SISTEMI OPERATIVI

(MODULO DI INFORMATICA II)

Concetti base e architettura

Prof. Luca Gherardi

Prof.ssa Patrizia Scandurra (anni precedenti)

Università degli Studi di Bergamo

a.a. 2012-13

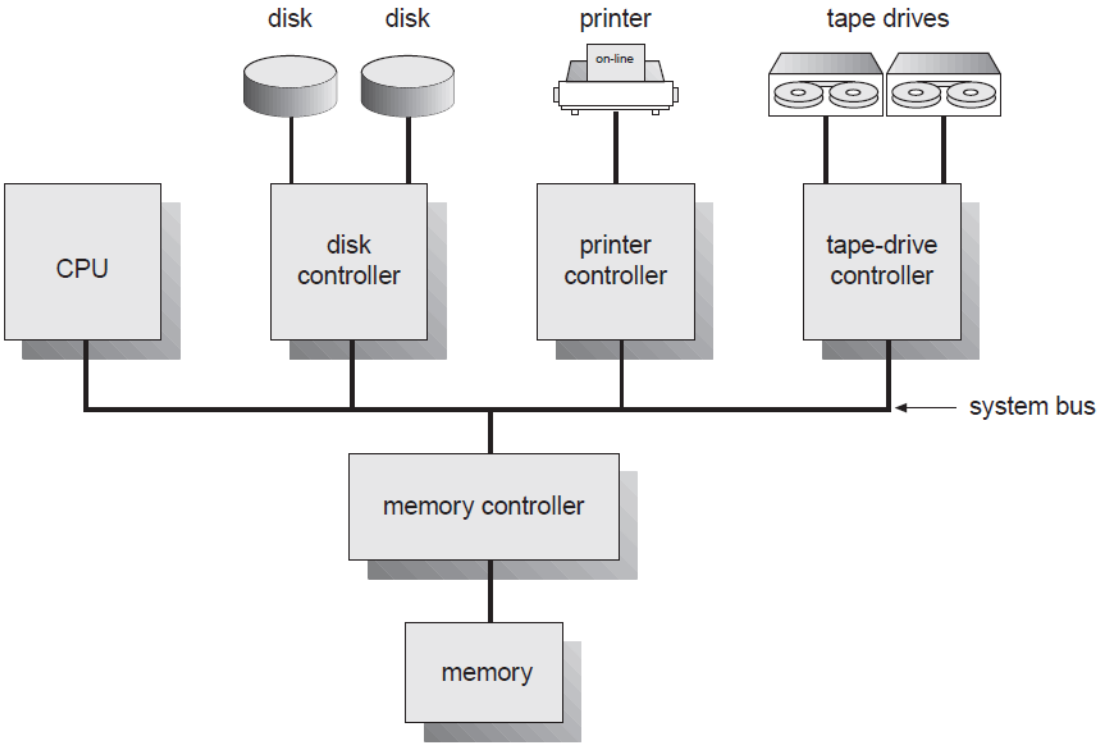
Sommario

- Funzionamento di un calcolatore (cenni)
- Obiettivi e funzioni di un sistema operativo
- Implementazione e avvio del sistema operativo
- Concetti di base sui sistemi operativi
  - Servizi di un sistema operativo
  - Funzionamento *Event-driven* e *Dual-Mode*
  - Servizi e chiamate di sistema
  - Programmi di sistema
- Struttura di un sistema operativo

FUNZIONAMENTO DI UN CALCOLATORE

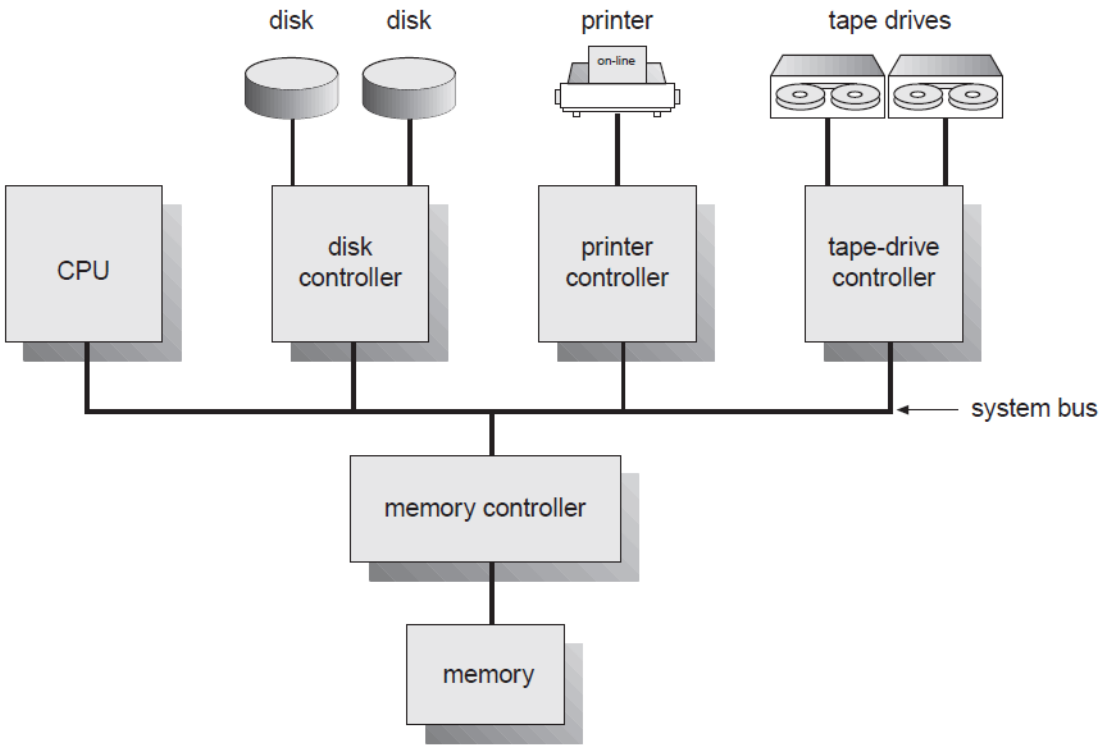
Funzionamento di un calcolatore

- Un calcolatore è composto da una CPU e diversi controllori di dispositivi
  - Comunicano tramite un bus
  - Operano in mondo concorrente contendendo l'accesso alla memoria centrale



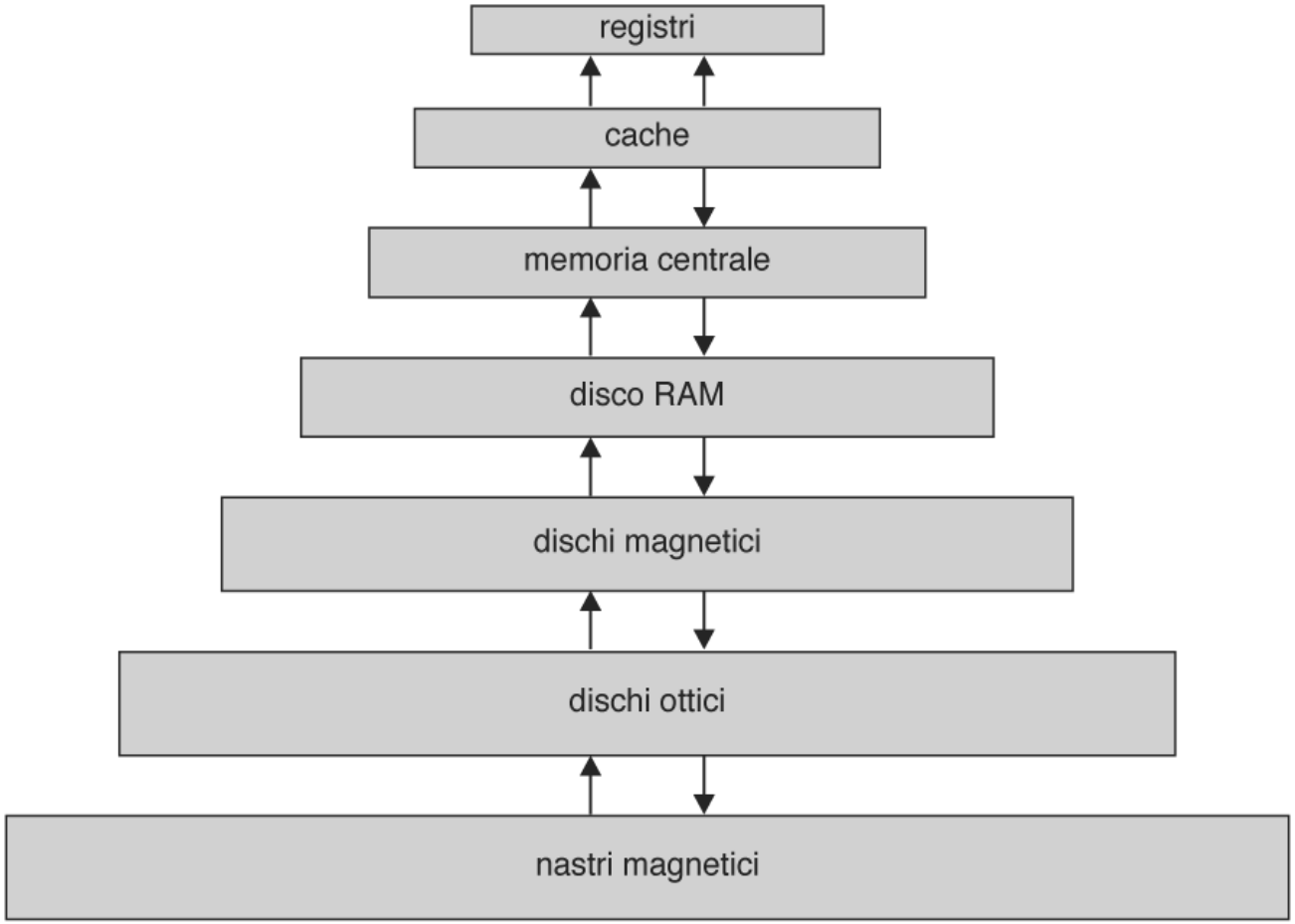
Funzionamento di un calcolatore

- L'I/O avviene **tra il dispositivo e il buffer locale del controller**
- La CPU guida poi lo spostamento dei dati (che viaggiano sul BUS) **dal buffer locale dei controller alla memoria**, e viceversa

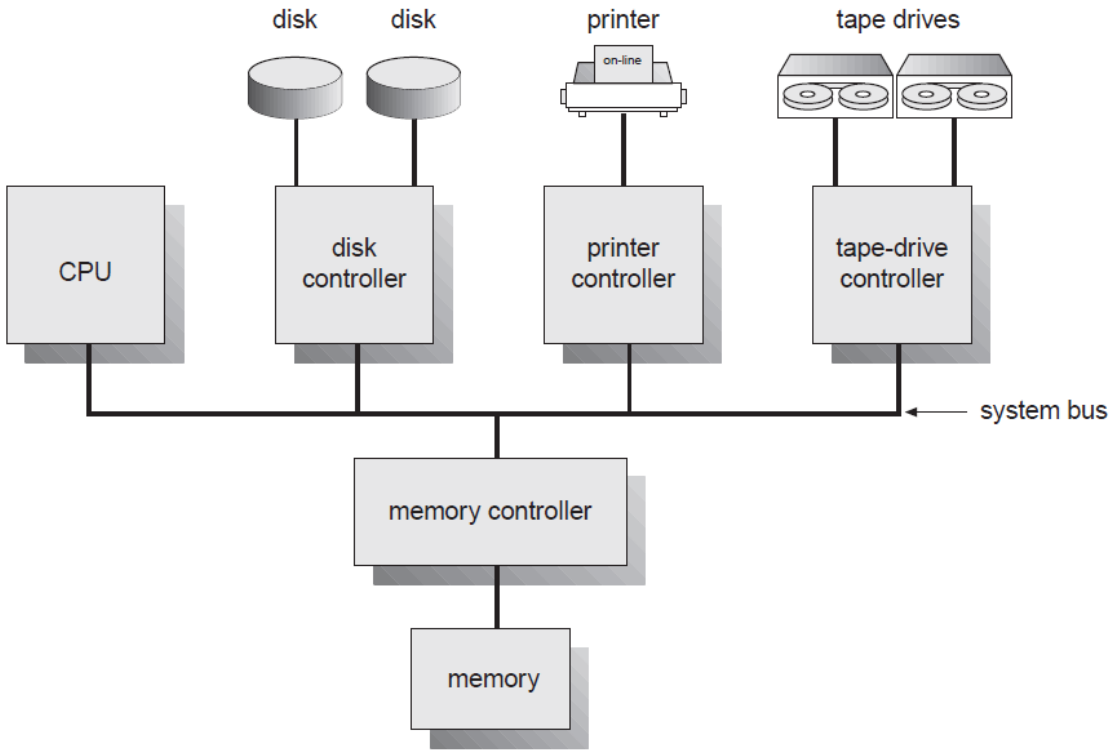


Funzionamento di un calcolatore

- Organizzazione della memoria



Funzionamento di un calcolatore – tecniche di I/O



1. I/O programmato
2. I/O gestito tramite interrupt
3. I/O con DMA
4. I/O con canali

Complessità Architetturale Crescente

Funzionamento di un calcolatore – tecniche di I/O

I/O programmato

- È tipicamente utilizzato da architetture di fascia bassa o al controllo industriale con basse necessità di I/O multiplo.
- Per ogni scrittura il processore esegue una istruzione di I/O
  - Per scrivere un blocco di n caratteri la CPU esegue n istruzioni di output carattere
- Per ricevere un dato la CPU legge in un ciclo stretto la porta di I/O in attesa del carattere
  - L'informazione di input disponibile viene tipicamente affidata ad un bit di controllo
- **Svantaggio**: non è ottimale nell'ipotesi in cui l'I/O di quantità di dati relativamente grosse occupi una parte rilevante del tempo macchina
  - Il microprocessore spenderà la maggior parte del suo tempo in operazioni di I/O

Funzionamento di un calcolatore – tecniche di I/O

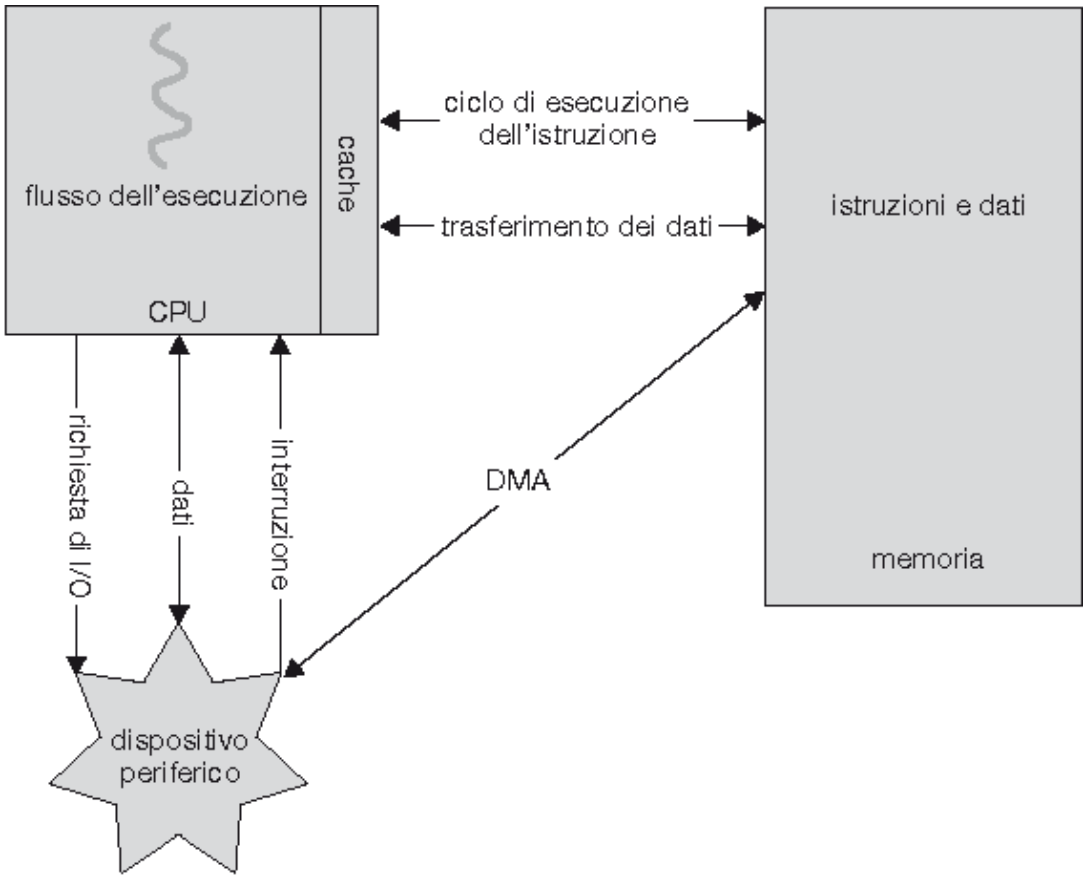
I/O tramite interrupt

- Permette di migliorare le prestazioni di letture/scritture multiple
- Scrittura:
  - La CPU lancia l'operazione di scrittura
  - Contemporaneamente continua altre operazione in attesa che un interrupt segnali il termine della scrittura
- Lettura:
  - La presenza di input viene segnalata tramite un interrupt

- La CPU incarica i dispositivi di eseguire le operazioni di I/O e nel mentre esegue altre attività

Funzionamento di un calcolatore – tecniche di I/O

- Come è possibile?
- Un controllore di dispositivo I/O dispone di una memoria interna (buffer)
- Per avviare un'operazione di read di un byte il controllore
  - copia i dati sul buffer
  - Informa il driver del dispositivo (tramite interrupt)
- Il driver cede il controllo al SO restituendo un puntatore ai dati nel buffer
- Oneroso per grosse dimensioni (molti byte)



DMA: un volta impostato il buffer, il controllore del dispositivo I/O trasferisce direttamente grandi porzioni di dati dal dispositivo alla memoria. È necessario un solo interrupt, quindi un solo interazione con la CPU per ogni blocco di byte.

