

# **INTRODUZIONE AI CONCETTI BASE DEI S.O.**

**Danilo Croce**

Ottobre 2024



# RIPARTIAMO DALLA DOMANDA DI IERI COS'È UN SISTEMA OPERATIVO?

- Un **moderno calcolatore** è tipicamente formato da:
  - uno o più processori;
  - memoria centrale;
  - dischi;
  - stampanti e altre periferiche di I/O.
- I dettagli di basso livello sono **molto complessi**.
- Gestire tutte queste componenti richiede uno strato intermedio software: il **Sistema Operativo**.



# LO ZOO DEI SISTEMI OPERATIVI

- 1. Sistemi Operativi per Mainframe:** computer di grandi dimensioni usati in grandi aziende e istituzioni.
  - **Funzioni principali:**
    - **Batch processing:** esecuzione di lavori senza interazione dell'utente (es. elaborazione reclami).
    - **Elaborazione di transazioni:** gestione di numerose richieste simultanee (es. prenotazioni aeree).
    - **Timesharing:** molti utenti eseguono lavori contemporaneamente (es. interrogazioni database).
  - **Esempi:** Z/OS (successore di OS/360), usato per applicazioni mission-critical (banche, e-commerce).
- 2. Sistemi Operativi per Server:** Utilizzati su server che servono più utenti attraverso la rete.
  - Forniscono servizi come: **File sharing, database, stampa, hosting web.**
  - Usati da provider di servizi internet e aziende per supportare infrastrutture critiche.
  - **Esempi: Linux, FreeBSD, Windows Server, Solaris.**



# LO ZOO DEI SISTEMI OPERATIVI

3. **Sistemi Operativi per Personal Computer:** Supportano un singolo utente con multiprogrammazione e, spesso, architetture multiprocessore.
  - Utilizzati per compiti quotidiani: **videoscrittura, navigazione web, fogli di calcolo.**
  - **Esempi: Windows, macOS, Linux, FreeBSD.**
  - **Caratteristiche:** facilità d'uso, interfaccia grafica, supporto per applicazioni di produttività.
4. **Sistemi Operativi per Smartphone e Tablet:** Usati in **dispositivi mobili** come **smartphone e tablet.**
  - Mercato dominato da:
    - **Android** (Google) e **iOS** (Apple).
  - Supportano **CPU multicore, GPS, fotocamere** e numerose **app di terze parti** (oltre 3 milioni su Google Play Store).



# LO ZOO DEI SISTEMI OPERATIVI

6. **Sistemi Operativi per IoT e Embedded:** Usati in **dispositivi connessi** (es. frigoriferi, lavatrici, telecamere di sicurezza).
  - Tipicamente **leggeri**, con funzioni specifiche e limitate.
  - Esempi: **Embedded Linux, QNX, TinyOS**.
  - **Caratteristiche:** footprint ridotto (es. **RIOT** può girare su meno di 10 KB).
7. **Sistemi Operativi Real-Time:** Progettati per rispettare **scadenze rigide** nei processi industriali e militari.
  - **Hard real-time:** risposte immediate e precise (es. robotica industriale).
  - **Soft real-time:** occasionali ritardi tollerabili (es. streaming multimediale).
  - Esempi: **eCos, VxWorks**.
8. **Sistemi Operativi per Smart Card:** Utilizzati in **smart card** per pagamenti, autenticazioni e altro. Alimentate tramite contatti o induzione, con capacità limitate.
  - **Java-oriented:** alcune eseguono applet Java per multiprogrammazione.
  - Esempi: Sistemi proprietari, **JavaCard** per applet multiprogrammate.



# COSA HANNO IN COMUNE?

- **Extended Machine**
  - Estensione della funzionalità hardware
  - Astrazione dell'hardware
  - Nascondere i dettagli al programmatore
- **Resource Manager**
  - Protegge l'uso simultaneo/non sicuro delle risorse
  - Condivisione equa delle risorse
  - Resource accounting/limiting



# CONCETTI “BASE” DI UN SISTEMA OPERATIVO

- Il sistema operativo offre funzionalità attraverso le chiamate di sistema
- Gruppi di chiamate di sistema implementano servizi, ad esempio
  - File System Service
  - Process Management Service
- I processi sono astrazioni a livello utente per eseguire un programma per conto dell'utente.
- Ogni processo ha il proprio spazio di indirizzamento
- I dati coinvolti nell'elaborazione vengono recuperati e memorizzati in file.
- I file persistono rispetto ai processi



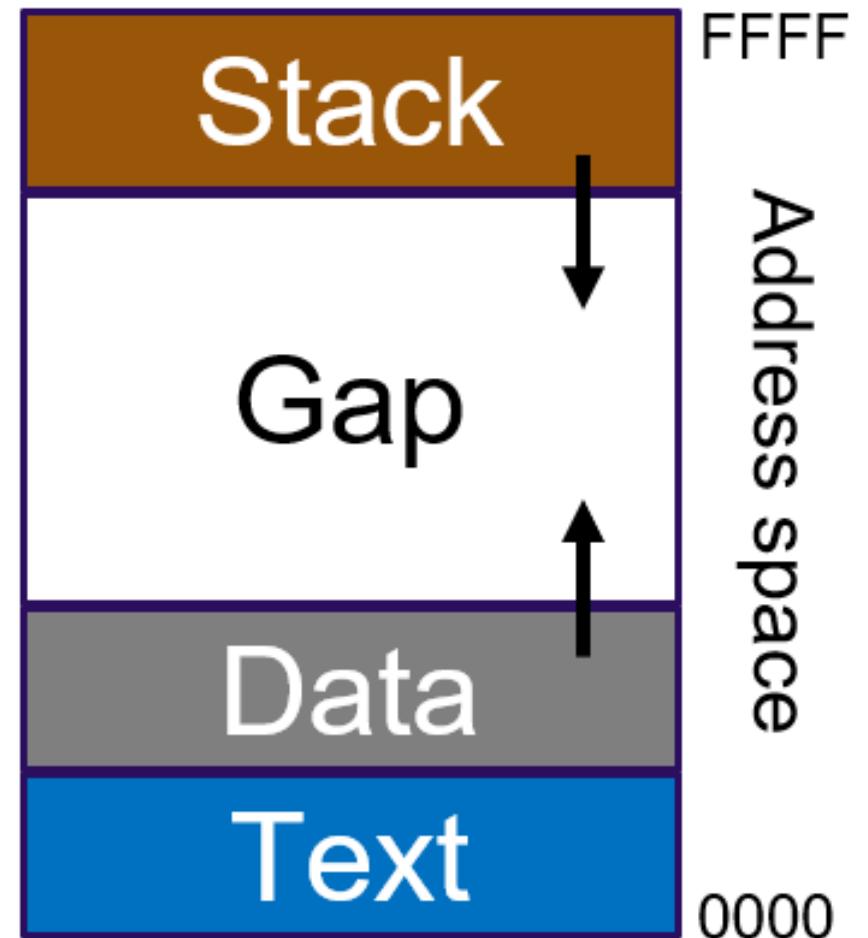
# COSA È UN PROCESSO?

