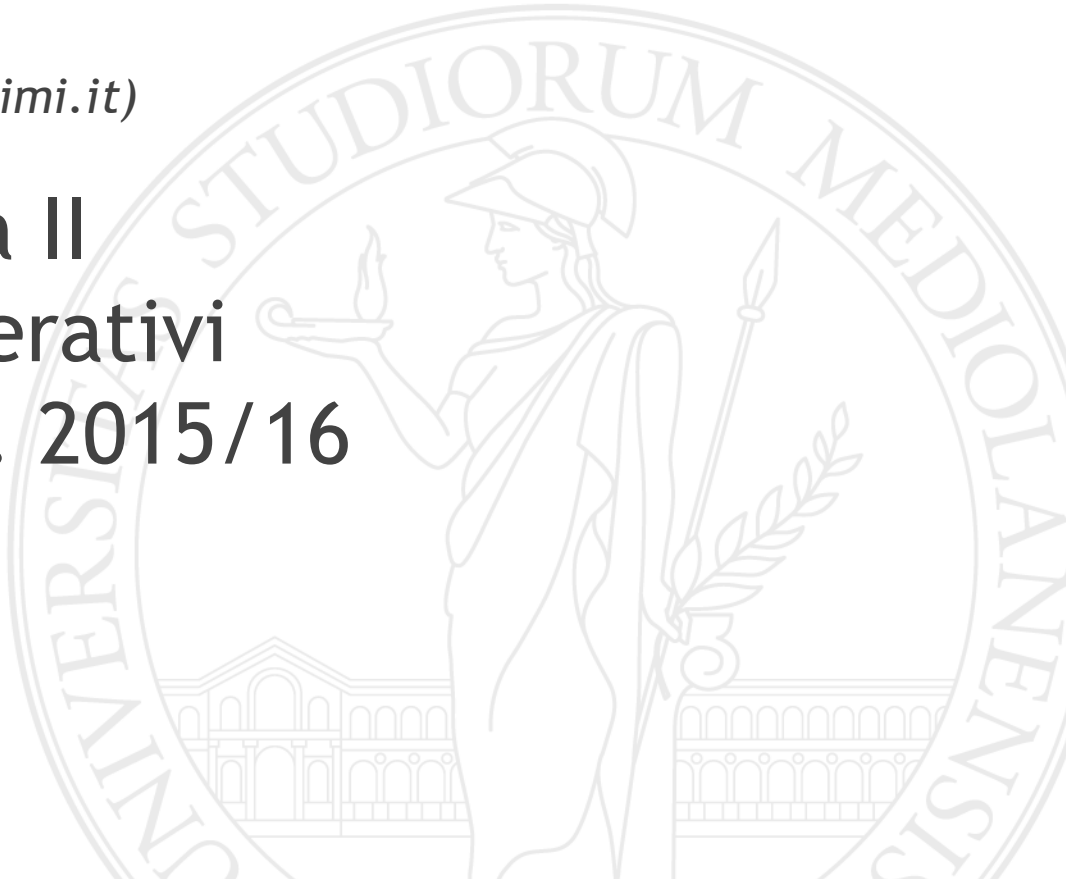
UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Alberto Ceselli
(alberto.ceselli@unimi.it)

Informatica II
Sistemi Operativi
DIGIP - a.a. 2015/16



Sistemi Operativi

(modulo di Informatica II)

Concetti base e architettura

Patrizia Scandurra

Università degli Studi di Bergamo

Sommario

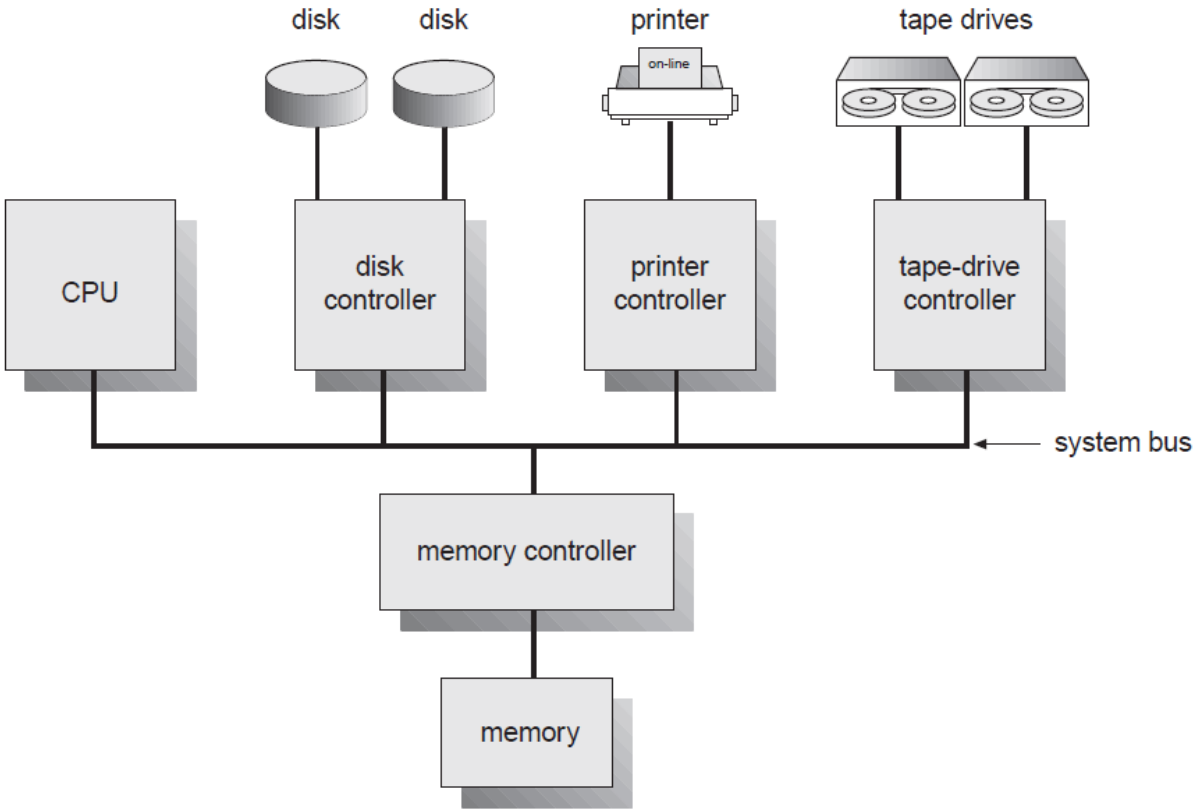
- Funzionamento di un calcolatore (cenni)
- Obiettivi e funzioni di un sistema operativo
- Implementazione e avvio del sistema operativo
- Architettura generale dei sistemi operativi
  - Servizi di un sistema operativo
  - Funzionamento *Event-driven* e *Dual-Mode*
  - Servizi e chiamate di sistema
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

Sommario

- Funzionamento di un calcolatore (cenni)
- Obiettivi e funzioni di un sistema operativo
- Implementazione e avvio del sistema operativo
- Architettura generale dei 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

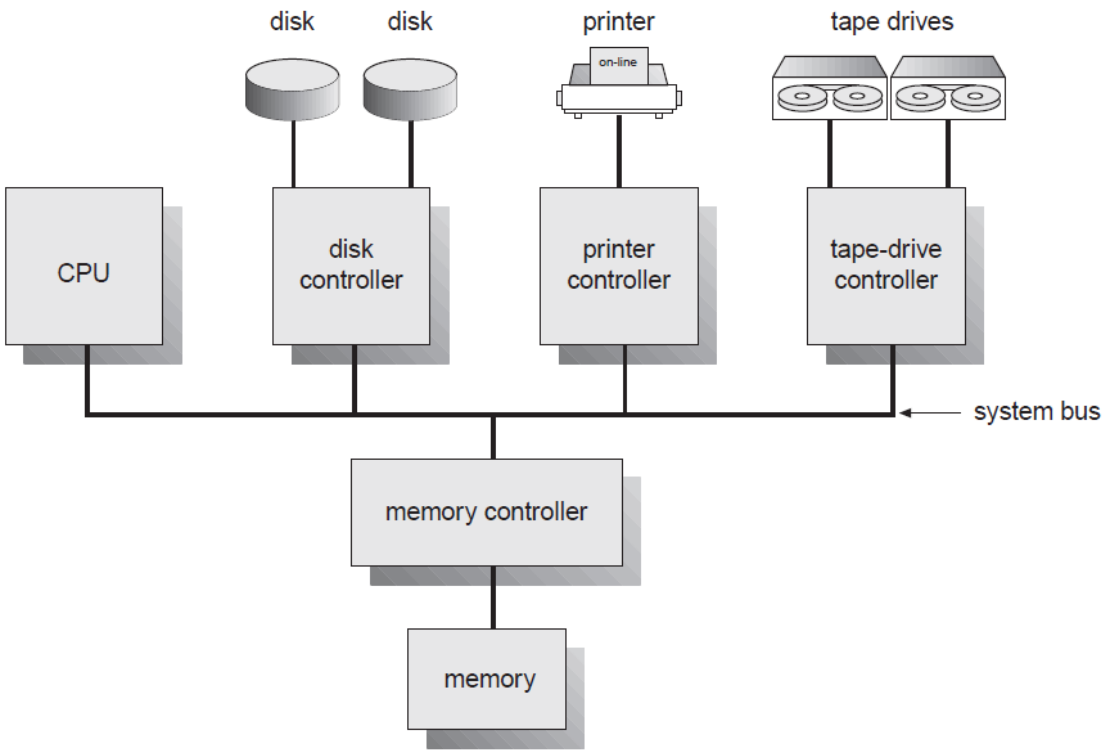
Funzionamento di un calcolatore

I controllori dei dispositivi e la CPU sono in esecuzione contemporaneamente



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 – 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 (modalità sincrona)

- Tipico di architetture hardware di fascia bassa o di controllo industriale con basse necessità di I/O multiplo
- Per scrivere un blocco di n caratteri (byte) la CPU esegue n istruzioni di output carattere!
- Per leggere un carattere la CPU controlla in un *ciclo stretto* la porta di I/O in attesa del carattere
- L'informazione di *input disponibile* o *fine scrittura* viene affidata ad un bit di controllo
- *Svantaggio*:
  - l'I/O di quantità di dati relativamente grosse occupa una parte rilevante del tempo macchina
  - Il processore spende la maggior parte del suo tempo in operazioni di I/O

Funzionamento di un calcolatore – tecniche di I/O

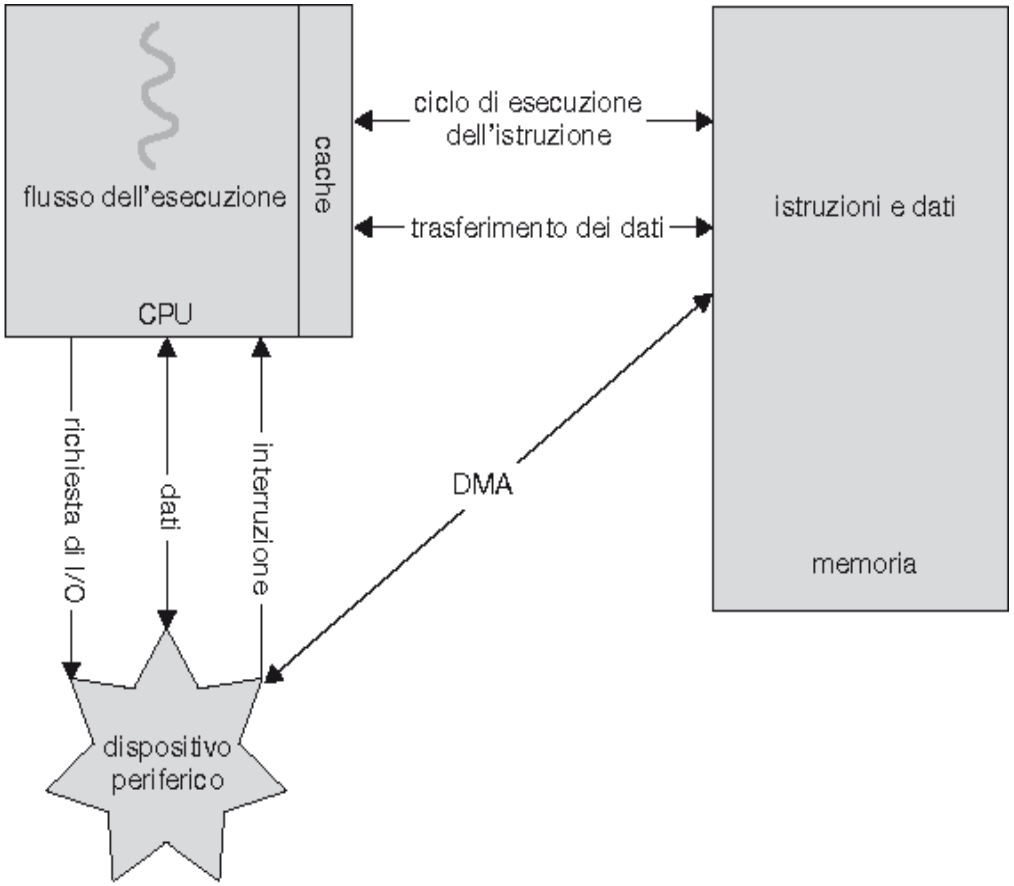
I/O tramite interrupt (modalità asincrona)

