

Sistemi Operativi e Reti di Calcolatori (SOReCa)

Corso di Laurea in *Ingegneria Informatica e Automatica (BIAR)*

Terzo Anno | Primo Semestre

A.A. 2024/2025

Inter-Process Communication (IPC)

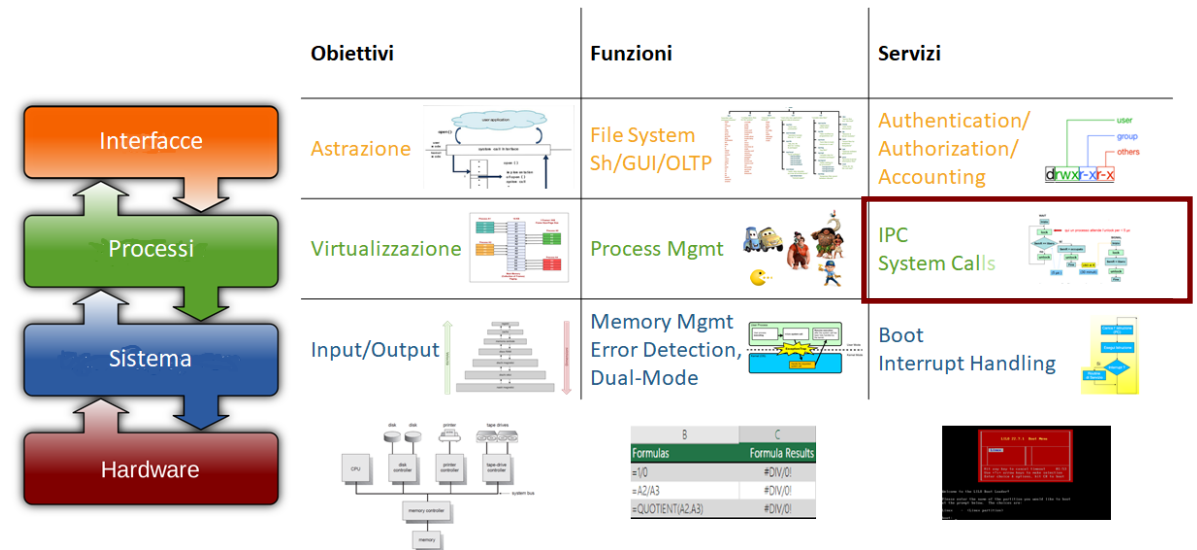
DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI



SAPIENZA
UNIVERSITÀ DI ROMA

Sistemi operativi (3 CFU)

- Il sistema operativo
- Concorrenza e sincronizzazione
- Deadlock
- Inter-process communication (IPC)
- Scheduling
- Memoria centrale e virtuale
- Memoria di massa e File system
- Sicurezza informatica



Lezioni: Settembre - Ottobre

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI



SAPIENZA
UNIVERSITÀ DI ROMA

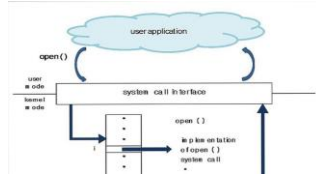

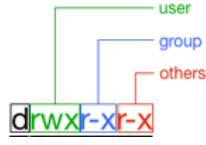
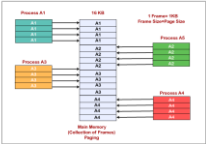

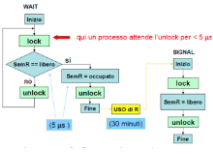
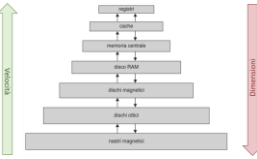
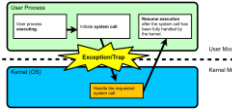
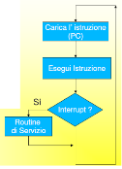
Organizzazione dei S.O.

Necessità di Comunicazione fra Processi

Operating Systems: Organizzazione

Tipologie di Sistemi Operativi

- Monolitico
- Stratificato
- Microkernel
- Modulari
- A macchine virtuali
- Client/Server


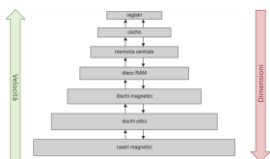

Obiettivi	Funzioni	Servizi
<p>Astrazione</p> 	<p>File System Sh/GUI/OLTP</p> 	<p>Authentication/ Authorization/ Accounting</p> 
<p>Virtualizzazione</p> 	<p>Process Mgmt</p> 	<p>IPC System Calls</p> 
<p>Input/Output</p> 	<p>Memory Mgmt Error Detection, Dual-Mode</p> 	<p>Boot Interrupt Handling</p> 

Operating Systems: Organizzazione

Monolitico

Generalmente:

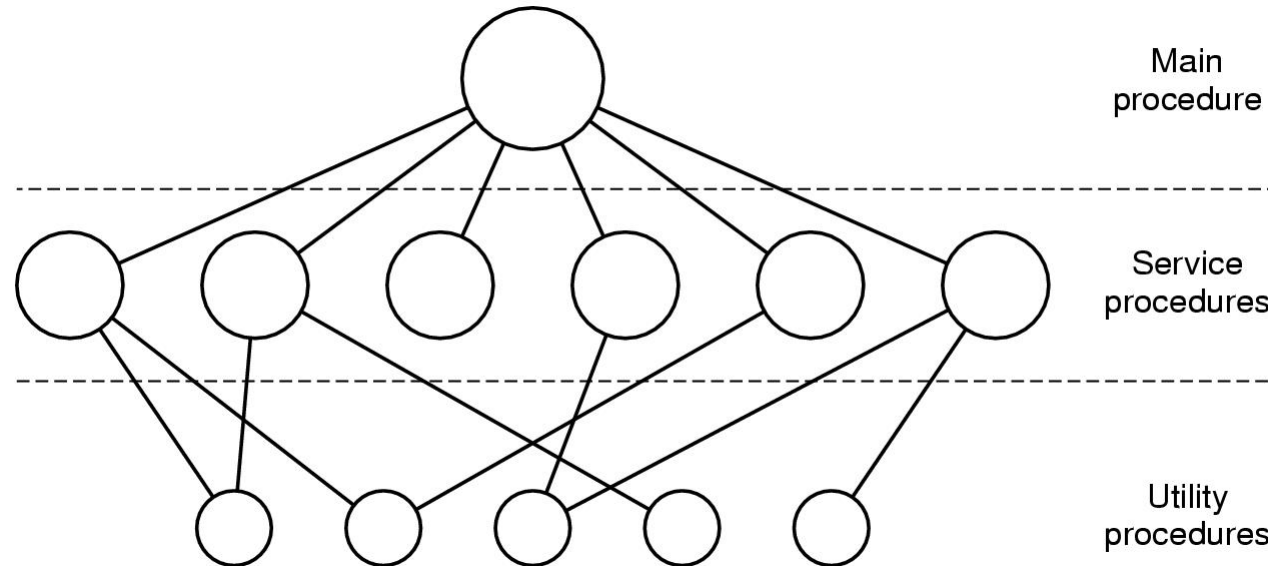
- **Interfaccia:** no Astrazione, no AA
- **Processi:** monoprocesso, no Virt, no IPS
- **Sistema:** no Dual-Mode, I/O etc incompleti

Obiettivi	Funzioni	Servizi
Astrazione	File System Sh/GUI/OLTP 	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt	IPC System Calls
Input/Output 	Dual-Mode	Boot Interrupt Handling 

Operating Systems: Organizzazione

Monolitico

Obiettivi	Funzioni	Servizi
Astrazione	File System Sh/GUI/DLTP	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt	IPC System Calls
Input/Output	Dual-Mode	Boot Interrupt Handling