- Concetto chiave in tutti i sistemi operativi
- **Definizione: un programma in esecuzione**
- Il processo è associato
  - a uno spazio di indirizzi
  - un insieme di risorse
    - registri
    - file aperti
    - allarmi
    - ...
- Il processo può essere pensato come un contenitore
  - Contiene tutte le informazioni necessarie per l'esecuzione del programma



# IL LAYOUT DI UN PROCESSO

- Layout dipende da:
  - Architettura della macchina
  - Il Sistema operativo
  - Il programma
- Very basic layout:
  - Stack: Active call data
  - Data: Program variables
  - Text: Program code



# IL CICLO DI VITA DI UN PROCESSO

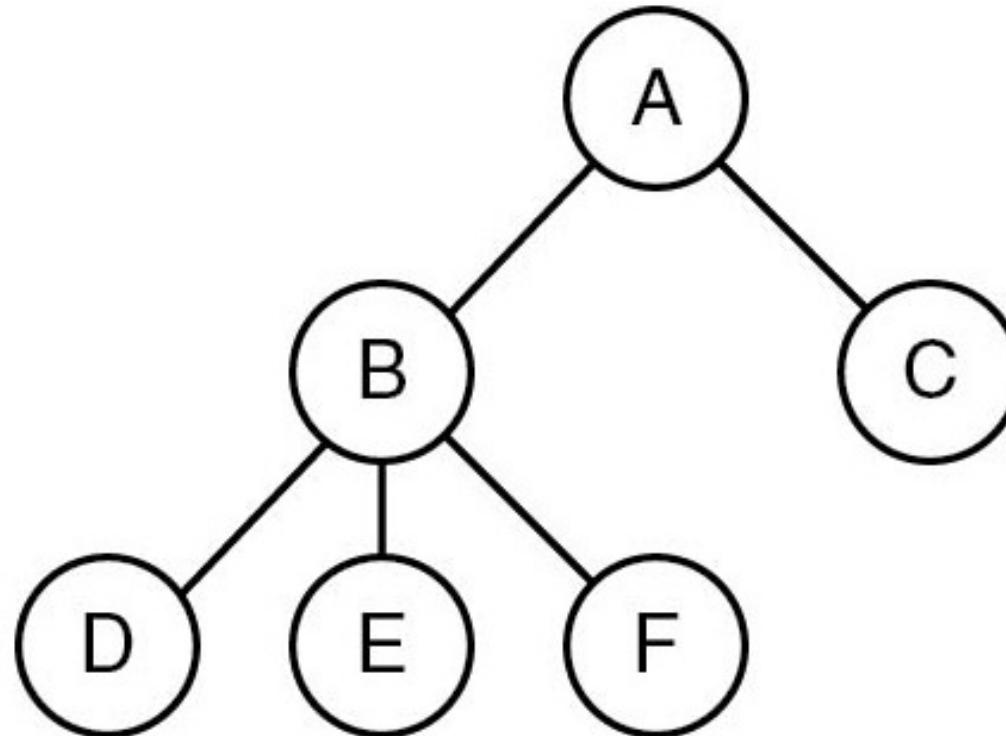
- Le informazioni sui processi sono conservate nella tabella dei processi del sistema operativo.

Un processo sospeso consiste in una voce della tabella dei processi (registri salvati e altre informazioni necessarie per riavviare il processo) e nel suo spazio degli indirizzi.

- Gestione dei processi:
  - Operazioni come la **creazione**, la **terminazione**, la **pausa** e la **ripresa** di un processo.
- Un processo può creare un altro processo
  - Conosciuto come processo "figlio"
  - Crea una gerarchia (o "albero") di processi



# ALBERO DEI PROCESSI



- Un albero di processi. Il processo A ha creato due processi figli, B e C. Il processo B ha creato tre processi figli, D, E e F.



# CHI “POSSIEDE” UN PROCESSO?

- I processi sono "di proprietà" di un utente, identificato da un UID.
  - Ogni processo ha tipicamente l'UID dell'utente che lo ha avviato.
  - Su UNIX, un processo figlio ha lo stesso UID del suo processo padre.
  - Gli utenti possono essere membri di gruppi, identificati da un GUID.

Un processo (superuser/root/administrator) è speciale: ha più privilegi !



# FILE

- **File:** astrazione di un dispositivo di memorizzazione (eventualmente) reale (ad esempio, un disco).
- È possibile leggere e scrivere dati da/su file fornendo una posizione e una quantità di dati da trasferire.
- I file vengono collezionati in directory (o cartelle)
  - Una directory conserva un identificatore per ogni file che contiene.
  - Una directory è un file a sé stante
  - La filosofia UNIX: “*Everything is a file*”.



# FILE

- Le directory e i file formano una gerarchia:
  - La gerarchia inizia dalla “directory principale” or “directory radice” (**root**):
    - /
  - È possibile accedere ai file tramite percorsi assoluti (*absolute path*):
    - /home/ast/todo-list
  - ... o percorsi relativi a partire dalla directory di lavoro corrente:
    - ../courses/slides1.pdf
- Altri filesystem possono essere montati (da **mount**) nella root:
  - /mnt/windows

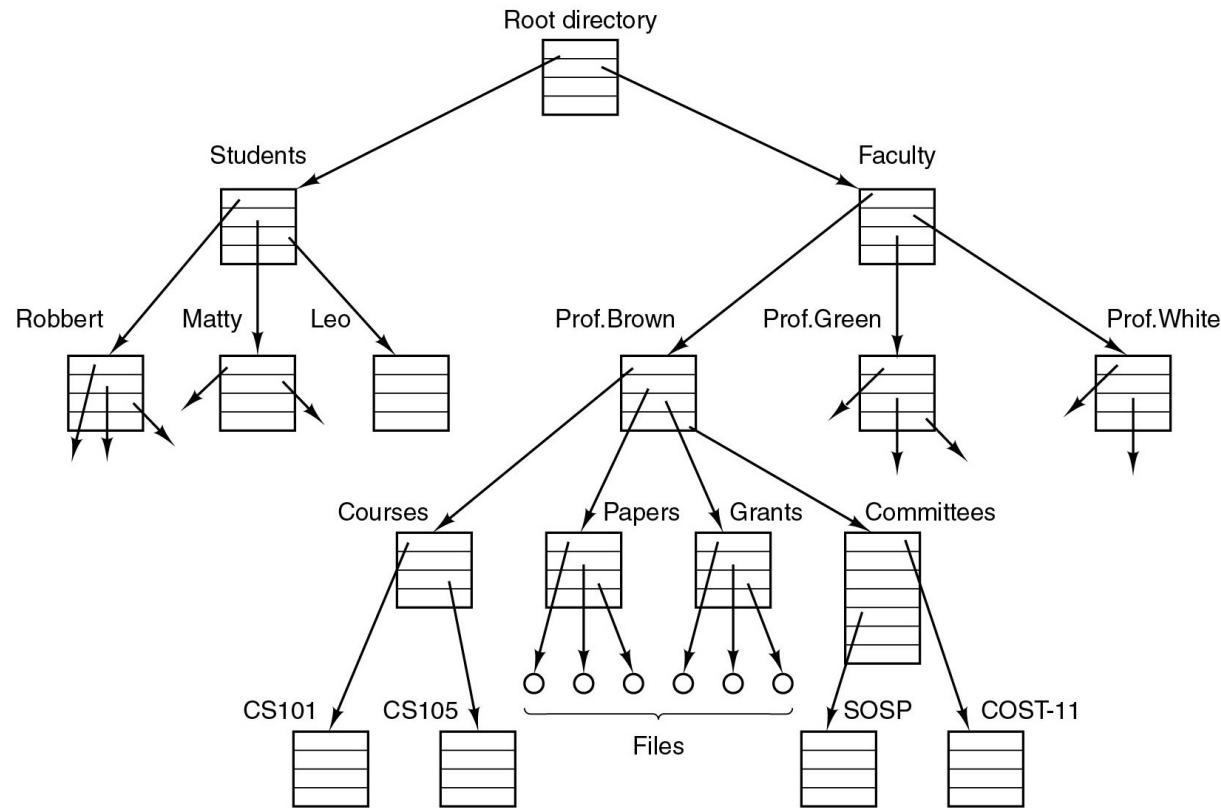


# DIRITTI DI ACCESSO

- I file sono "protetti" da tuple a tre bit per il **proprietario (owner)**, il **gruppo (group)** e gli **altri utenti (other users)**.
- Le tuple contengono un bit (r)ead, (w)rite e un bit e(x)ecute (ma sono disponibili più bit)
- Example:
  - rwxr-x--x** myuser my group 14492 Dec 4 18:04 my file
    - **Owner** is allowed to execute, modify, read the file
    - **Group** is allowed to read and execute the file
    - **Other** users are only allowed to execute the file