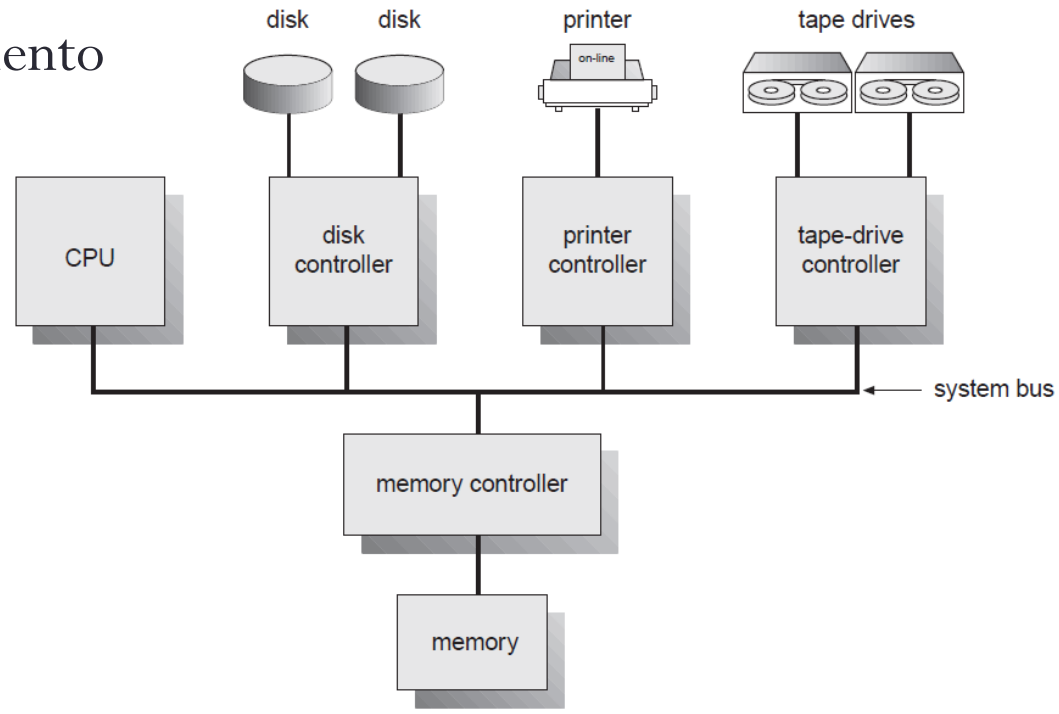
OBIETTIVI E FUNZIONI DI UN SO

Obiettivi di un sistema operativo (1)

- Astrazione

  Alzare il livello di astrazione dei componenti del sistema di elaborazione (macchina di von Neumann)

- Astrazione del comportamento
- Semplificazione dell'uso
- Più facile da programmare dell'hardware sottostante
- Semplificazione nella implementazione dei programmi applicativi



Obiettivi di un sistema operativo (2)

- Virtualizzazione

  Creazione di un'immagine del sistema di elaborazione dedicata a ciascun programma in esecuzione

- Indipendenza dalla presenza di altri programmi
- Gestione ottimizzata delle risorse
- Allocazione e condivisione ordinata delle risorse rispetto a *spazio/tempo*

Funzione principale di un SO

- Un SO **fornisce l'ambiente per l'esecuzione dei programmi**

Più in dettaglio …..

Funzioni (1)

- Gestione del processore

- **Gestione dei processi: un processo è un programma in esecuzione**

- Creazione e terminazione dei processi
- Sospensione e riattivazione dei processi
- Schedulazione dei processi
- Sincronizzazione tra processi
- Gestione di situazioni di stallo (deadlock)
- Comunicazioni tra processi (memoria condivisa o scambio di messaggi)

Funzioni (2)

- Gestione della memoria centrale

- **Processi in memoria centrale per esecuzione**

- Multiprogrammazione
- Allocazione e deallocazione della memoria ai processi
- Caricamento e scaricamento di processi e di loro porzioni in memoria centrale
- Protezione della memoria centrale

Funzioni (3)

- Gestione delle periferiche

- **Omogeneità di interazione (verso hardware eterogeneo)**

- Configurazione e inizializzazione
- Interfaccia generale e omogenea
- Gestione ottimizzata dei dispositivi di I/O, memorizzazione di massa e rete informatica
- Protezione delle periferiche
- Bufferizzazione
- Caching

Funzioni (4)

- Gestione del file system

- **File e albero del file system**

- Nasconde all'utente l'organizzazione della memoria attraverso il concetto di **file**
- File e directory
  - Creazione e cancellazione
  - Lettura e scrittura
  - Copiatura
  - Ricerca
  - Salvataggio e ripristino
  - Protezione e sicurezza (diritti di accesso)

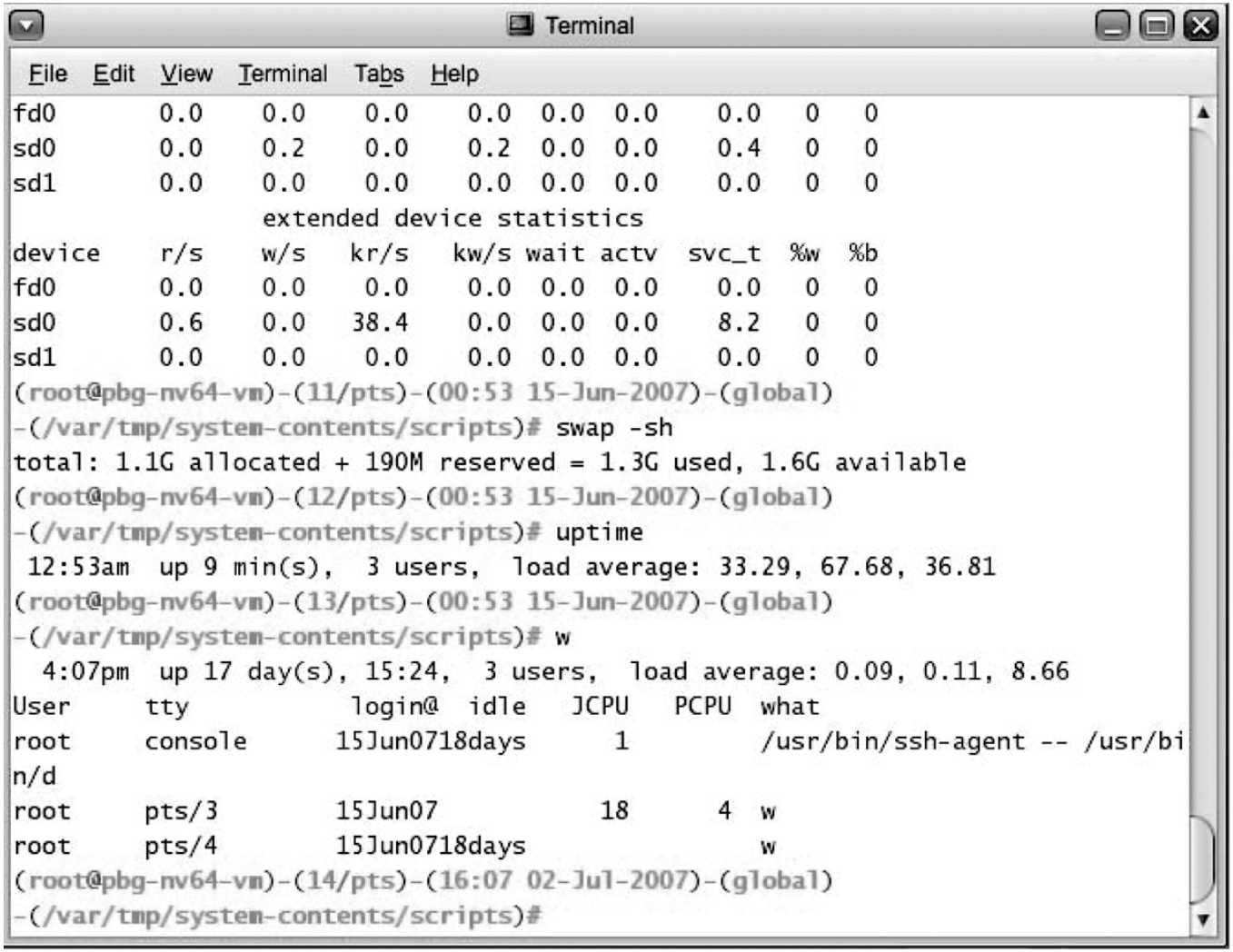
Funzioni (5)

- Gestione dell'interfaccia utente

- **Interazione con utenti e processi attraverso istruzioni di controllo**

- Interfacce a riga di comando: interprete dei comandi o shell (programma che legge ed interpreta istruzioni di controllo)
- Interfacce grafiche
- Interfacce a lotti
  - comandi e direttive codificati in file ed eseguiti a lotti
- L'interfaccia è separata dall'esecuzione dei comandi (flessibilità)!!!

Funzioni (5)


Terminal
File Edit View Terminal Tabs Help
fd0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
sd0 0.0 0.2 0.0 0.2 0.0 0.0 0.4 0 0
sd1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
extended device statistics
device r/s w/s kr/s kw/s wait actv svc_t %w %b
fd0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
sd0 0.6 0.0 38.4 0.0 0.0 0.0 8.2 0 0
sd1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# uptime
12:53am up 9 min(s), 3 users, load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# w
4:07pm up 17 day(s), 15:24, 3 users, load average: 0.09, 0.11, 8.66
User tty login@ idle JCPU PCPU what
root console 15Jun0718days 1 /usr/bin/ssh-agent -- /usr/bi
n/d
root pts/3 15Jun07 18 4 w
root pts/4 15Jun0718days w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
-(/var/tmp/system-contents/scripts)#

Funzioni (5)


Grab File Edit Capture Window Help
15:06 EDT Mon 2 Jul
fig-dir
Network
Freedom Stripe
Macintosh HD
Untitled
Untitled 2
ZPBG
ZPBGE
iDisk
Peter Baer Galvin's iPod
pbg
Applications
Documents
Games
Utilities
tmp
Desktop
Favorites
Music
Movies
Pictures
Sites
Public
Preferences
Library
cti
projects
consult
fig-1.06a.pdf
fig-1.06a.svg
fig-2.0.png
fig-2.0.svg
fig-2.0a
TIFF
5 items
1 of 5 items selected, 734.3 GB available, 5.1 GB used
Address Book
Apple Computer Inc.
main 1-800-MY-APPLE
other 800-275-2273
home page http://www.apple.com
work 1 Infinite Loop
Cupertino CA 95014
United States
2 found
PCalc
Dictionary and Thesaurus
operating system
Dictionary
op•er•at•ing sys•tem
noun
the software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.
Trash

Funzioni (6)

- Rilevamento d'errori
- **Capacità di rilevare errori nella CPU o nei dispositivi**
- Esempi:
  - Divisione per 0
  - Fine della carta
  - Guasto di una connessione di rete
- Per ogni errore è necessario eseguire una appropriata azione di recovery

SERVIZI OFFERTI DA UN SO

Servizi del sistema operativo (1)

Il SO fornisce **servizi** ai programmi e agli utenti dei programmi

- **Elaborazione**
  - il sistema deve potere caricare un programma in memoria e eseguirlo
- **Operazioni di I/O** – in generale gli utenti non possono controllare direttamente i dispositivi di I/O; è il SO 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

Servizi del sistema operativo (2)

- **Comunicazione** fra processi in esecuzione sullo stesso computer o in esecuzione su computer differenti collegati fra loro in rete
  - Le comunicazioni possono avvenire attraverso
    - una *memoria condivisa* o
    - attraverso *scambio di messaggi*

- **Rilevamento degli errori** – il sistema assicura un calcolo corretto per individuare errori
  - nella CPU
  - nell'hardware della memoria,
  - nei dispositivi di I/O e nei programmi dell'utente

Servizi del sistema operativo (3)

Esistono **servizi aggiuntivi** per una gestione/monitoraggio efficiente del sistema

- **Allocazione delle risorse** – *quando ci sono più utenti o più processi* contemporaneamente in funzione, le risorse devono essere assegnate a ciascuno di loro

- **Contabilità (accounting)**– *mantiene traccia di quali utenti usano le risorse* e di quale tipo e genere di risorse si tratta per stilare statistiche d'uso o per la contabilità

- **Protezione e sicurezza** – implica la *garanzia* che tutti gli *accessi* alle risorse del sistema siano *controllati*

Implementazione del SO

- **Tradizionalmente scritti in linguaggio assembly, oggi possono essere 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 facile da capire e da mettere a punto
- Un SO è molto più flessibile e manutenibile se è scritto in un linguaggio ad alto livello
- I SO basati sui linguaggi si affidano alla ***sicurezza* del linguaggio**
  - Java è considerato un linguaggio sicuro
  - Linguaggi ritenuti sicuri sono utili per implementare SO su HW limitati (palmari, smart card, ecc..) che non hanno protezione fornita da HW

Generazione di un SO

- Tipicamente lo stesso SO può essere eseguito su più macchine
- Il sistema deve essere quindi configurato (generazione di sistema)

- Parametri
  - CPU
  - Memoria (indirizzo legale finale, i.e. dimensione)
  - Formattazione del disco di avvio (partizioni, dimensioni, contenuto)
  - Dispositivi disponibili
  - Opzioni (e.g. algoritmo di schedulazione, max. num. processi_)
- Vale anche per le macchine virtuali

AVVIO DI UN SO

Avvio (boot) del SO (1)

I passi sono:

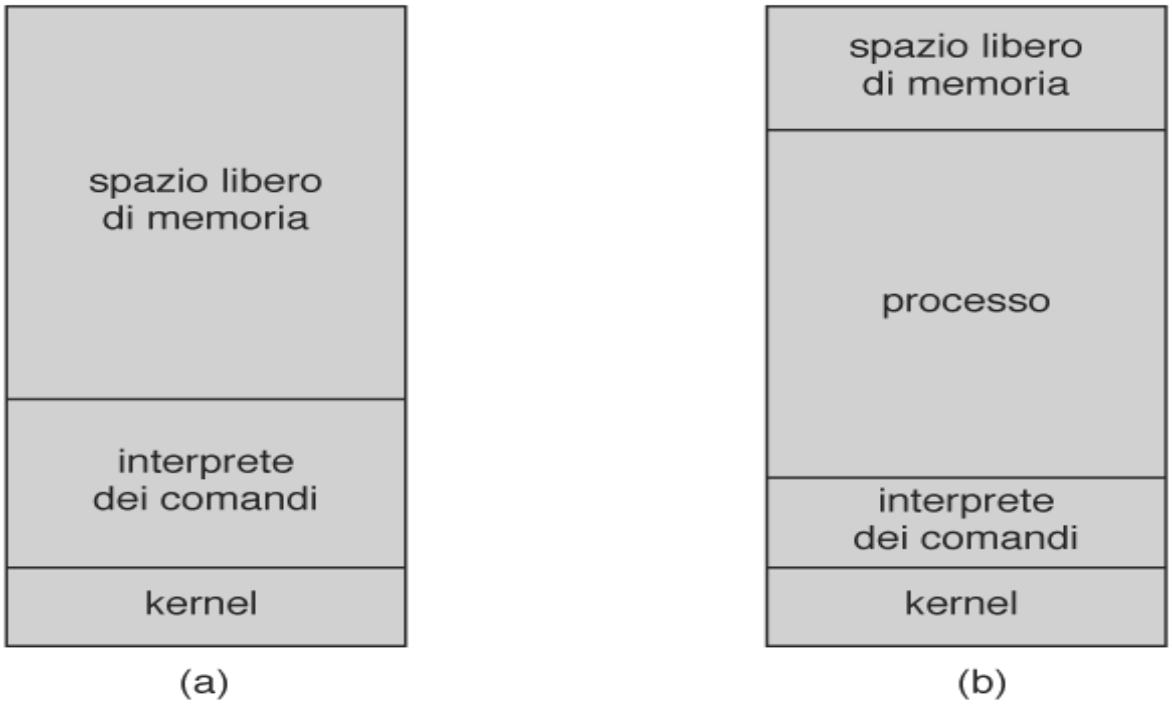
- Accensione
- Esecuzione del BIOS (in ROM o EEPROM) – *Basic I/O System*
  - Inizializza tutti i registri della CPU, i controllori delle periferiche e la memoria centrale
- Esecuzione del *bootstrap program o bootstrap loader*
  - caricamento del kernel del SO da una unità di memoria ed esecuzione
- Controllo del sistema da parte del SO

Avvio (boot) del SO (2)

Dove risiede il SO?

