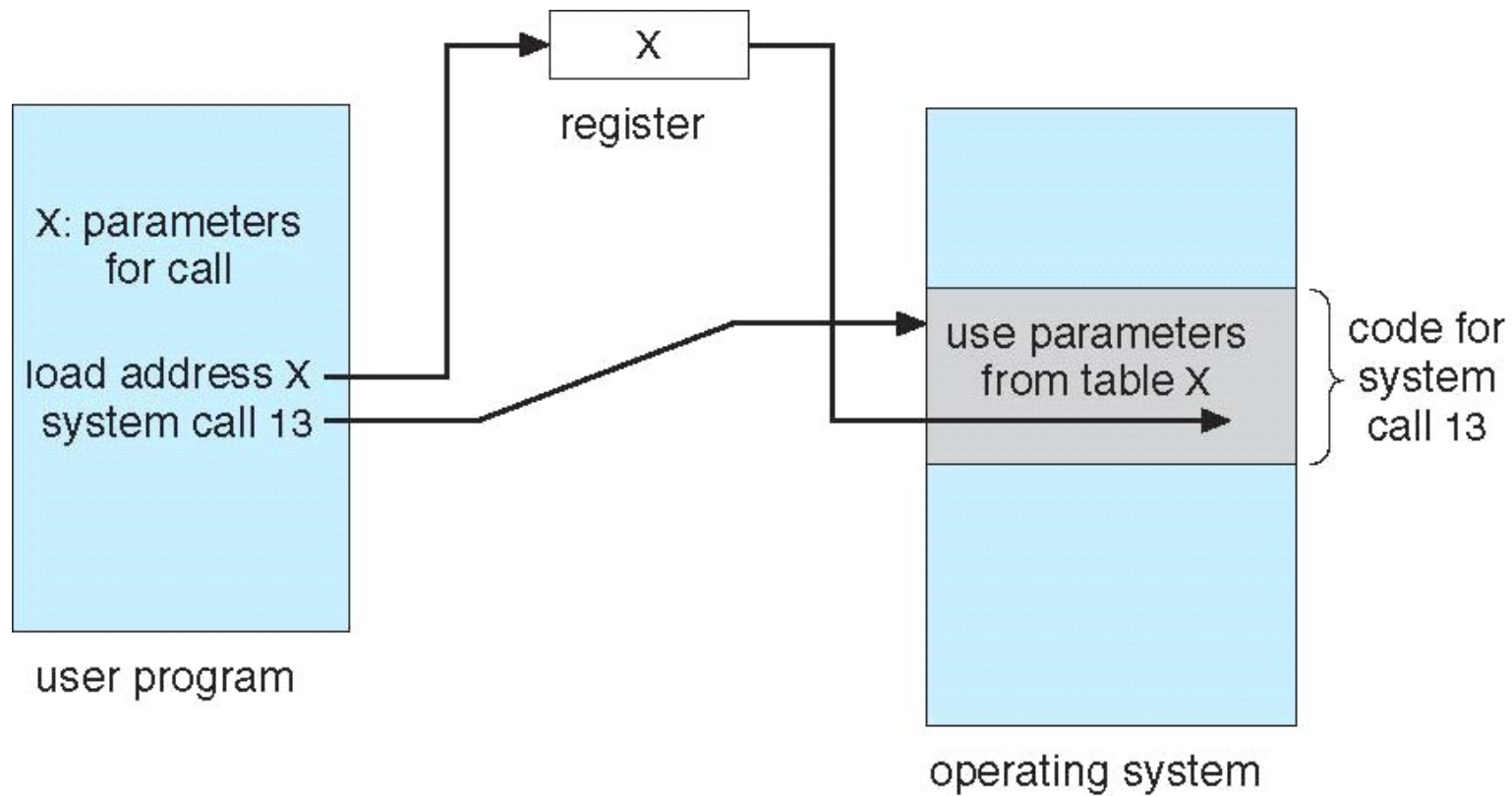


System calls – passaggio di parametri

- Le System call forniscono l'interfaccia di programmazione tra un programma e il sistema operativo.
 - Generalmente a basso livello, in alcuni casi con sintassi simile ai linguaggi di alto livello.
 - Alcuni sistemi hanno system call con sintassi simile a quella di linguaggi come C, e C++.

- Generalmente vengono usati tre metodi per il passaggio dei parametri tra un programma in esecuzione e il sistema operativo:
 - 1 Passaggio dei parametri in *registri*.
 - 2 Memorizzazione dei parametri in una *tabella in memoria* centrale e passaggio dell'indirizzo della tabella, come un parametro, in un registro.
 - 3 *Push* (inserimento) dei parametri in uno *stack* e *pop* (prelievo) dallo stesso dal sistema operativo.