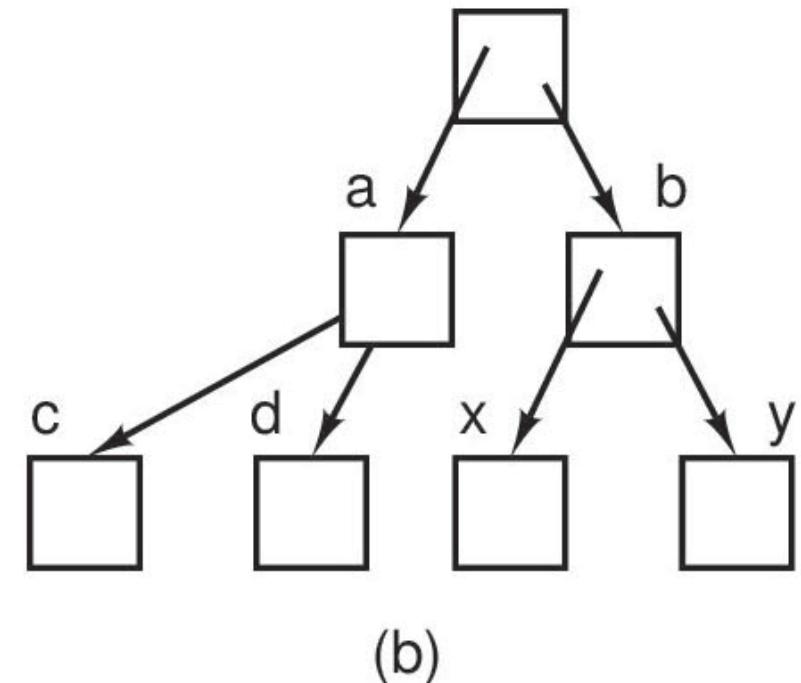
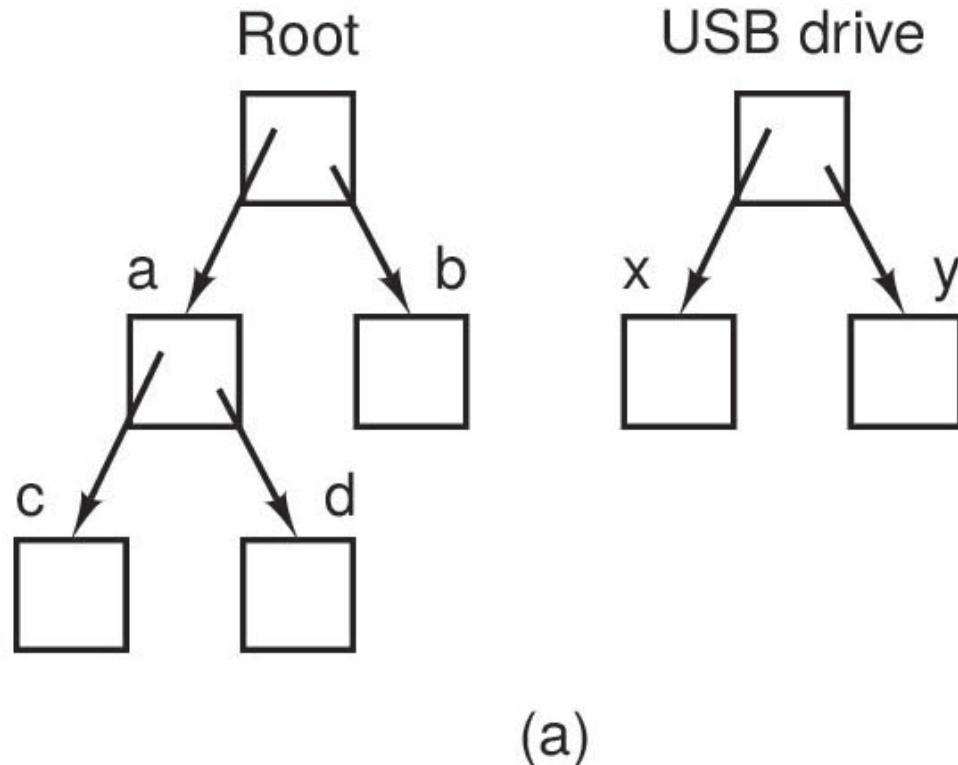
# UN ESEMPIO DI ORGANIZZAZIONE DI UN FILE SYSTEM



Un file system per un dipartimento universitario.



# FILE E RISORSE



- a) Prima del *mount*, i file dell'unità USB non sono accessibili.
- b) Dopo aver eseguito l'operazione di *mount*, fanno parte della gerarchia dei file.



# ACCESSO AI FILE: UN ESEMPIO

```
herbertb@sleet:~/tmp$ pwd  
/home/herbertb/tmp  
  
herbertb@sleet:~/tmp$ mkdir os  
  
Herb ertb@sleet:~/tmp$ cat > os/hello.sh  
#!/bin/sh  
  
echo "hello world!"  
  
herbertb@sleet:~/tmp$ chmod 744 os/hello.sh  
  
herbertb@sleet:~/tmp$ os/hello.sh  
hello world!  
  
herbertb@sleet:~/tmp$ chmod 644 os          ← What happens here?  
  
herbertb@sleet:~/tmp$ cat > os/qq.txt      ← Will this work?  
  
herbertb@sleet:~/tmp$ ls os/                ← Will this work?  
  
herbertb@sleet:~/tmp$ os/hello.sh          ← Will this work?  
  
herbertb@sleet:~/tmp$ rm os/hello.sh        ← Will this work?
```



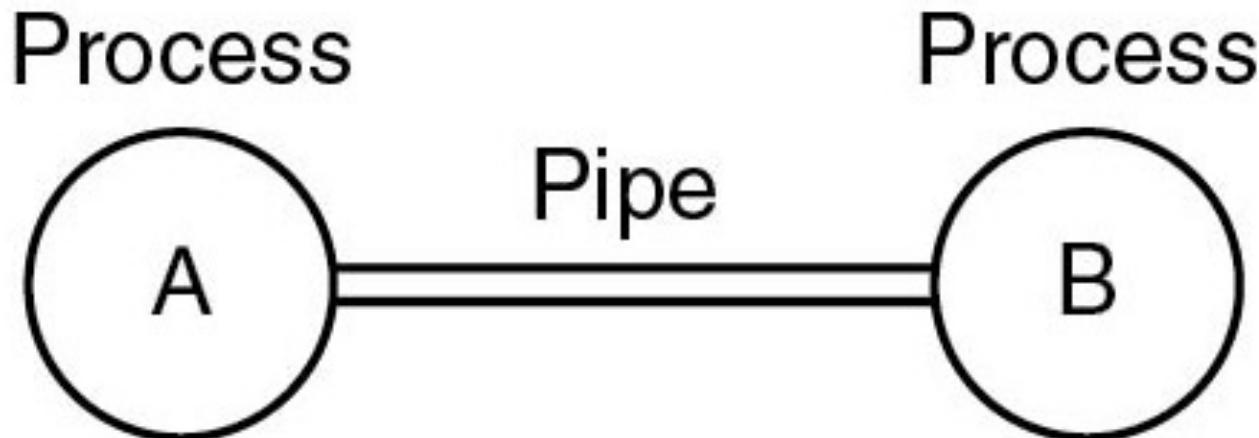
# “EVERYTHING IS A FILE”:

- Hardware devices are abstracted as files:  
**Block** special files, esempio i dischi:  
`b rw-rw---- 1 root root 8, 2 Dec 4 18:04 /dev/sda2`
- **Character** special files, ad esempio le porte seriali:  
`c rw-rw---- 1 root root 4, 64 Dec 4 18:04 /dev/ttyS0`
- Altri file speciali:  
symbolic links  
named/anonymous FIFOs (sockets/pipes)
- “Everything is a file *descriptor*”



# FILE E PIPE

- Pipes: pseudo file che consentono ai processi di comunicare su un canale FIFO
- Deve essere impostato in anticipo
- Sembra un file "normale" per leggere e scrivere da/sui processi in esecuzione.