- Nelle tradizionali console giochi e PDA, il SO risiede **in memorie ROM-like (*firmware*)**
  - costoso
  - non facilmente modificabile (anche se protetto da virus)
  - più lento eseguire da ROM che da RAM
- In generale, il SO è **nel disco (in una partizione detta *attiva*)** ed il *bootstrap program* è in memoria ROM o EEPROM
  - Il BIOS cerca periferiche avviabili e dà il controllo al bootstrap program
  - Il bootstrap program si avvale di un frammento di codice (che sà in quale parte del disco si trova il SO) che risiede in un blocco del disco ad una posizione fissa (tipicamente 0) – ***boot block*** *o* ***master boot record (MBR)***
  - La partizione del disco che contiene il boot block è detta di *avvio* – ***boot partition***

MS-DOS avvio ed esecuzione



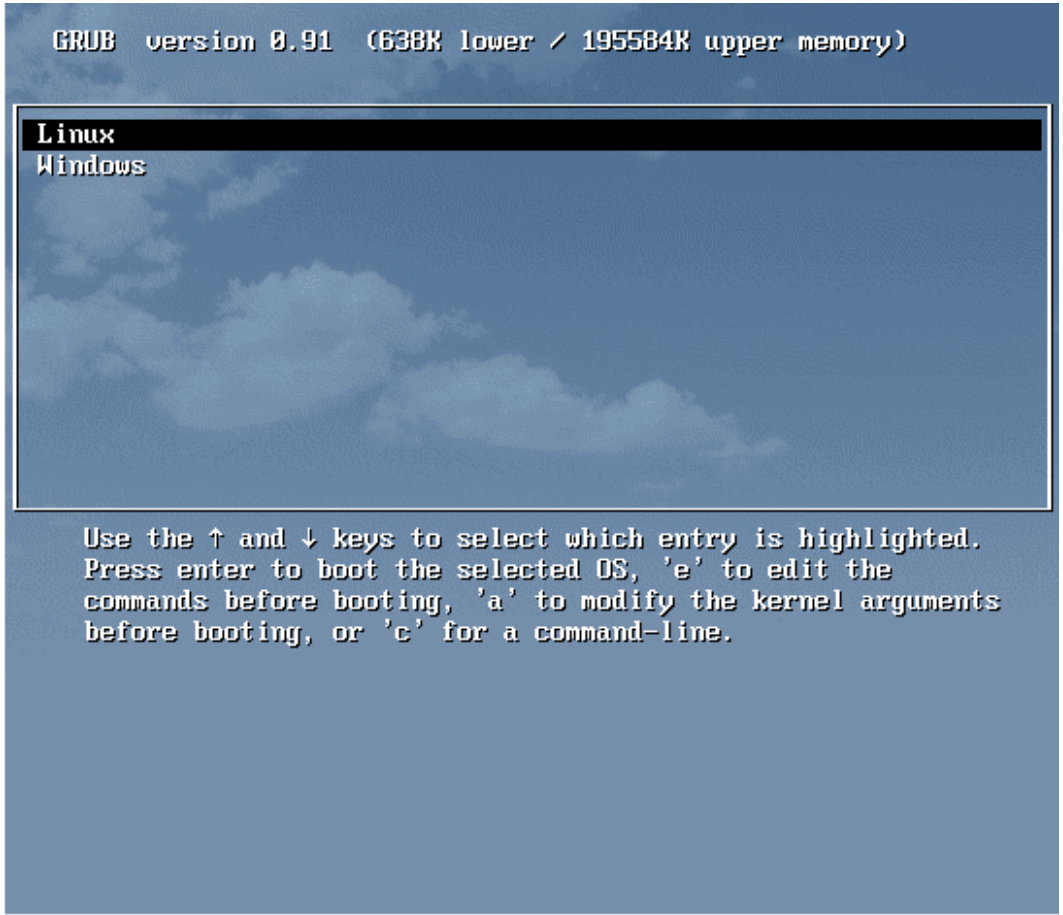
- L'interprete carica il programma in memoria.
- Esso sovrascrive parte della memoria occupata da se stesso.

(a) System startup (b) Running a program

Boot loader per i SO moderni

- GNU GRUB (GRand Unified Bootloader) è il boot loader utilizzato nelle recenti *distro* di GNU/Linux

- Il GRUB
  - Cerca SO disponibili
  - Avvia schermata di selezione
  - Avvia SO desiderato



DUAL MODE E INTERRUZIONI

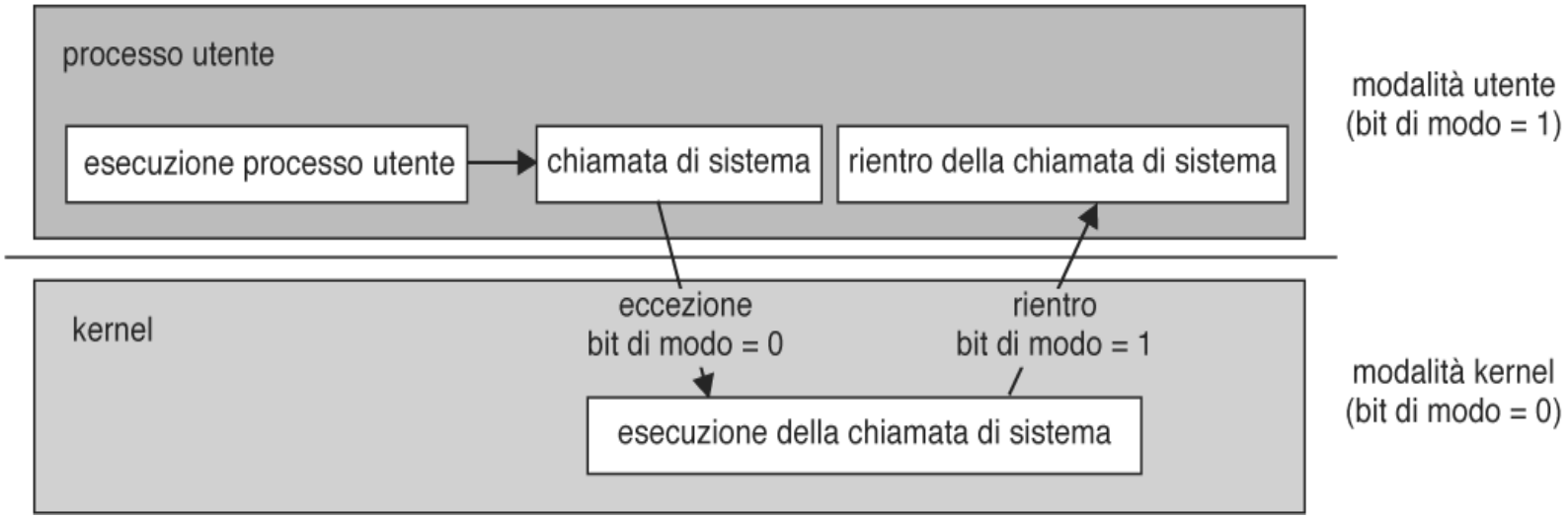
Funzionamento Dual-Mode (1)

- Una volta avviato il sistema operativo, la CPU può operare in due modalità
  1. **User mode** – la CPU sta eseguendo codice di un utente
  2. **Kernel mode** (anche supervisor mode, system mode, monitor mode) – la CPU sta eseguendo codice del sistema operativo

- La CPU ha un **Mode bit** nel *registro di stato* (Process Status Word o PSW) che indica in quale modo si trova: kernel (0) o user (1)

Funzionamento Dual-Mode (2)

- Protezione dei registri da accessi erronei/intenzionali
- L'accesso completo all'hardware solo in modo kernel
- Passaggio controllato da modo utente a modo kernel
  - interruzioni
  - chiamate di sistema (system call), trap/fault

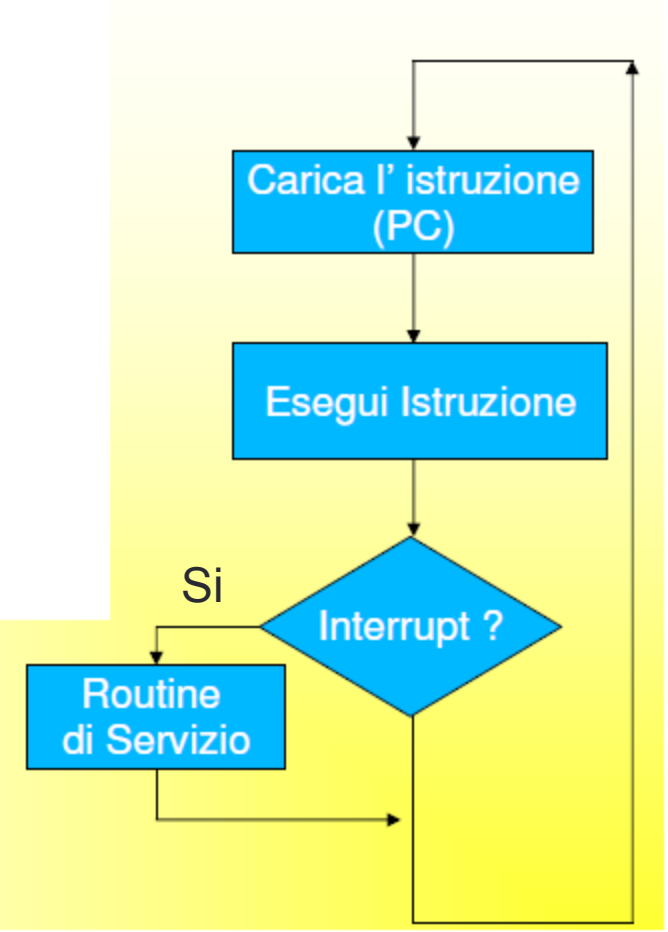


Funzionamento event-driven del SO

- Dopo il caricamento in memoria, il SO lancia un processo speciale (*init* in Unix) e poi **attende segnali di interrupt** (eventi asincroni)
- Tre forme di interrupt:
  - (1) Interrupt hardware
  - Interrupt software
    - (2) Chiamate di sistema (normale funzionamento) ovvero invocazioni di servizi (funzioni) del SO
    - (3) Trap o eccezioni software (anche in seguito a chiamate di sistema)

Interruzioni (1)

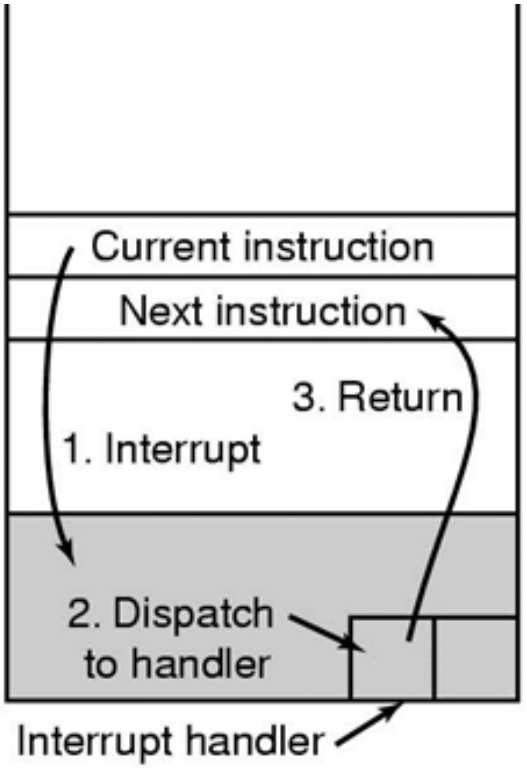
- Il sistema esegue sequenzialmente le istruzioni
- All'arrivo di un interrupt (I/O guidato da interrupt), la CPU invoca un "gestore" (*routine di servizio*) opportuno attraverso l'**interrupt vector**
  - Il **vettore di interrupt contiene gli indirizzi in memoria di tutte le routine di servizio**
- L'HW deve salvare l'indirizzo dell'istruzione interrotta, in modo tale che, una volta ultimata la gestione di interrupt, possa essere ripristinato.
- NOTA: **Interrupt in arrivo sono disabilitati mentre un altro interrupt viene gestito**, per evitare che vadano perduti



Interruzioni (2)

Passi necessari alla gestione di una interruzione:

1. CPU rileva l'interruzione e legge l'indirizzo del dispositivo (*device*) sul bus
   - salvataggio dei registri PC e PSW (stato della CPU) sullo stack, passaggio in modo kernel
2. L'indirizzo del dispositivo viene usato come indice nella tabella dei gestori delle interruzioni (*interrupt vector*)
3. Il gestore selezionato prende il controllo e svolge le operazioni necessarie
4. Quando il gestore termina:
   - ripristino PC e PSW , ritorno in modo utente
   - si esegue l'istruzione successiva a quella interrotta



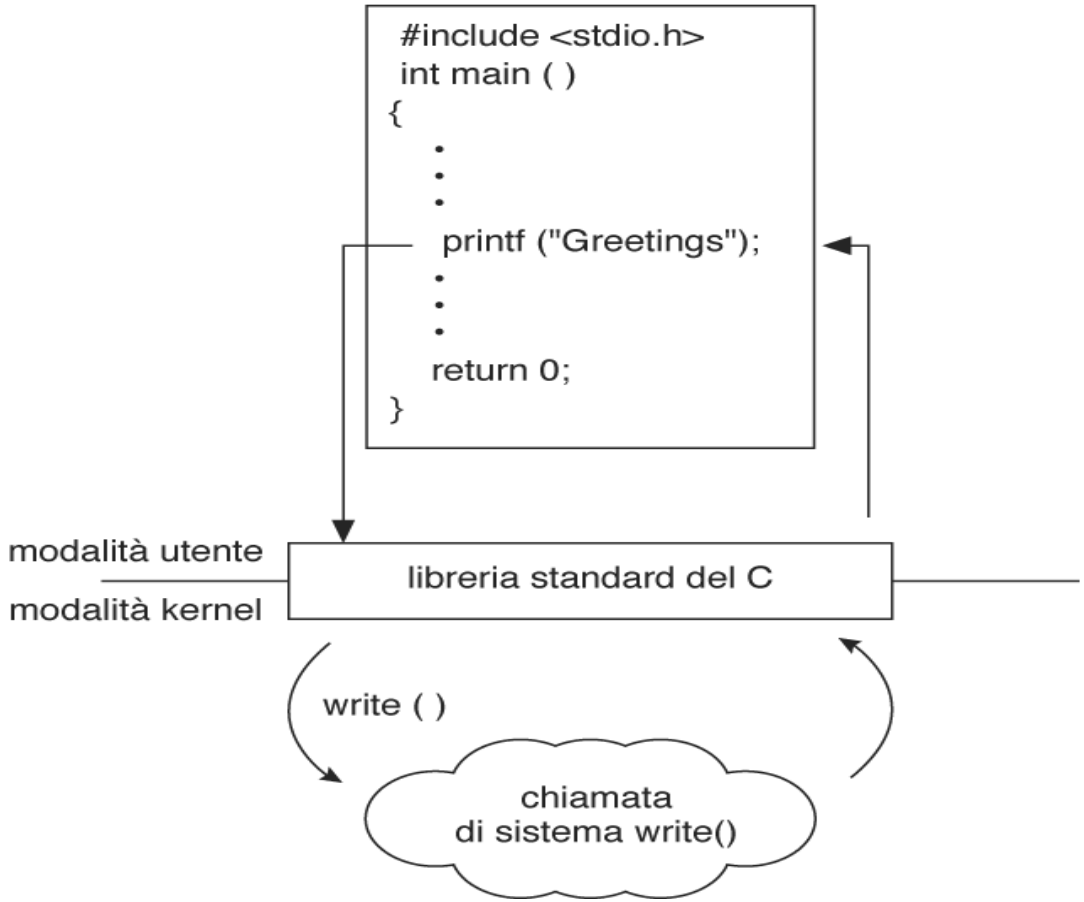
CHIAMATE DI SISTEMA

Chiamate di sistema (1)