System calls: passaggio di parametri via tabella



Tipi di system calls – controllo di processi

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory
- Dump memory if error
- **Debugger** for determining **bugs, single step** execution
- **Locks** for managing access to shared data between processes

Tipi di system calls – gestione di file e dispositivi

■ File management

- create file, delete file
- open, close file
- read, write, reposition
- get and set file attributes

■ Device management

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

Tipi di system calls – altro

- Information maintenance
 - get time or date, set time or date
 - get system data, set system data
 - get and set process, file, or device attributes

- Communications
 - create, delete communication connection
 - send, receive messages if **message passing model** to **host name** or **process name** (from **client** to **server**)
 - **Shared-memory model** create and gain access to memory regions
 - transfer status information
 - attach and detach remote devices

- Protection
 - Control access to resources
 - Get and set permissions
 - Allow and deny user access

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 dei dispositivi di I/O
- ◆ Gestione memoria principale
- ◆ Protezione
- ◆ Gestione memoria secondaria
- ◆ Networking
- ◆ Gestione file system
- ◆ Interprete dei comandi

- ◆ **Vediamo ora come sono collegati tra loro**

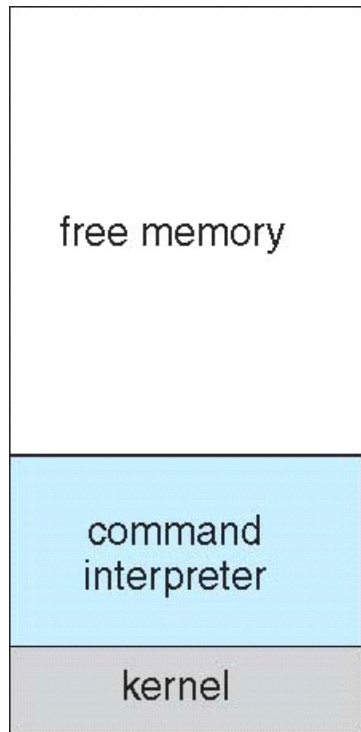
Struttura dei sistemi operativi



- ♦ **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

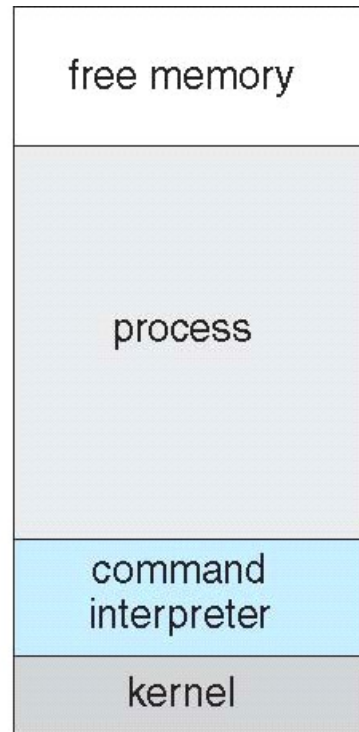
Struttura dei sistemi operativi

MS-DOS



(a)

Avvio

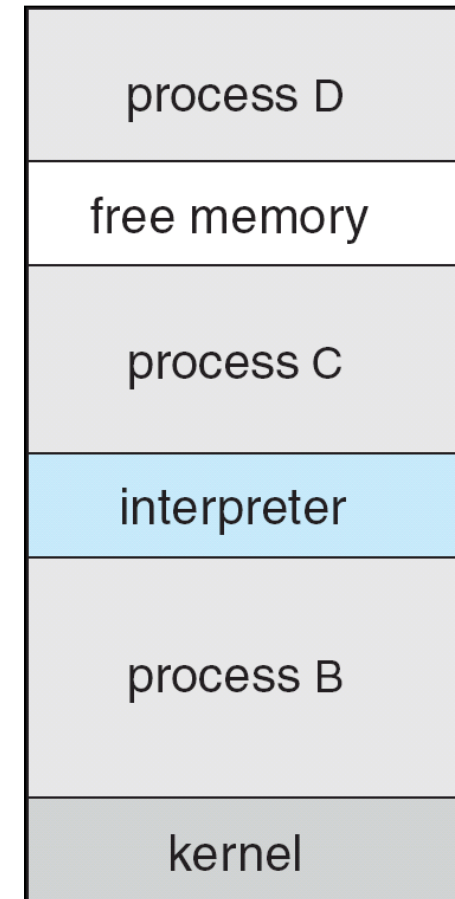


(b)

esecuzione di un programma

vs.