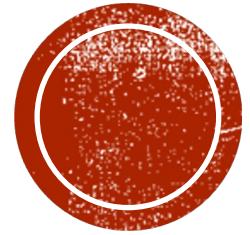
Due processi collegati da una pipe



# TERMINI IMPORTANTI

- Path
- Folder / map / directory
- Root directory
- Working directory
- File descriptor
- Mounting
- Block/character special files
- Pipe





# **LE CHIAMATE DI SISTEMA (o SYSTEM CALL)**

# CHIAMATE DI SISTEMA

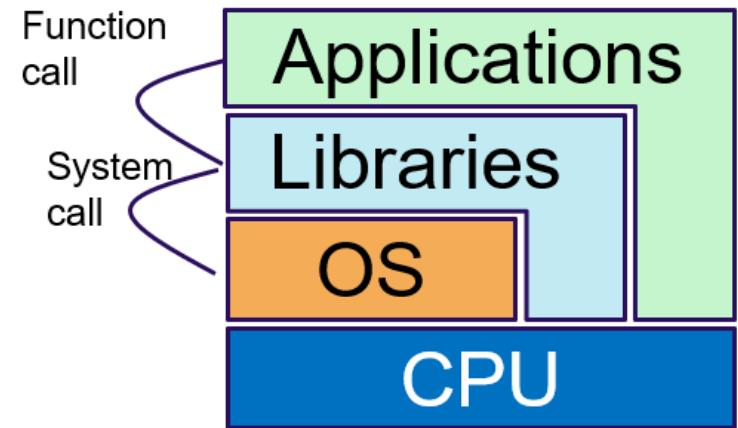
Le chiamate di sistema sono l'**interfaccia che il sistema operativo offre alle applicazioni per richiedere servizi**.

## Problema:

- Il meccanismo delle chiamate di sistema è **altamente specifico** del sistema operativo e dell'hardware.
- La **necessità di efficienza esaspera** questo problema

## Soluzione:

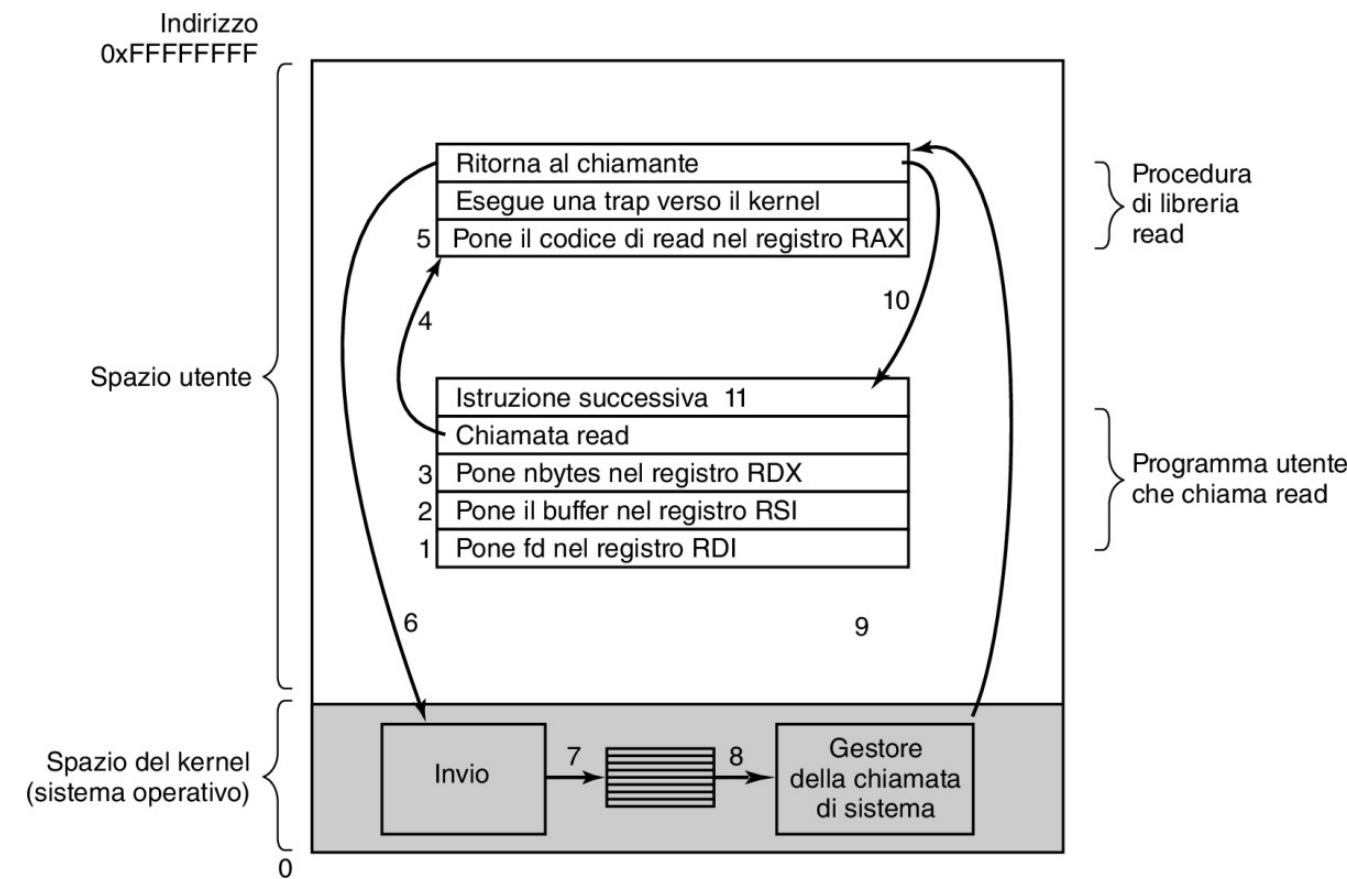
- Incapsulare le chiamate di sistema nella libreria C (`libc`)
- Tipicamente esporta **una chiamata di libreria per ogni chiamata di sistema**
- UNIX `libc` si basa sulla libreria C POSIX
- Si noti che esistono molte librerie C UNIX... :-/



# I 10 PASSI PER EFFETTUARE LA CHIAMATA DI SISTEMA

read(fd, buffer, nbytes)