- **Chiamata di sistema**: interazione elementare tra un programma utente e il SO
- Con le chiamate di sistema i programmi utente accedono ai **servizi del SO** disponibili come istruzioni in:
  - linguaggio **assembler**
  - linguaggi di programmazione di più alto livello come **C e C++**
    - permettono alle chiamate di sistema di avvenire direttamente
  - **Java non lo permette**
    - poiché una chiamata di sistema è specifica del SO e dà come risultato un codice specifico della piattaforma
    - tuttavia se richiesto, è possibile attraverso *metodi nativi*
      - Java può richiamare metodi scritti in un altro linguaggio (C o C++) che eseguono la chiamata di sistema

Chiamata di sistema (*System Call*) – esempio Standard C Library

- Un programma C che invoca la chiamata di libreria **printf()**, che a sua volta chiama la funzione di sistema **write()**



Progettare chiamate di sistema (1)

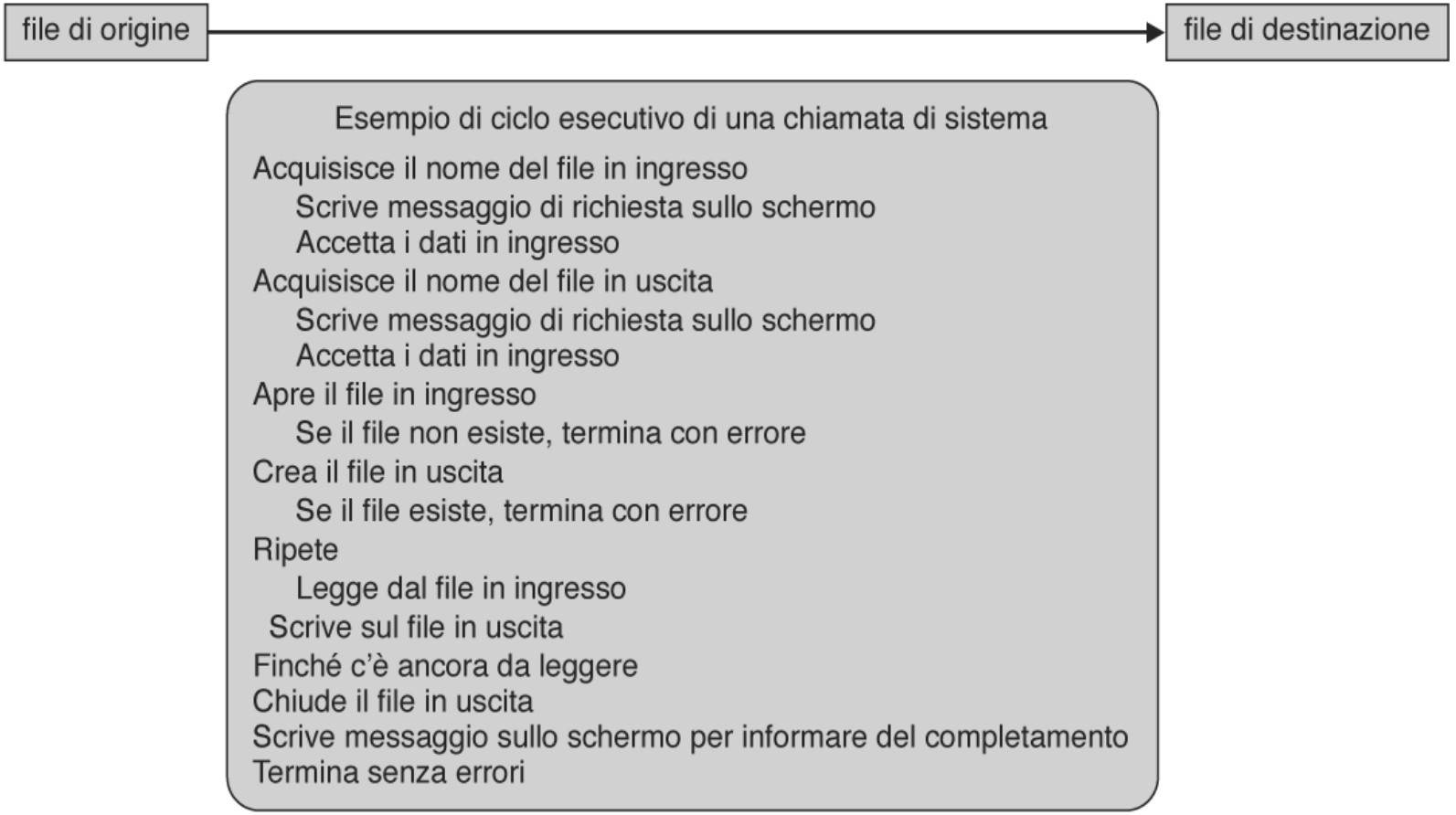
Esempio: programma per copiare un file in un altro file

Richiede diverse chiamate di sistema per:

- Ricavare i nomi dei due file
  - Tramite richiesta del nome
  - Tramite browser dei file (molte più chiamate)
- Aprire il file di lettura e creare il file di scrittura
  - Possibili errore: non esiste file input o accesso negato (come risolvo? Avviso l'utente e termino?)
  - Possibili errore: esiste già file output (come risolvo? sovrascrivo?)
- Lettura e scrittura
  - Possibili errori: end of file, memoria insufficiente per scivere
- Chiusura dei file, avviso utente, terminazione programma

Progettare chiamate di sistema (2)

- Esempio: chiamata di sistema per **copiare un file in un altro file**



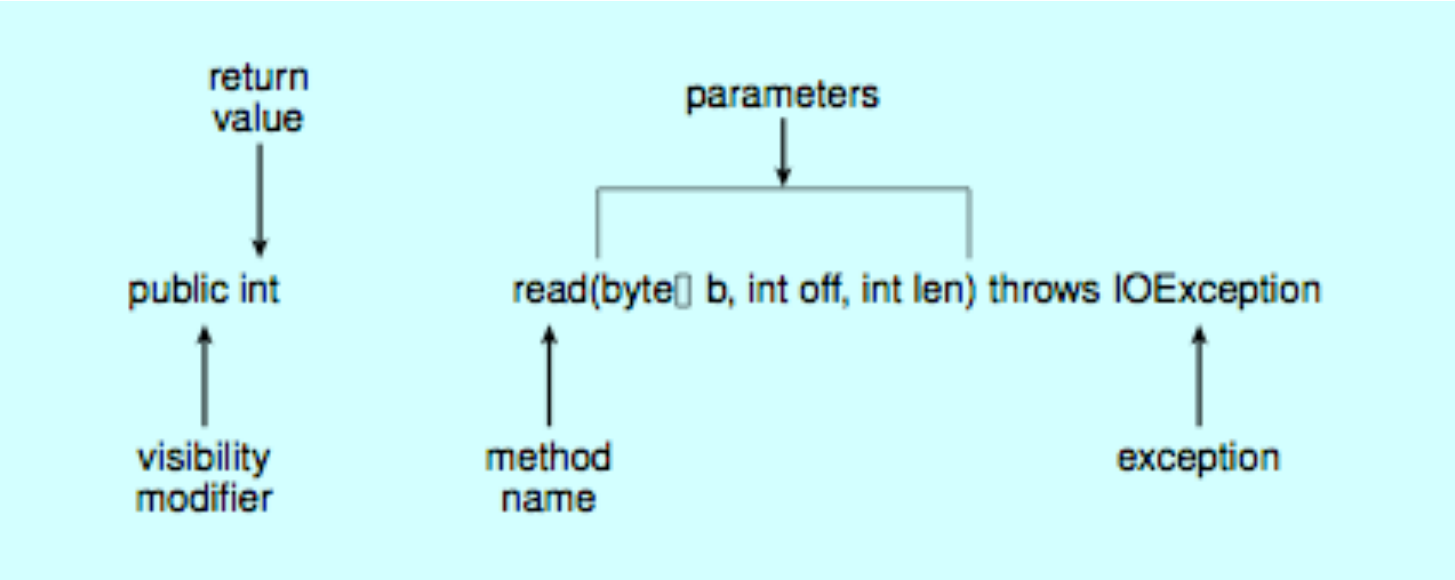
API per le chiamate di sistema

- Il programmatore ha davvero bisogno di sapere quali sono le chiamate di sistema per progettare un programma?

- Il programmatore tipicamente non invoca le chiamate di sistema ma delle funzioni messe a disposizione da un API
- È un'interfaccia verso le chiamate di sistema
- Motivi:
  - Portabilità
  - Semplicità

Progettare chiamate di sistema
Esempio di una API Standard

- Considera il imetodo Java **read()**



byte[] b – il buffer di destinazione dei dati letti

int off - offset iniziale in b dove vengono posti i dati

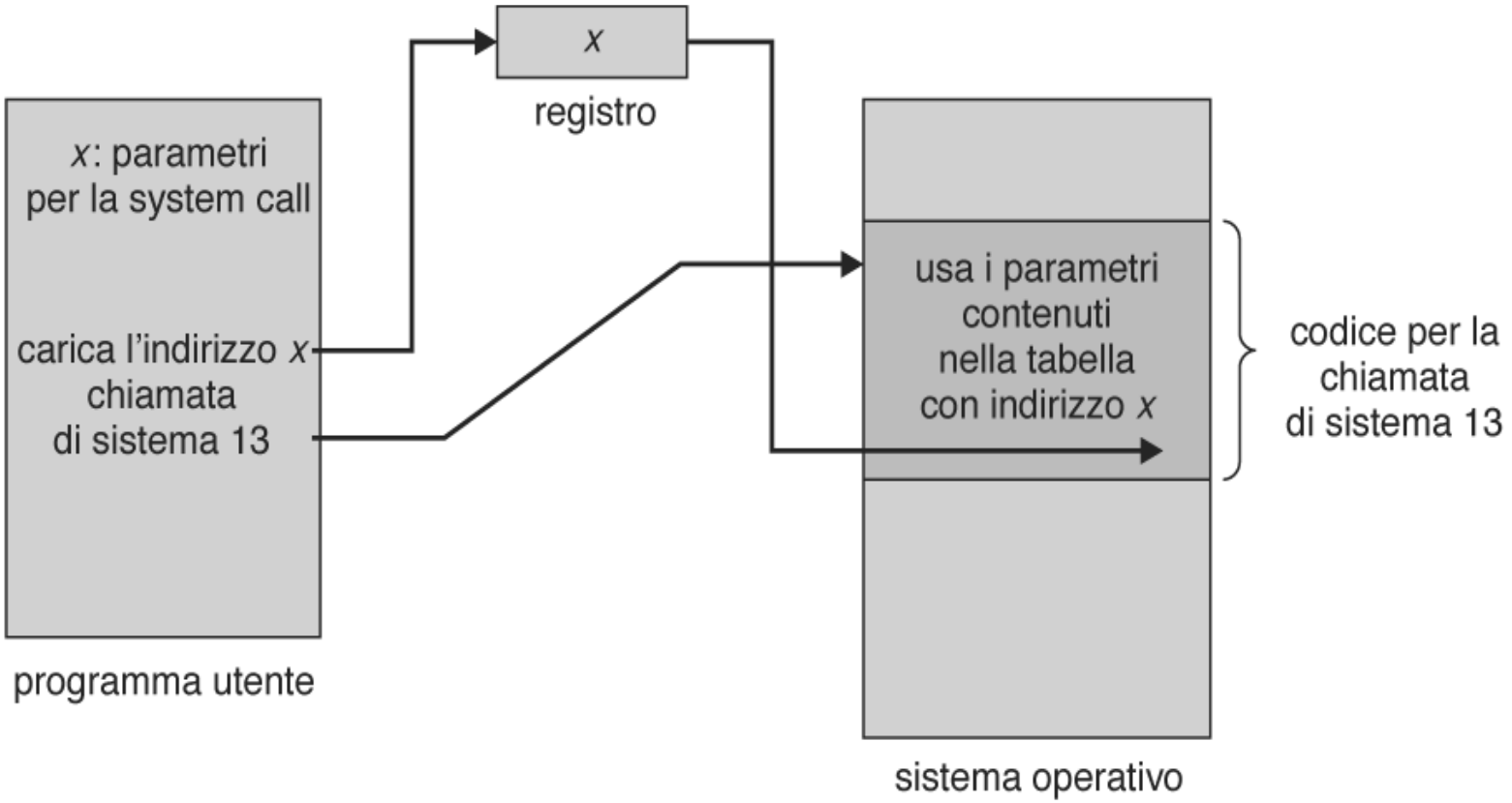
int len – il massimo numero di byte da leggere

Chiamate di sistema – passaggio dei parametri

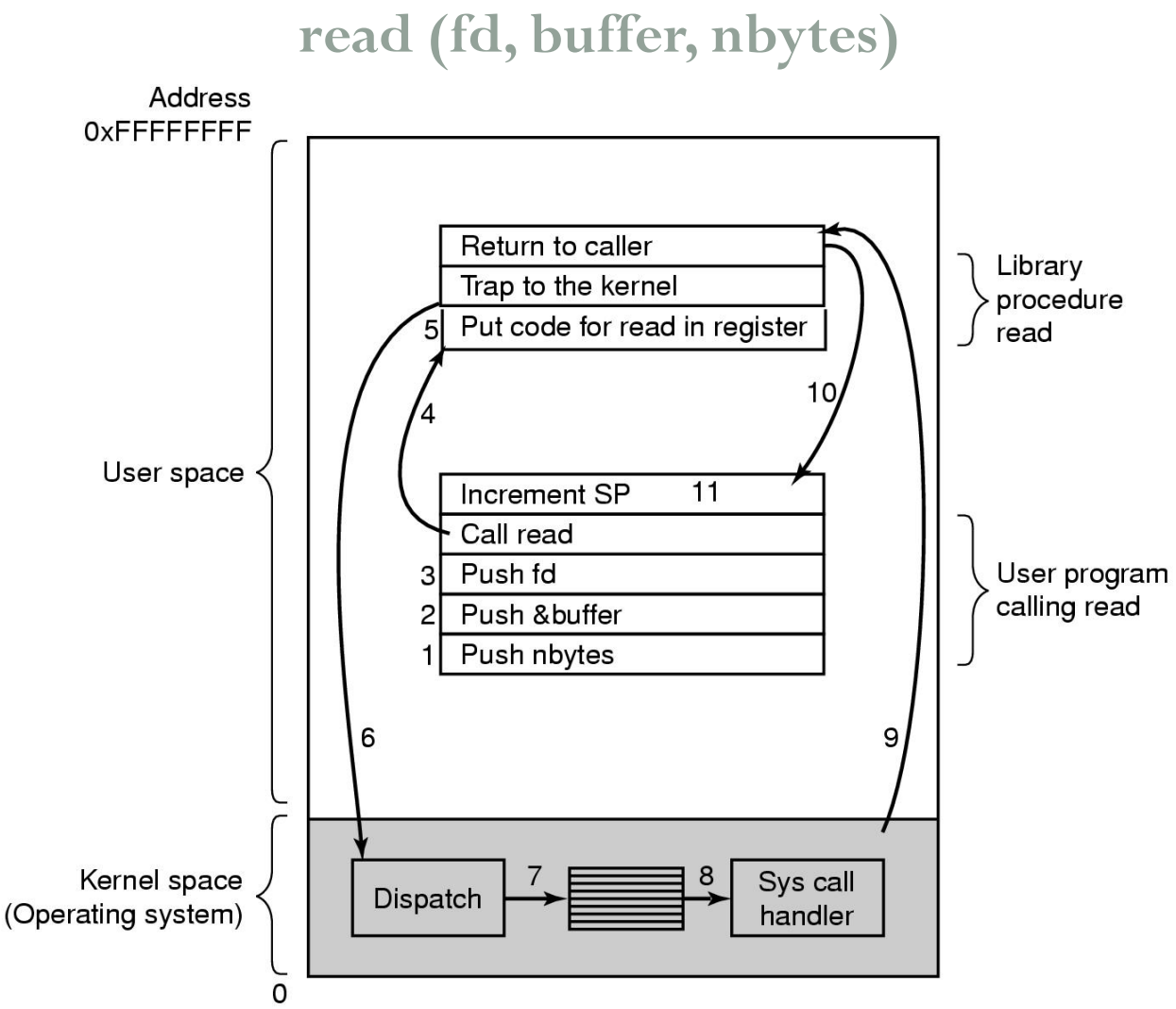
- Le chiamate di sistema possono richiedere lo scambio di informazioni (***parametri***) tra un programma e il SO
- Lo scambio dei parametri avviene **in generale in 3 modi**:
  1. Attraverso **registri**
     - attuabile solo se il numero dei parametri non supera quello dei registri
  2. Attraverso una memoria provvisoria detta **pila (stack)** posti dal programma e prelevati dal SO
     - Soluzione preferita, perché non limitano il numero o la lunghezza dei parametri

Chiamate di sistema – passaggio dei parametri (cont.)