SO che 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

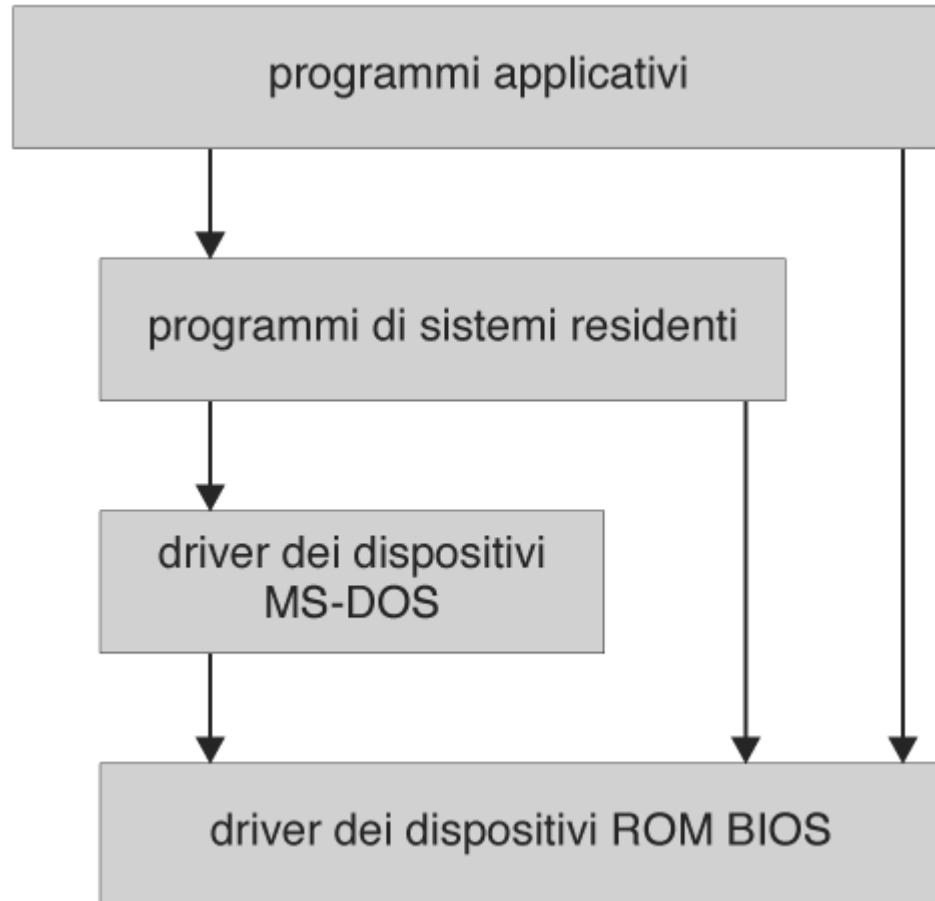
- Manutenzione difficile
- Vulnerabilità agli errori e agli attacchi
- Blocco del sistema
- Insufficiente protezione fornita dal SO

Operating Systems: Organizzazione

Monolitico

nati come sistemi piccoli, semplici e limitati, per poi crescere al di là dello scopo originario (es. MS-DOS, UNIX originario)

Obiettivi	Funzioni	Servizi
Astrazione	File System Sh/GUI/DLTP	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt	IPC System Calls
Input/Output	Dual-Mode	Boot Interrupt Handling



Es. MS-DOS:

- Dipendente dall'HW per cui sono stati pensati (Intel 8086, 8088: senza bit di stato)
- libertà ai programmi applicativi di accedere direttamente all'hardware
- No ASLR (Address Space Layout Randomization): spostamento casuale della posizione di moduli e strutture in memoria ad ogni esecuzione
- No DEP (Data Execution Prevention) impedisce determinati settori di memoria, ad es. lo stack, dall'essere eseguito

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI



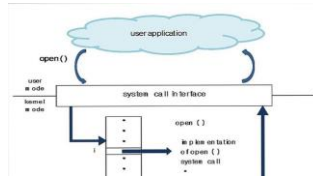
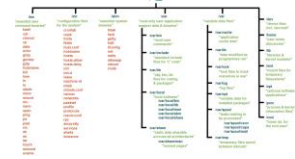

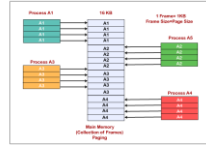

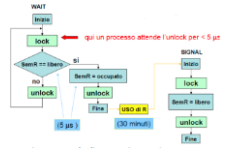
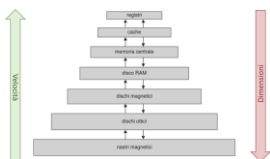
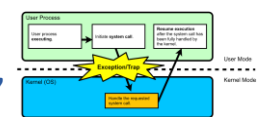
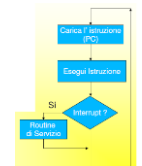
SAPIENZA
UNIVERSITÀ DI ROMA

Operating Systems: Organizzazione

Stratificato

Generalmente:

- **Interfaccia:** programmi di sistema
- **Processi:** gestione memoria, processi, comunicazione
- **Sistema:** Dual-Mode, I/O etc completi

Obiettivi	Funzioni	Servizi
Astrazione 	File System Sh/GUI/OLTP 	Authentication/ Authorization/ Accounting 
Virtualizzazione 	Process Mgmt 	IPC System Calls 
Input/Output 	Memory Mgmt Error Detection, Dual-Mode 	Boot Interrupt Handling 

Operating Systems: Organizzazione

Stratificato: UNIX

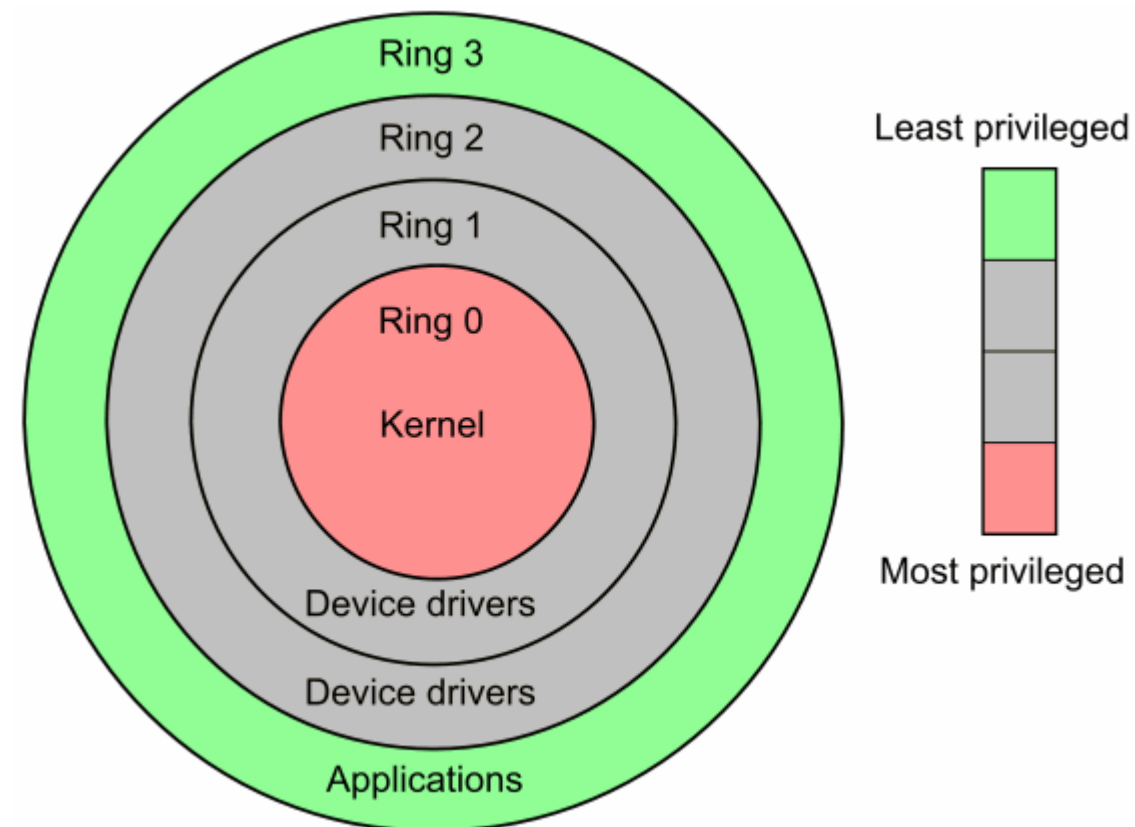
Obiettivi	Funzioni	Servizi
Astrazione	File System S/N/GU/O/LP	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt	IPC System Calls
Input/Output	Memory Mgmt Error Detection Dual-Mode	Boot Interrupt Handling

UNIX consiste di due parti separabili:

1. **Programmi di Sistema** (es. **libc**, **init**, **inet**, **daemon**, etc)
2. **Nucleo (kernel)**: 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..