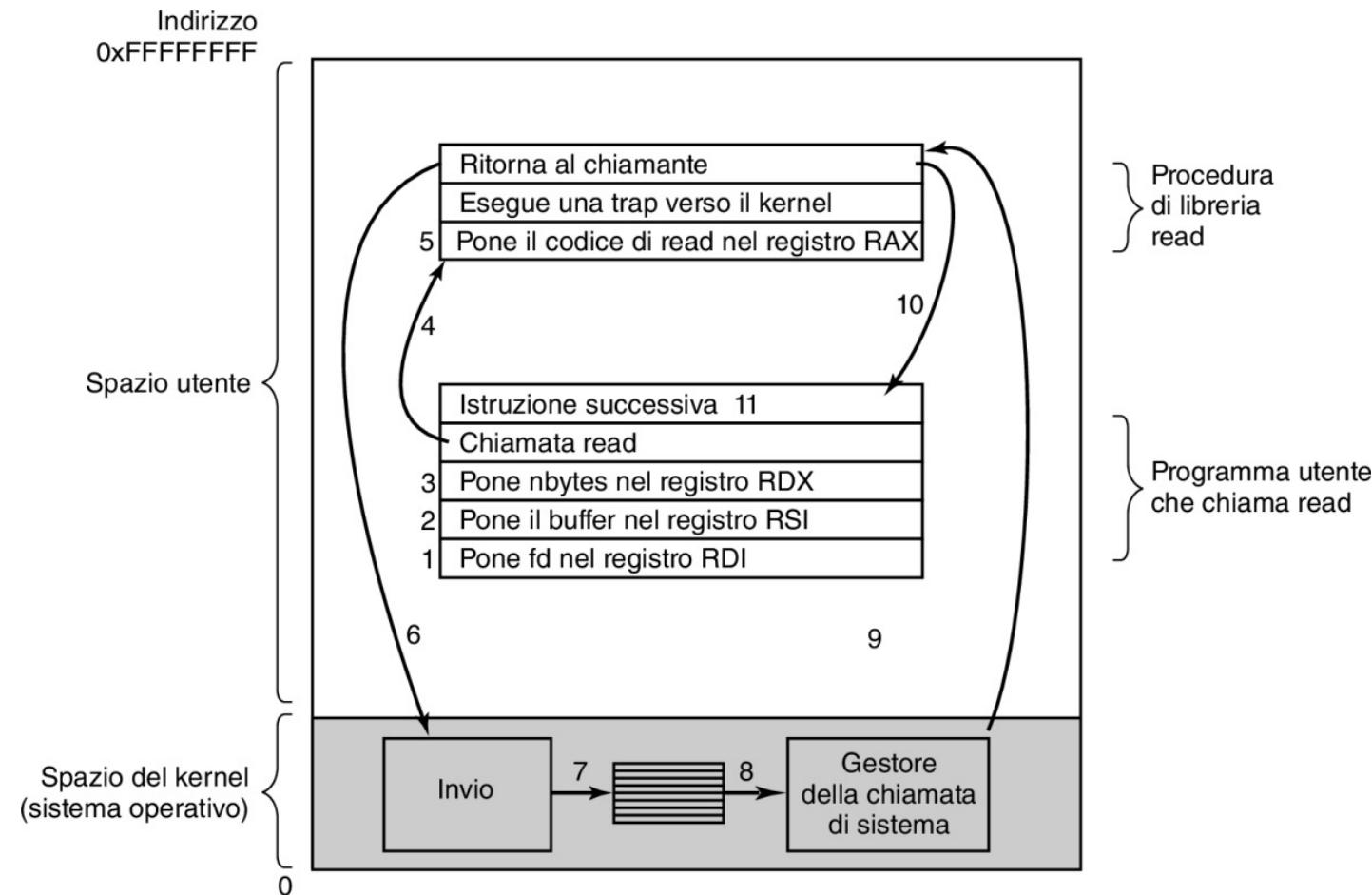
- **Preparazione dei parametri:** il programma chiamante prepara i parametri (i.e., fd, buffer, nbytes)
  - solitamente memorizzandoli nei registri (es, RDI, RSI, RDX) o nello stack (se di più)
- **Chiamata alla procedura di libreria:** Il programma effettua la chiamata alla procedura di libreria (step 4)
- **Collocazione del numero di chiamata di sistema:** colloca il numero della chiamata di sistema in un registro, come RAX (passaggio 5).
  - Questo numero identifica quale funzione del kernel deve essere eseguita (es. read, write, open, ecc.).
  - Il kernel può consultare una tabella di chiamate di sistema, dove ogni numero è associato a un gestore specifico.
- **Passaggio a modalità kernel:** si esegue un'istruzione "trap" (ad esempio, SYSCALL in x86-64)
  - passare dalla modalità utente a quella kernel (passaggio 6).
  - L'istruzione "trap" è simile a una chiamata di procedura ma cambia la modalità in modalità kernel
  - può saltare solo a indirizzi specifici o indici di una tabella di memoria, a differenza della chiamata di procedura.



# I 10 PASSI PER EFFETTUARE LA CHIAMATA DI SISTEMA

read(fd, buffer, nbytes)

- **Esecuzione del gestore di chiamate di sistema:** Il gestore di chiamate di sistema specifico viene eseguito (passaggio 8).
- **Ritorno alla procedura di libreria utente:** dopo, il controllo può essere restituito alla procedura di libreria utente, all'istruzione successiva all'istruzione "trap" (passaggio 9).
- **Possibilità di blocco:** La chiamata di sistema può bloccare il chiamante, ad esempio, se l'input desiderato non è disponibile.
  - Il sistema operativo può quindi eseguire altri processi
- **Ripresa dopo il blocco:** Quando l'input o le condizioni desiderate sono disponibili, il processo bloccato viene ripreso,
  - Tornando alla procedura di libreria utente e procedendo all'istruzione successiva (passo 10).



# CHIAMATE DI SISTEMA PER LA GESTIONE DEI PROCESSI

Call	Description
pid fork()	Creare un processo figlio identico al genitore
pid waitpid( pid, &statloc, options)	Attendere che un processo figlio termini
s = execve(name, argv, environp)	Sostituire l'immagine centrale di un processo
exit(status)	Terminare l'esecuzione del processo e restituire lo stato

- Alcune delle principali chiamate di sistema POSIX.
- Il codice di ritorno s è -1 se si è verificato un errore. I codici di ritorno sono i seguenti: pid è l'id di un processo



# CHIAMATE DI SISTEMA PER LA GESTIONE DEI PROCESSI

Call	Description
fd = open(file, how, ...)	Apre un file per la lettura, la scrittura o entrambe le operazioni.
s = close(fd)	Chiude un file aperto
n = read(fd, buffer, nbytes)	Legge dati da un file in un buffer
n = write(fd, buffer, nbytes)	Scrive dati da un buffer in un file
Position = lseek(fd, offset, whence)	Sposta il puntatore del file
s = stat(name, &buf)	Ottiene informazioni sullo stato di un file

- Il codice di ritorno s è -1 se si è verificato un errore. I codici di ritorno sono i seguenti: fd è un descrittore di file, n è un conteggio di byte, position è un offset all'interno del file.



# CHIAMATE DI SISTEMA PER LA GESTIONE DEL FILE SYSTEM

Call	Description
s = mkdir(name, mode)	Crea una nuova directory
s = rmdir(name)	Rimuove una directory vuota
s = link(name1 , name2)	Crea una nuova voce, nome2, che punta a nome1
s = unlink(name)	Rimuove una voce della directory
s = mount(special, name, flag)	Monta un file system
s = umount(special)	Smonta un file system



# ALTRÉ CHIAMATE DI SISTEMA

Call	Description
s = chdir(dirname)	Cambia la directory di lavoro
s = chmod(name, mode)	Modifica i bit di protezione di un file
s = kill(pid, signal)	Invia un segnale a un processo (NON UCCIDE)
s = time(&seconds)	Ottiene il tempo trascorso dal 1° gennaio 1970



# UN ESEMPIO

```
#define TRUE 1

while (TRUE) {
    type_prompt( );
    read_command(command, parameters);

    if (fork( ) != 0) {
        /* Parent code. */
        waitpid(-1, &status, 0);
    } else {
        /* Child code. */
        execve(command, parameters, 0);
    }
}
```

/\* repeat forever \*/  
/\* display prompt on the screen \*/  
/\* read input from terminal \*/  
  
/\* fork off child process \*/  
  
/\* wait for child to exit \*/  
  
/\* execute command \*/

Si assume che TRUE sia definito come 1.