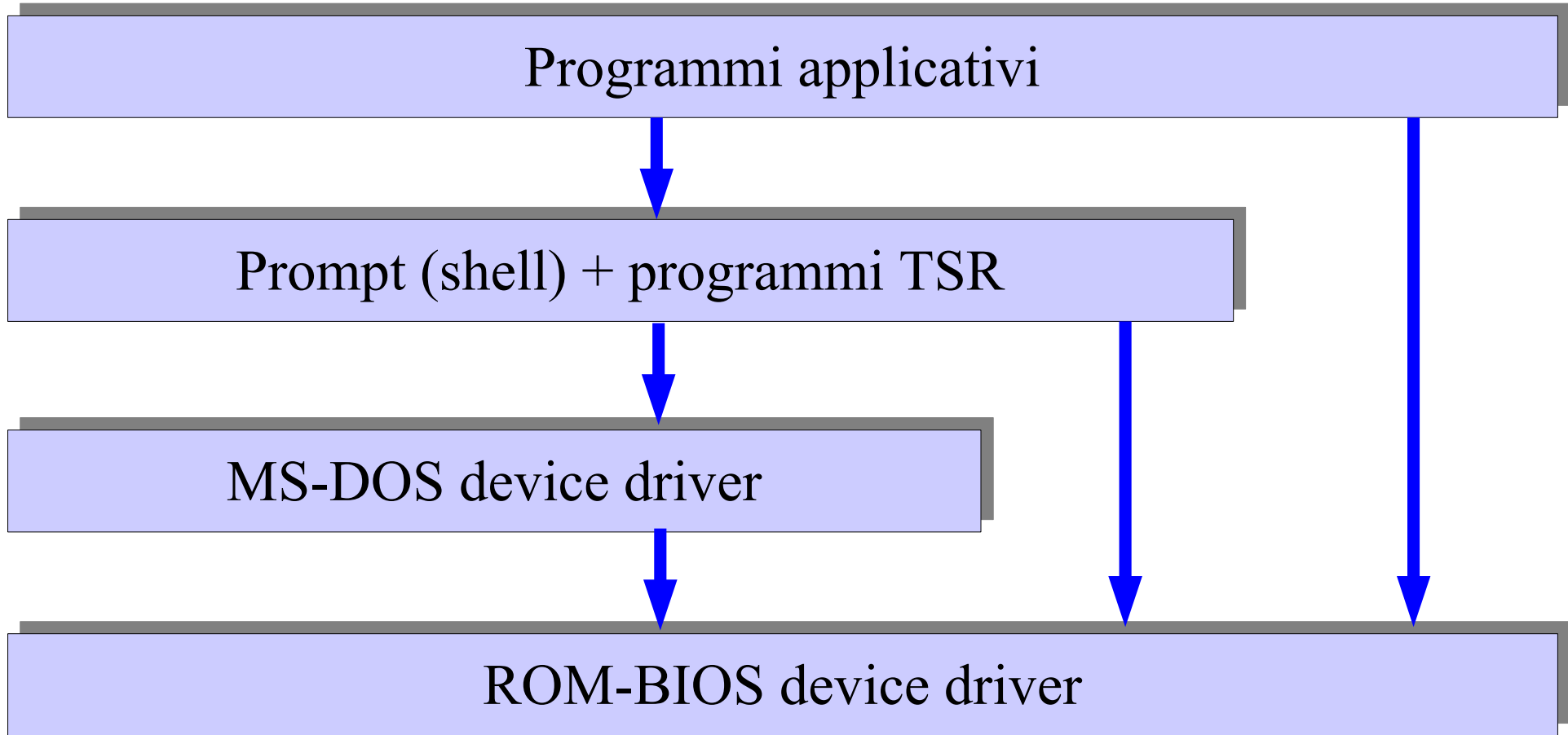
UNIX



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:**

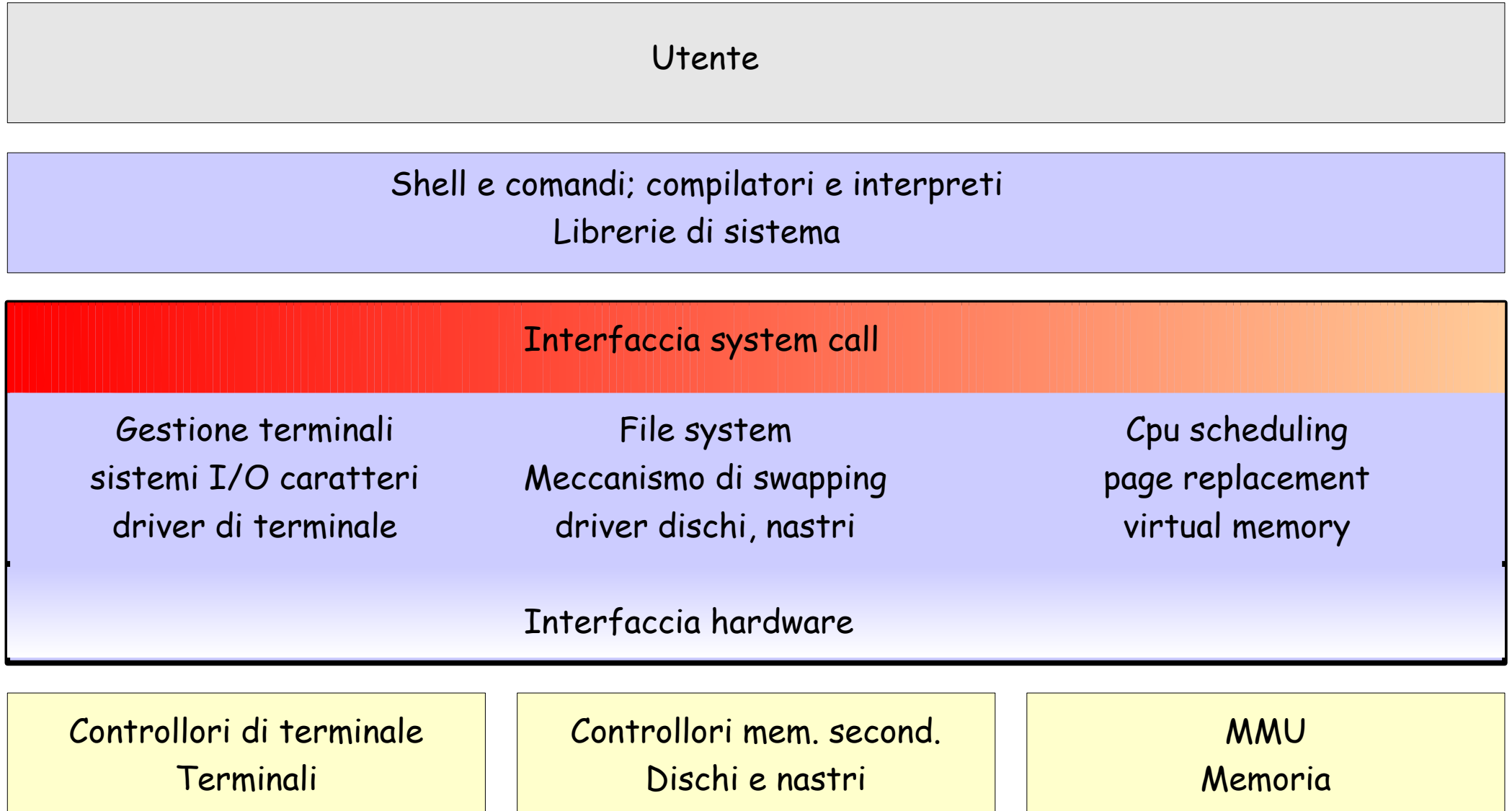
- ◆ i progettisti di MS-DOS erano legati all'hardware dell'epoca
- ◆ 8086, 8088, 80286 non avevano la modalità protetta (kernel)

UNIX



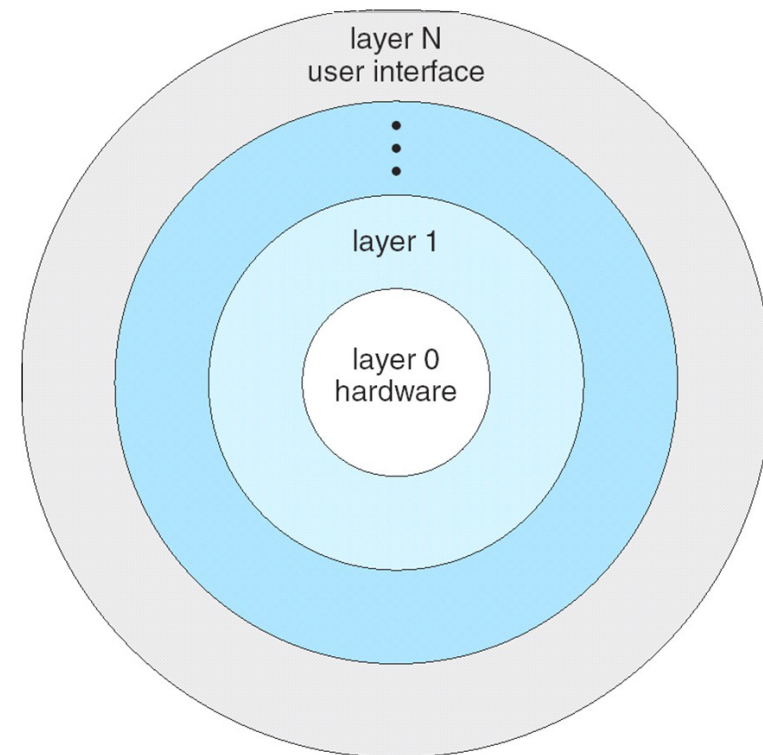
- ♦ **Anche UNIX è poco strutturato**
- ♦ **E' suddiviso in due parti**
 - ♦ kernel
 - ♦ programmi di sistema
- ♦ **Il kernel è delimitato**
 - ♦ in basso dall'hardware
 - ♦ in alto dal livello delle system call
- ♦ **Motivazioni**
 - ♦ anche Unix inizialmente fu limitato dalle limitazioni hardware...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