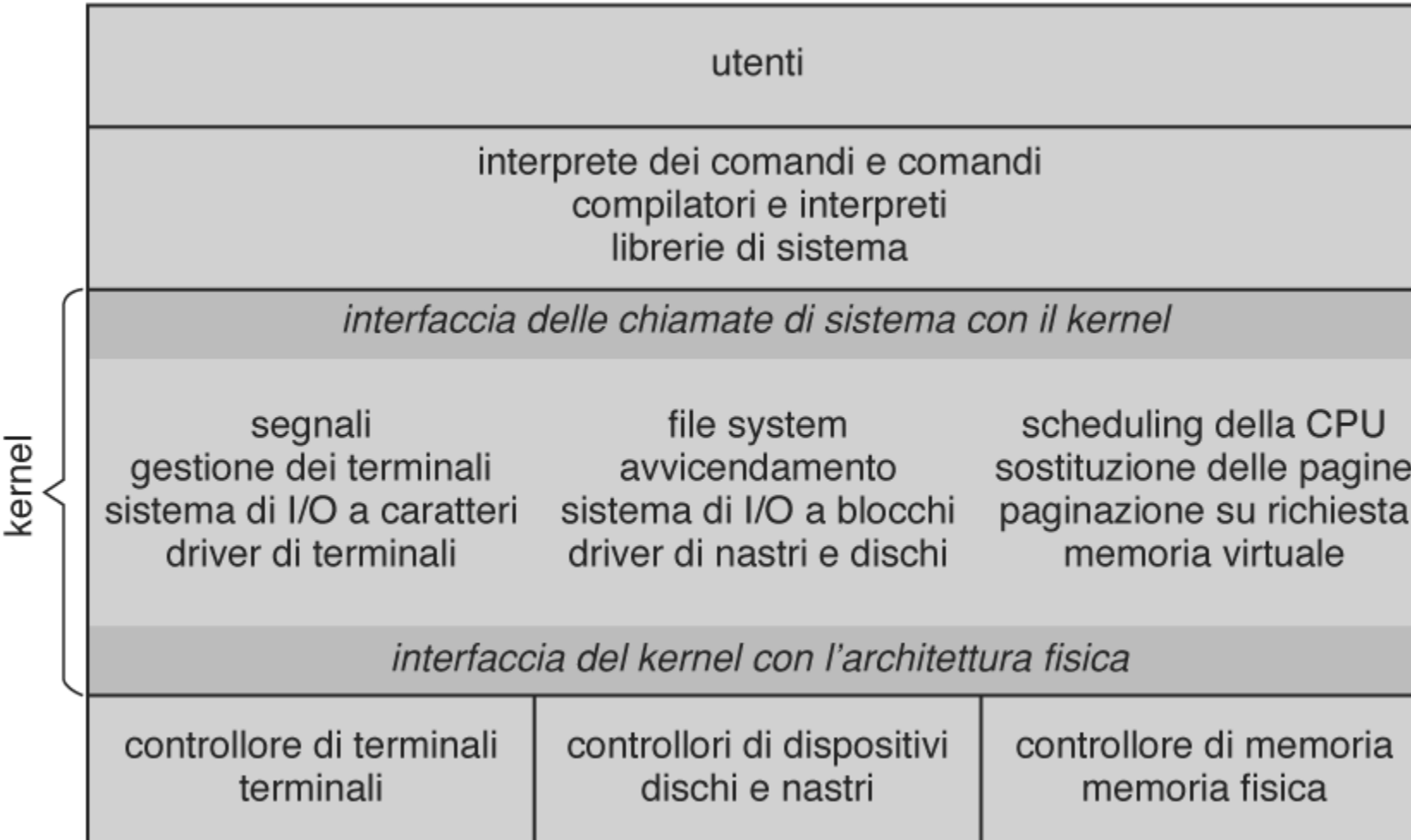
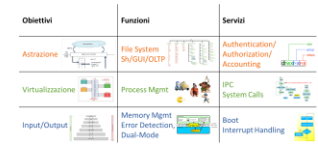
Separazione gerarchica o a livelli delle funzioni

- 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



Operating Systems: Organizzazione

Stratificato: UNIX



Vantaggi:

- Ogni strato offre una virtualizzazione di un certo numero di funzioni (macchina virtuale)
- La dipendenza dall'hardware è limitata al livello più basso (portabilità)

Svantaggi:



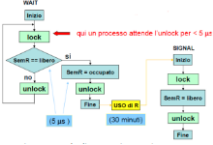
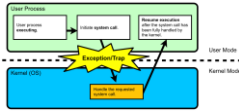
- La portabilità si paga con minor efficienza perché una chiamata di sistema deve attraversare più strati (eventuale adattamento dei dati passati)
- Più difficile da progettare

Operating Systems: Organizzazione

MicroKernel

Generalmente:

- **Interfaccia:** programmi di sistema (no kernel)
- **Processi:** gestione memoria, processi, comunicazione
- **Sistema:** Dual-Mode

Obiettivi	Funzioni	Servizi
Astrazione	File System Sh/GUI/OLTP 	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt 	IPC System Calls 
Input/Output	Memory Mgmt Error Detection, Dual-Mode 	Boot Interrupt Handling

Operating Systems: Organizzazione

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

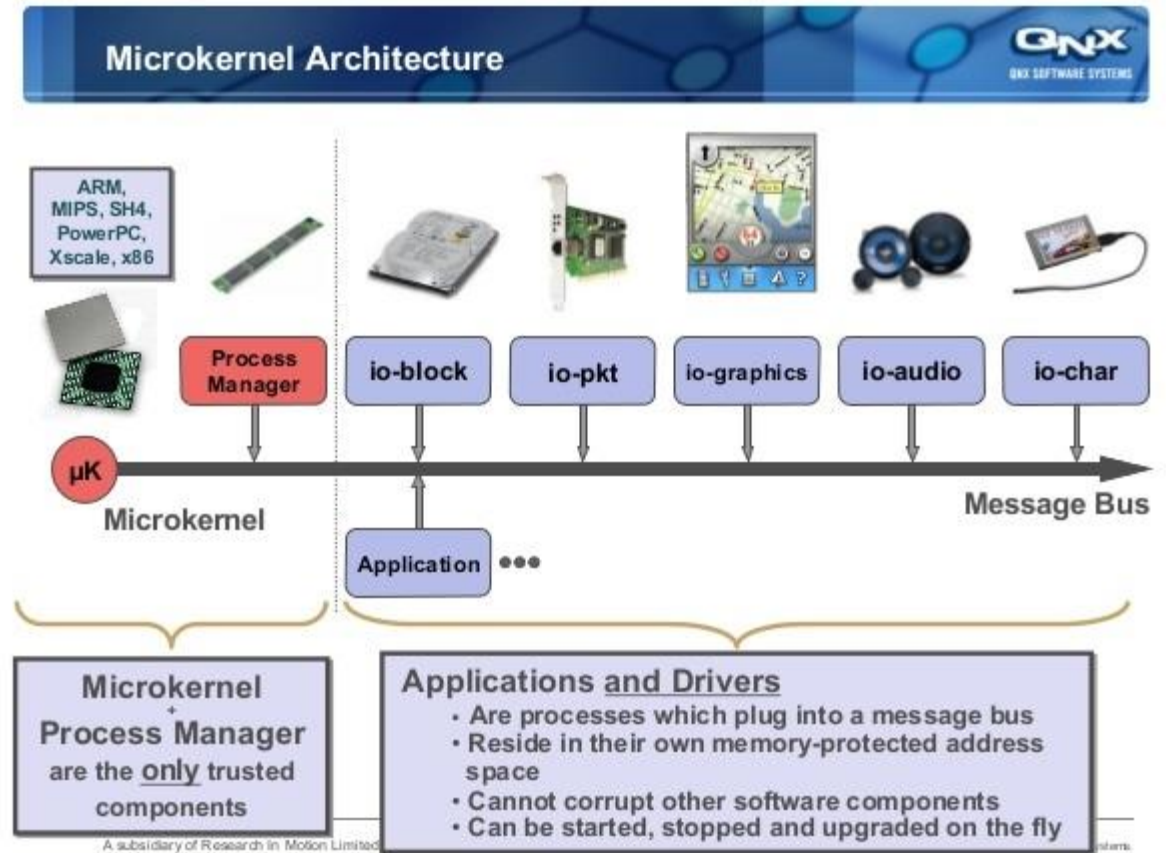
Vantaggi:

- Ottimale per i sistemi Real Time (RTOS)
- 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)

Svantaggi:

- I microkernel possono soffrire di calo di prestazioni per l'aumento di sovraccarico di funzioni di sistema

Obiettivi	Funzioni	Servizi
Astrazione	File System SV/GUI/DLTP	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt	IPC System Calls
Input/Output	Memory Mgmt Error Detection Dual-Mode	Host Interrupt Handling



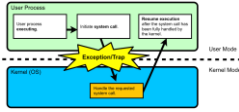



Operating Systems: Organizzazione

Modulare (Kernel a Moduli Funzionali)

Generalmente:

- **Interfaccia:** Funzioni e Servizi operati da moduli
- **Processi:** gestione interna al kernel
- **Sistema:** operati da moduli appositi

Obiettivi	Funzioni	Servizi
Astrazione	File System Sh/GUI/OLTP	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt 	IPC System Calls 
Input/Output	Memory Mgmt Error Detection, Dual-Mode 	Boot Interrupt Handling 

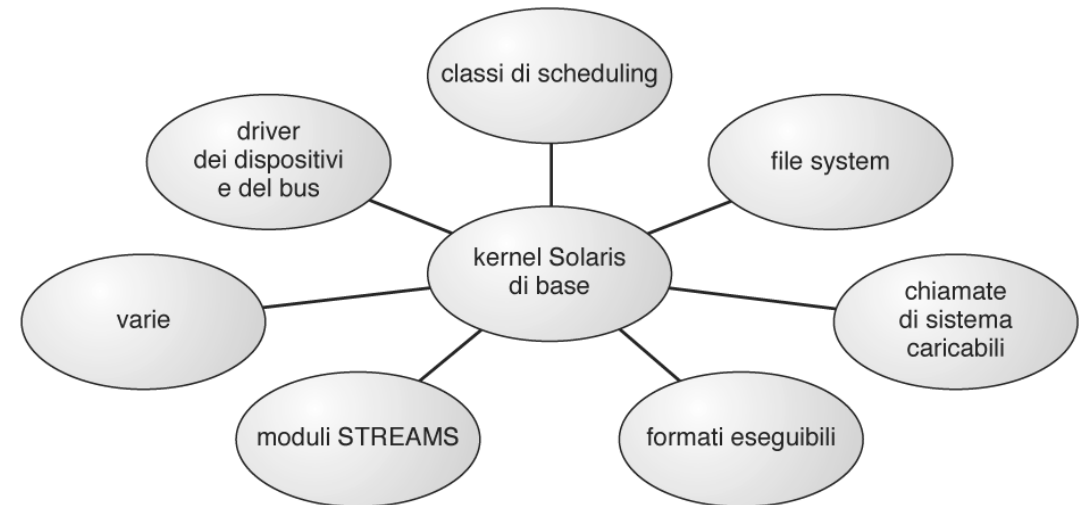
Operating Systems: Organizzazione

Modulare (Kernel a Moduli Funzionali)