- 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
- *Svantaggio*: Permette di migliorare le prestazioni di letture/scritture multiple, ma ancora oneroso è il coinvolgimento della CPU per grosse dimensioni di dati

Funzionamento di un calcolatore – tecniche di I/O

I/O con DMA (asincrona)

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



Sommario

- Funzionamento di un calcolatore (cenni)
- Obiettivi e funzioni di un sistema operativo
- Implementazione e avvio del sistema operativo
- Architettura generale dei 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

Funzione principale di un SO

- Un SO **fornisce l'ambiente per l'esecuzione dei programmi**
- Due principali obiettivi realizzativi:
  - *Astrazione*
  - *Virtualizzazione*

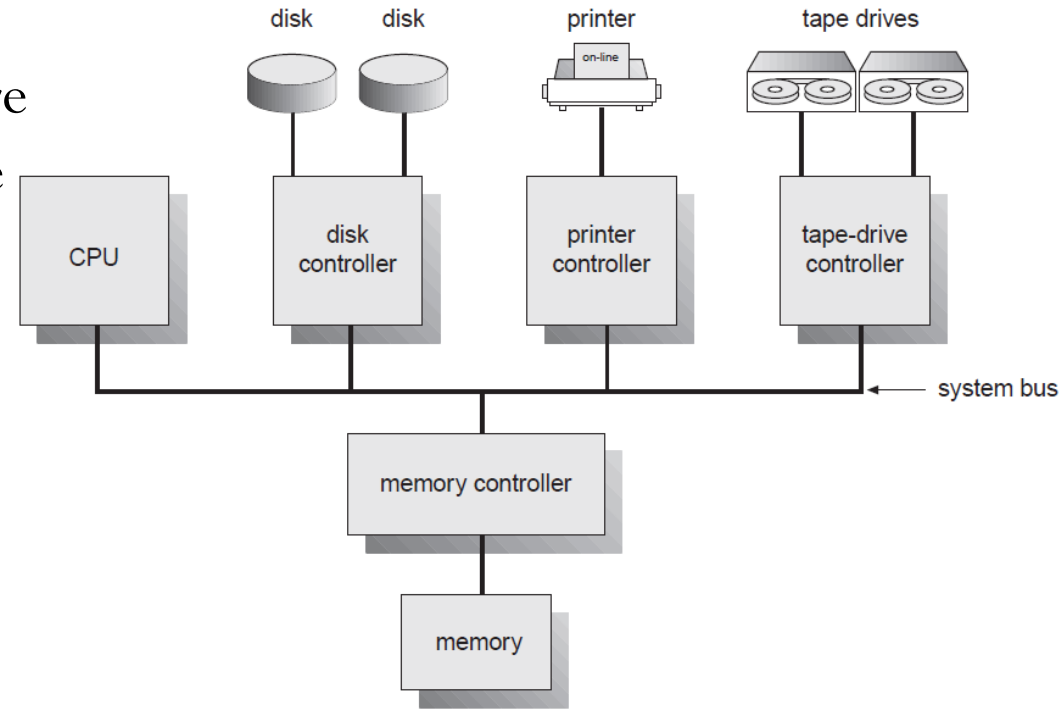
Più in dettaglio …..

Obiettivi (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 (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*
    - Esempio di risorsa condivisa risp. al tempo: CPU, stampante
    - Esempio di risorsa condivisa risp. allo spazio: memoria, disco

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

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 di I/O**

---> ***omogeneità di interazione verso hardware eterogeno***

  - 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*

- File e directory
  - Creazione e cancellazione
  - Lettura e scrittura
  - Copiatura
  - Ricerca
  - Salvataggio e ripristino
  - Protezione e sicurezza

Funzioni (5)

- Gestione dell'**interfaccia utente**

  ---> ***interazione con utenti e processi attraverso istruzioni di controllo***

  - Le **istruzioni di controllo** permettono di accedere ai servizi (funzioni) del SO
    - a livello utente
    - a livello programmi (*chiamate di sistema*)
  - **Interprete** è il programma che legge ed interpreta le istruzioni di controllo
    - Interfaccia a riga di commando o *shell*
    - *Interfaccia grafica*
    - *Interfaccia a lotti*
  - L'interfaccia è separata dall'esecuzione dei comandi (flessibilità)!!!

Sommario

- Funzionamento di un calcolatore (cenni)
- Obiettivi e funzioni di un sistema operativo
- La vita del sistema operativo
- Architettura generale dei 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

La vita del SO

- Implementazione
- Generazione
- Avvio
- Utilizzo

Implementazione del SO

- Implementazione
  - Prime generazioni: assembly
  - Terza generazione: linguaggi di programmazione ad «alto» livello, che costituiscono anche parte del sistema
    - IBM OS/360 >> FORTRAN
    - UNIX >> C
    - Commodore >> Basic
  - SO moderni
    - implementati con linguaggi di programmazione ad alto livello (vantaggi / svantaggi?)
    - Offrono supporto alla programmazione in diversi linguaggi (es. supporto alla programmazione concorrente)

Generazione del SO

- Procedura **System generation** or **sysgen**
- I SO sono progettati per funzionare su di una classe di macchine; ma devono essere configurati per ogni HW specifico
- Parametri:
  - CPU (velocità elaborazione, Single Core/Dual Core/Quad Core, dimensione cache, ecc..)
  - Memoria (indirizzo legale finale, dimensione, ecc..)
  - Formattazione del disco di avvio (partizioni, dimensioni, contenuto)
  - Dispositivi disponibili e relativi driver
  - Opzioni utente (ad esempio, l'algoritmo di schedulazione)
- Valgono anche per le macchine virtuali!

Avvio (boot) del SO (2)

Dove risiede il SO?

- **Opzione 1: in memorie ROM (*firmware*)**
- **Opzione 2: su memorie di massa**

Vantaggi / Svantaggi?

Avvio (boot) del SO (2)

Dove risiede il SO?

- **Opzione 1: in memorie ROM (*firmware*)**
  - Es. dispositivi mobili od orientati al multimedia (console giochi
  - (S) costoso
  - (S) non facilmente modificabile
  - (V) protetto da virus e da danni accidentali
- **Opzione 2: su memorie di massa**
  - (V) più economico e flessibile (S) più fragile
  - (S) se il SO è su disco, e il disco è gestito dal SO, chi fa partire il SO?
  - 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***

Avvio (boot) del SO (2)

Dove risiede il SO?

- **S**e il SO è su disco, e il disco è gestito dal SO, chi fa partire il SO?
  - Il BIOS (Basic I/O System) : programma di avviamento in ROM o EEPROM)

Avvio (boot) del SO (1)

I passi sono (accensione e …):

- Esecuzione del BIOS (Basic I/O System)
  - Inizializza tutti i registri della CPU, i controllori delle periferiche di I/O e la memoria centrale
  - Cerca periferiche avviabili e dà il controllo al *bootstrap program* o *bootstrap loader*

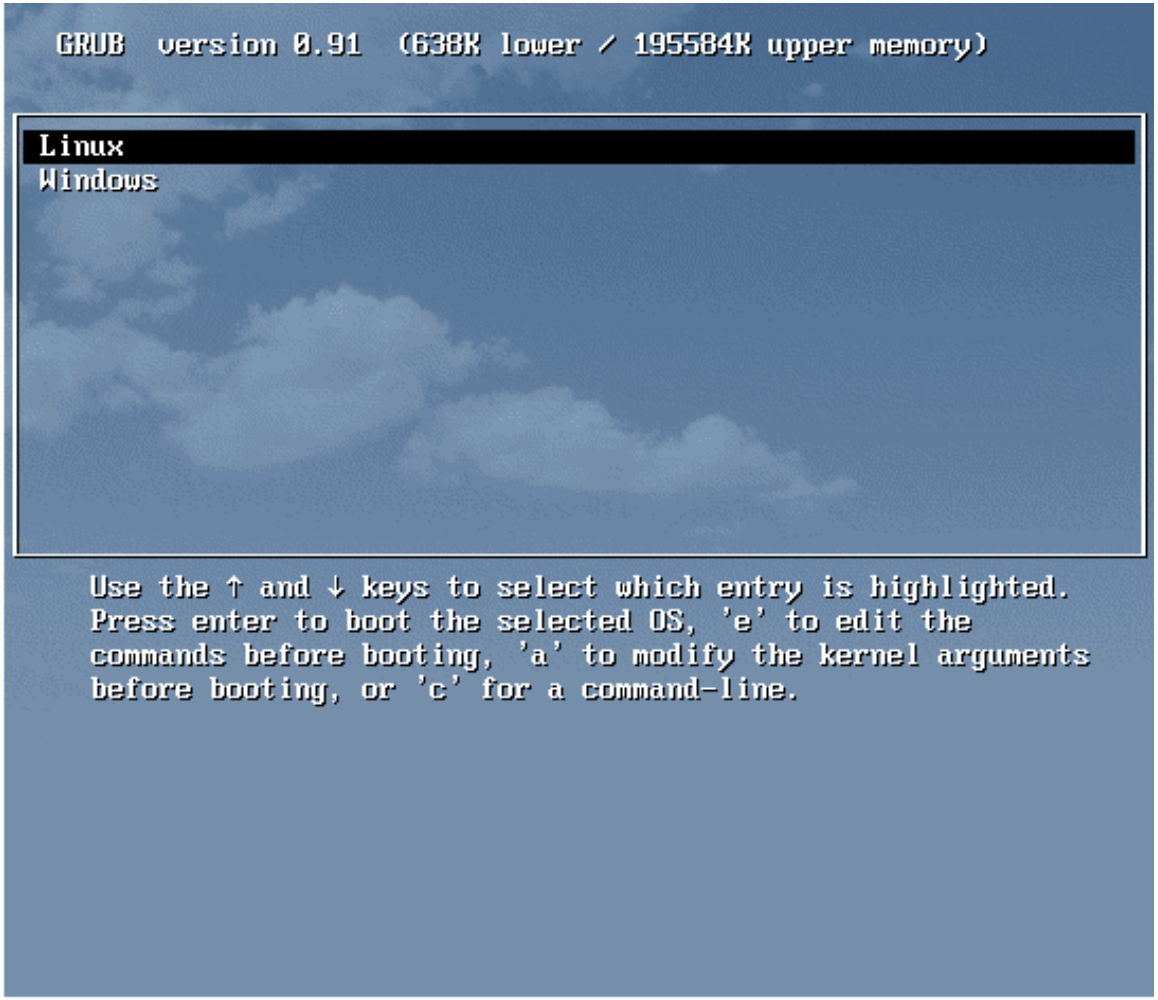
Avvio (boot) del SO (1)

I passi sono (accensione e …):

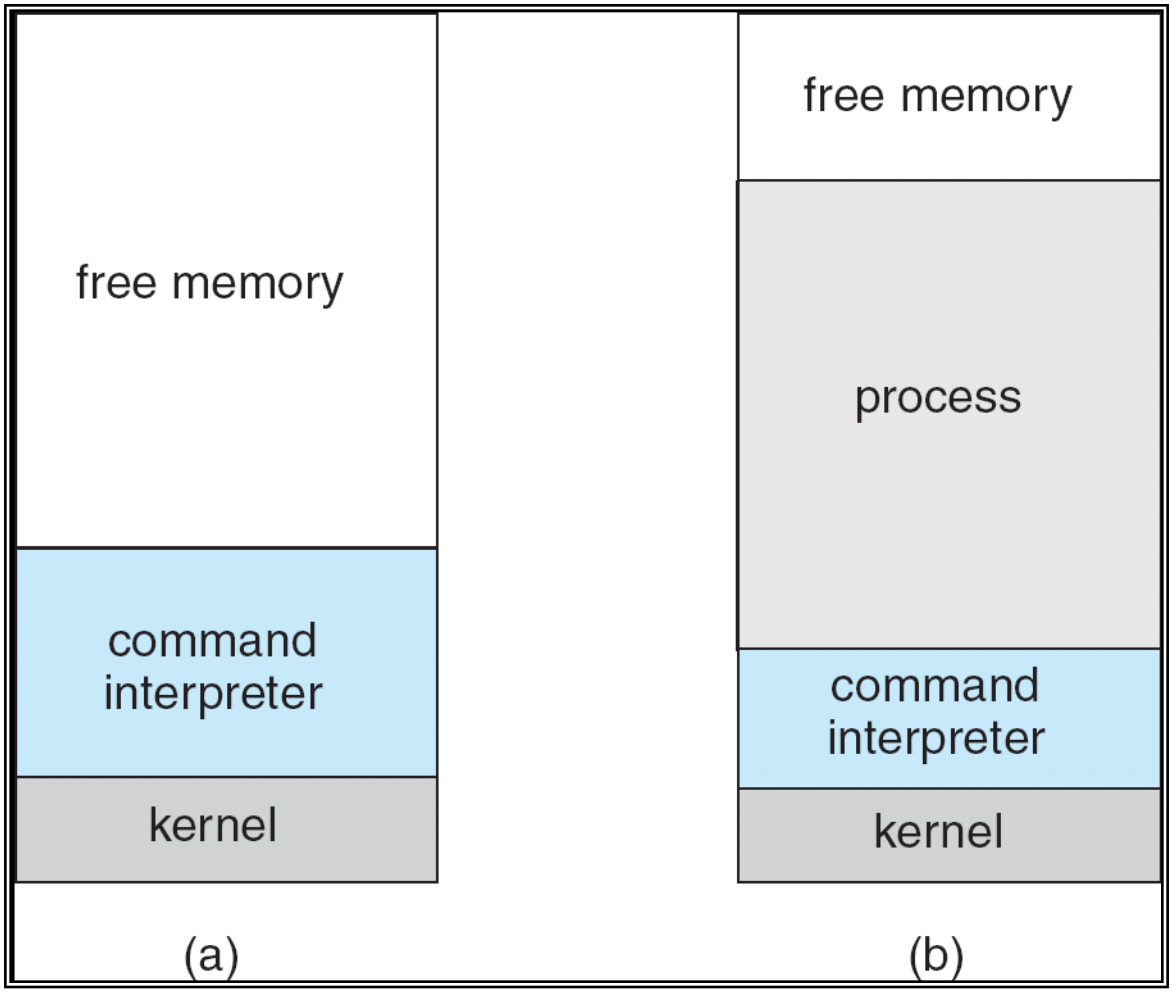
- Esecuzione del BIOS (Basic I/O System)
- Il bootstrap program si avvale di un frammento di codice particolare
  - che risiede in un blocco del disco ad una posizione fissa (tipicamente 0) – ***boot block*** *o* ***master boot record (MBR)***
  - e che conosce la posizione su memoria di massa del SO
  - La partizione del disco che contiene il boot block è detta di boot (o avvio, o ***boot partition)***
  - Viene caricato in RAM il **kernel** del SO da una unità di memoria (anche un'altra partizione, un disco differente, un'unità di rete …)
  - Viene effettuato un controllo del sistema