UNIX



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



- ◆ **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 moderni tendono ad avere meno strati

OS/2

Applicazioni

API

API Extension

Subsystem

Subsystem

Subsystem

System Kernel

Gestione memoria
Scheduling
Gestione device

Device
Driver

Device
Driver

Device
Driver

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 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

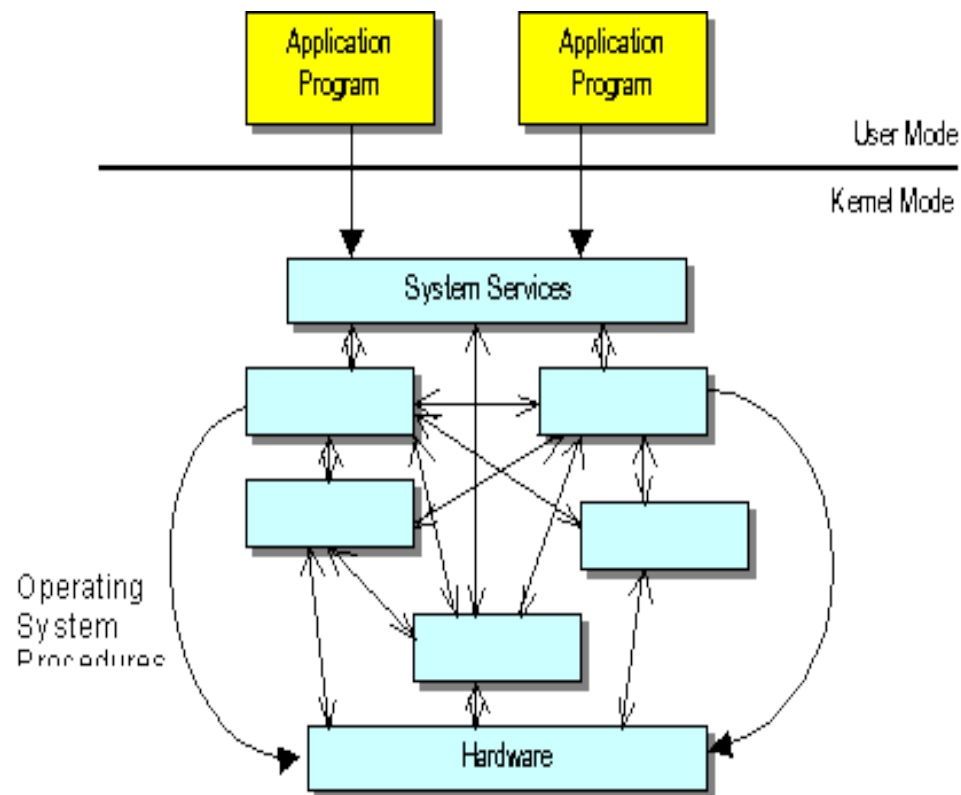
◆ Kernel Monolitici