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

Obiettivi	Funzioni	Servizi
Astrazione	File System Sh/GUI/OLTP	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt	IPC System Calls
Input/Output	Memory Mgmt Error Detection Dual-Mode	Boot Interrupt Handling




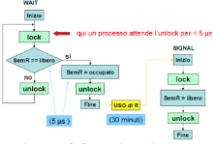
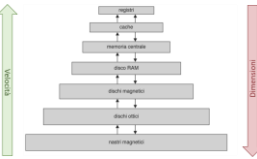
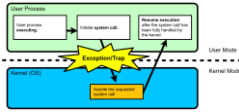

Stratificato: maggiormente flessibile
MicroKernel: più efficiente (no messaggi)

Operating Systems: Organizzazione

Macchine Virtuali






Generalmente:

- **Interfaccia:** minimale
- **Processi:** gestione interna al kernel
- **Sistema:** operati da moduli appositi

Obiettivi	Funzioni	Servizi
Astrazione	File System Sh/GUI/OLTP	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt 	IPC System Calls 
Input/Output 	Memory Mgmt Error Detection, Dual-Mode 	Boot Interrupt Handling 

Operating Systems: Organizzazione

Macchine Virtuali

Obiettivi	Funzioni	Servizi
Astrazione	File System Sh/GUI/OLTP	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt 	IPC System Calls 
Input/Output 	Memory Mgmt Error Detection, Dual-Mode 	Boot Interrupt Handling 

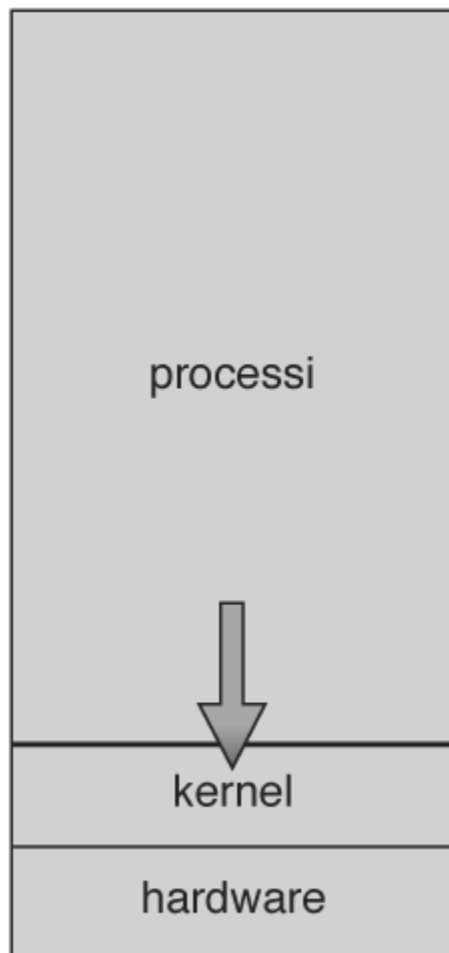
L'approccio a strati trova il suo logico sbocco nel concetto di macchina virtuale

- 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

Operating Systems: Organizzazione

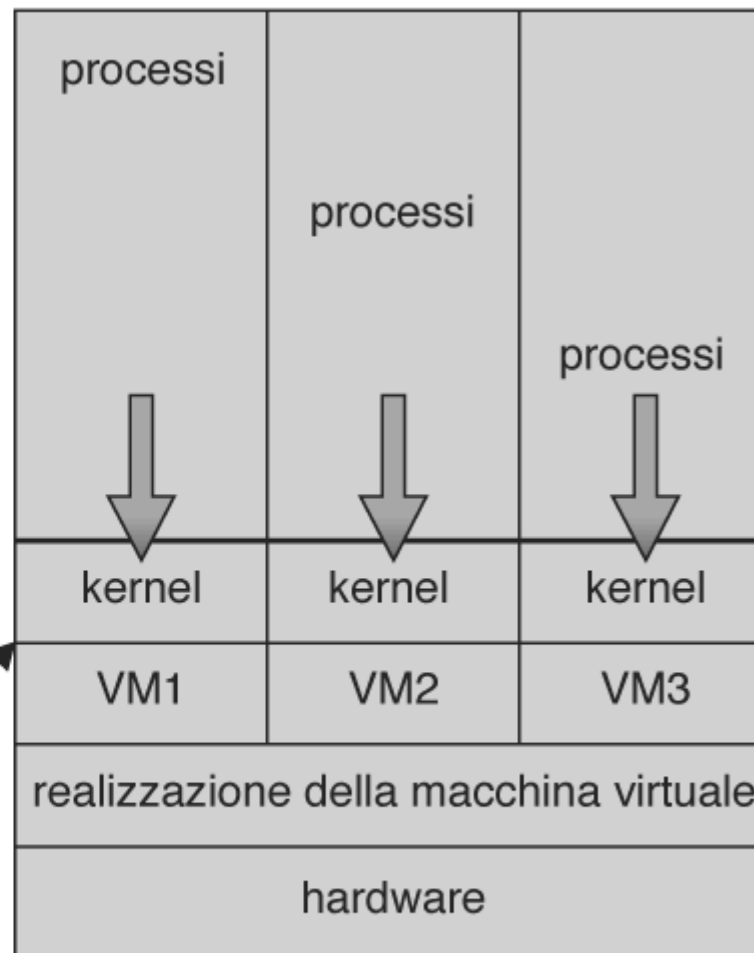
Macchine Virtuali

Non-virtual Machine



(a)

Virtual Machine Virtual Machine Virtual Machine



(b)

Obiettivi	Funzioni	Servizi
Astrazione	File System Sh/GUI/OLTP	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt	IPC System Calls
Input/Output	Memory Mgmt Error Detection, Dual-Mode	Boot Interrupt Handling

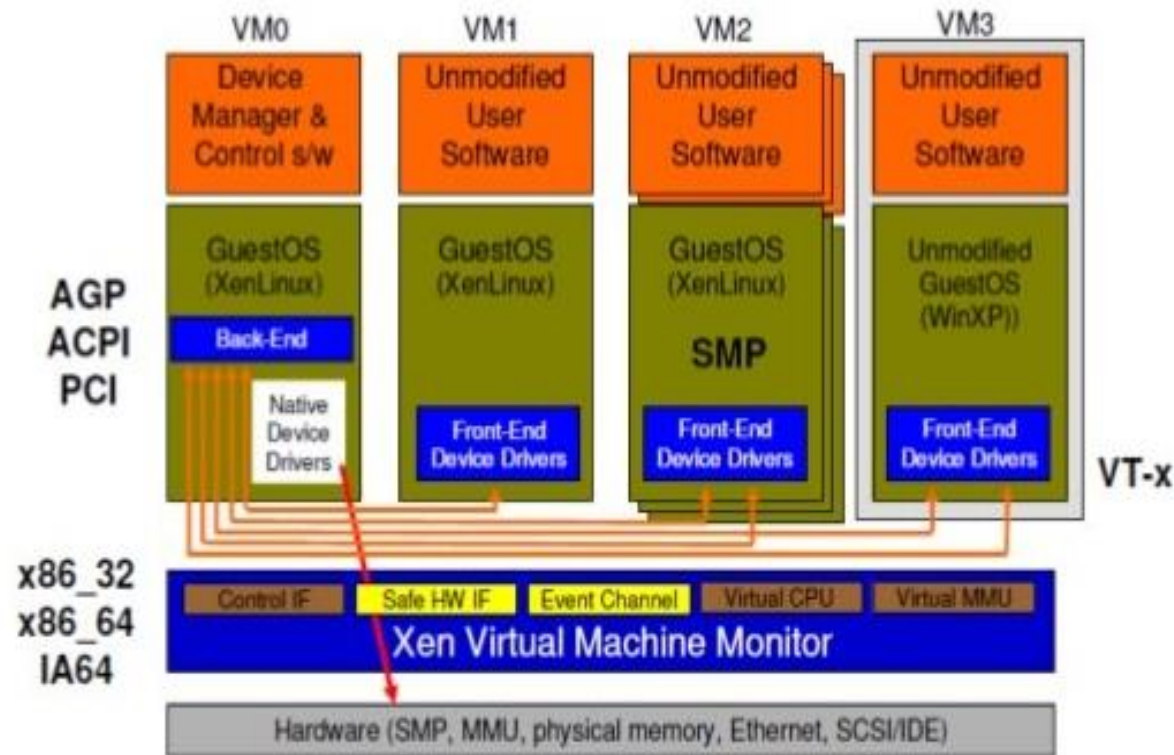
Il computer fisico mette in condivisione le proprie risorse per generare macchine virtuali