Boot loader per i SO moderni

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



MS-DOS avvio ed esecuzione



(a) At system startup (b) running a program

Sommario

- Funzionamento di un calcolatore (cenni)
- Obiettivi e funzioni di un sistema operativo
- Implementazione e avvio del sistema operativo
- Architettura generale dei 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

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*

Sommario

- Funzionamento di un calcolatore (cenni)
- Obiettivi e funzioni di un sistema operativo
- Implementazione e avvio del sistema operativo
- Architettura generale dei 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 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:
  - Interrupt hardware
  - Interrupt software
    - Chiamate di sistema (normale funzionamento) ovvero invocazioni di servizi (funzioni) del SO
    - Trap o eccezioni software (anche in seguito a chiamate di sistema)

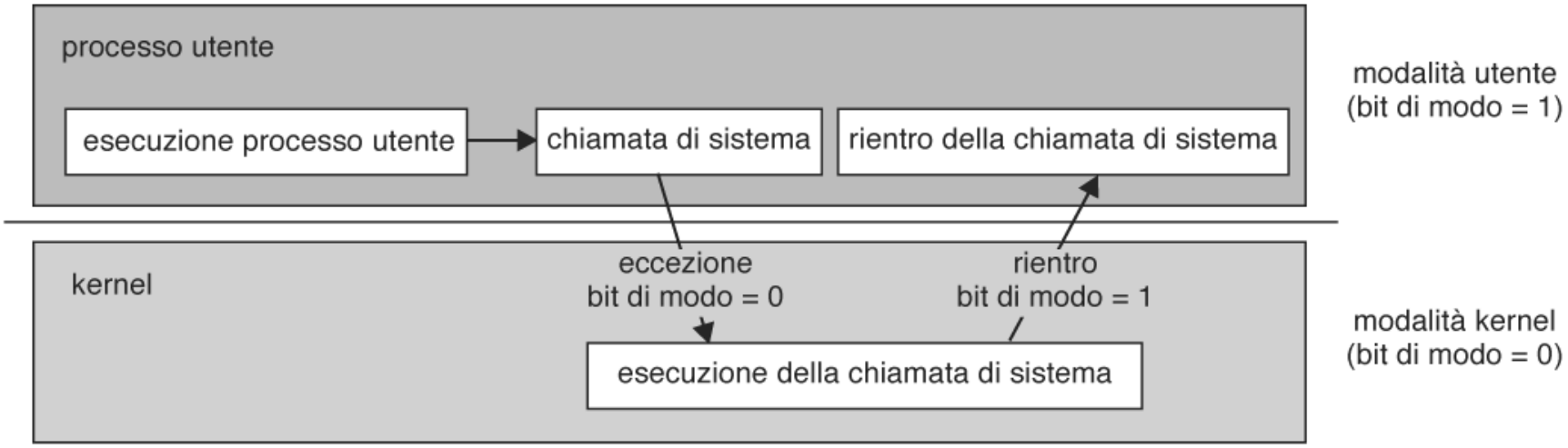
Funzionamento Dual-Mode (1)

- L'hardware deve fornire supporto per differenziare tra due modi di funzionamento
  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)

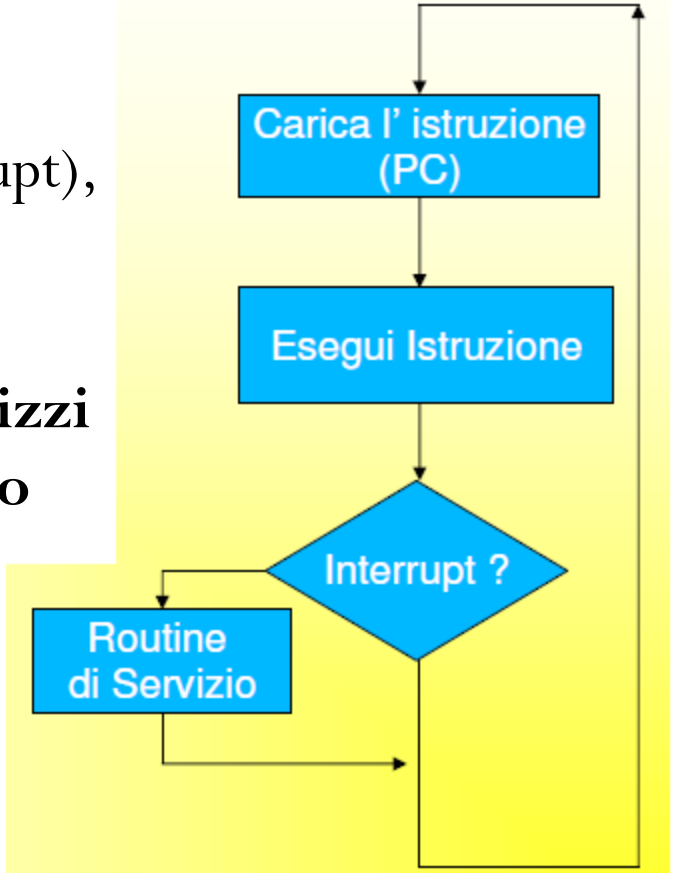
Funzionamento Dual-Mode (2)

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



Interruzioni (1)

- La CPU esegue sequenzialmente le istruzioni
- All'arrivo di un interrupt (I/O guidato da interrupt), la CPU invoca un "gestore" (*routine di servizio*) opportuno attraverso il **vettore di interrupt**
  - 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.

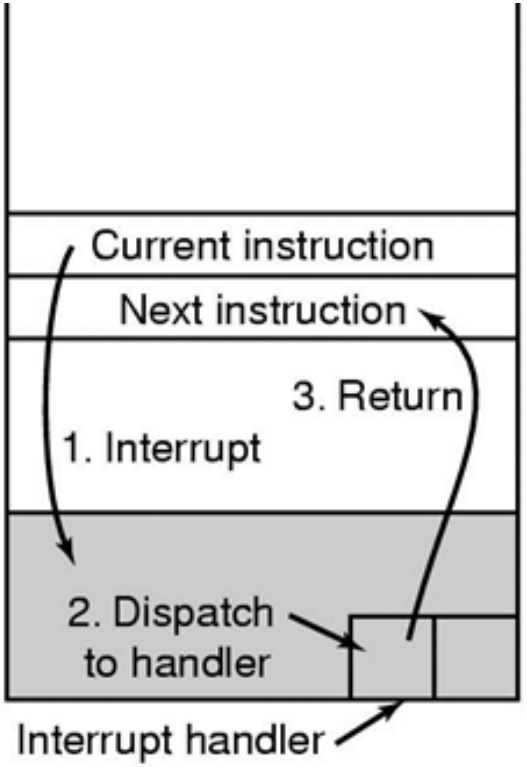


- NOTA: **Interrupt in arrivo sono disabilitati mentre un altro interrupt viene gestito**, per evitare che vadano perduti

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 (*vettore di interrupt*)
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



Sommario

- Funzionamento di un calcolatore (cenni)
- Obiettivi e funzioni di un sistema operativo
- Implementazione e avvio del sistema operativo
- Architettura generale dei 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

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 (JNI)*
      - Java può richiamare metodi scritti in un altro linguaggio (C o C++) che eseguono la chiamata di sistema

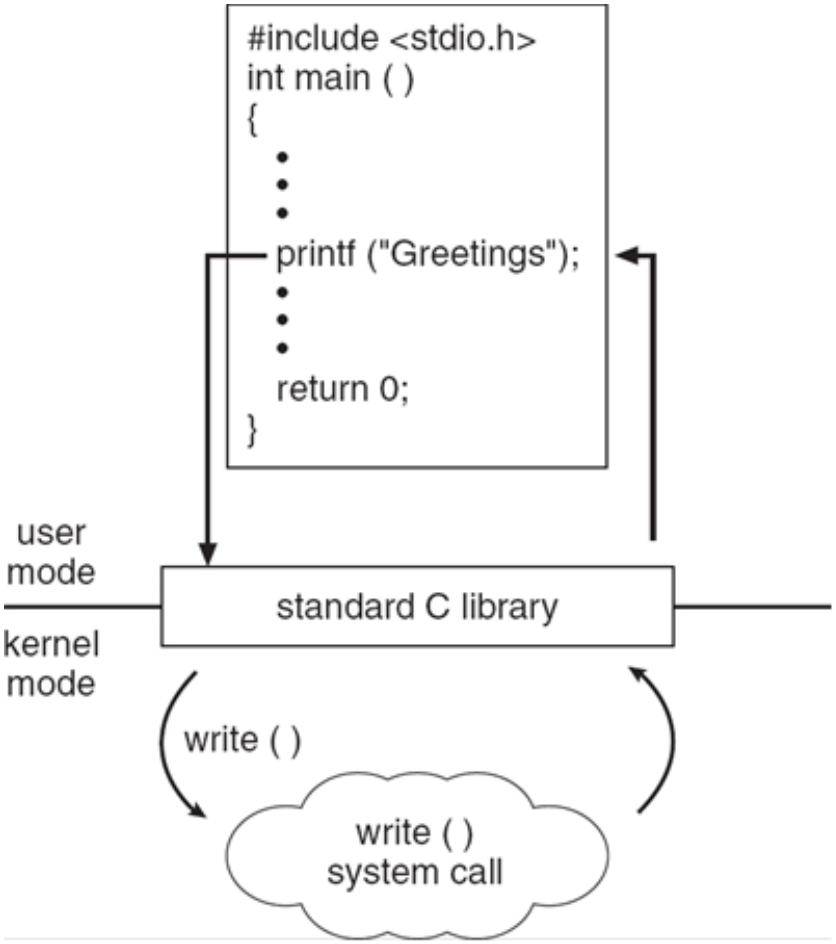
API per le chiamate di sistema

(API = Application Programmer's Interface)

- 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 fornite dal **sistema di supporto run-time**
  - L'insieme di funzioni sviluppate in librerie incluse in un compilatore
  - È un'interfaccia verso le chiamate di sistema
- Motivi:
  - Portabilità
  - Semplicità

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

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

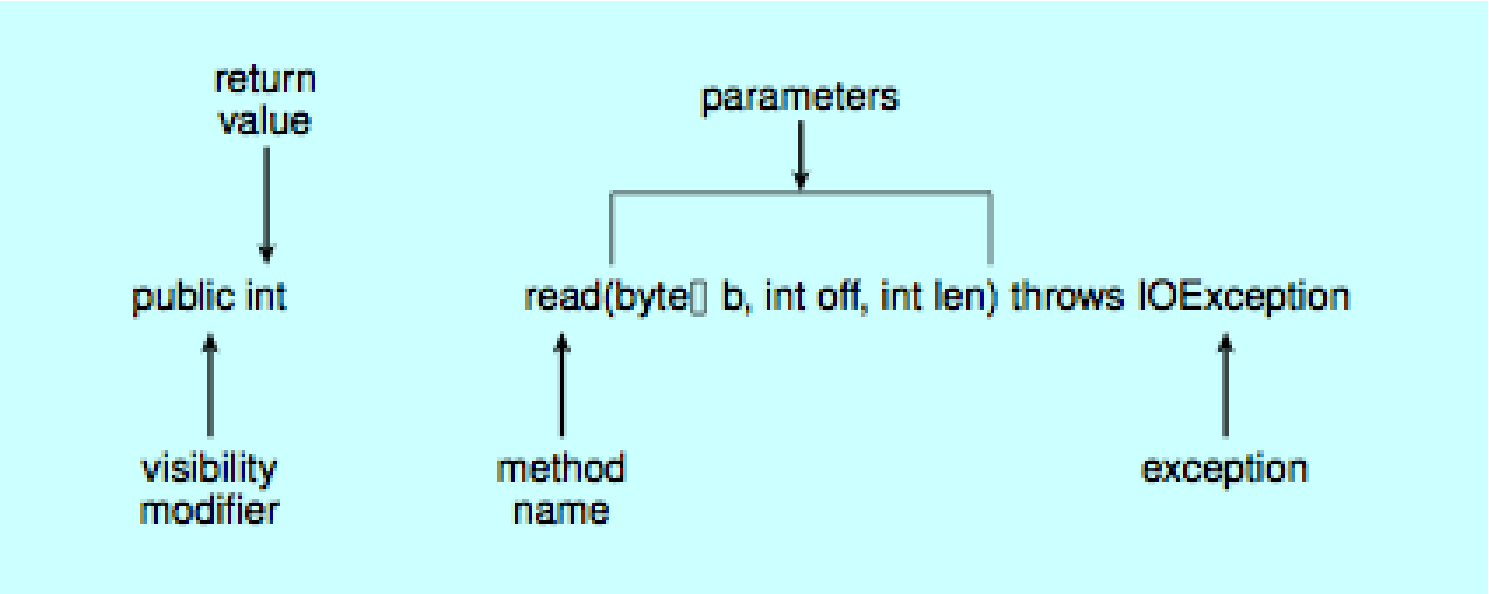
Esempio di ciclo esecutivo di una chiamata di sistema