3. Attraverso una **tabella in memoria**, e l'indirizzo X della tabella viene passato come parametro in un registro
   - Usato da Linux



Esempio di chiamata di Sistema (1)

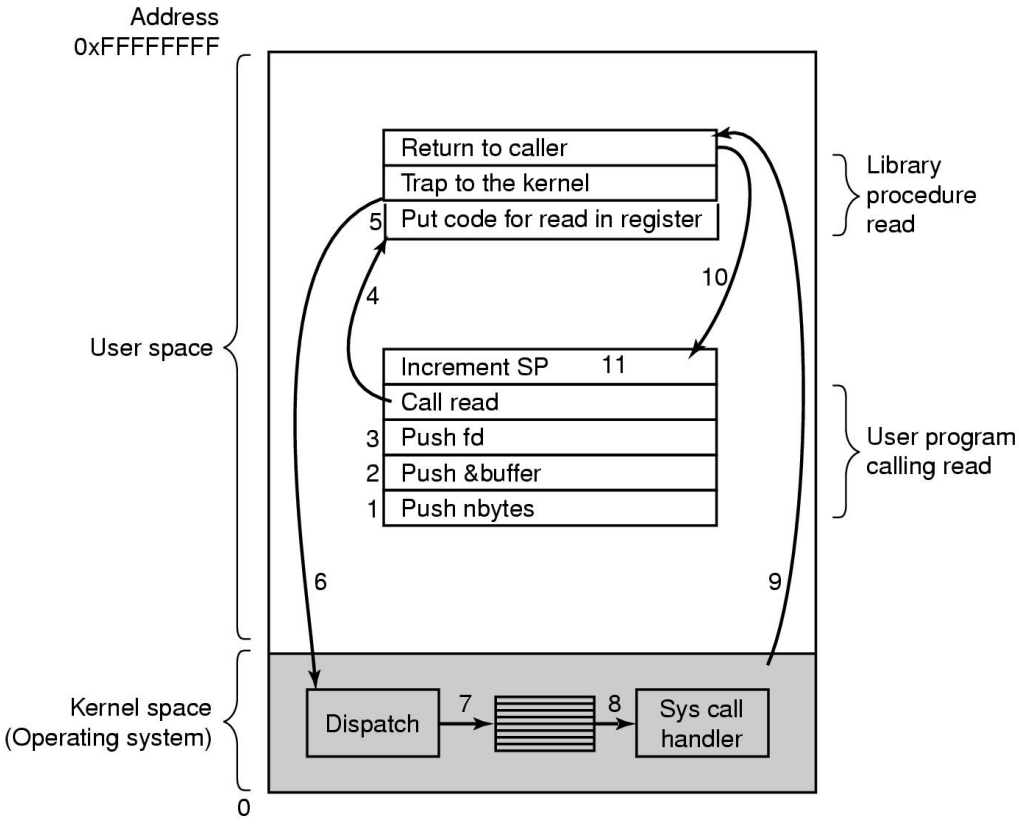


Esempio di chiamata di Sistema (2)

Dettaglio dell'esecuzione di
read (fd, buffer, nbytes)

1-3. Salvataggio dei parametri sullo stack

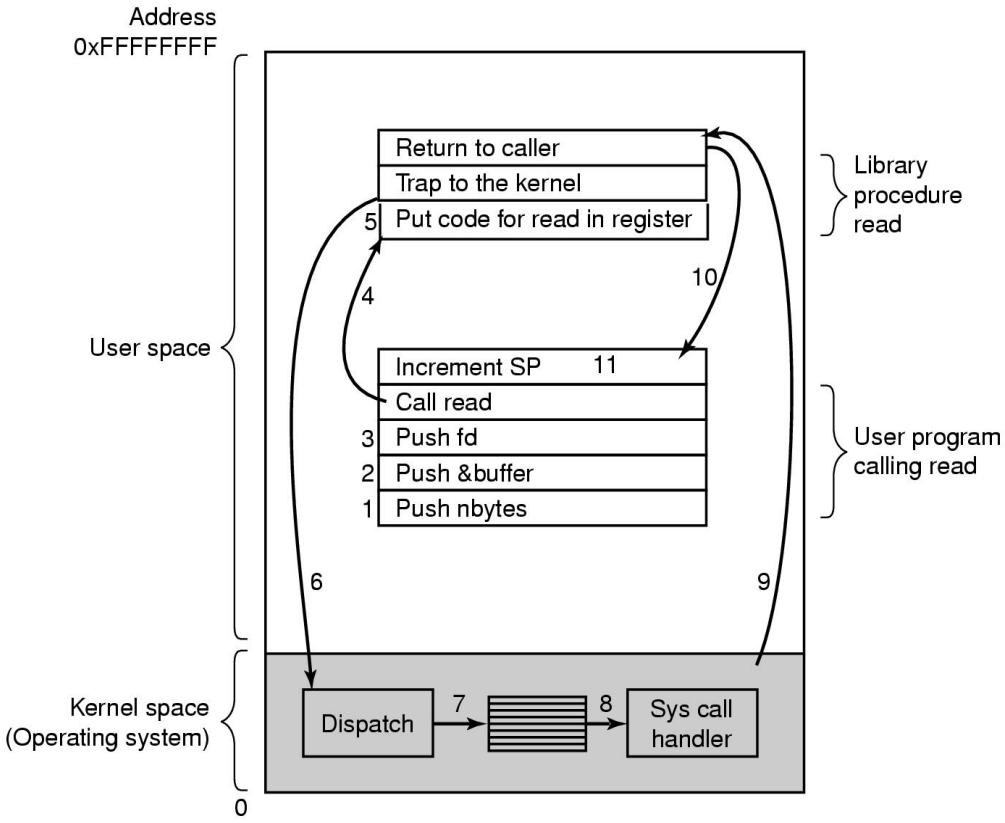
4. Chiamata della funzione di libreria *read*



Esempio di chiamata di Sistema (3)

Dettaglio dell'esecuzione di **read (fd, buffer, nbytes)**

5. Caricamento del codice della system call in un registro fissato Rx
6. Esecuzione **TRAP**
   - passaggio in *kernel mode*, salto al codice del *dispatcher*

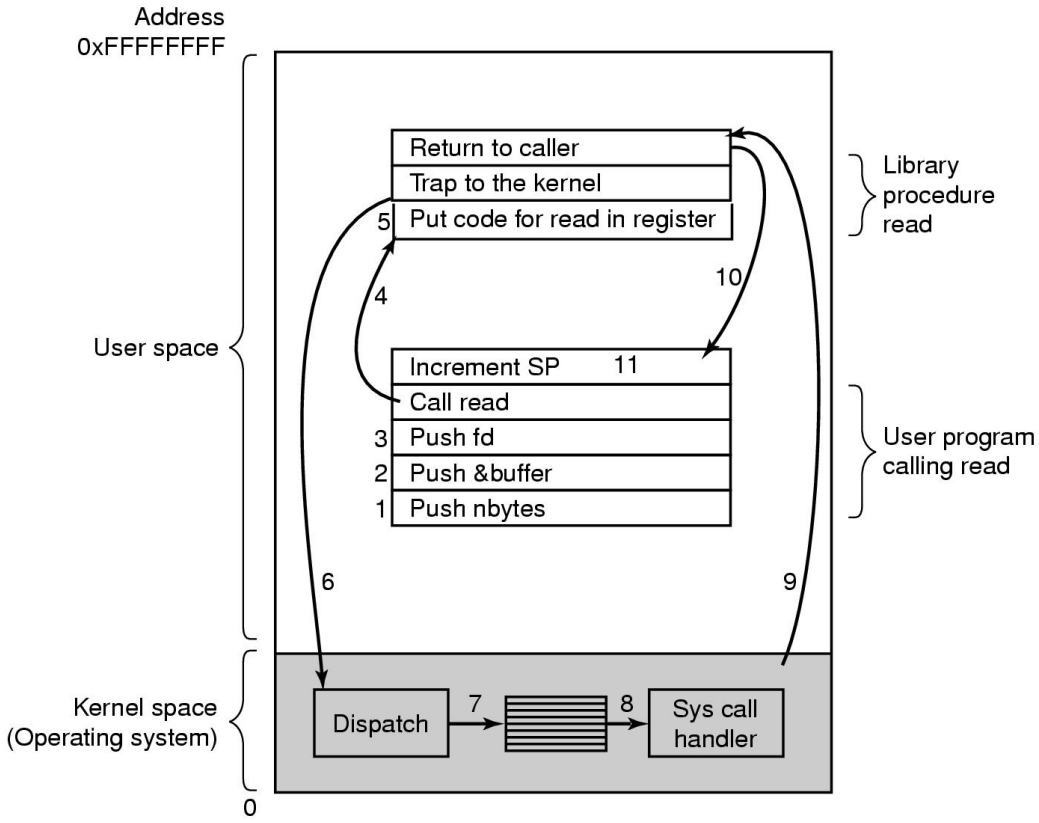


Esempio di chiamata di Sistema (4)

Dettaglio dell'esecuzione di
read (fd, buffer, nbytes)

7-8. Selezione della system call secondo il codice in Rx ed invocazione del gestore appropriato

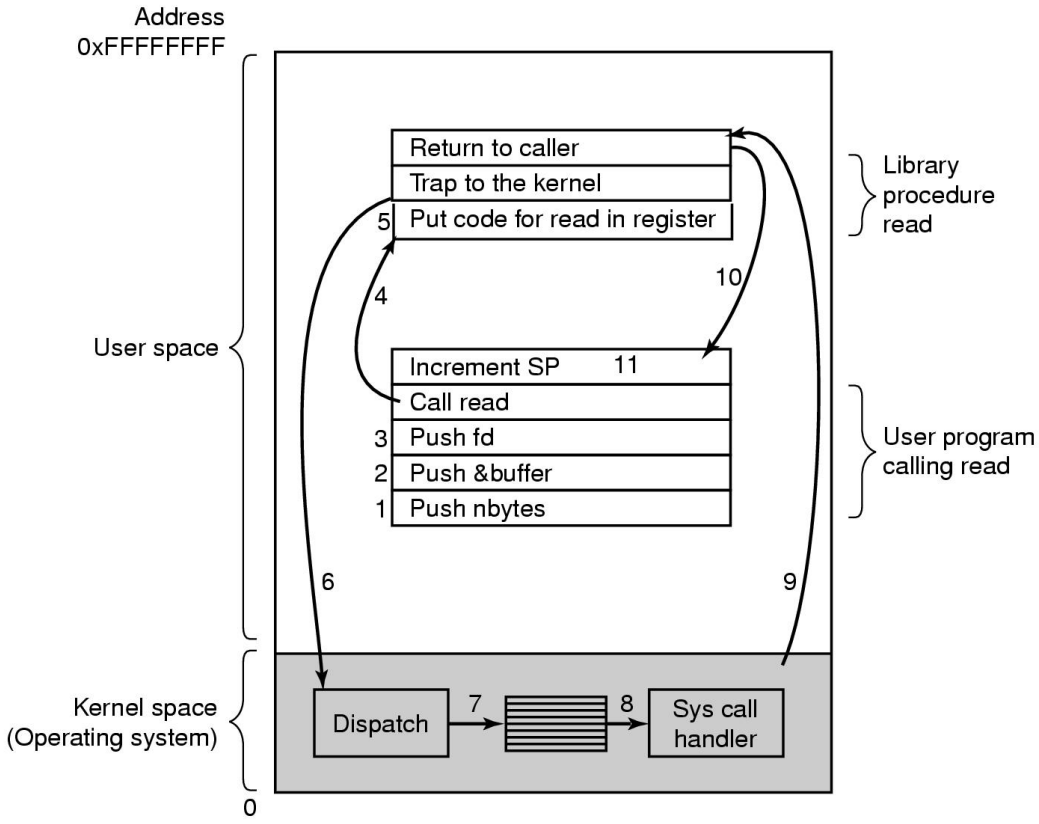
9. Ritorno alla funzione di libreria
   - ripristino user mode, caricamento del program counter PC



Esempio di chiamata di Sistema (5)

Dettaglio dell'esecuzione di
read (fd, buffer, nbytes)

10-11. Ritorno al codice utente (nel modo usuale) ed incremento dello stack pointer SP per rimuovere I parametri della chiamata a read



Chiamate di sistema

- Controllo dei processi
  - terminazione normale e anormale
  - caricamento, esecuzione
  - creazione e arresto di un processo
  - esame e impostazione degli attributi di un processo
  - attesa per il tempo indicato
  - attesa e segnalazione di un evento
  - assegnazione e rilascio di memoria
- Gestione dei file
  - creazione e cancellazione di file
  - apertura, chiusura
  - lettura, scrittura, posizionamento
  - esame e impostazione degli attributi di un file
- Gestione dei dispositivi
  - richiesta e rilascio di un dispositivo
  - lettura, scrittura, posizionamento
  - esame e impostazione degli attributi di un dispositivo
  - inserimento logico ed esclusione logica di un dispositivo
- Gestione delle informazioni
  - esame e impostazione dell'ora e della data
  - esame e impostazione dei dati del sistema
  - esame e impostazione degli attributi dei processi, file e dispositivi
- Comunicazione
  - creazione e chiusura di una connessione
  - invio e ricezione di messaggi
  - informazioni sullo stato di un trasferimento
  - inserimento ed esclusione di dispositivi remoti

System call / API di Unix

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

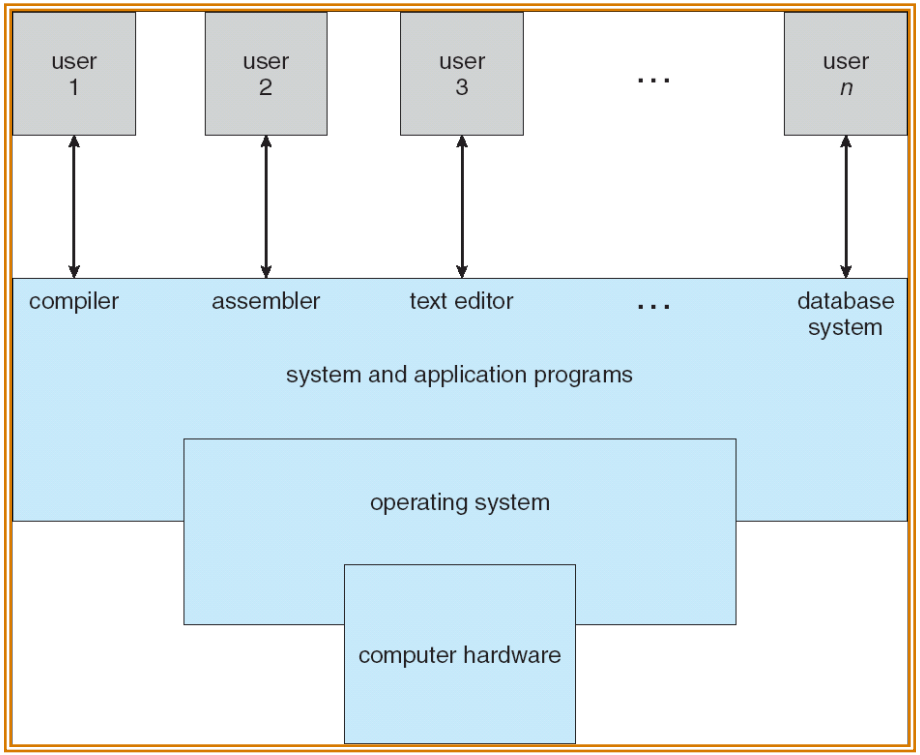
System call / API di Windows

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

PROGRAMMI DI SISTEMA E PROGRAMMI UTENTE

Programmi di sistema

- I programmi di sistema sono il punto di contatto tra il Sistema Operativo e l'utente
- Possono essere semplici interfacce verso le chiamate di sistema oppure complessi programmi



Programmi di sistema

- La visione del SO offerta agli utenti comuni è definita in genere dai **programmi di sistema** piuttosto che dalle effettive chiamate di sistema (che interessano ai programmatori di sistema)
- I programmi di sistema forniscono un ambiente opportuno per la **gestione delle risorse** e lo **sviluppo di applicazioni**
  - Gestione dei file
  - Informazioni di stato (data, ora, spazio su disco, ecc.., ma anche prestazioni e debug)
  - Modifica dei file (es. programmi elaborazioni testo)
  - Supporto ai linguaggi di programmazione (compilatori, assemblatori, debugger, ecc..)
  - Caricamento ed esecuzione dei programmi
  - Comunicazioni (posta elettronica, login remoto, ecc..)