- 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

Operating Systems: Organizzazione

Macchine Virtuali: Vmware, Xen, Hyper-V, KVM, etc

Obiettivi	Funzioni	Servizi
Astrazione	File System Sh/GUI/OLTP	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt	IPC System Calls
Input/Output	Memory Mgmt Error Detection, Dual-Mode	Boot Interrupt Handling



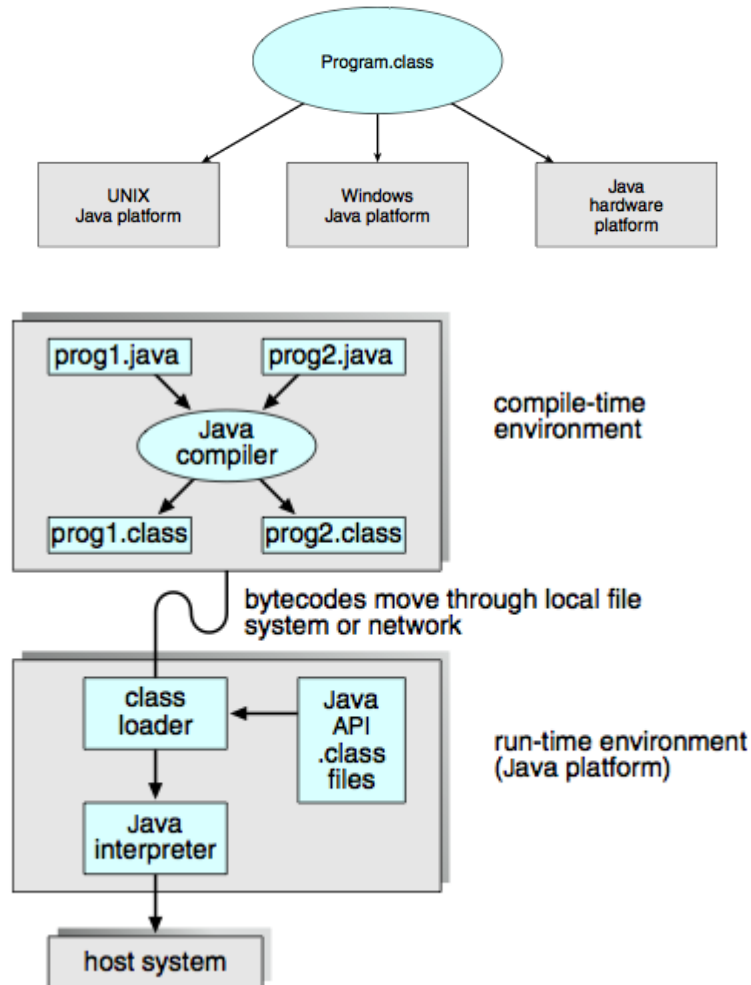
Lo strato di virtualizzazione è scritto per funzionare in modalità utente

- Trasforma HW in macchine virtuali che ospitano SO
- Il disco virtuale è in realtà un file (Si possono così creare snapshot)

Operating Systems: Organizzazione

Macchine Virtuali: Java Virtual Machine

Obiettivi	Funzioni	Servizi
Astrazione	File System Sh/GUI/OLTP	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt	IPC System Calls
Input/Output	Memory Mgmt Error Detection, Dual-Mode	Boot Interrupt Handling



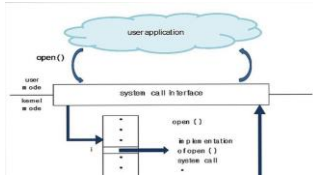

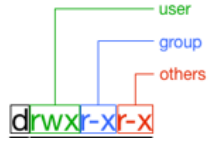
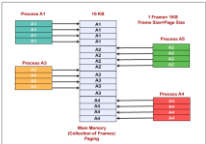


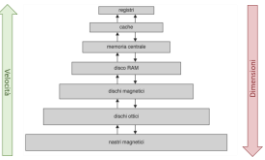
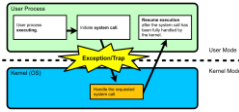

- I programmi Java compilati sono bytecode indipendenti dall'architettura ed eseguiti da una Java Virtual Machine (JVM)
- La JVM è un calcolatore astratto su un sistema ospitante vero
- La JVM utilizza un interprete puro, senza influenzare troppo negativamente le prestazioni, in quanto utilizza un Just In Time compiler (JIT)
- porzioni di codice eseguite numerose volte vengono compilate per aumentare le prestazioni
- Diversi HotSpot (client e server). La JVM ufficiale è quella mantenuta da Oracle

Operating Systems: Organizzazione

Client/Server (Distribuito)

Generalmente:

- **Interfaccia:** transattiva tra processi
- **Processi:** client e server
- **Sistema:** operati da moduli appositi

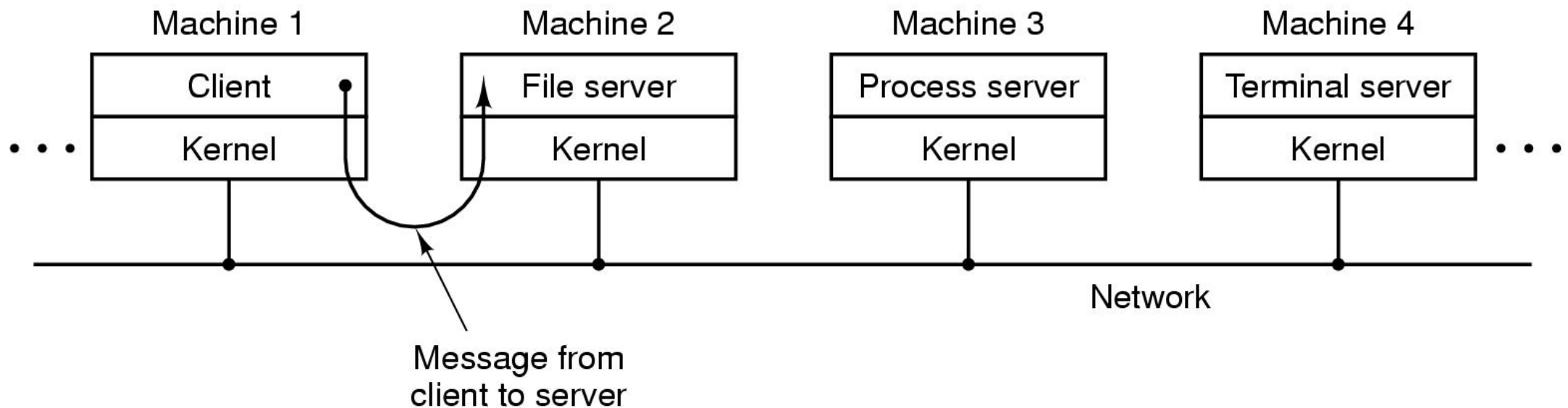
Obiettivi	Funzioni	Servizi
Astrazione 	File System Sh/GUI/OLTP 	Authentication/Authorization/Accounting 
Virtualizzazione 	Process Mgmt 	IPC System Calls 
Input/Output 	Memory Mgmt Error Detection, Dual-Mode 	Boot Interrupt Handling 

Operating Systems: Organizzazione

Client/Server (Distribuito)










Obiettivi	Funzioni	Servizi
Astrazione	File System Sh/GUI/OLTP	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt	IPC System Calls
Input/Output	Memory Mgmt Error Detection, Dual-Mode	Boot Interrupt Handling

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

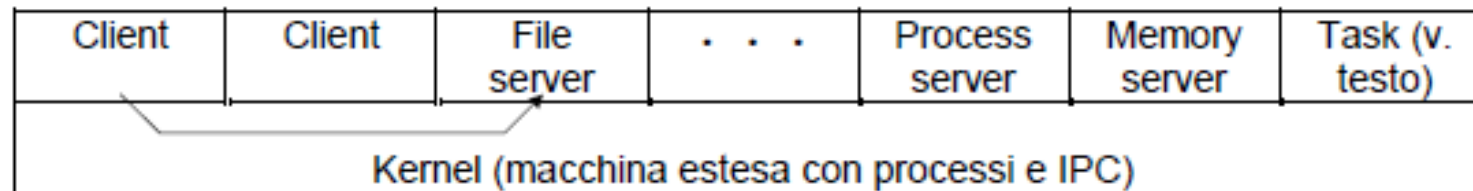


Operating Systems: Organizzazione

Client/Server (Distribuito)

Obiettivi	Funzioni	Servizi
Astrazione 	File System Sh/GUI/OLTP 	Authentication/ Authorization/ Accounting 
Virtualizzazione 	Process Mgmt 	IPC System Calls 
Input/Output 	Memory Mgmt Error Detection, Dual-Mode 	Boot Interrupt Handling 

- 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



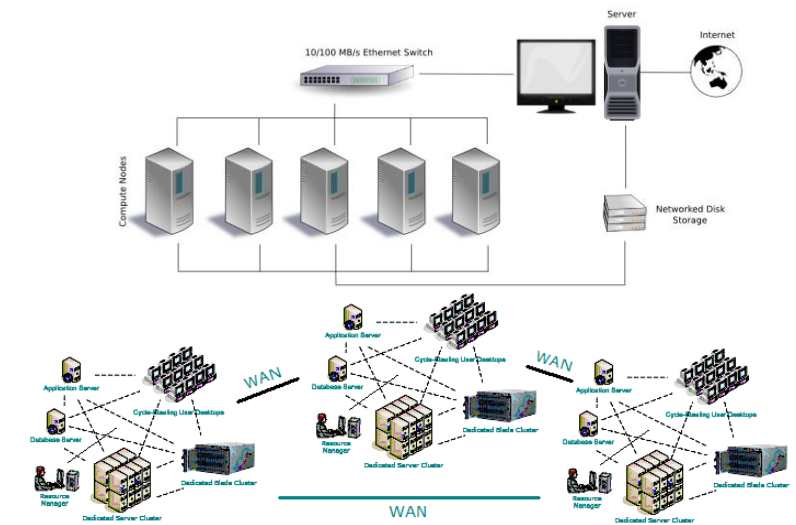
Operating Systems: Organizzazione

Client/Server (Distribuito): es. Clusters

In generale, un cluster si riferisce a un insieme di computer collegati tra loro.