Acquisisce il nome del file in ingresso
    Scrive messaggio di richiesta sullo schermo
    Accetta i dati in ingresso
Acquisisce il nome del file in uscita
    Scrive messaggio di richiesta sullo schermo
    Accetta i dati in ingresso
Apre il file in ingresso
    Se il file non esiste, termina con errore
Crea il file in uscita
    Se il file esiste, termina con errore
Ripete
    Legge dal file in ingresso
  Scrive sul file in uscita
Finché c'è ancora da leggere
Chiude il file in uscita
Scrive messaggio sullo schermo per informare del completamento
Termina senza errori

Progettare chiamate di sistema

Esempio di una API Standard

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



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

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

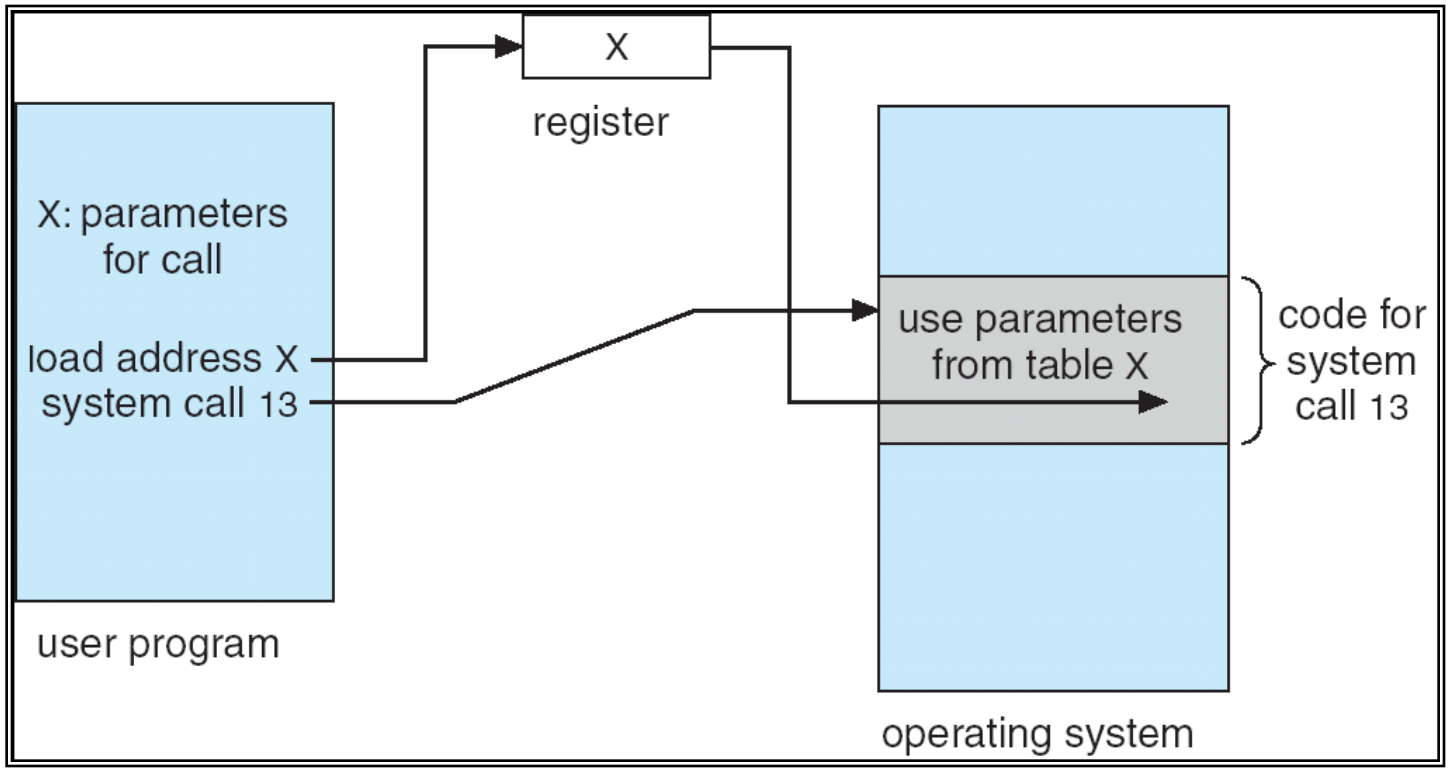
int len – il massimo numero di byte da leggere

Chiamate di sistema – passaggio dei parametri

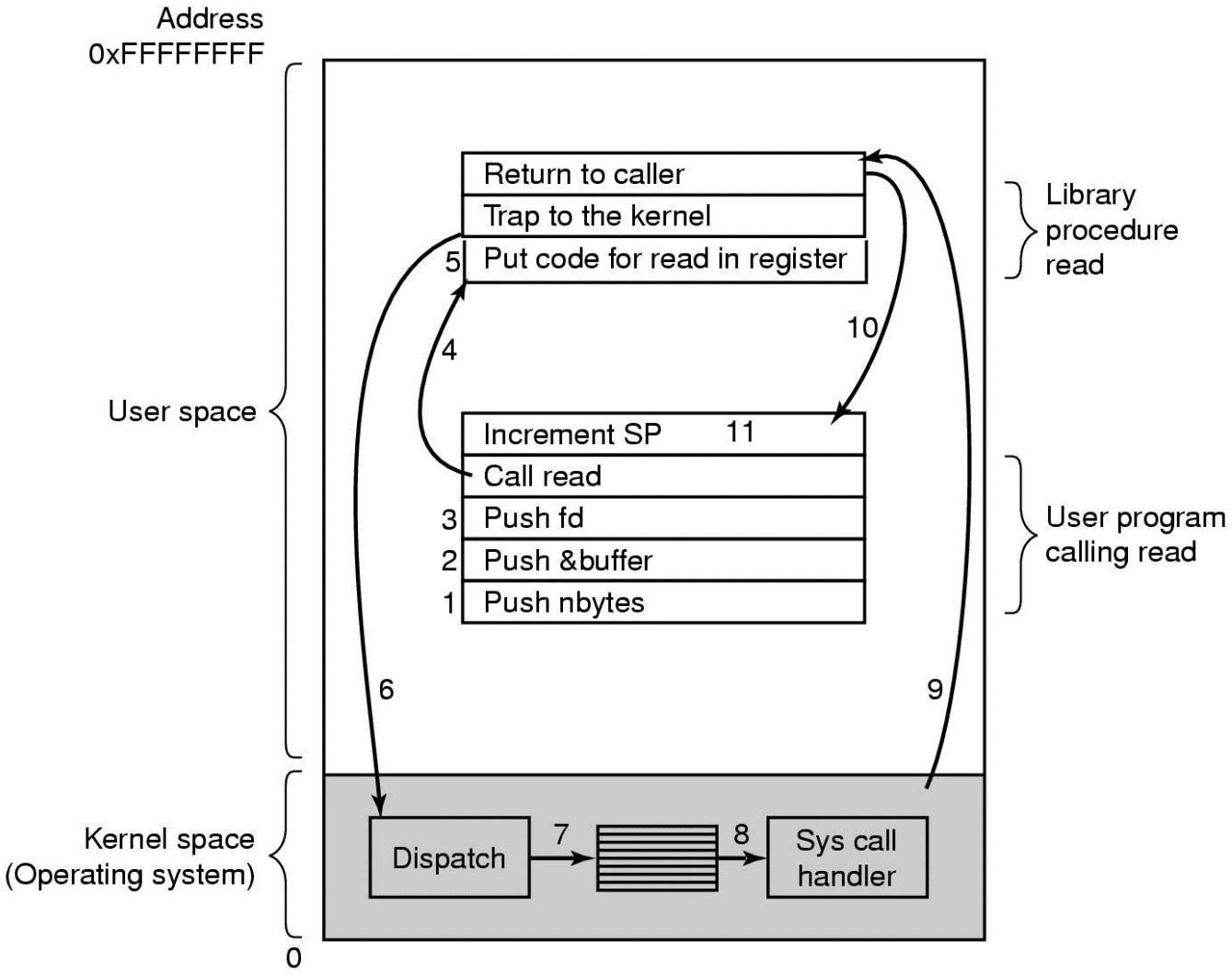
- Le chiamate di sistema possono richiedere lo scambio di informazioni (***parametri***) tra un programma e il SO
- Lo scambio dei parametri avviene **in generale in 3 modi**:
  1. Attraverso **registri**
     - attuabile solo se il numero dei parametri non supera quello dei registri
  2. Attraverso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



Gli 11 passi necessari per effettuare la chiamata di sistema

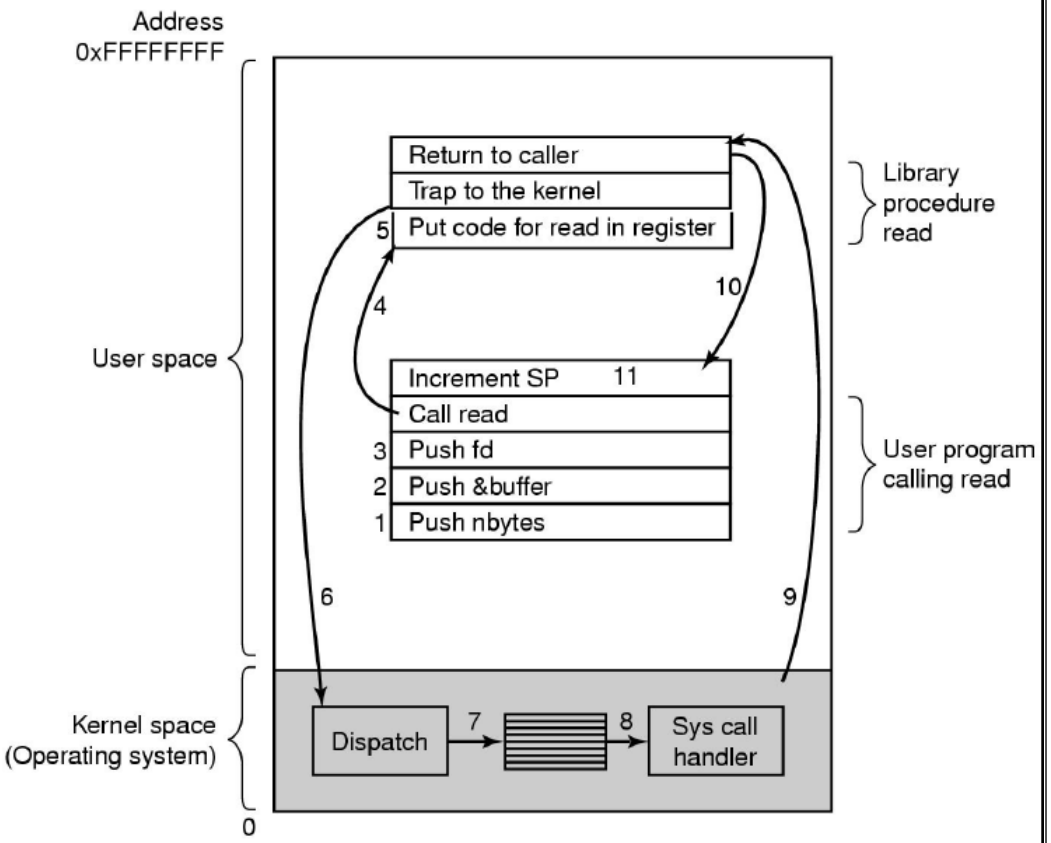
read (fd, buffer, nbytes)

Esempio di chiamata di Sistema (2)