SO e programmi utente (1)

- Il **software utente** può dunque essere:
  1. **applicativo** o
  2. **di sistema** cioè "di base" (es. shell)
- **La seconda categoria non va confusa con il SO!**

- L'utente quindi utilizza i programmi di sistema e non le chiamate di sistema.
- Una GUI e una shell offrono due modalità di I/O differenti, ma usano le stesse chiamate di sistema

SO e programmi utente (2)

- Differenze tra sw utente e il SO:
  - il sw utente, anche se di sistema, non accede direttamente alle risorse fisiche
  - il sw utente è eseguito con la CPU in **user mode** (se disponibile); ciò impedisce:
    - l'accesso diretto alle risorse fisiche, nonché
    - manomissioni del SO
- viceversa il SO, eseguito con la CPU in **kernel mode**, ***non*** ha limiti nell'accesso all'hardware

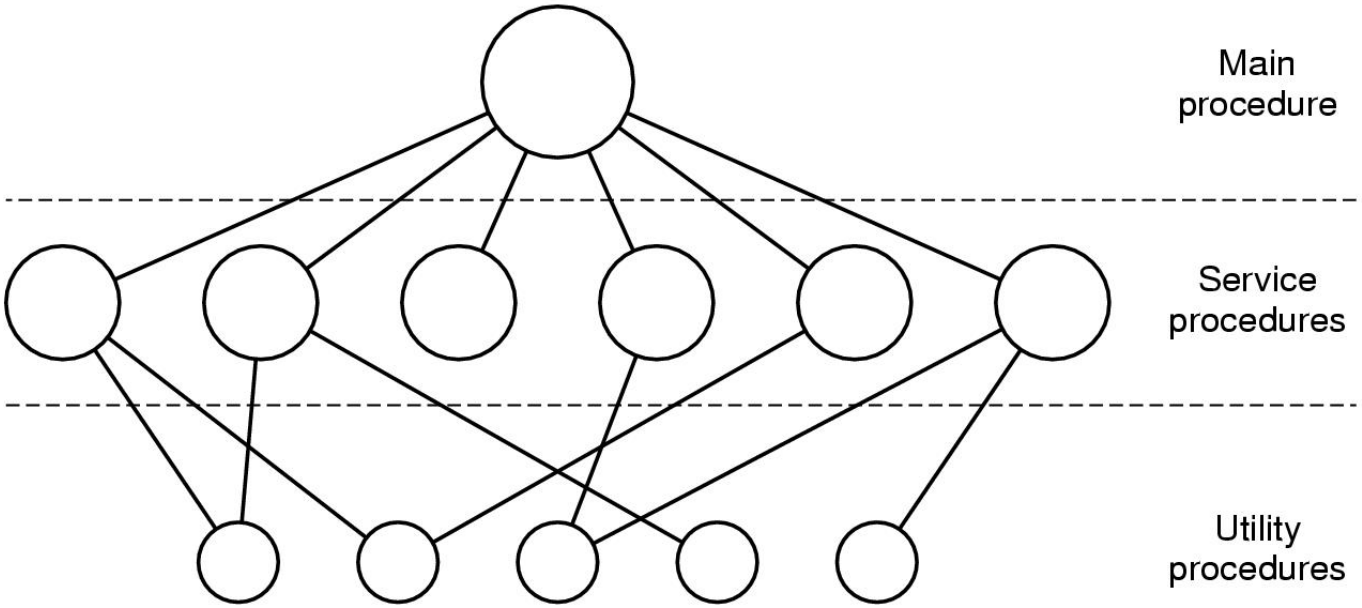
STRUTTURA DI UN SO

Struttura di un sistema operativo

- Come sono organizzate le varie componenti di un SO?
- Questa scelta influenza fortemente la progettazione del SO!
  - Sistema monolitico
  - Sistema stratificato
  - Sistema a microkernel
  - Sistema a moduli funzionali
  - Sistema a macchine virtuali
  - Sistema client/server

Struttura monolitica (1)

- Alcuni SO commerciali non hanno una struttura ben definita
  - procedure di servizio compilate in un unico oggetto
  - ogni procedura può chiamare tutte le altre
  - ogni processo esegue parzialmente in modo kernel
  - system call bloccanti



Struttura monolitica (1)

- Molti SO commerciali non hanno una struttura ben definita
  - procedure di servizio compilate in un unico oggetto
  - ogni procedura può chiamare tutte le altre
  - ogni processo esegue parzialmente in modo kernel
  - system call bloccanti

- Che problemi comporta tutto ciò?
  - Manutenzione difficile
  - Vulnerabilità agli errori e agli attacchi
  - Blocco del sistema
  - La protezione fornita dal SO non è sufficiente

Struttura monolitica (3)

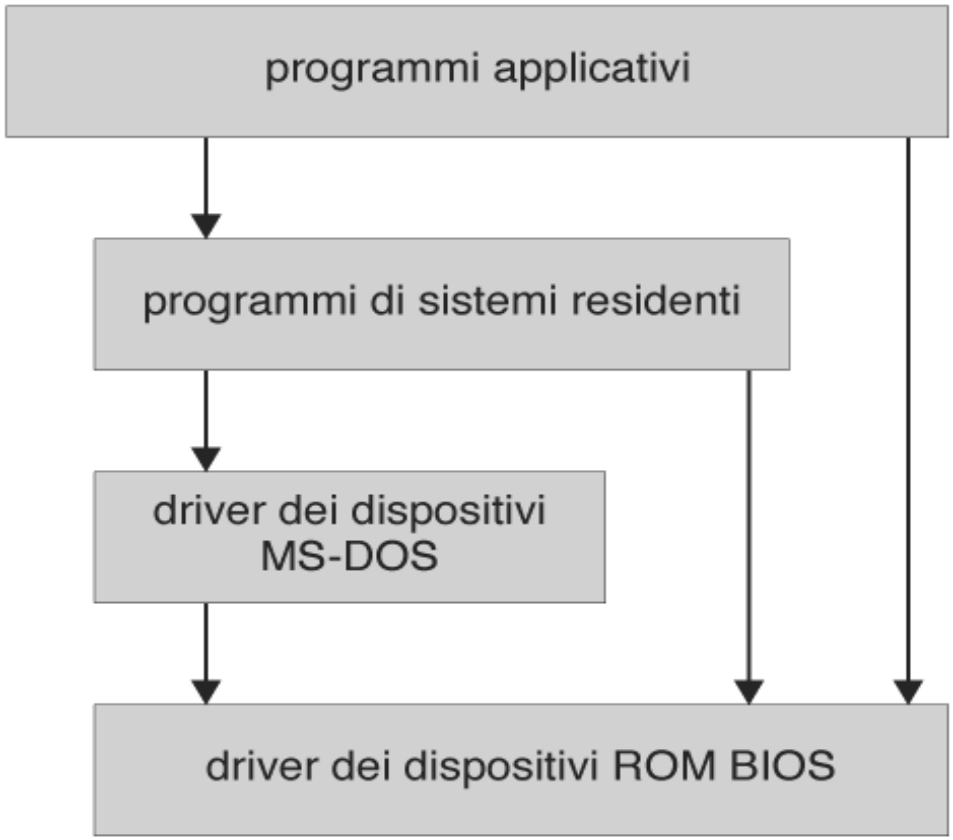
- Spesso nascono come sistemi piccoli, semplici e limitati, per poi crescere al di là dello scopo originario

- L'**MS-DOS** ne è un esempio
  - Funzionava su un hardware limitato (l'8088 dell'Intel)
  - SO molto vulnerabile perché lascia libertà ai programmi applicativi di accedere direttamente all'hardware (non ha modalità utente)

- Un altro esempio di SO con struttura semplice è l'originario **Unix**
  - anch'esso inizialmente limitato dall'hardware

La struttura dell'MS-DOS

- Le interfacce e i livelli di funzionalità non sono ben separati
  - Ad es. i programmi applicativi possono accedere alle procedure di I/O di base per scrivere direttamente sullo schermo e sui dischi

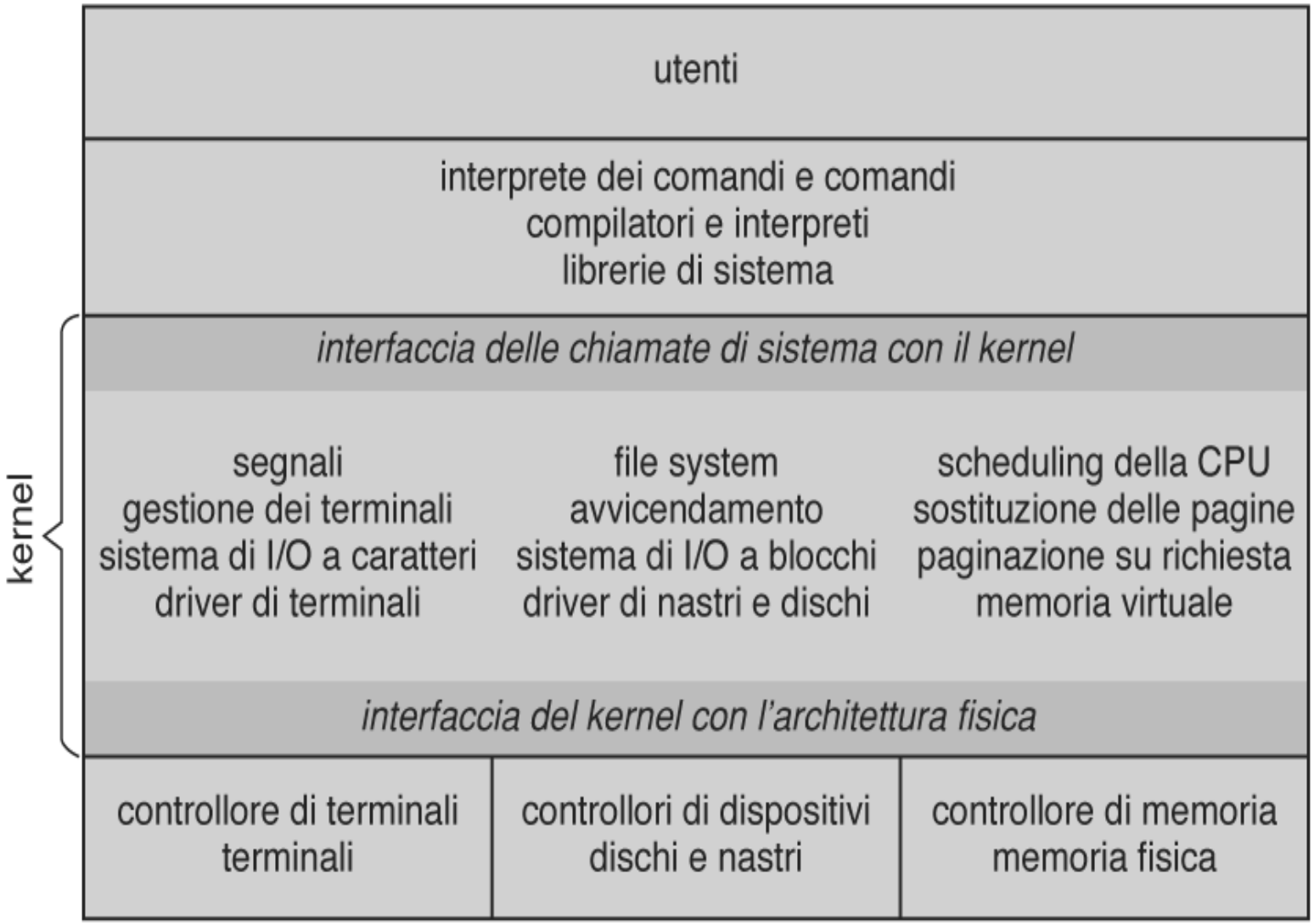


Struttura del sistema UNIX (1)

UNIX consiste di **due parti separabili**:

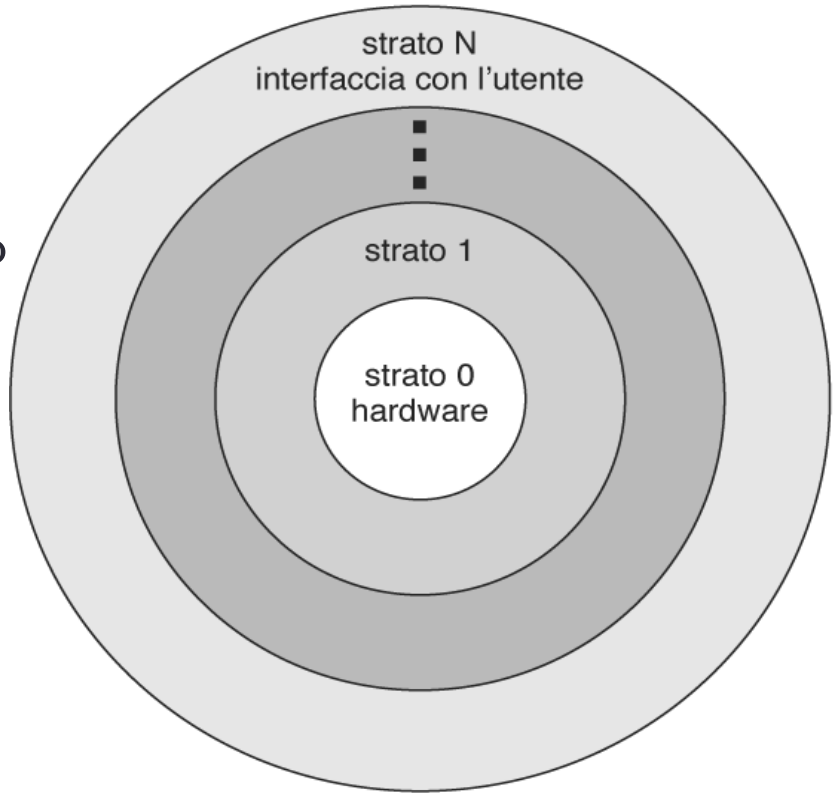
1. **Programmi di sistema**
2. **Nucleo** (*kernel):*
   - Consiste in qualsiasi parte si trovi sotto l'interfaccia di chiamata di sistema e sopra l'hardware fisico
   - Un'enorme quantità di funzioni combinate in un solo livello:
     - gestione del file-system, schedulazione della CPU, gestione della memoria, ecc..