- ◆ 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

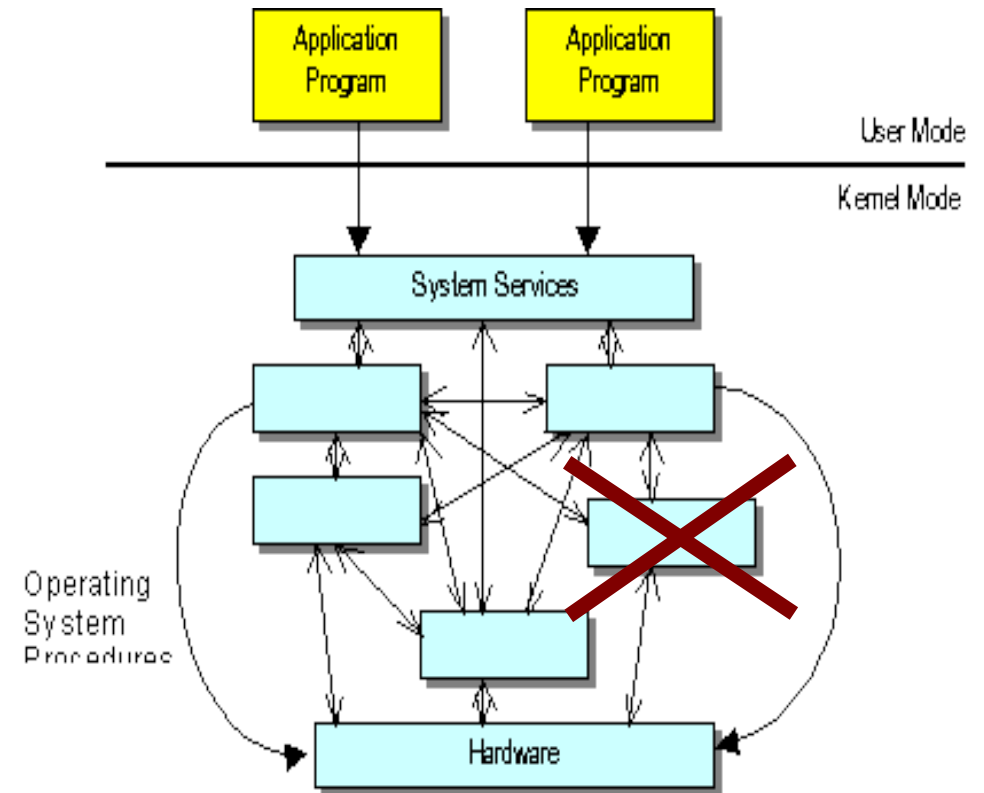
◆ Kernel Monolitici

- ◆ 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

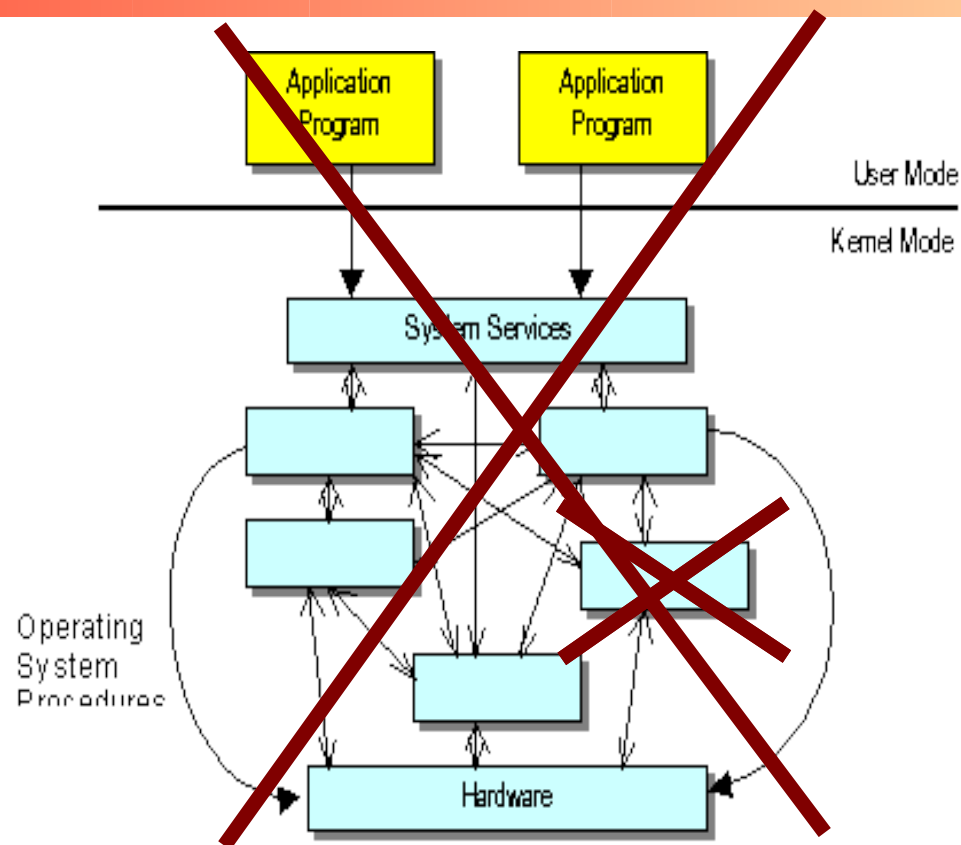
◆ Kernel Monolitici

- ◆ 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**

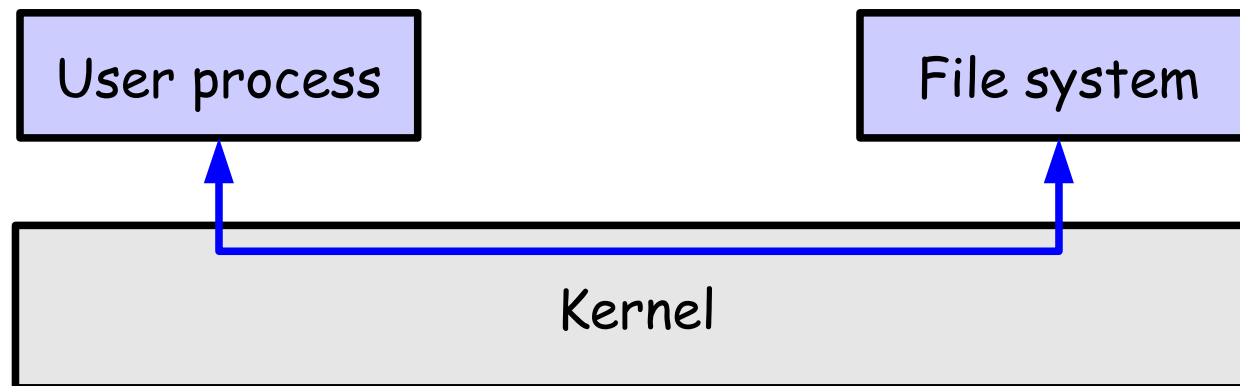
- ◆ per accedere ad un file, un processo interagisce con il processo gestore del file system

- ◆ **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 server
- ♦ La comunicazione è basata su *message passing*