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

1-3. Salvataggio dei parametri sullo stack

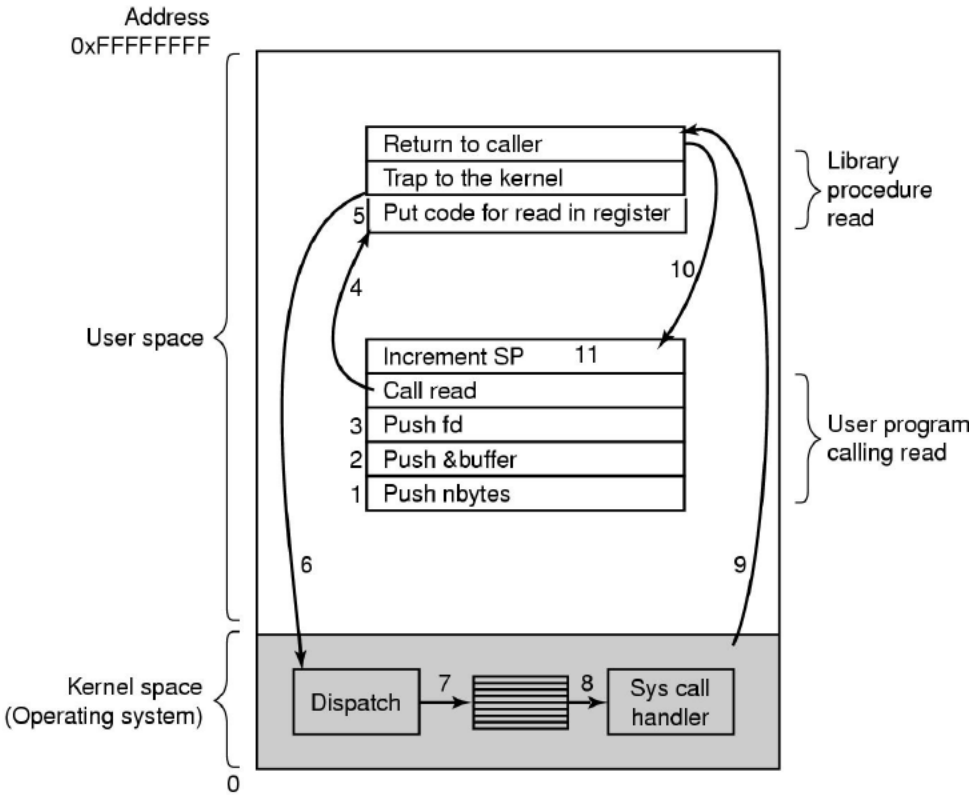
4. Chiamata della funzione di libreria *read*



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*

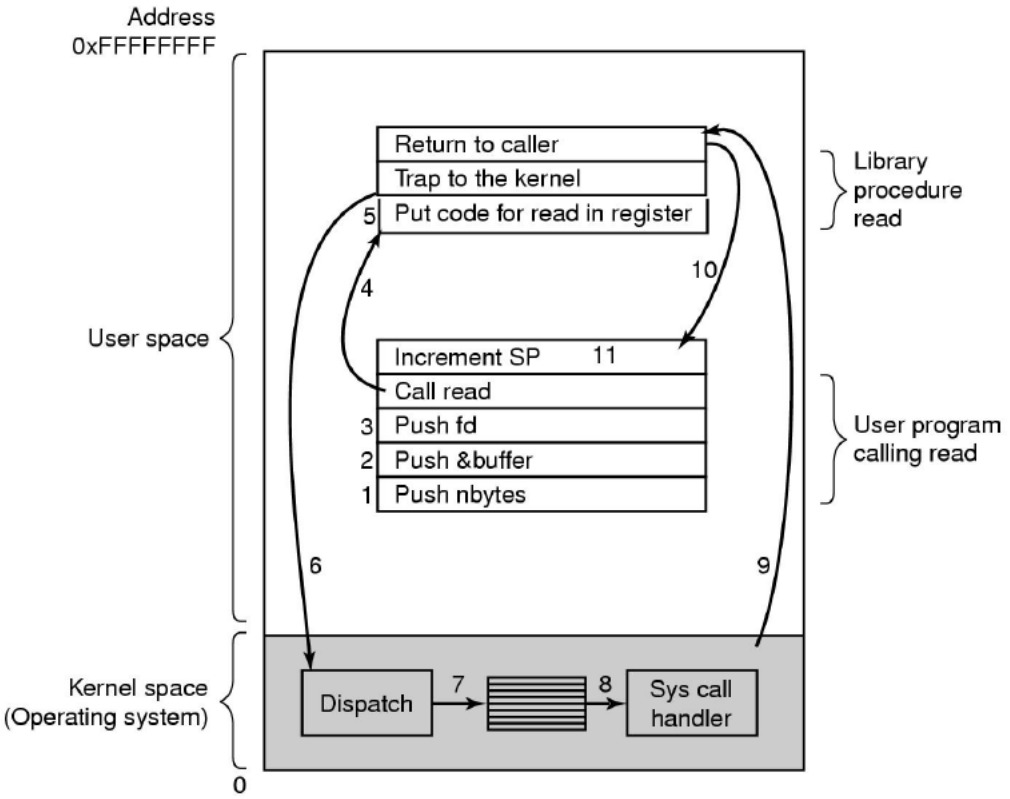


Esempio di chiamata di Sistema (4)

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

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

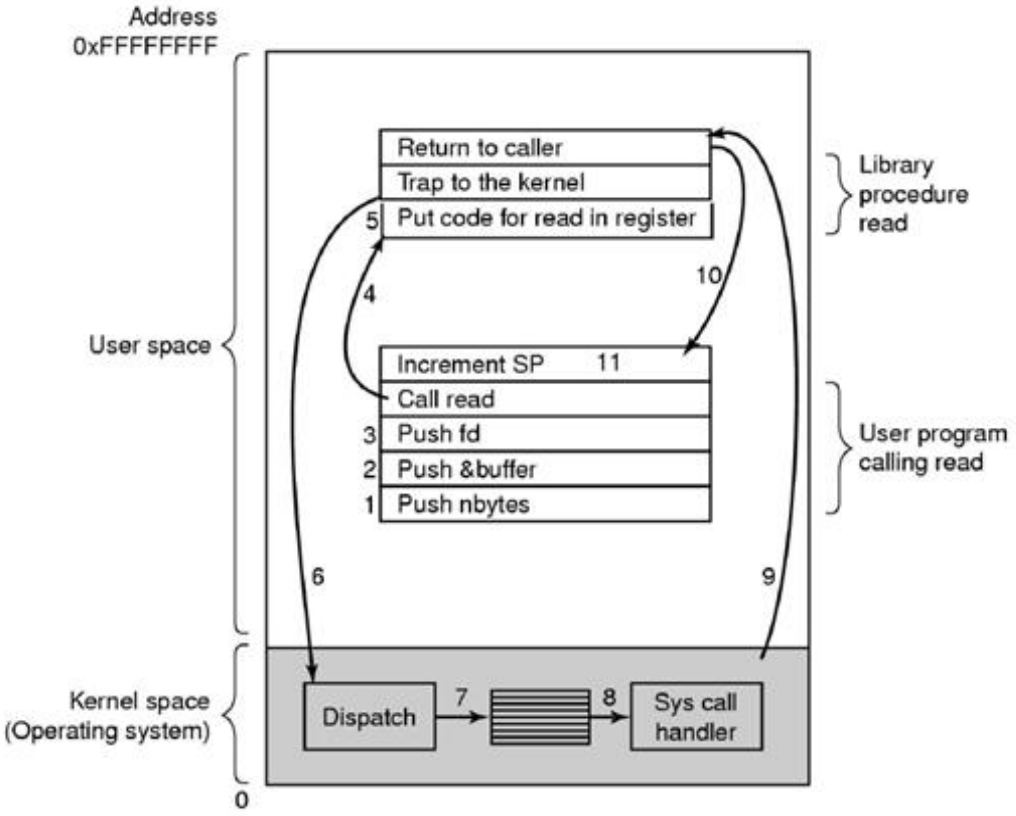
9. Ritorno alla funzione di libreria
   - ripristino user mode, caricamento del program counter PC



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



Categorie di 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

Relazione tra funzioni di API e system call ignota e comunque non fissa

Presumibili funzioni API in relazione 1-1 con system call: **Win32 API**

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

Sommario

- Funzionamento di un calcolatore (cenni)
- Obiettivi e funzioni di un sistema operativo
- Implementazione e avvio del sistema operativo
- Architettura generale dei 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

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 sono il **punto di contatto tra il SO e l'utente**
  - Possono essere **semplici interfacce verso le chiamate di sistema** oppure **complessi programmi**
- 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..)
  - 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!

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

Sommario

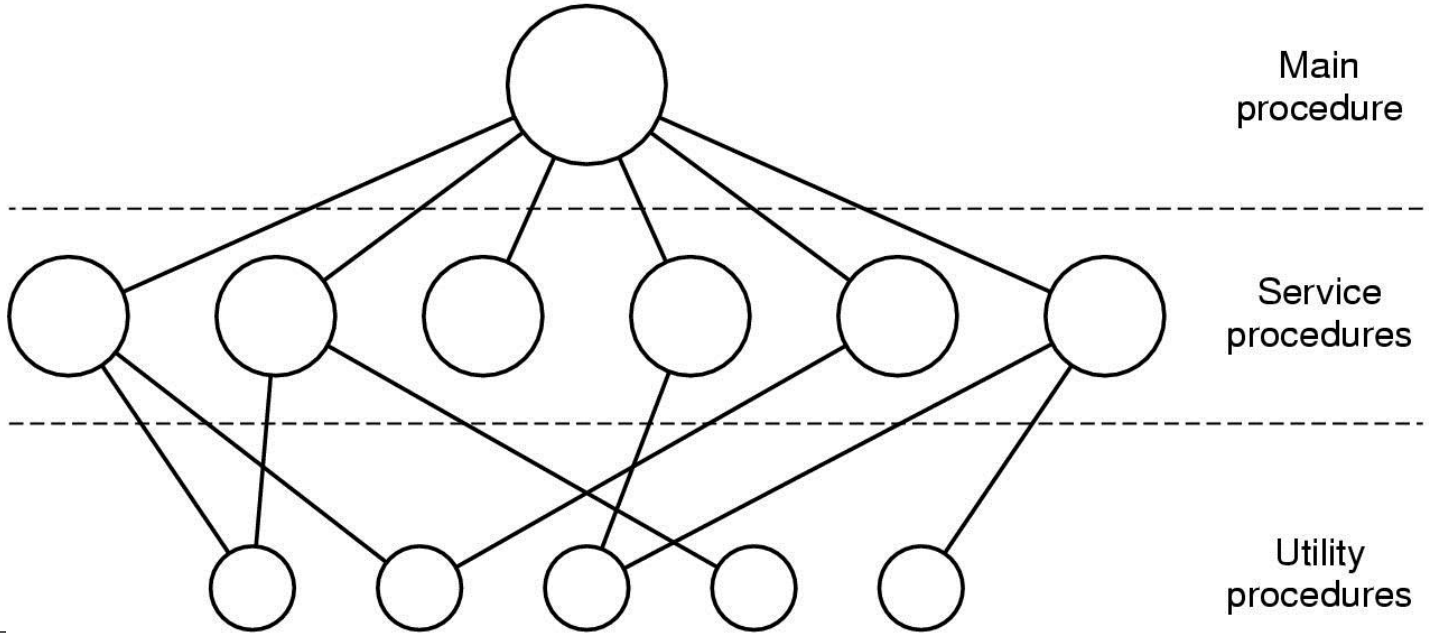
- Funzionamento di un calcolatore (cenni)
- Obiettivi e funzioni di un sistema operativo
- Implementazione e avvio del sistema operativo
- Architettura generale dei sistemi operativi
  - Servizi di un sistema operativo
  - Funzionamento *Event-driven* e *Dual-Mode*
  - Servizi e chiamate di sistema
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

Struttura di un sistema operativo

- Come sono organizzate le varie componenti di un SO?
  - Struttura monolitica
  - Struttura stratificata
  - Struttura a microkernel
  - Struttura a moduli funzionali
  - Struttura ibrida
  - Struttura a macchine virtuali
  - Struttura client/server

Struttura monolitica (1)

- Molti SO commerciali non avevano (e ancora 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

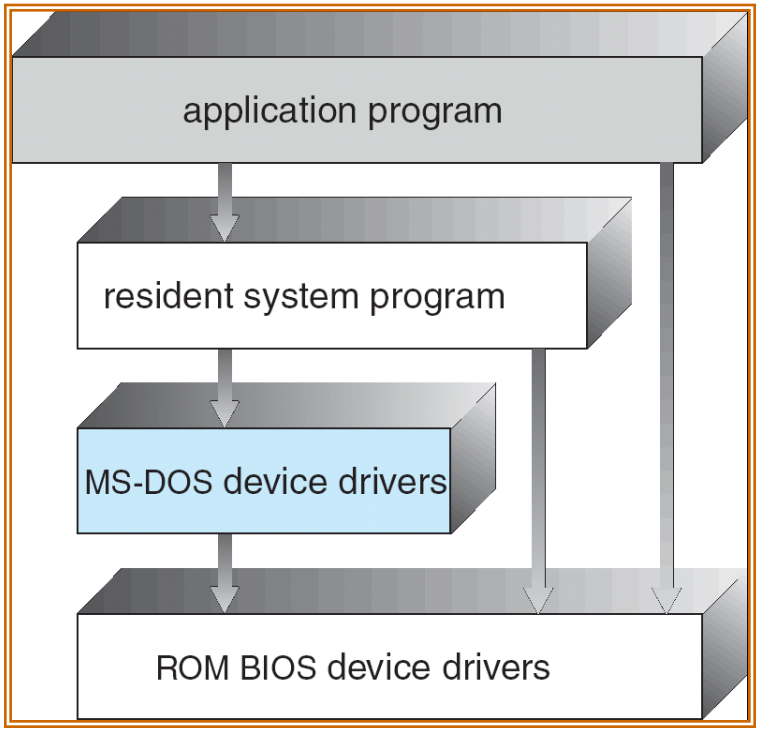


Struttura monolitica (2)