# API DI WINDOWS

UNIX	Win32	Descrizione
fork	CreateProcess	<b>Crea un nuovo processo</b>
waitpid	WaitForSingleObject	Può attendere l'uscita di un processo
execve	(none)	<b>Createprocess = fork + execve</b>
exit	ExitProcess	Termina l'esecuzione
open	createFile	<b>Crea un file o apre un file esistente</b>
close	CloseHandle	<b>Chiude un file</b>
read	ReadFile	<b>Legge dati da un file</b>
Write	WriteFile	<b>Scrive dati in un file</b>
lseek	SetFilePointer	<b>Sposta il puntatore del file</b>
stat	GetFileAttributesEx	<b>Ottiene vari attributi del file</b>
mkdir	CreateDirectory	<b>Crea una nuova directory</b>

- Le chiamate API Win32 che corrispondono grosso modo alle chiamate UNIX

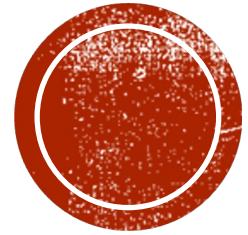


# API DI WINDOWS (2)

- Le chiamate API Win32 che corrispondono grosso modo alle chiamate UNIX

UNIX	Win32	Description
Iseek	SetFilePointer	Move the tile 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, so no umount
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	GetLocamme	Get the current time



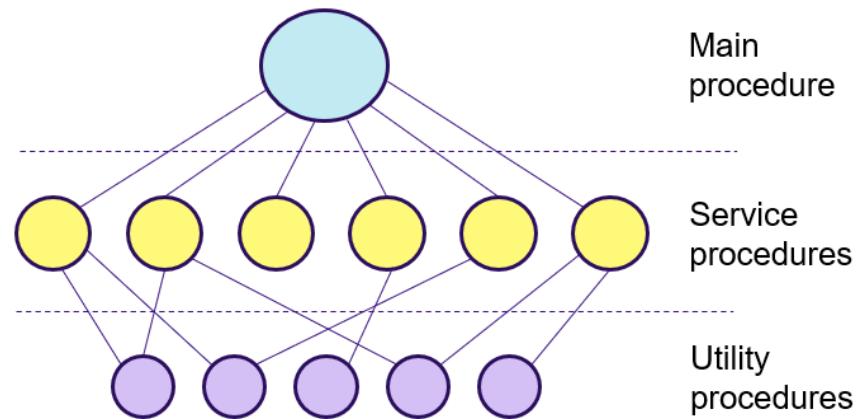


# STRUTTURA DI UN SISTEMA OPERATIVO



# STRUTTURA DI UN SISTEMA OPERATIVO: MONOLITICO

- Il programma principale invoca le chiamate di sistema richieste
- Il kernel è un blocco monolitico con:
  - Procedure di servizio che eseguono le chiamate di Sistema
  - Procedure di utilità che aiutano a implementare le procedure di servizio



# ALCUNE CONSIDERAZIONI

- **Sistemi Operativi Monolitici:** un approccio al design «tutto in uno»
  - il kernel è un'unica unità grande e interconnessa.
  - tutte le funzioni del sistema operativo, come la gestione dei processi, la gestione della memoria e la gestione dei dispositivi di I/O, sono strettamente integrate in un unico spazio di indirizzamento.
- **Flessibilità vs Complessità:**
  - offre una certa flessibilità in termini di prestazioni e design.
  - tuttavia, dato che tutto è strettamente interconnesso, un bug o un errore in una parte del sistema può causare problemi in altre parti, potenzialmente portando a crash sistematici.
- **Compilazione e Collegamento:**
  - tutte le funzioni e procedure del sistema operativo devono essere compilate e collegate in un unico eseguibile del kernel.



# ALTRÉ CONSIDERAZIONI

- **Mancanza di occultamento:**

- tutte le procedure possono, in teoria, accedere a qualsiasi altra procedura o variabile all'interno del kernel.
- Non c'è un vero e proprio "occultamento" o isolamento tra le diverse parti del sistema.

- **Utilizzo di «trap»:**

- meccanismo attraverso il quale un programma può richiedere i servizi del sistema operativo.
  - avviene attraverso interruzioni software che trasferiscono il controllo al sistema operativo.

- **Una «struttura a tre strati»**

- una suddivisione del sistema in livelli, spesso user mode, kernel mode e hardware, con il «trap» che agisce come meccanismo di comunicazione tra questi livelli.



# ALTRÉ CONSIDERAZIONI

## ▪ **Estensioni caricabili:**

- Molti sistemi operativi **permettono di caricare dinamicamente componenti aggiuntivi**, come driver di dispositivi o file system, senza dover riavviare o ricompilare l'intero sistema operativo.
- Questi componenti possono essere caricati e scaricati a seconda delle necessità, offrendo una certa modularità anche in un sistema monolitico.

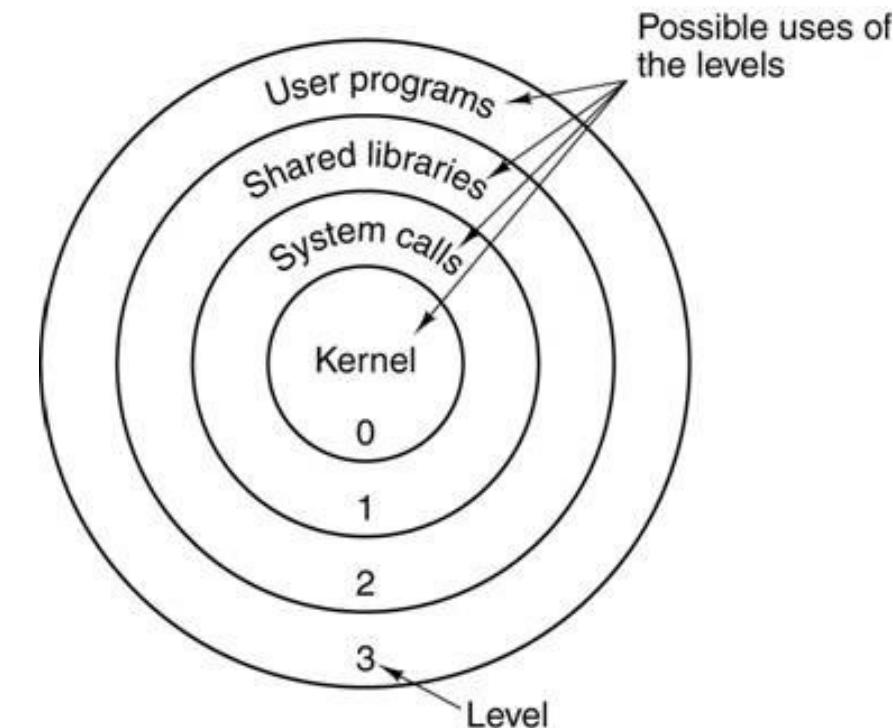
## ▪ **Librerie Condivise e DLL:**

- Sia UNIX che Windows supportano l'idea di librerie di codice che possono essere condivise tra più programmi.
  - In UNIX, queste sono chiamate "librerie condivise",
  - In Windows sono chiamate "Dynamic Link Libraries" (DLL).
- contengono codice che può essere eseguito da più programmi contemporaneamente, riducendo la necessità di avere copie multiple del medesimo codice in memoria.