Struttura del sistema UNIX (2)



Struttura stratificata (1)

- Separazione gerarchica o a livelli delle funzioni
- Il SO è diviso in un certo numero di strati (livelli, o *layers), ognuno costruito sulla sommità dello strato inferiore*
- *Lo* strato inferiore (il livello 0) è l'hardware; quello più elevato (il livello N) è l'interfaccia utente
- L'interfaccia degli strati è stabile
- L'implementazione può variare



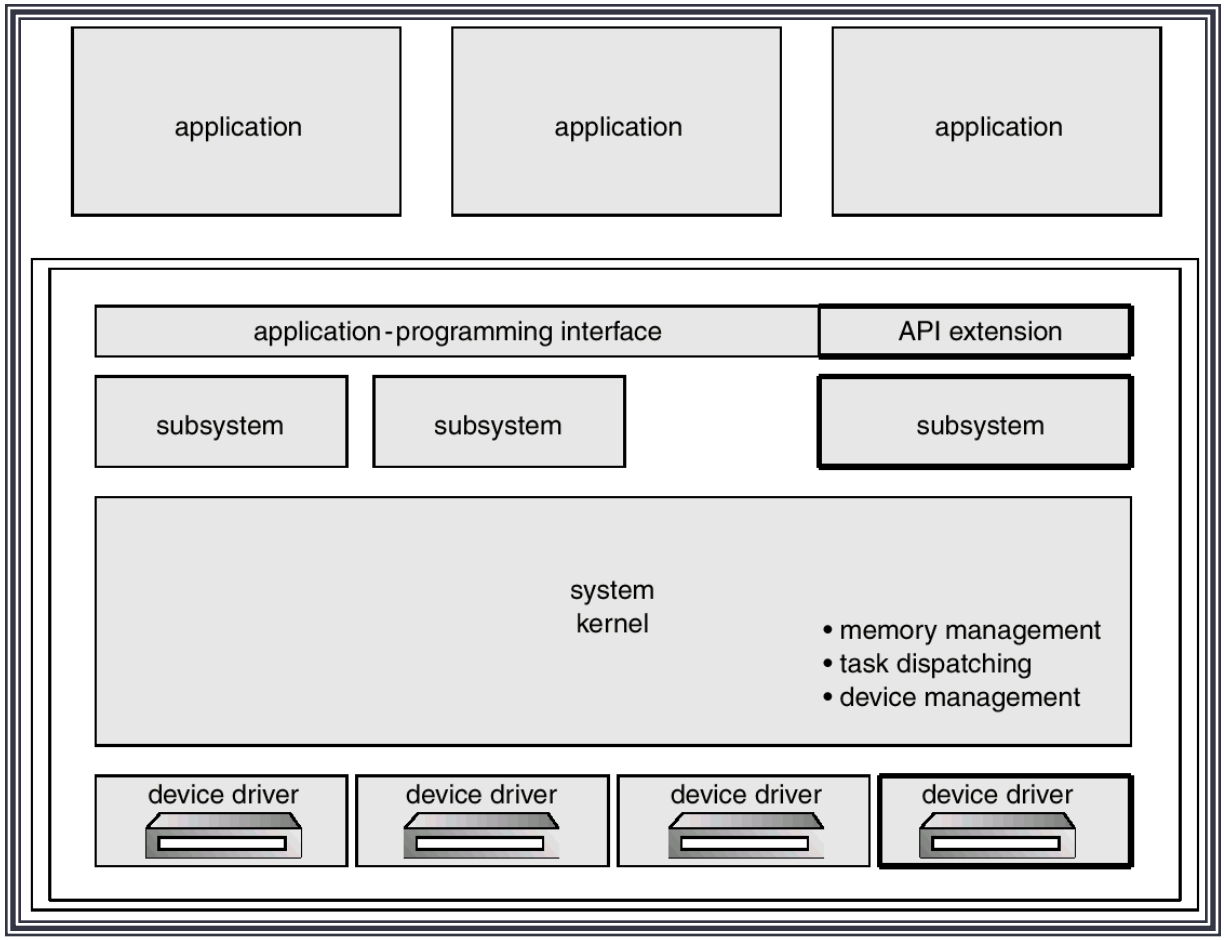
Struttura stratificata (2)

- **Vantaggi/svantaggi**:

- Ogni strato offre una virtualizzazione di un certo numero di funzioni (macchina virtuale)
- La dipendenza dall'hardware è limitata al livello più basso (portabilità)
- La portabilità si paga con minor efficienza perché una chiamata di sistema deve attraversare più strati
  - Con eventuale adattamento dei dati passati
- Più difficile da progettare

La struttura a strati di IBM OS/2

- Una nicchia nel settore bancario, ma attualmente lo sviluppo di IBM OS/2 è congelato ed il supporto tecnico è cessato (dal 31 Dic 2006)



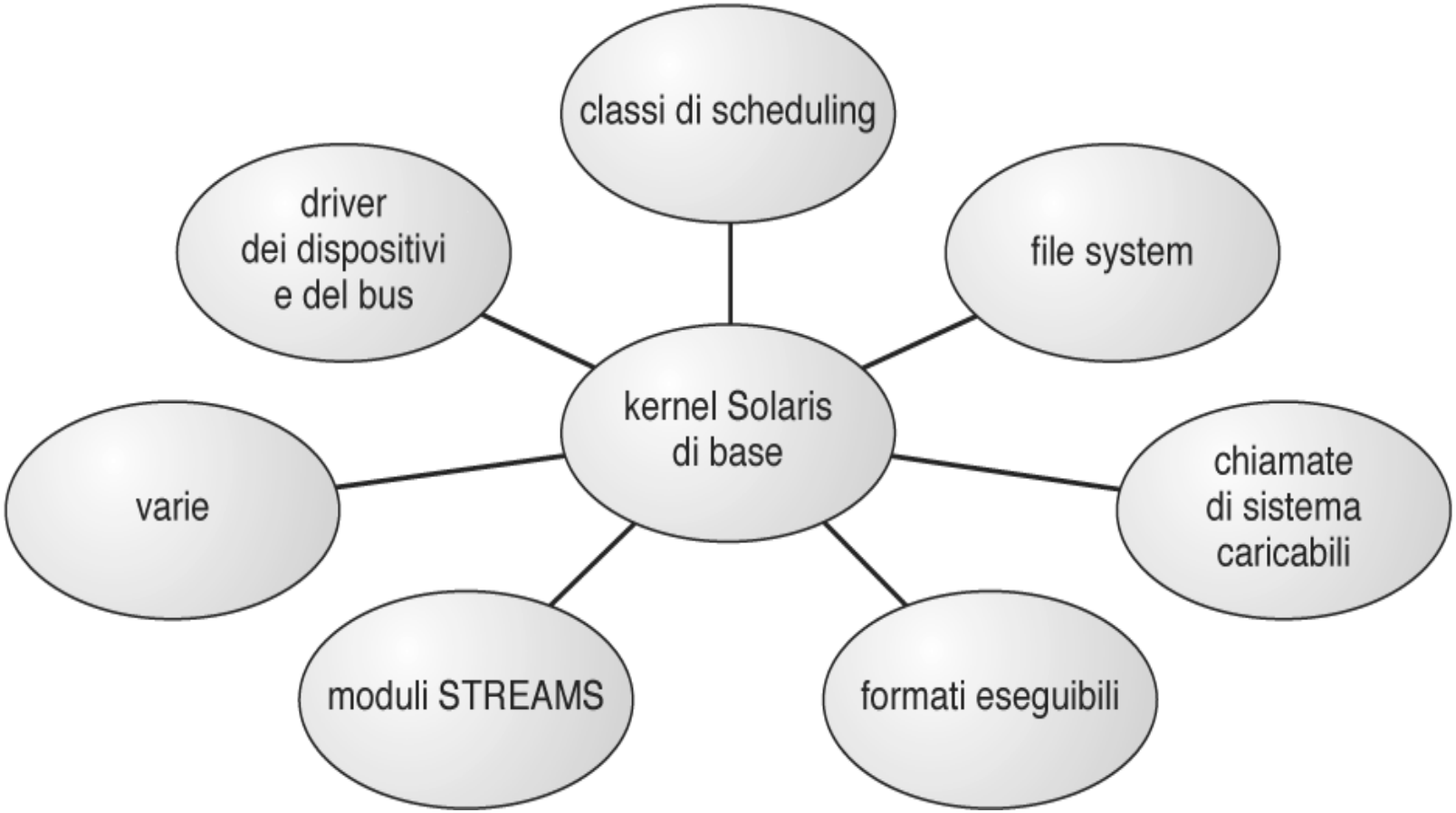
Struttura a microkernel

- Sposta il maggior numero di funzionalità possibili **dal kernel allo spazio utente** (come *programmi di sistema*)
- Il microkernel offre solo servizi per gestire **processi, memoria e comunicazione.**
- La comunicazione tra moduli avviene tramite **scambio di messaggi**
- **Vantaggi/svantaggi**:
  - Facilità di estendere il SO (lasciando invariato il microkernel)
  - Maggiore portabilità del SO da un'architettura HW a un'altra
  - Maggiore affidabilità e sicurezza (meno codice viene eseguito in modalità kernel)
  - I microkernel possono soffrire di calo di prestazioni per l'aumento di sovraccarico di funzioni di sistema

Struttura modulare

- Moderna metodologia di progetto che realizza **kernel modulari**
  - attraverso tecniche object-oriented
  - ogni componente core è separata e interagisce con le altre attraverso interfacce ben note
    - A differenza della struttura a stati i moduli possono comunicare con tutti gli altri, non solo con quelli di livello inferiore
  - **ogni componente è caricabile dinamicamente nel kernel** al momento del boot e/o durante l'esecuzione
- Simile alla struttura a strati, ma con maggiore flessibilità
- **Simile alla struttura a microkernel ma più efficiente perché i moduli non richiedono l'invio di messaggi per la comunicazione**

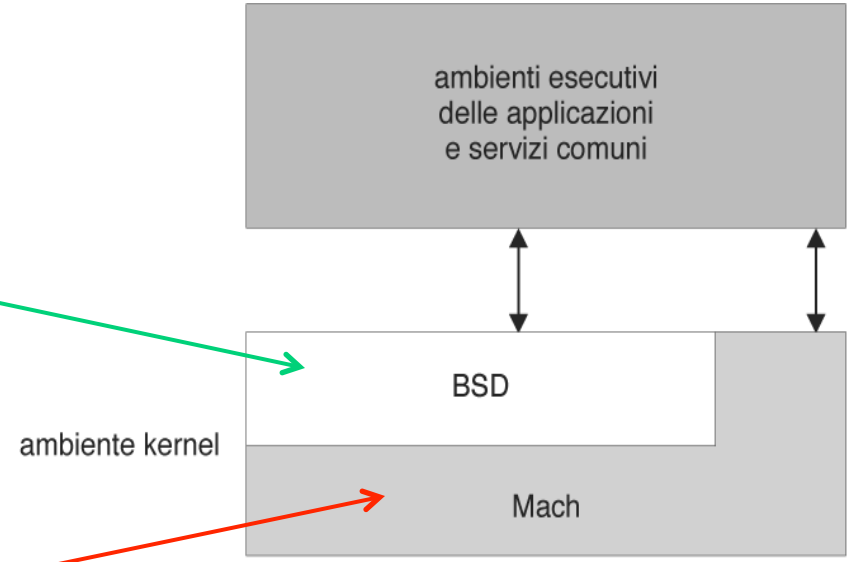
I moduli caricabili di Sun Solaris



La struttura di Mac OS X

Struttura ibrida

- Struttura a 2 strati**: microkernel Mach** (by Carnegie Mellon University, alla base di molti sistemi commerciali), e il **kernel modulare BSD** (Berkely SW Distribution)
  - Le applicazioni e i servizi comuni possono però accedere ad entrambi
- In aggiunta a Mach e a BSD, esistono ***anche moduli caricabili dinamicamente*** (*estensioni del kernel*)



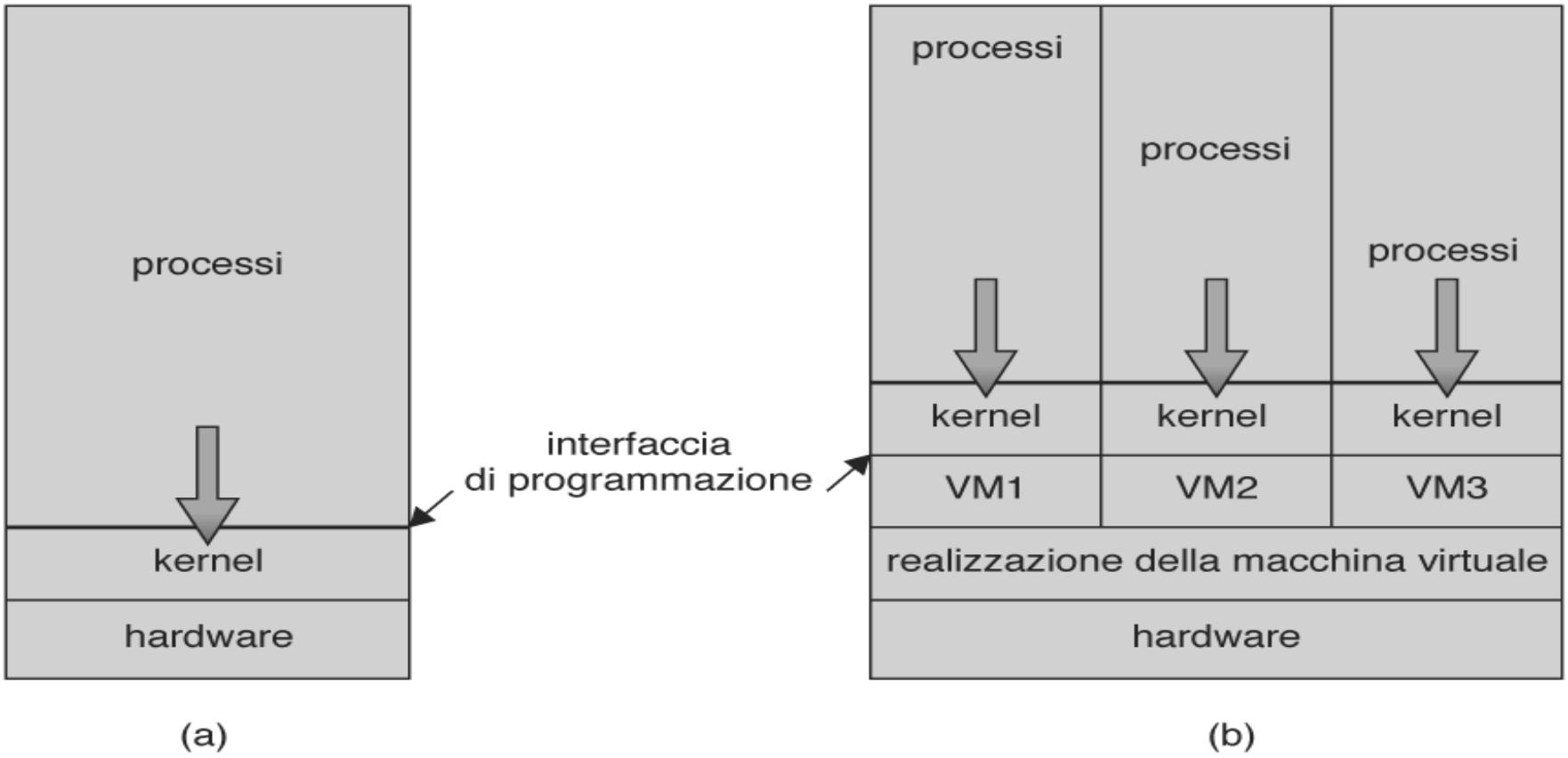
Interprete a linea di comando, API POSIX, servizi per il file system e la rete

Gestione memoria, comunicazione tra processi (scambio dei messaggi), e scheduling

Struttura a macchine virtuali (1)

- L'approccio a strati trova il suo logico sbocco nel concetto di *macchina virtuale*
  - Costruzione gerarchica di macchine astratte (o virtuali) in esecuzione su uno stesso hardware
  - Una macchina virtuale fornisce un'interfaccia software che è identica al puro hardware sottostante
- Il SO crea l'illusione che un processo ospite abbia un proprio processore ed una propria memoria (virtuale)
- Il processo ospite è tipicamente un SO

Struttura a macchine virtuali (2)



Non-virtual Machine

Virtual Machine

Ogni macchina virtuale può eseguire il proprio SO!