- 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, senza *bit di modo*)
    - SO molto vulnerabile perché lascia libertà ai programmi applicativi di accedere direttamente all'hardware
    - DOMANDA: ho qualche **vantaggio**?
  - 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

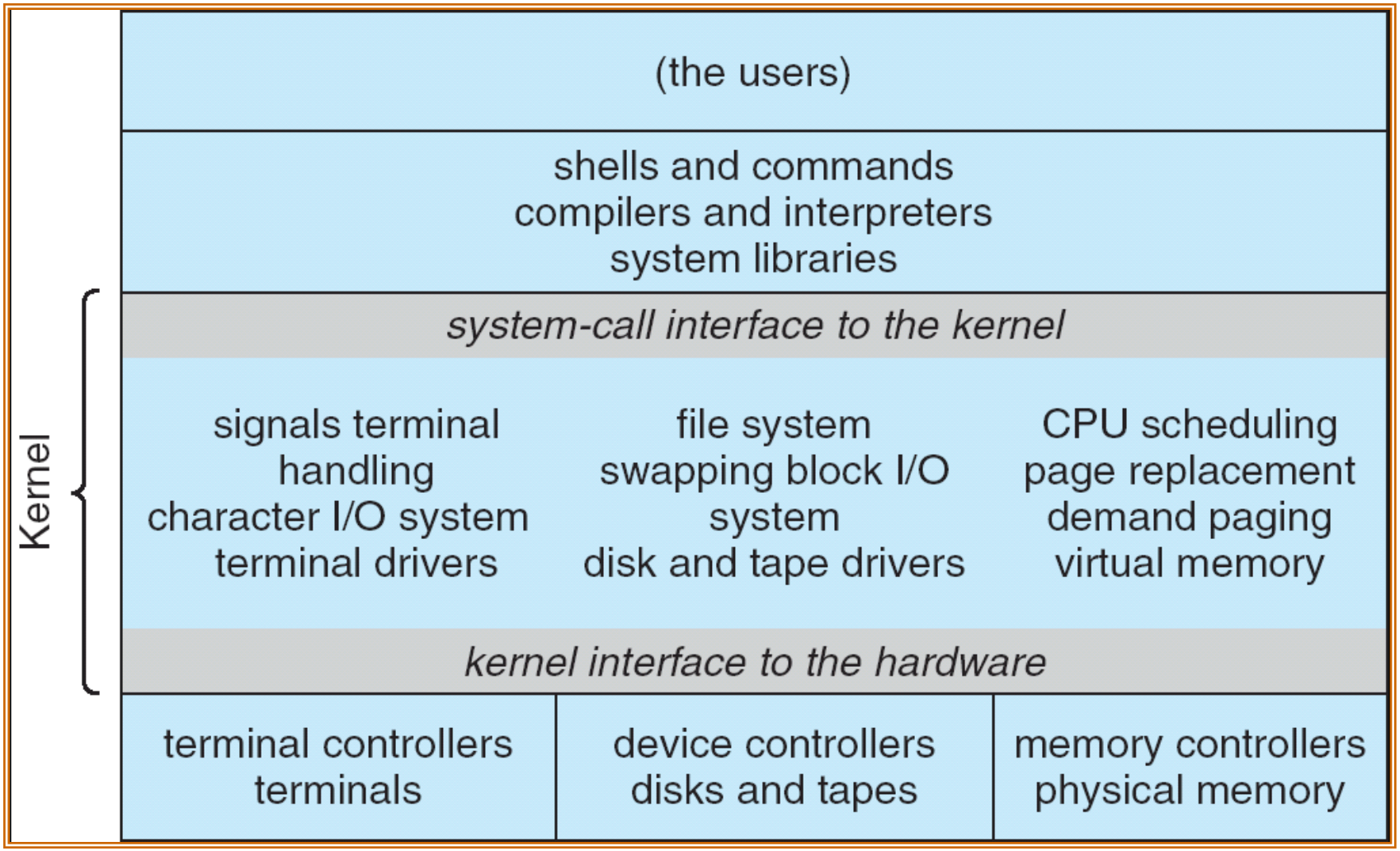


Struttura del sistema UNIX (1)

UNIX consiste di **due parti separabili**:

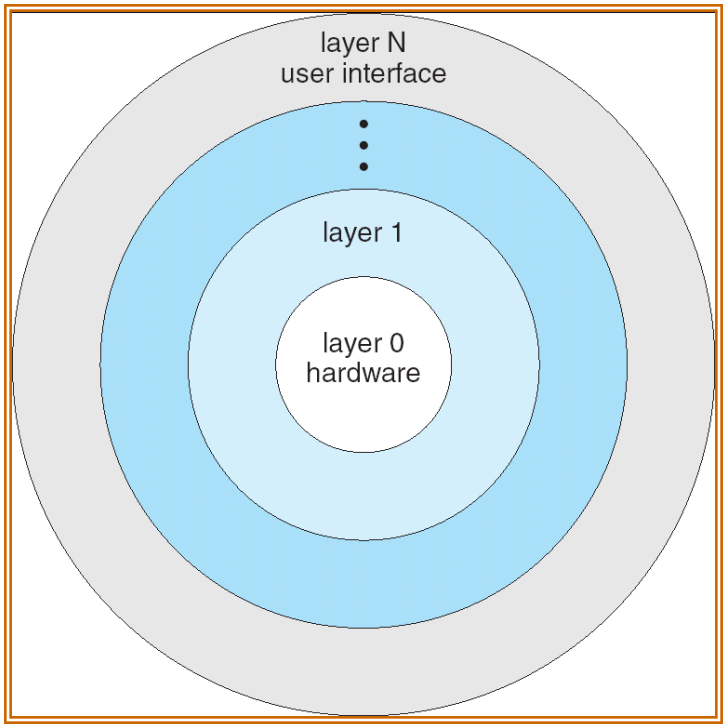
1. **Programmi di sistema**
2. **Nucleo** (*kernel):*
   - Consiste in qualsiasi parte si trovi sotto l'interfaccia di chiamata di sistema e sopra l'hardware fisico
   - Un'enorme quantità di funzioni combinate in un solo livello:
     - gestione del file-system, schedulazione della CPU, gestione della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

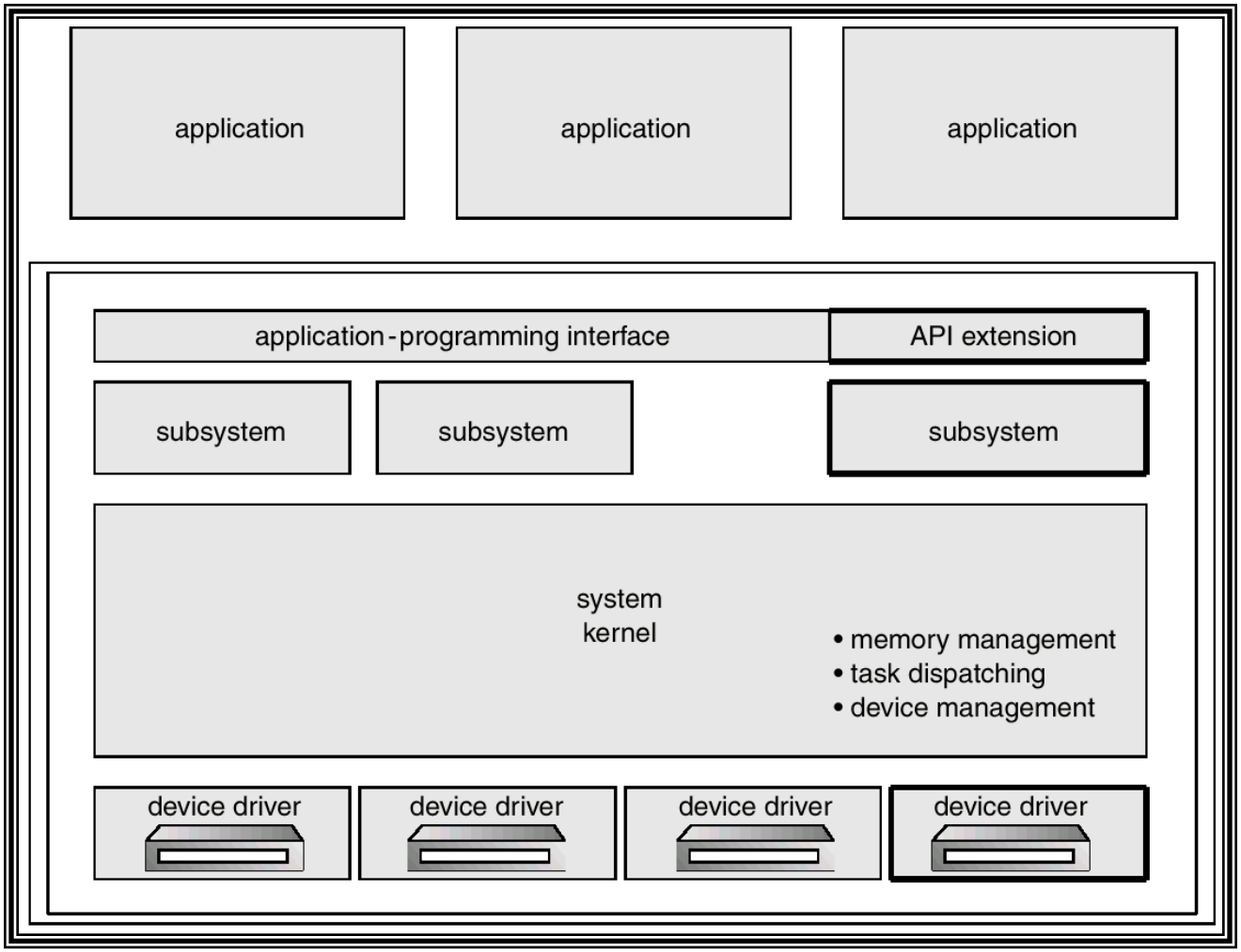


Struttura stratificata (2)

- **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
  - Più *difficile da progettar*e

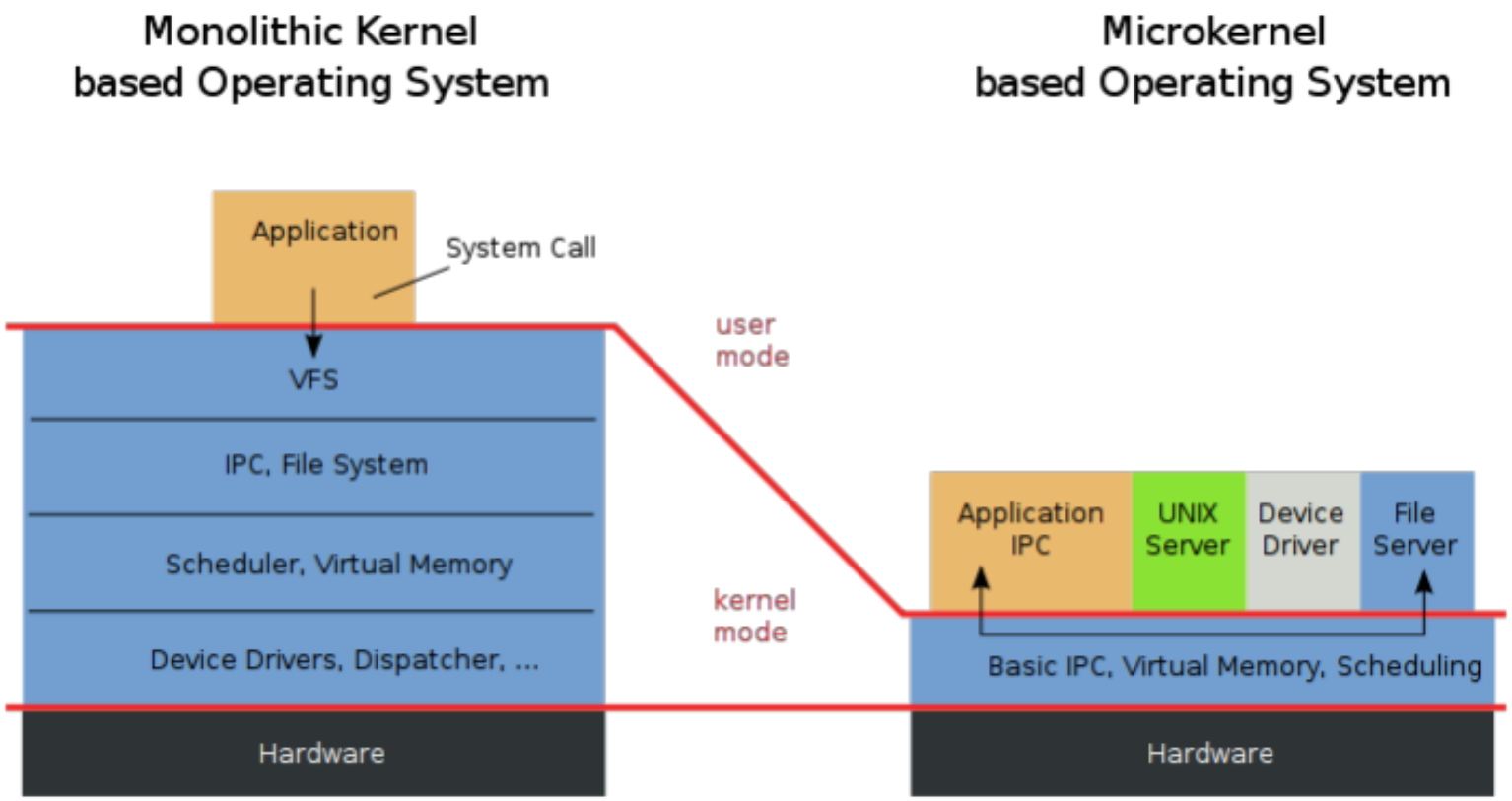
La struttura a strati di IBM OS/2

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 (1)