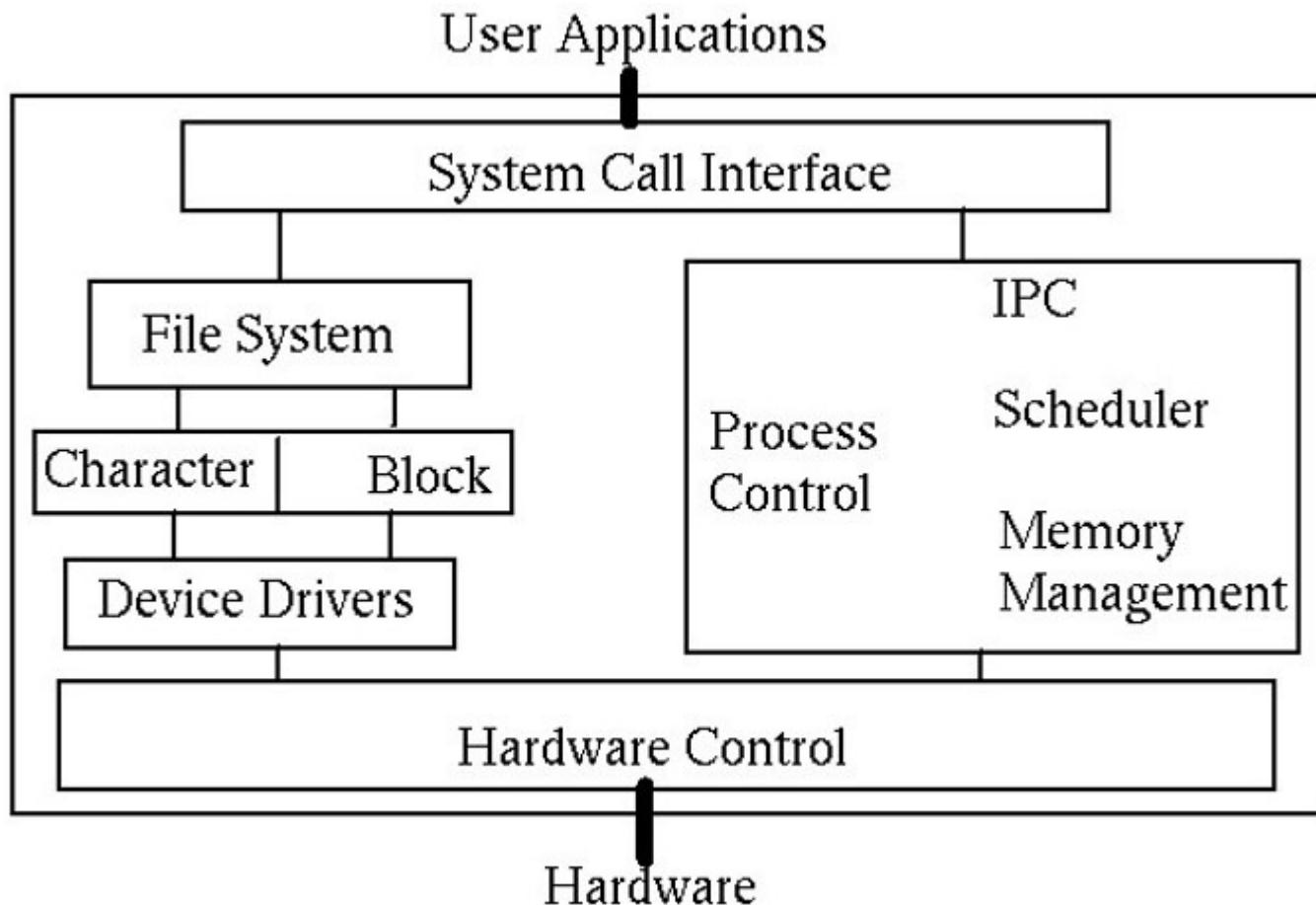
# STRUTTURA DI UN SISTEMA OPERATIVO: MONOLITICO

- L'organizzazione stratificata dei sistemi operativi è una generalizzazione dell'approccio monolitico.
- Il sistema THE fu uno dei primi a implementare questa idea, con sei livelli gerarchici.
  - Questi livelli gestivano l'allocazione del processore, la memoria, la comunicazione, l'I/O, i dispositivi, e gli utenti.
- Il sistema MULTICS usava anelli concentrici per definire i privilegi, con livelli interni più privilegiati di quelli esterni.
- Vantaggi:
  - Protezione delle risorse e dati critici.
  - Separazione chiara dei compiti (es. valutazione degli studenti).



# STRUTTURA DI UN SISTEMA OPERATIVO: MONOLITICO

- **Kernel Unificato:** Tutte le funzionalità centralizzate in un unico kernel.
- **Interconnessione:** Ogni componente ha la capacità di richiamare qualsiasi altro componente.
- **Scalabilità:** Questa struttura può diventare complessa e meno gestibile con l'evoluzione del sistema.



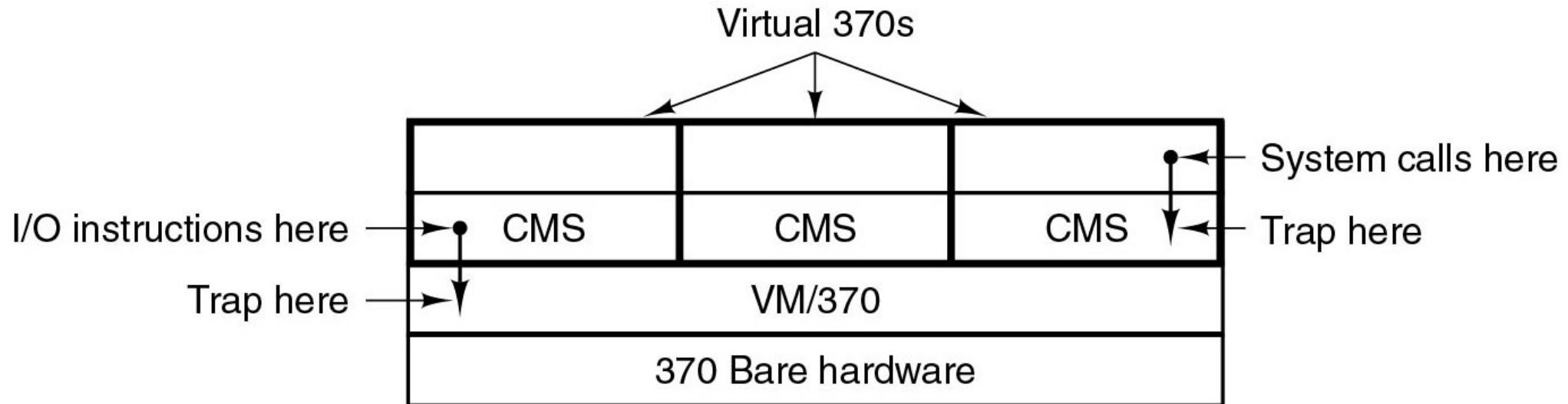
# ORIGINE DELLE MACCHINE VIRTUALI

- Le **macchine virtuali (VM)** permettono l'esecuzione di più sistemi operativi su un unico hardware fisico, simulando un ambiente separato per ciascuno.
- **Origini:** Il sistema **VM/370** di IBM (anni '70) è stato uno dei primi a implementare macchine virtuali, creando ambienti virtuali identici all'hardware fisico, capaci di eseguire diversi sistemi operativi contemporaneamente.
- **Vantaggi:**
  - **Isolamento:** Ogni VM opera indipendentemente.
  - **Flessibilità:** Possibilità di eseguire diversi sistemi operativi simultaneamente (es. OS/360, CMS).
  - **Gestione Semplificata:** Facilita la gestione e manutenzione separando multiprogrammazione e risorse hardware.
- **Uso Moderno:** Il successore **z/VM** viene utilizzato sui mainframe IBM serie Z per far girare più sistemi operativi completi, come **Linux**, in ambienti ad alta intensità di transazioni e dati.



# STRUTTURA DI UN SISTEMA OPERATIVO: VIRTUALIZZAZIONE (1 DI 2)

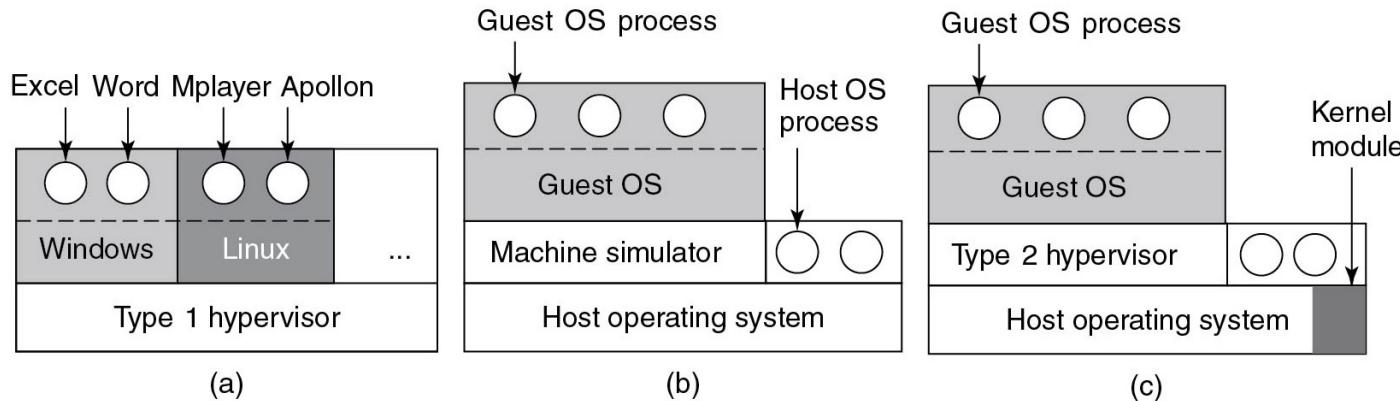
- Inventato negli anni '70 per separare la multiprogrammazione dalla macchina estesa
- Oggi di nuovo interesse in diversi ambiti
- N interfacce di chiamata di sistema indipendenti dal sistema operativo