 - ♦ il microkernel si occupa di smistare i messaggi fra i vari processi



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**

- ♦ *maggiore inefficienza*

- ♦ 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

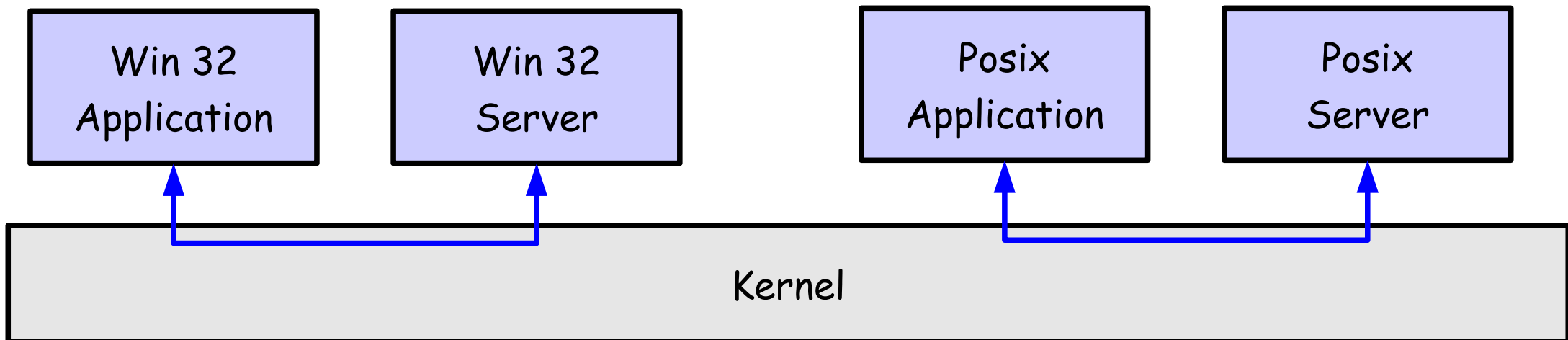
Kernel Ibridi



- ◆ **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**
 - ◆ Es. XNU (MAC OS X kernel)

Windows NT 4.0 / 2000

- ♦ **Windows NT è dotato di diverse API**
 - ♦ Win32, OS/2, Posix
- ♦ **Le funzionalità delle diverse API sono implementate tramite processi server**