- **High Performance Computer (HPC):** fisicamente situati uno vicino all'altro, al fine di risolvere i problemi in modo più efficiente. Generalmente, eseguono la stessa immagine di Sistema Operativo.
- **Grid Computing:** uso di una griglia computazionale (stazioni di lavoro, server blade, ecc.) applicando le risorse della griglia, tramite rete, a un singolo problema allo stesso tempo, mentre si superano i confini politici e teorici.
- **High Availability (HA):** un sistema informatico funge da sistema di backup per uno o più sistemi primari, tutti situati uno vicino all'altro. Quando c'è un guasto in un sistema primario, le applicazioni critiche in esecuzione su quel sistema vengono trasferite al sistema di backup designato.
 - **Load Balancing (LB):** come HA ma tutti i sistemi funzionanti ed aderenti al cluster dividono il carico di lavoro (Attivo/Attivo).
 - **Cluster Geografico:** come HA ma i sistemi sono situati a distanza.
- **Three Tier:** architetture si sistema a 3 livelli: presentazione (web Server), elaborazione (App), dati (DB). Ognuno su server diversi, in HA/LB e SO distinti

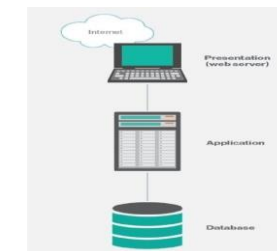
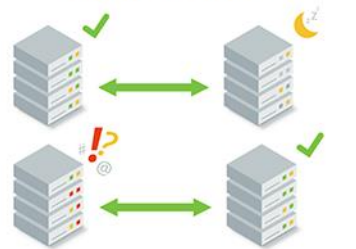
Obiettivi	Funzioni	Servizi
Astrazione	File System Shell/CLI/FTP	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt	IPC System Calls
Input/Output	Memory Mgmt Error Detection Fault-Mode	Mail Internet Handling



Active / Active Design



Active / Passive



DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

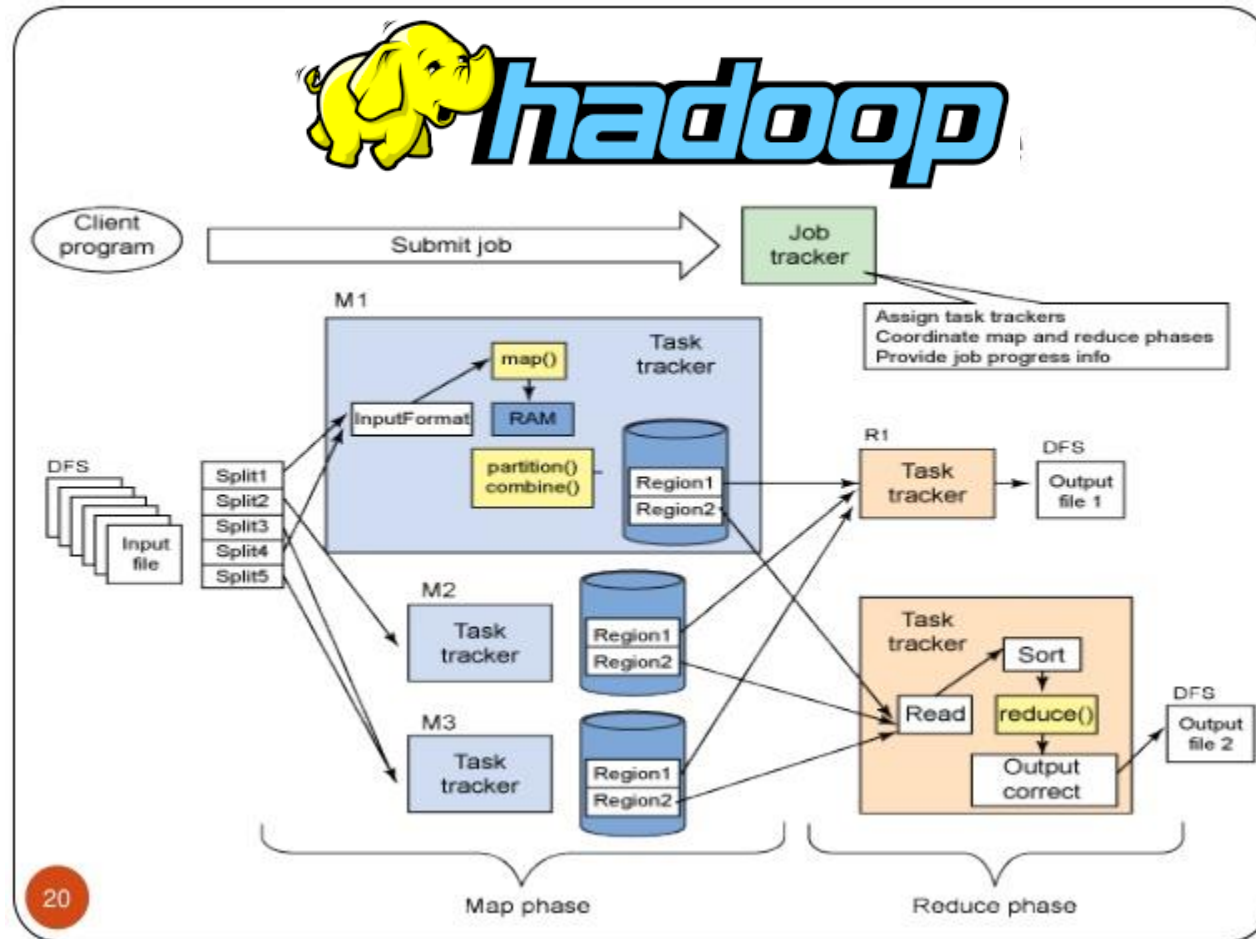


SAPIENZA
UNIVERSITÀ DI ROMA

Operating Systems: Organizzazione

Client/Server (Distribuito): es. Hadoop

Obiettivi	Funzioni	Servizi
Astrazione	File System sh/GUI/OLTP	Authentication/ Authorization/ Accounting
Virtualizzazione	Process Mgmt	IPC System Calls
Input/Output	Memory Mgmt Error Detection Dual Mode	Host Interrupt Handling



- **Interfaccia:** HDFS (Hadoop Distributed File System)
- **Processi:** Map/Reduce (capacità di dividere ed elaborare terabyte di dati in parallelo, ottenendo risultati più rapidi.)
- **Sistema:** si basa sui SO sottostanti

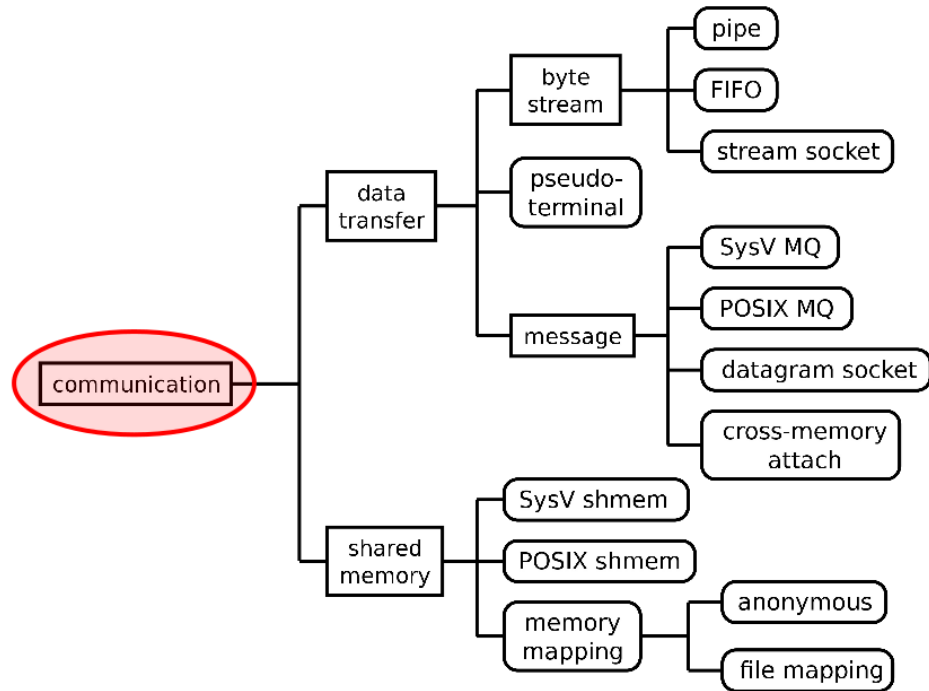
IPC

Comunicazione fra Processi

Operating Systems: Obiettivi Funzioni Servizi

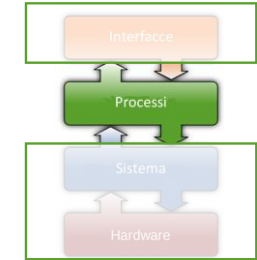
IPC

Communication



Inter-Process Communication

- **Segnale** (signals), per eventi asincroni
- **Pipe**, per reindirizzare il risultato della elaborazione
- **Socket**, scambio di datagram
- **Code di Messaggi** (Message Queue), per comunicazioni in multicast
- **Memoria Condivisa** (Shared Memory), per condividere dati fra processi con trust reciproco
- **Semaforo** (Semaphore), per coordinare processi che lavorano su una stessa risorsa (-> critical region)



Permettere ai processi di comunicare fra di loro.