The structure of VM/370 with Conversational Monitor System (CMS)

# STRUTTURA DI UN SISTEMA OPERATIVO: VIRTUALIZZAZIONE (2 DI 2)

- Virtual machine monitor (VMM) o Hypervisor emula l'hardware
  - **Type 1:** VMM viene eseguito sul "pezzo di ferro" (direttamente su HW, come Xen)
  - **Type 2:** VMM ospitato nel sistema operativo (esempio: QEMU)
    - In teoria non c'è nessuna specializzazione da parte del Sistema operativo sottostante (fig. b)
    - Nella pratica esistono dei moduli del Kernel usati per ottimizzare il processo di simulazione (fig. c)



(a) A type 1 hypervisor. (b) A pure type 2 hypervisor. (c) A practical type 2 hypervisor



# CONTAINER

- I container possono eseguire più istanze di un sistema operativo su una singola macchina.
- Ogni container condivide il kernel del sistema operativo host e i file binari e le librerie.
  - Il container non contiene il sistema operativo completo e può quindi essere leggero
- Gli svantaggi dei container
  - Non è possibile eseguire un container con un sistema operativo completamente diverso da quello dell'host
  - A differenza delle macchine virtuali, non esiste un rigido partizionamento delle risorse.
  - I container sono isolati a livello di processo
    - Se un container altera la stabilità del kernel sottostante, ciò può influire sugli altri container.



# STRUTTURA DI UN SISTEMA OPERATIVO: EXOKERNEL

- **Idea:** Separare il controllo delle risorse dalla macchina estesa
- Simile a un VMM/Hypervisor, ma:
  - Exokernel non emula l'hardware
  - Fornisce solo una condivisione sicura delle risorse a basso livello
- Ogni macchina virtuale a livello utente esegue il suo sistema operativo, ma è limitata a utilizzare solo le risorse assegnate.
- Rispetto ad altri approcci, l'exokernel elimina la necessità di mappature complesse, concentrandosi solo su quale macchina virtuale ha accesso a quali risorse.



# **STRUTTURA DI UN SISTEMA OPERATIVO: UNIKERNEL**

- Gli unikernel sono sistemi minimi basati su LibOS, progettati per eseguire una singola applicazione su una macchina virtuale
  - Esempio WebServer
- Questi sistemi contengono solo la funzionalità necessaria per supportare l'applicazione specifica, come un server web, su una macchina virtuale.
- Gli unikernel sono altamente efficienti poiché non richiedono protezione tra il sistema operativo (LibOS)
  - esiste solo un'applicazione per macchina virtuale.
- Il concetto degli unikernel è stato recentemente riscoperto, offrendo una soluzione leggera ed efficiente per eseguire applicazioni isolate su macchine virtuali.

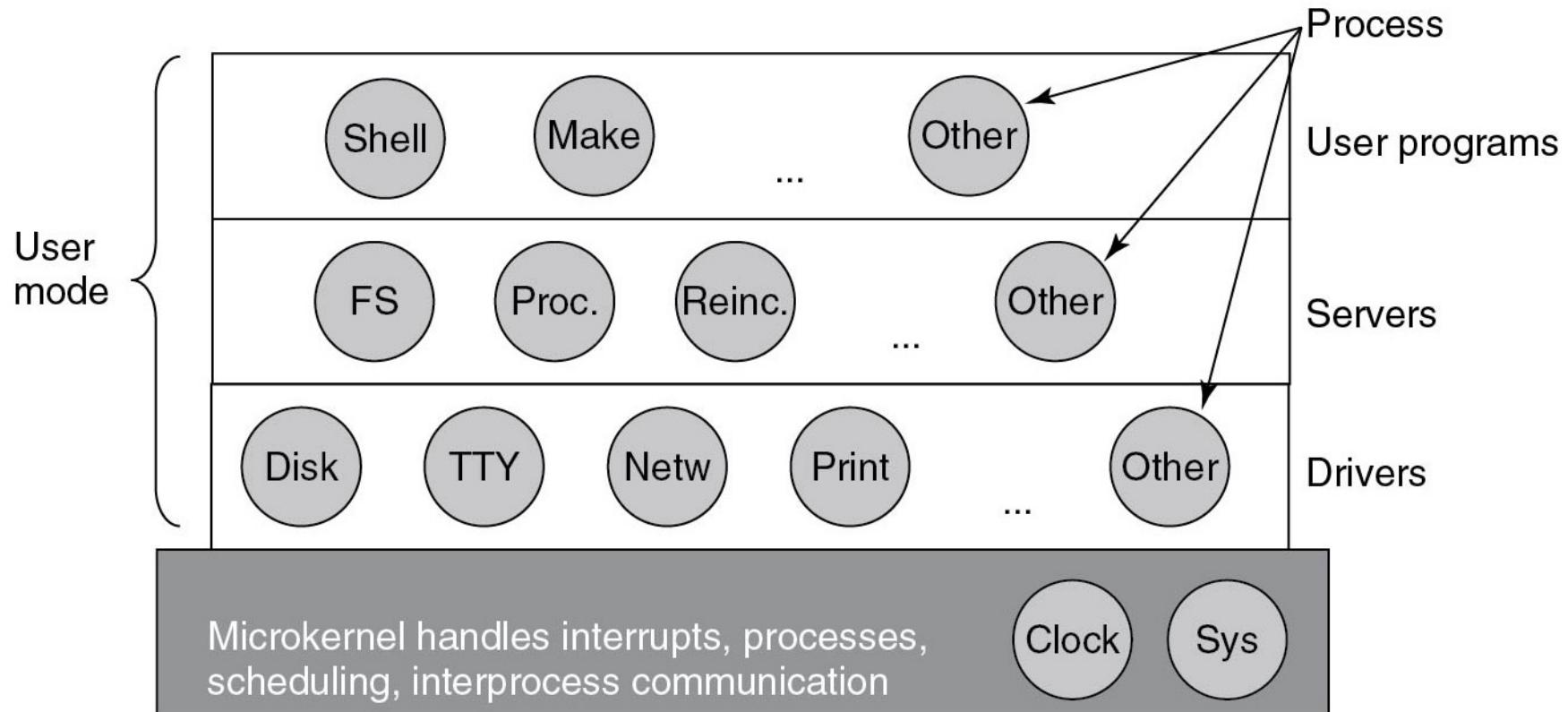


# STRUTTURA DI UN SISTEMA OPERATIVO: MICROKERNEL-BASED CLIENT/SERVER (1 OF 3)

- Organizza le *service procedure* che vengono eseguiti in modo separato  
processes => **System Servers/Drivers**
- I processi di sistema comunicano attraverso il passaggio di messaggi
- Le chiamate di sistema si basano sullo stesso meccanismo di messaggistica
- Meccanismo di messaggistica implementato nel kernel minimale => **Microkernel**



# STRUTTURA DI UN SISTEMA OPERATIVO: MICROKERNEL-BASED CLIENT/SERVER (2 OF 3)



Struttura semplificata del sistema MINIX



# STRUTTURA DI UN SISTEMA OPERATIVO: MICROKERNEL-BASED CLIENT/SERVER (3 OF 3)

- **Pro:** è più facile aderire al Principle of Least Authority (POLA):
  - Trusted Computing Base (TCB) relativamente “piccolo”
  - Ogni processo del sistema operativo può fare solo ciò che è necessario per svolgere il proprio compito
  - La compromissione, ad esempio, del driver di stampa non influisce sul resto del sistema operativo.
- **Contra:**
  - Il passaggio di messaggi è più lento di una chiamata di funzione (come in un kernel monolitico)