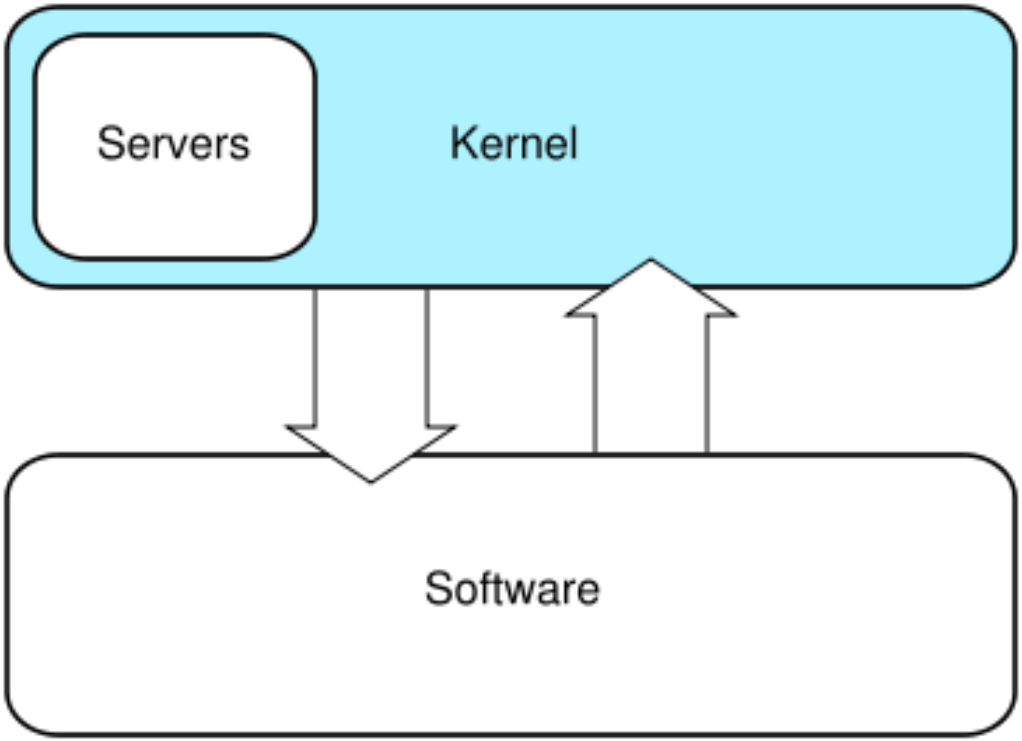
- Sposta il maggior numero di funzioni **dal kernel allo spazio utente** (come *programmi di sistema*)
- La comunicazione tra moduli avviene tramite **scambio di messaggi: IPC (Inter-process procedure call)**



Struttura a microkernel (2)

- **Esempi:**
  - **Microkernel Mach** (by Carnegie Mellon University, alla base di molti moderni sistemi operativi tra cui True64 UNIX e Mac OS X)
  - **WindowsNT** (in verità kernel ibrido!)
- **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)
  - Calo di prestazioni per l'aumento di sovraccarico indotto dall'esecuzione di processi di sistema in modalità utente.
    - Ad es. la prima versione di WindowsNT, basata su microkernel stratificato, aveva prestazioni inferiori a Windows95

Kernel ibrido

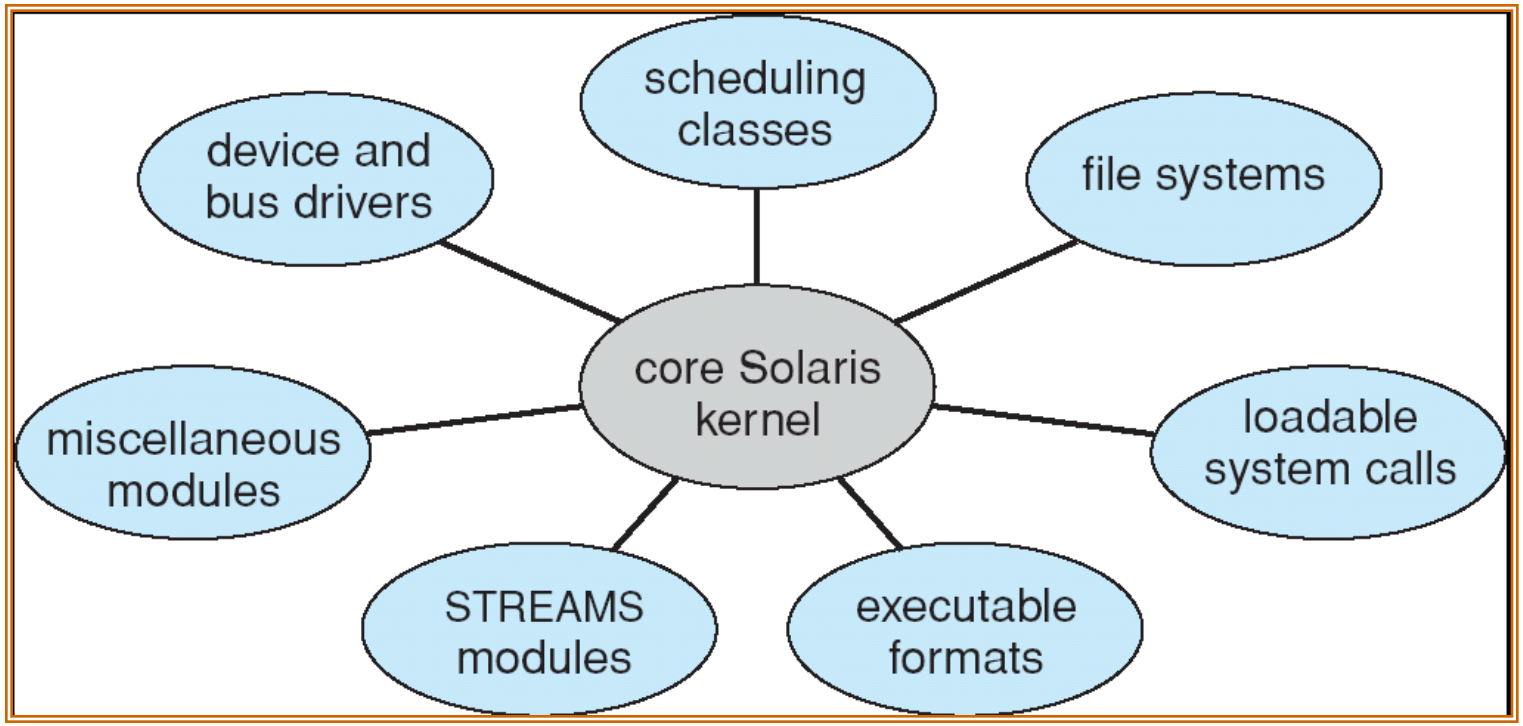


E’ come l’architettura a microkernel (cioè con un nucleo centrale e tanti “pezzettini” che girano sopra questo nucleo) solo che questi “pezzettini” funzionano anch’essi in modalità kernel e non in modalità utente!

Struttura modulare

- Moderna metodologia di progetto che realizza **kernel modulari caricabili dinamicamente**
  - **ogni componente è caricabile dinamicamente nel kernel** al momento del boot e/o durante l'esecuzione
  - Ad esempio, il kernel Linux carica dinamicamente alcuni moduli per supportare driver delle periferiche e file system
- Simile alla struttura a strati, perchè ogni sezione del kernel ha interface protette ben definite, ma **maggiore flessibilità della struttura a strati perchè ogni modulo può chiamare qualsiasi alro modulo**
- Simile alla struttura a microkernel ma più efficiente perchè **i moduli non richiedono l'invio di messaggi per la comunicazione**
- **Esempi: Solaris, Linux, Windows, Mac OS X**

I moduli caricabili di Sun Solaris

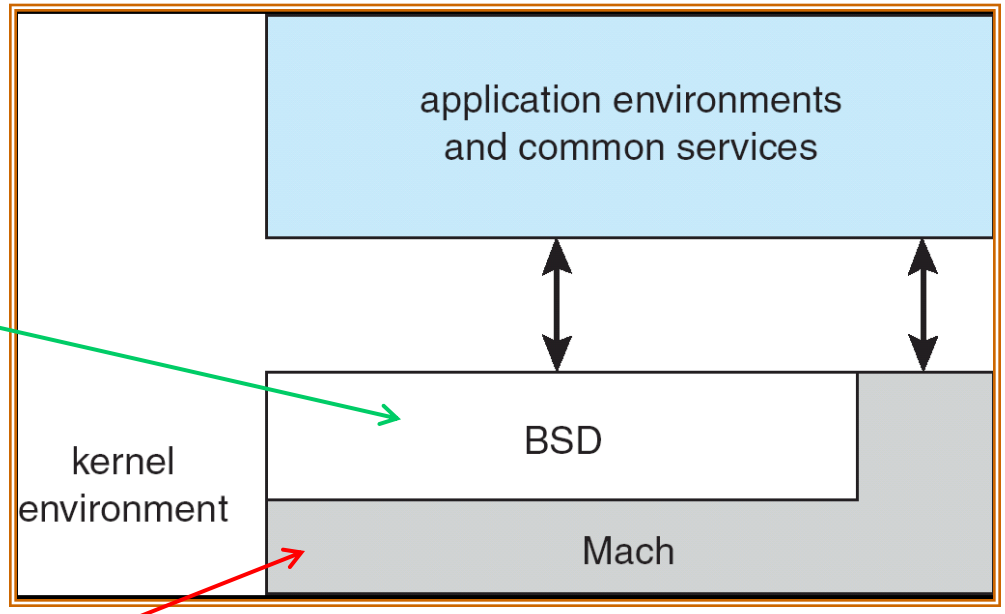


Struttura ibrida

Struttura Mac OS X

- Struttura a 2 strati:
  1. **microkernel Mach (by Carnegie Mellon University)**
  2. **kernel 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



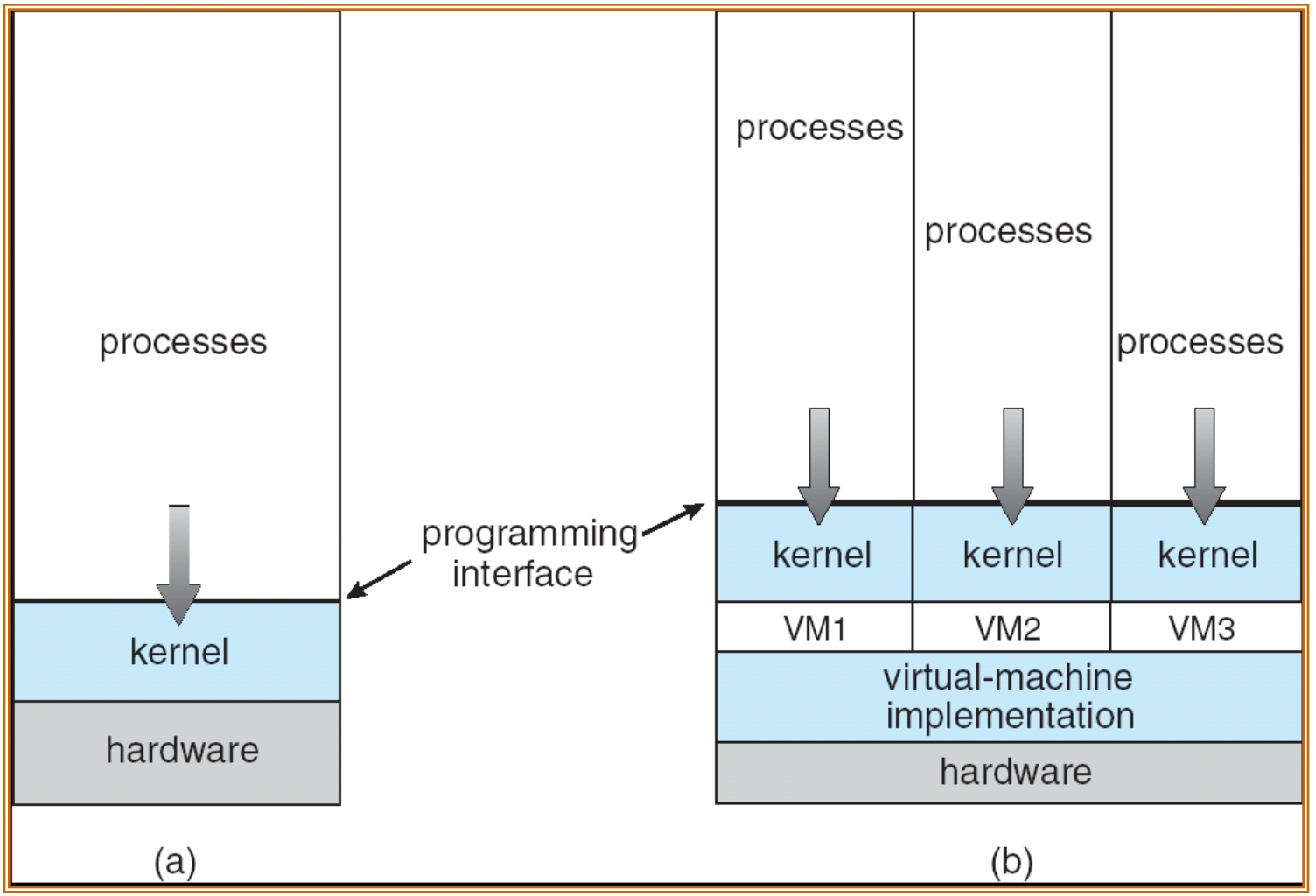
gestione memoria, comunicazione tra processi (scambio dei messaggi), e scheduling