Un **processo indipendente** non può influenzare o essere influenzato dagli altri processi in esecuzione nel sistema

→ non condivide risorse con altri processi

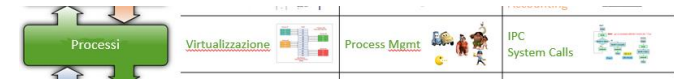
Un **processo cooperante** può influenzare o essere influenzato da altri processi in esecuzione nel sistema

→ condivide risorse con altri processi

(es. Client-Server, Compiler-Assembler-Loader, Produttore-Consumatore)

Operating Systems: IPC

Cooperazione



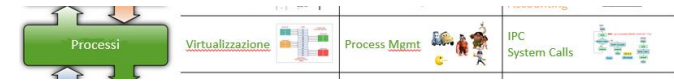
Cooperazione = lavoro congiunto di processi per raggiungere scopi applicativi comuni con condivisione e scambio di informazioni

Vantaggi del processo di cooperazione:

- **Condivisione delle informazioni** (ad es. un file o una directory condivisa) → Accesso concorrente (!!!)
- **Velocizzazione della computazione** (possibile in verità solo con più CPU o canali di I/O) → Esecuzione di sotto-attività in parallelo
- **Modularità**: dividendo le funzioni del sistema in processi o thread separati (es. sistemi operativi modulari)
- **Convenienza** (un singolo utente può lavorare su molte attività) → Multi tasking

Operating Systems: IPC

Processi Cooperanti



- Hanno uno scopo applicativo comune
- Possono condividere informazioni → **Comunicazione**
- Possono influenzare o essere influenzati da altri processi → **Sincronizzazione**

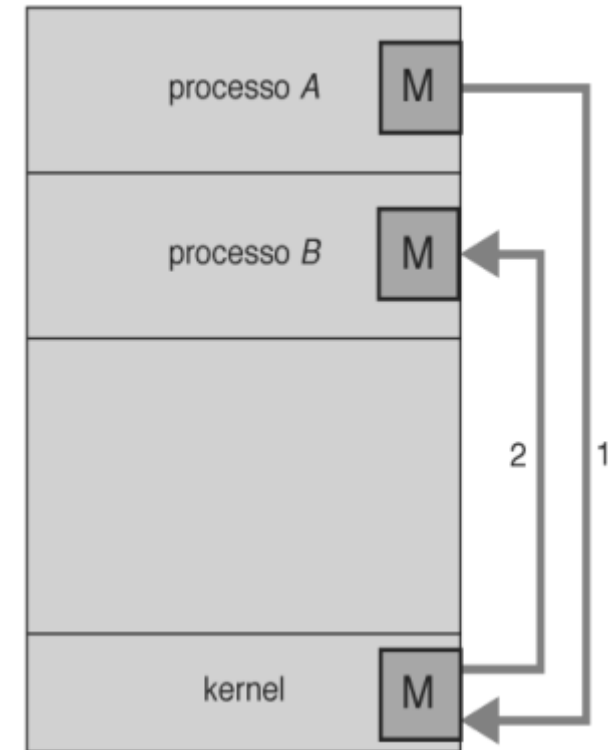
- Quantità di informazioni da trasmettere
- Velocità di esecuzione
- Scalabilità
- Semplicità di uso nelle applicazioni
- Omogeneità delle comunicazioni
- Integrazione nel linguaggio di programmazione
- Affidabilità
- Sicurezza
- Protezione

Operating Systems: IPC

IPC: Implementazioni a Scambio di Messaggi

- Quantità di informazioni da trasmettere
- Velocità di esecuzione
- Scalabilità
- Semplicità di uso nelle applicazioni
- Omogeneità delle comunicazioni
- Integrazione nel linguaggio di programmazione
- Affidabilità
- Sicurezza
- Protezione

(es. Segnale, Socket, Message Queue, Semaforo)

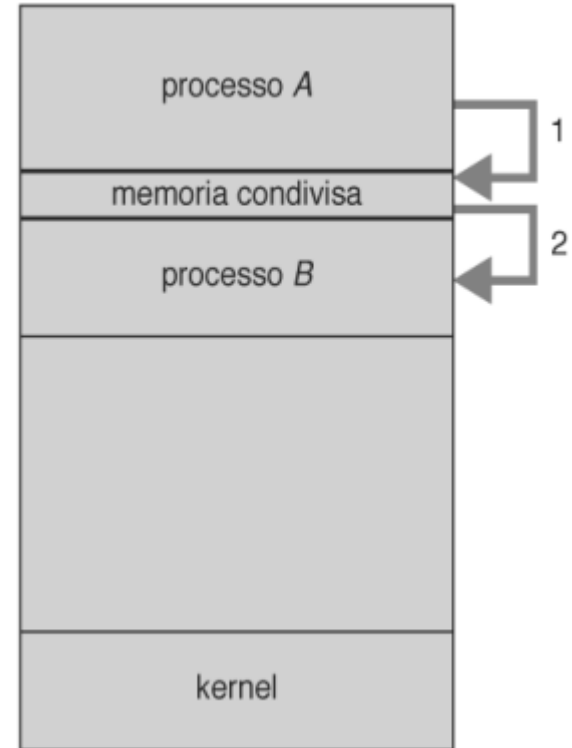


Operating Systems: IPC

IPC: Implementazioni a Memoria Condivisa

- Quantità di informazioni da trasmettere
- Velocità di esecuzione
- Scalabilità
- Semplicità di uso nelle applicazioni
- Omogeneità delle comunicazioni
- Integrazione nel linguaggio di programmazione
- Affidabilità
- Sicurezza
- Protezione

(es. Pipe, Memoria Condivisa)



Operating Systems: IPC

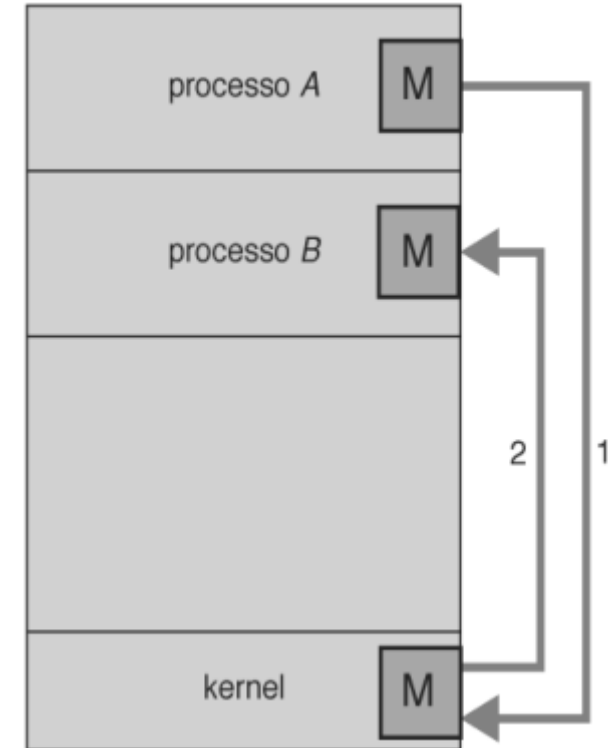
IPC: Implementazioni a Scambio di Messaggi

Segnale

Socket

Message Queue

Semaforo



Nei sistemi UNIX-like per notificare ad un processo che si è verificato un particolare evento si usa un segnale

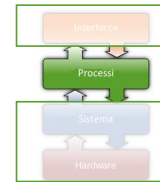
Operating Systems: Obiettivi Funzioni Servizi

IPC: Signal in Unix

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from <code>abort(3)</code>
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from <code>alarm(2)</code>
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at terminal
SIGTTIN	21,21,26	Stop	Terminal input for background process
SIGTTOU	22,22,27	Stop	Terminal output for background process

The signals **SIGKILL** and **SIGSTOP** cannot be caught, blocked, or ignored.

Permettere ai processi di comunicare fra di loro.