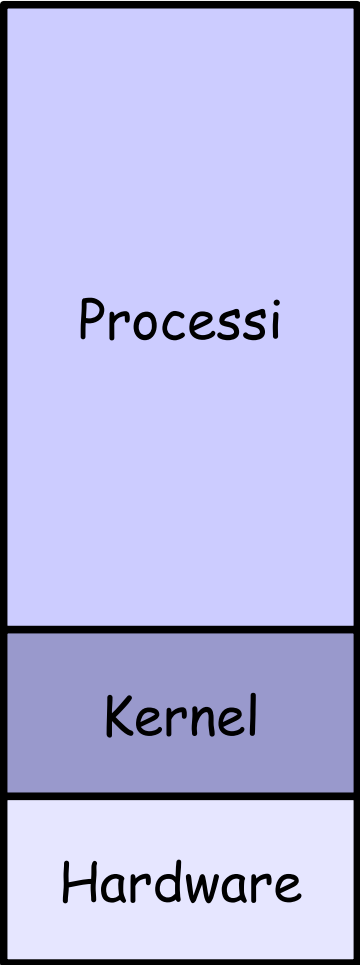


Macchine virtuali

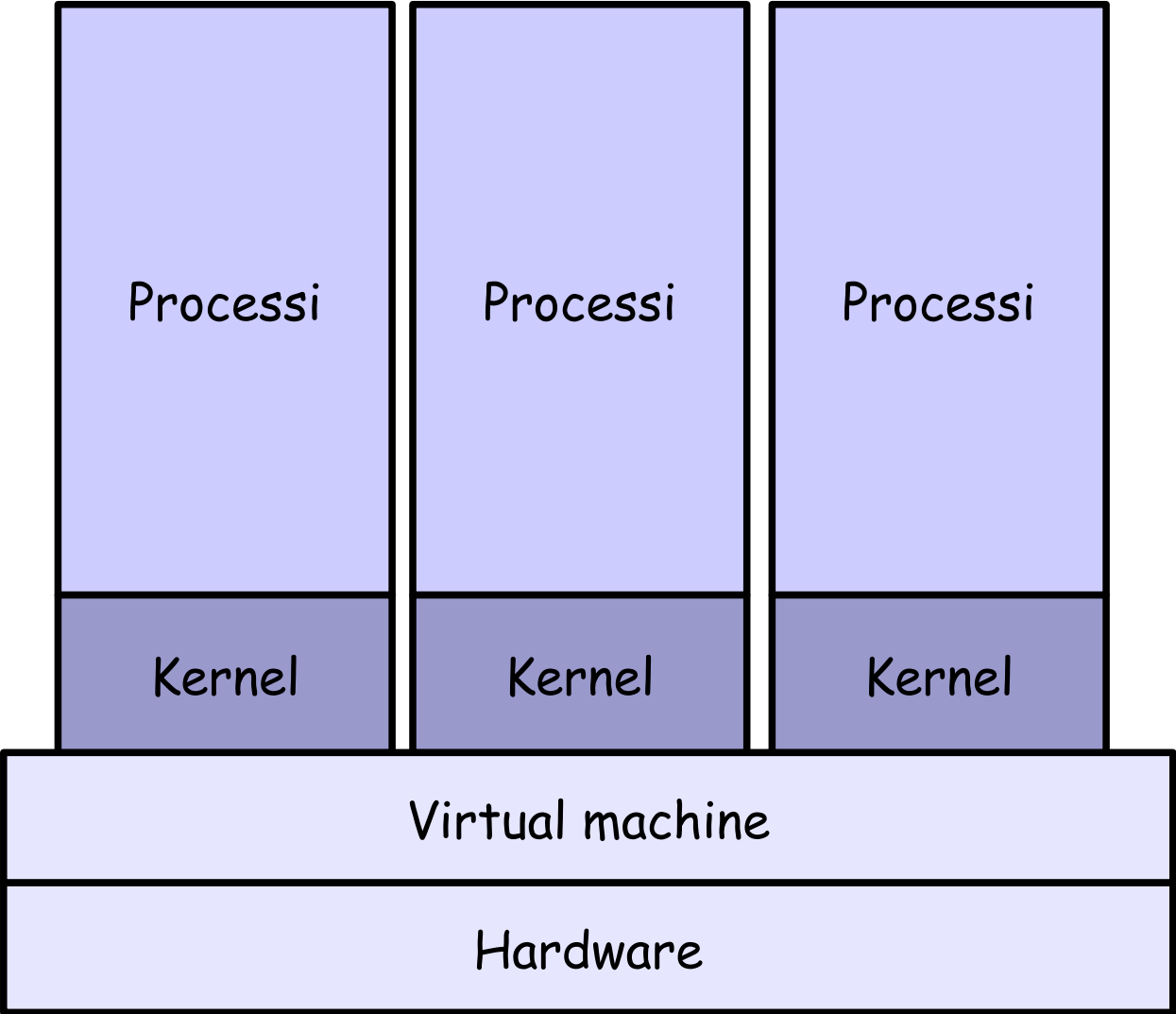


- ♦ **E' un approccio diverso al multitasking**
 - ♦ invece di creare l'illusione di molteplici processi che posseggono la propria CPU e la propria memoria...
 - ♦ si crea l'astrazione di un macchina virtuale
- ♦ **Le macchine virtuali**
 - ♦ emulano il funzionamento dell'hardware
 - ♦ è possibile eseguire 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) vengono eseguiti dalla Java Virtual Machine (JVM)**
- ♦ **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

Esempi di VM



- ◆ **Mac-On-Linux: (free SW, SVM, Dual)**
- ◆ **Vmware: (proprietary, SCVM, Dual)**
- ◆ **VirtualBOX: (dual licencing, SCVM, Dual)**
- ◆ **VirtualPC/Virtual Server: (proprietary, SVM, Dual)**
- ◆ **PearPC: (free, SVM, User-Mode User-Access)**
- ◆ **Solaris “zones” (aka containers): Native, SVN (OSLV).**

Progettazione di un sistema operativo

- ◆ **Definizione del problema**
 - ◆ definire gli obiettivi del sistema che si vuole realizzare
 - ◆ definire i vincoli 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 (“come” deve essere fatto)
- ♦ **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