Struttura a macchine virtuali (3)

- **Il computer fisico mette in condivisione le proprie risorse per generare macchine virtuali**

- La schedulazione della CPU e la memoria virtuale può dare l'impressione che ogni *processo ospite* (solitamente un SO) abbia un proprio hardware dedicato – *virtual HW*
- Lo spooling e il file system possono fornire lettori di schede e stampanti in linea virtuali, e altri dispositivi – *virtual devices, virtual memory*

Struttura a macchine virtuali (4)

Vantaggi

- Comporta la **completa protezione delle risorse** di sistema perché ogni macchina virtuale è isolata da tutte le altre
  - Questo isolamento, tuttavia, non permette la condivisione diretta delle risorse
- **Una VM è uno strumento perfetto per l'emulazione di altri SO**
  - utile per avere una macchina multi-boot
- **Una VM è uno strumento per sviluppare/debug di nuovi SO**
  - Tutto si svolge sulla macchina virtuale, invece che su quella fisica, quindi non c'è pericolo di causare danni
  - Le normali operazioni di sviluppo e manutenzione del sistema non rendono il calcolatore inutilizzabile
- Possibilità di distribuire applicazioni attraverso le macchine virtuali (richiede standardizzazione)

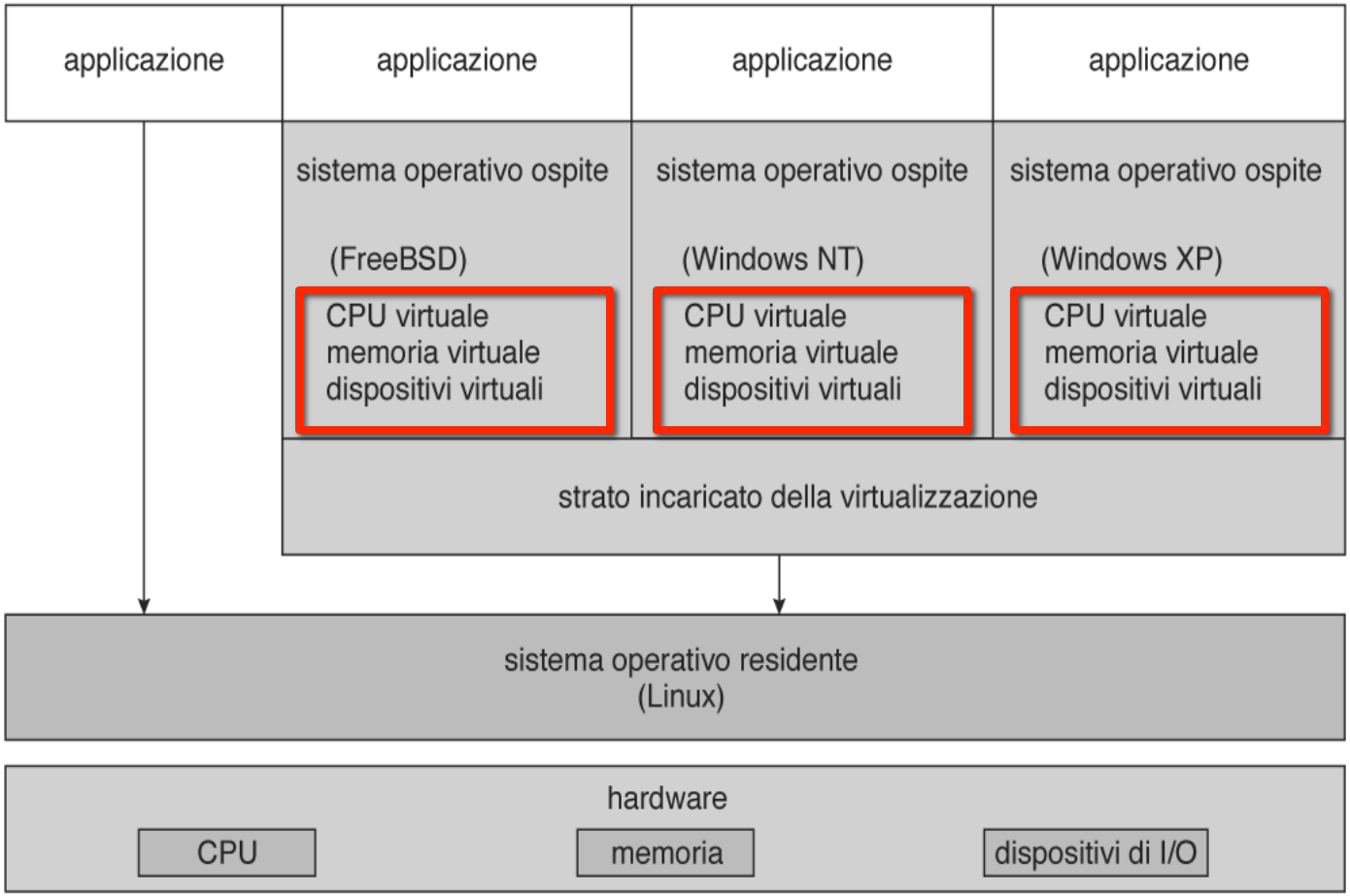
Struttura a macchine virtuali (5)

Svantaggi

- **Difficile da implementare** in quanto richiede tecniche sofisticate di virtualizzazione per creare un esatto duplicato della macchina sottostante
- **Prestazioni limitate in origine, non ora**
  - grazie ai progressi delle tecniche di virtualizzazione per infrastrutture di Cloud Computing
  - E al supporto hardware (alcuni processori offrono supporto HW per la virtualizzazione)
  - AMD offre due ulteriori modalità per la CPU
    - Host: modalità kernel virtuale
    - Guest: modalità utente virtuale

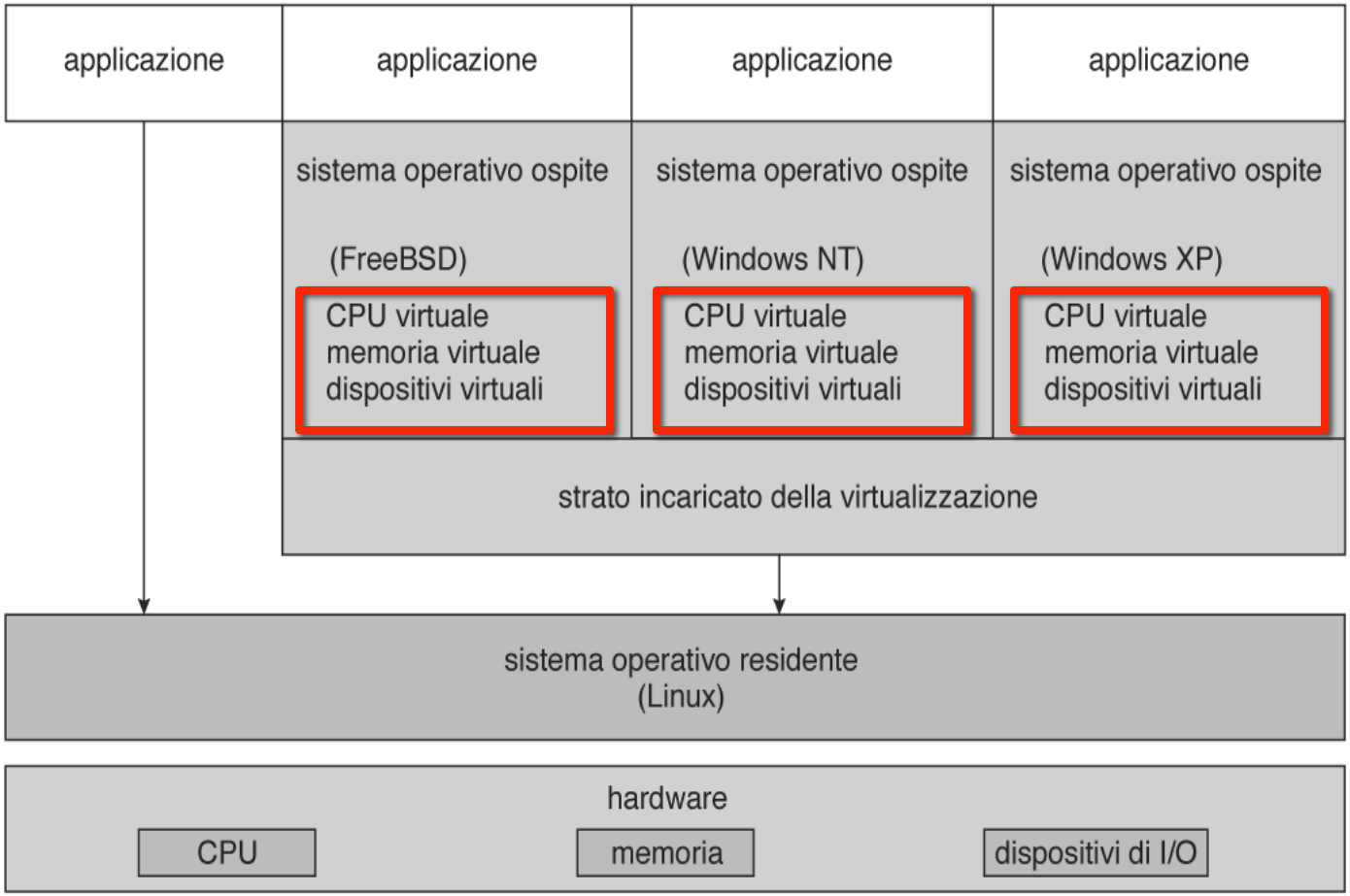
Architettura VMware

- Lo strato di virtualizzazione è scritto per funzionare in modalità utente
- Trasforma HW in macchine virtuali che ospitano SO



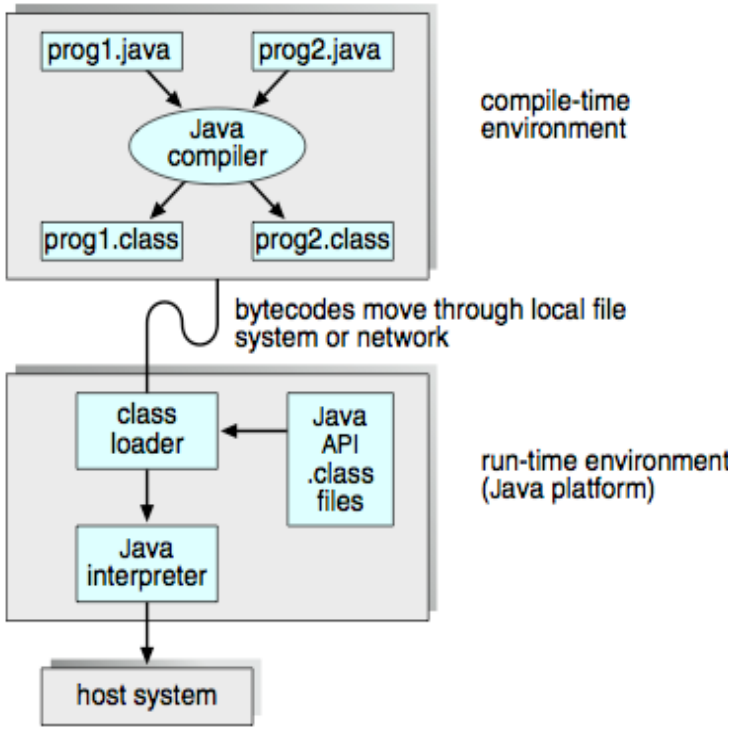
Architettura Vmware

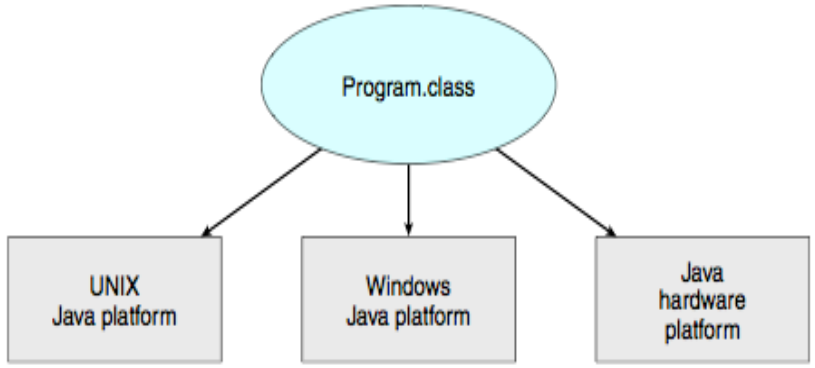
- Il disco virtuale è in realtà un file
- Si possono cosi creare snapshot



La Java Virtual Machine (JVM) (1)

- I programmi Java compilati sono bytecode indipendenti dall'architettura ed eseguiti da una Java Virtual Machine (JVM)
- La JVM è un calcolatore astratto su un sistema ospitante vero



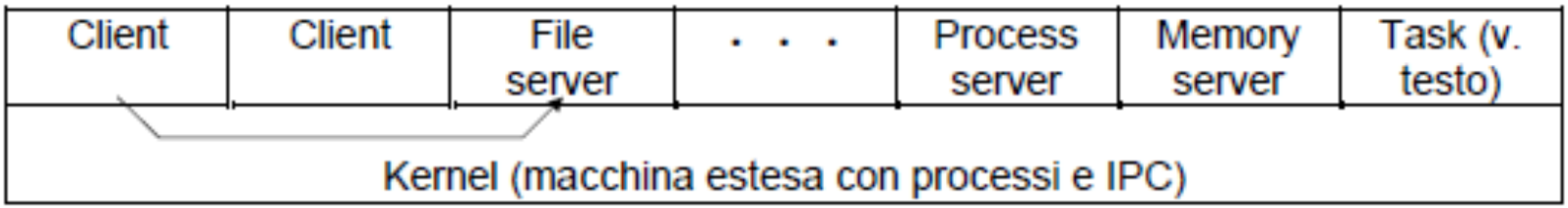


La Java Virtual Machine (JVM) (1)

- Come viene gestita la memoria?
  - Garbage collector (invisibile all'utente a differenza di C++)
- La JVM utilizza un interprete puro per eseguire il codice? Questo influenza le prestazioni?
  - In realtà non eccessivamente, utilizza un Just In Time compiler (JIT)
  - Diversi HotSpot (client e server)

Struttura client/server (1)

- I processi usano i servizi messi a disposizione dal kernel per comunicare
- Un *processo utente* (*processo client*) richiede un servizio (ad es. lettura di un file) ad un processo di SO (processo *server*)
- **Client e server operano in *modalità utente***
- I processi server realizzano le politiche di gestione delle risorse
- Il **kernel** include solo i dati che descrivono un processo e realizza i meccanismi di comunicazione tra processi



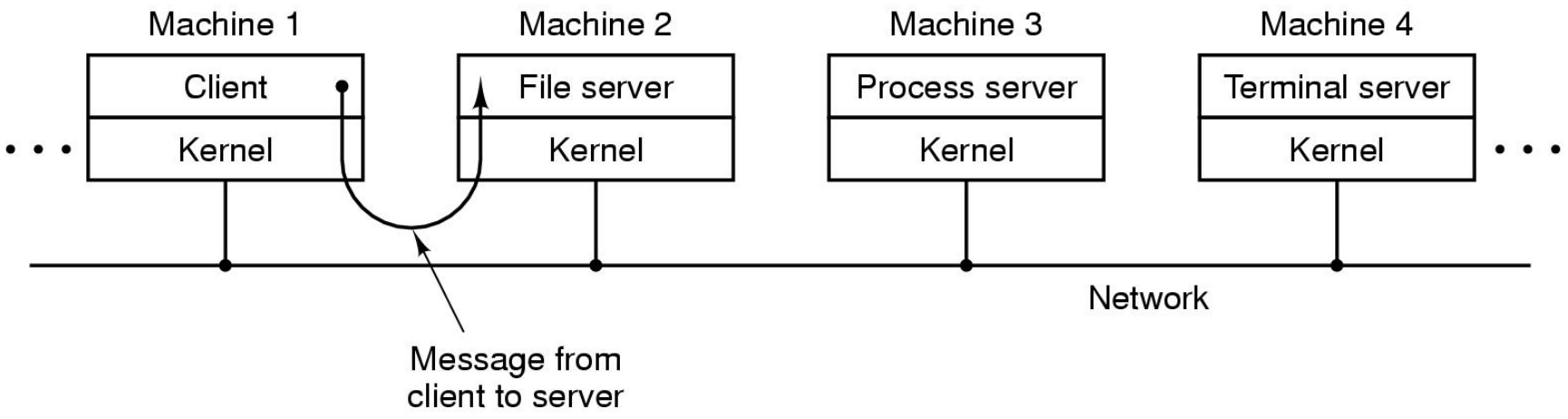
Struttura client/server (2)

- **Vantaggi/svantaggi**
  - **Modularità: suddivisione della funzionalità del SO in moduli**
  - **Portabilità: separazione tra *politiche* e *meccanismi di comunicazione***
    - Le politiche determinano che cosa sarà fatto
    - I meccanismi determinano come fare qualcosa
    - La separazione tra politica e meccanismo permette la max flessibilità se le decisioni della politica sono da cambiare in seguito
  - **Perdita di efficienza**: comunicazione sistemi client-server

- Windows NT 3.0 adottava un modello ispirato al client/server
- Struttura di SO studiata in ambito accademico (MACH, Minix)

Modello client/server nei sistemi distribuiti

- Più calcolatori tra loro collegati, **ciascuno con una propria copia del kernel** ed un certo numero di processi client e/o server
- I processi client richiedono servizi inviando le richieste tramite il kernel



Caratteristiche dei moderni SO

- **Architettura a micro-kernel**
  - SO semplice, detto *(micro-)kernel*
- **Multi-threading**
  - Suddivisione di uno stesso processo in flussi paralleli (vedremo più avanti nel corso)
- **Multi-processing simmetrico**
  - Più processori che condividono memoria e dispositivi di I/O, comunicano tramite bus, possono effettuare le stesse operazioni
- **SO distribuiti**
  - Sistemi multi-computer