Android: struttura ibrida

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 concorrente su uno stesso hardware
  - Una macchina virtuale fornisce un'interfaccia che è identica al puro hardware sottostante
  - Esempio: Intel ha fornito sul Pentium una modalità virtuale 8086 per eseguire i vecchi programmi MS-DOS
- Il SO crea l'illusione che un processo ospite abbia un proprio processore ed una propria memoria (macchina virtuale)

Struttura a macchine virtuali (2)



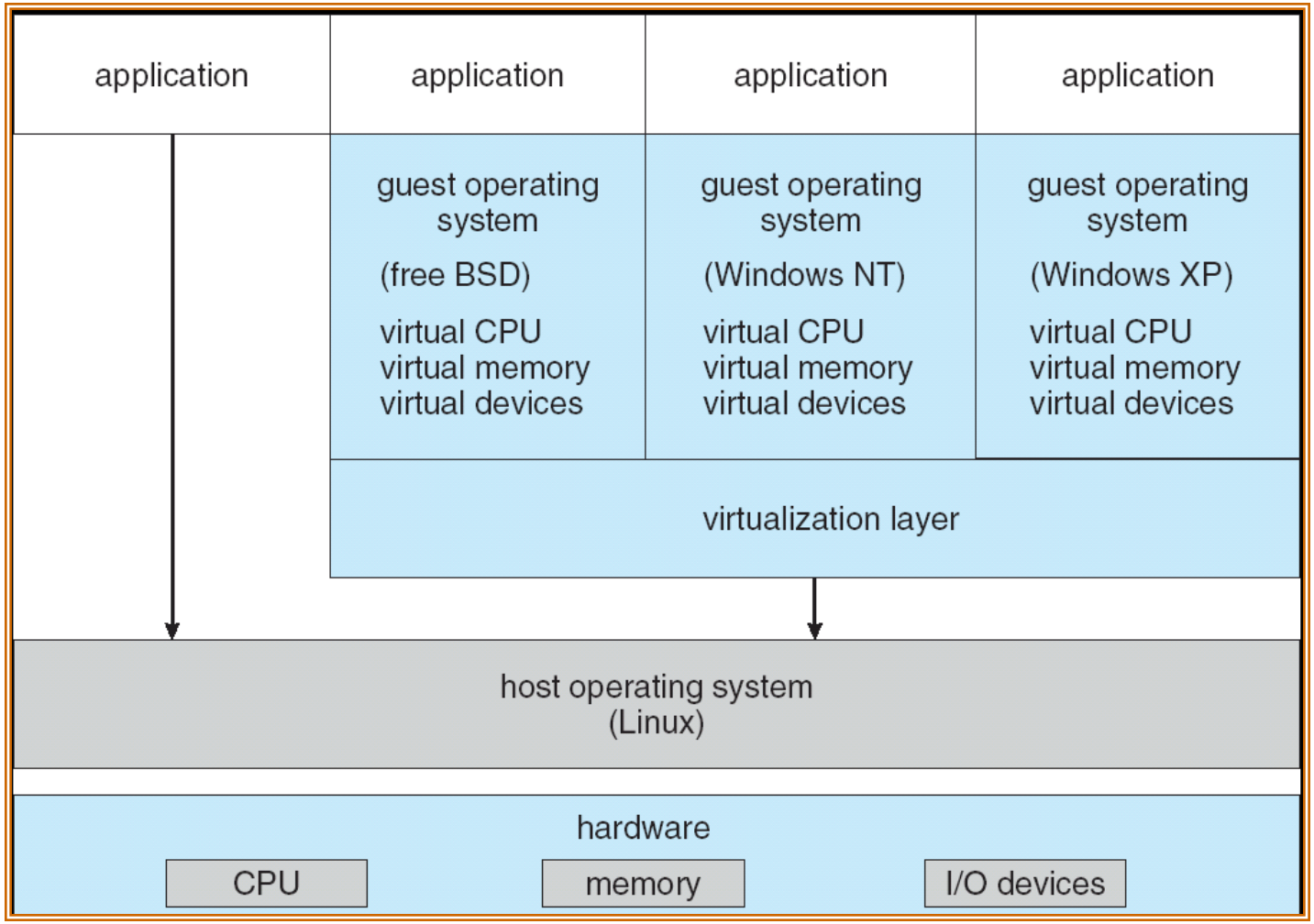
Non-virtual Machine

Virtual Machine

Ogni macchina virtuale può eseguire il proprio SO!

Architettura VMware

Nota applicazione commerciale che converte l'apparato fisico Intel 80x86 in macchine virtuali.



Struttura a macchine virtuali (3)

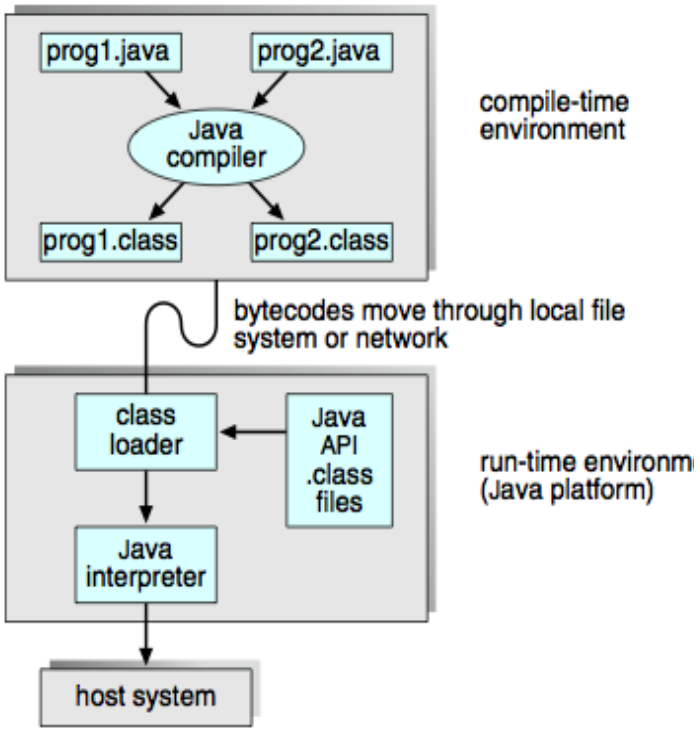
- **Il computer fisico mette a condivisione le proprie risorse per generare macchine virtuali**
  - La schedulazione della CPU e la memoria virtuale può dare l'impressione che ogni *processo ospite* (solitamente un SO) abbia un proprio processore dedicato – *virtual CPU*
  - Lo spooling e il file system possono fornire lettori di schede e stampanti in linea virtuali, e altri dispositivi – *virtual devices, virtual memory*

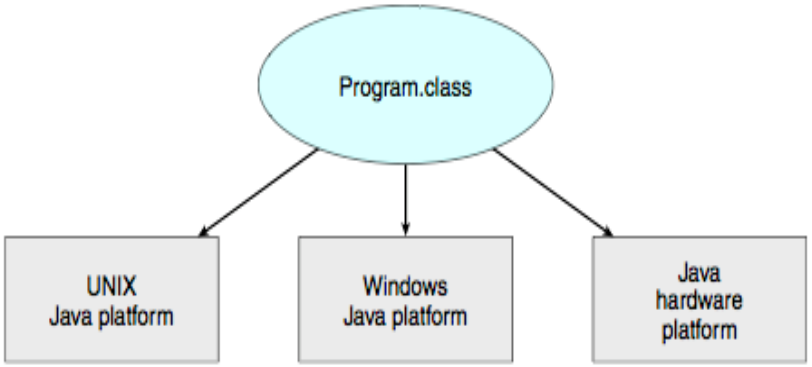
Struttura a macchine virtuali (4)

- **Vantaggi/svantaggi**
  - Comporta la **completa protezione delle risorse** di sistema perchè 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 fare danni
    - Le normali operazioni di sistema raramente fanno sì che si debba interrompere lo sviluppo del sistema
  - **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 il *Cloud Computing* e al supporto HW per la virtualizzazione

La Java Virtual Machine (JVM) (1)

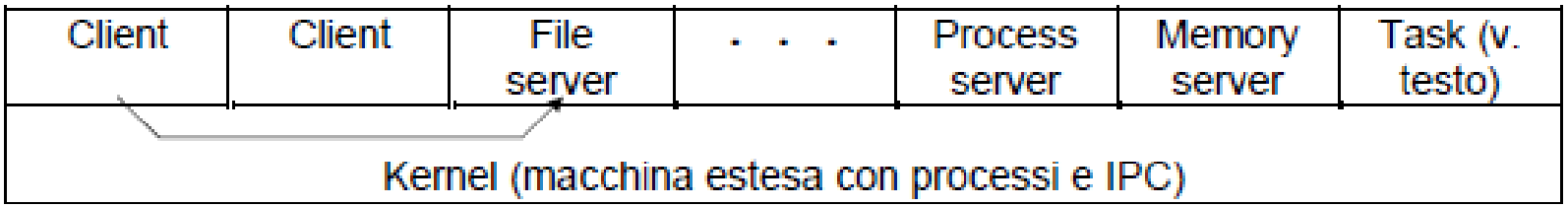
- I programmi Java compilati sono bytecode indipendenti dall'architettura ed eseguiti da una Java Virtual Machine (JVM)
- La JVM è un calcolatore astratto su un sistema ospitante vero





Struttura client/server (1)

- 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

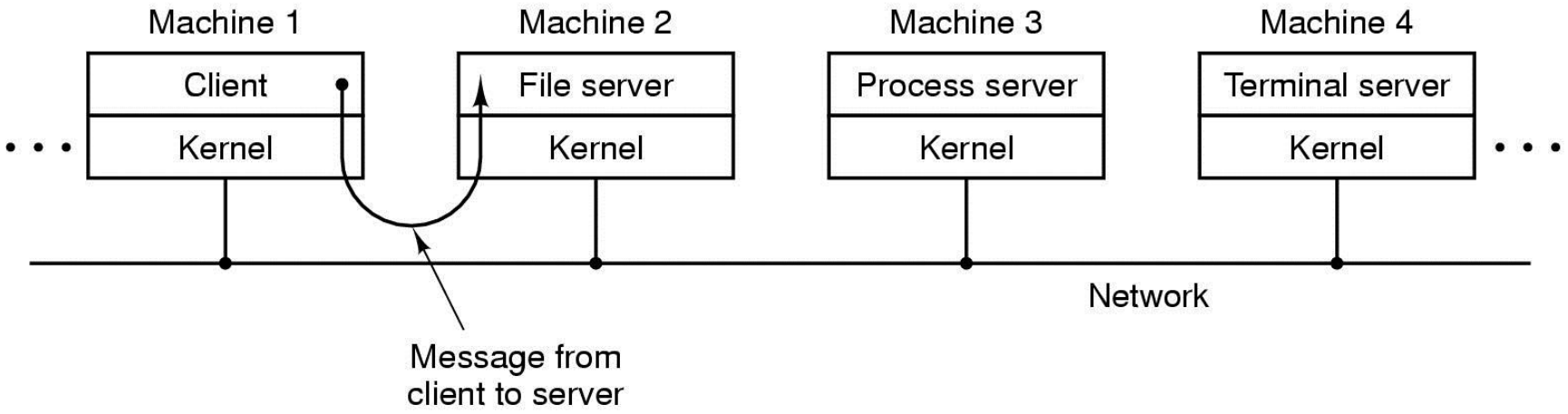


Struttura client/server (2)

- **Vantaggi/svantaggi**
  - **Modularità: suddivisione della funzionalità del SO in moduli**
  - **Portabilità**
  - **Perdita di efficienza**: comunicazione sistemi client-server
- Windows NT 3.0 adottava un modello ispirato al client/server
- Studiato 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