- **Sincroni**: sono inviati allo stesso processo che ha causato la generazione del segnale (es. divisione per 0, accesso illegale alla memoria)
- **Asincroni**: sono inviati ad un processo differente da quella che ha causato la generazione del segnale (es. intercettazione di una combinazione di tasti (ctrl + c, ctrl+alt+canc, scadenza di un timer, etc))

Gestione dei Segnali con i Thread

I segnali vengono elaborati secondo questo schema:

1. il verificarsi di un particolare evento genera un segnale
2. il segnale generato viene consegnato ad un processo
3. il segnale viene gestito

I segnali possono essere gestiti attraverso

- Il gestore predefinito dello specifico segnale (esiste per ogni segnale)
- Un funzione di gestione definita dall'utente (override del gestore predefinito)

Per i processi a singolo thread la gestione dei segnali è semplice

Per i processi multithread è necessario decidere a quale thread inviare il processo. Diverse opzioni:

1. **UNO**: Consegnare il segnale al thread a cui il segnale viene applicato (sempre per segnali **asincroni**)
2. **TUTTI**: Consegnare il segnale ad ogni thread del processo (istruire i processi su quali segnali **sincroni** accettare e quali ignorare)
3. **MOLTI**: Consegnare il segnale al alcuni thread del processo (istruire i processi su quali segnali **sincroni** accettare e quali ignorare)
4. **Specifico**: Designare un thread specifico che riceva tutti i segnali per il processo

Operating Systems: IPC

Gestione dei Socket

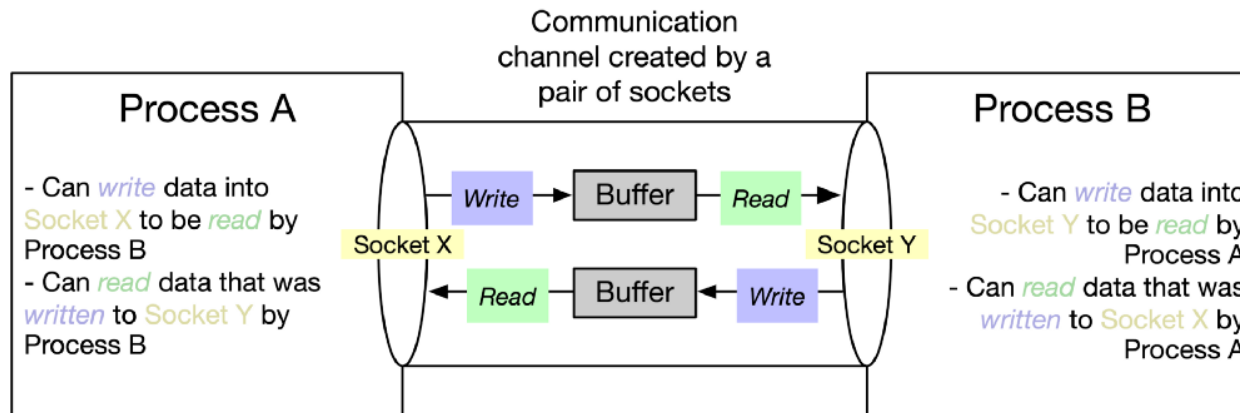
Socket = estremo di un canale di comunicazione

Operating Systems: Obiettivi Funzioni Servizi

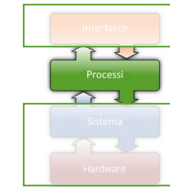
IPC: Socket (stream/datagram) in Unix

One-way communication

Flusso (stream) o blocchi di dati (Datagram) Unidirezionale



Permettere a 2 processi di scambiarsi informazioni (anche di notevoli dimensioni) fra loro.



Tutte le connessioni devono essere uniche: se un processo vuole n connessioni verso un server deve creare n socket

Vantaggi:

- Semplice
- Efficiente

Svantaggi:

- Di basso livello: permette la trasmissione di un flusso non strutturato di byte
- È responsabilità di Client e Server interpretare e organizzare i dati in forme complesse

Operating Systems: IPC

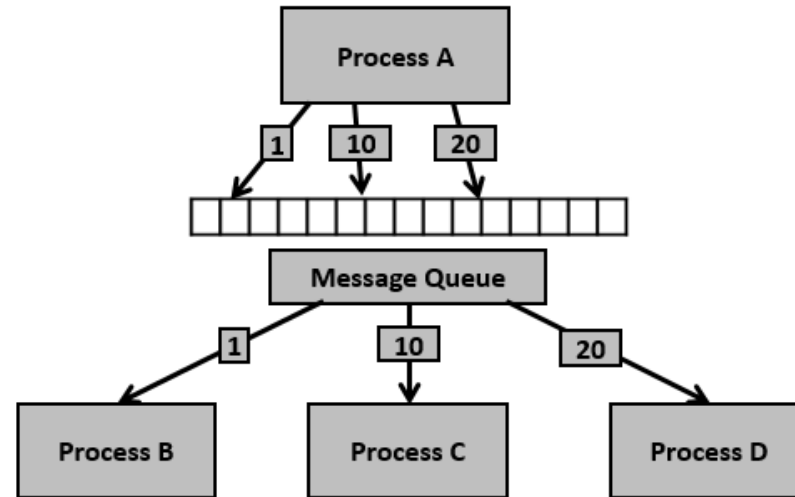
Gestione delle Code di Messaggi

Il SO implementa codice per la realizzazione e la gestione della risorsa condivisa

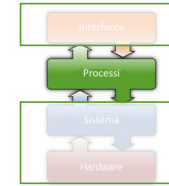
Operating Systems: Obiettivi Funzioni Servizi

IPC: Message Queue in Unix

Multicast communication



Permettere a molteplici processi di scambiarsi informazioni non riservate.

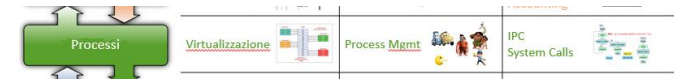


Sincronizzazione per l'accesso ai messaggi gestita implicitamente dal SO fornendo due operazioni:

- send (messaggio)
- receive (messaggio)

Operating Systems: IPC

Gestione delle Code di Messaggi



Contenuto messaggio

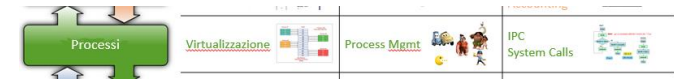
- Processo mittente
- Processo destinatario
- Informazioni da trasmettere
- Eventuali altre informazioni di gestione dello scambio messaggi

Dimensione messaggio

- Fissa
- Facile implementazione a livello SO, difficile a livello applicazione
- Variabile
- Difficile implementazione a livello SO, facile a livello applicazione

Operating Systems: IPC

Gestione delle Code di Messaggi: Canali



Due processi che vogliono comunicare, devono:

- stabilire un canale di comunicazione tra di loro
- scambiare messaggi mediante send/receive

Implementazione di un canale di comunicazione:

- Fisica (come memoria condivisa, hardware bus) o
- Logica (come le proprietà logiche)

1. La denominazione dei processi

- comunicazione diretta (simmetrica o asimmetrica)
- comunicazione indiretta

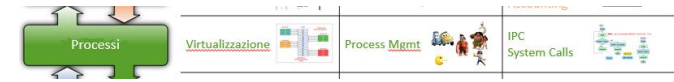
2. La sincronizzazione: Il passaggio dei msg può essere bloccante oppure non bloccante (ovvero **sincrono** oppure **asincrono**)

3. La bufferizzazione: i messaggi scambiati risiedono in una coda temporanea

- I processi devono conoscere **esplicitamente** il nome del destinatario o del mittente:
 - **send (P, msg)** – manda un messaggio al processo P
 - **receive (Q, msg)** – riceve un messaggio dal processo Q
- Proprietà di un canale di comunicazione:
 - Le connessioni sono stabilite automaticamente
 - Una connessione è associata esattamente a due processi (connessione binaria)
 - Fra ogni coppia di processi esiste esattamente una connessione
 - La connessione può essere unidirezionale, ma di norma è bidirezionale

Operating Systems: IPC

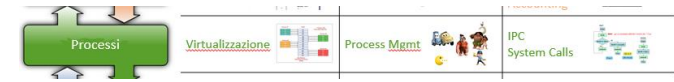
Gestione delle Code di Messaggi: Comunicazione Asimmetrica



- Solo il processo che invia il messaggio deve conoscere **esplicitamente** il nome del destinatario o del mittente:
 - **send (P, msg)** – manda un messaggio al processo P
 - **receive (id, msg)** – riceve un messaggio da un processo il cui identificatore è salvato in id

Operating Systems: IPC

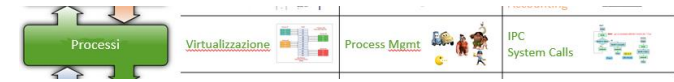
Gestione delle Code di Messaggi: Comunicazione Indiretta



- I messaggi sono mandati e ricevuti attraverso una **mailbox** o **porte**
 - Ciascuna mailbox ha un identificatore univoco
 - I processi possono comunicare solo se hanno una mailbox condivisa
- Le primitive sono definite come
 - **send** (M, msg) – manda un messaggio alla mailbox M
 - **receive** (M, msg) – riceve un messaggio dalla mailbox M
- Viene stabilita una connessione fra due processi solo se entrambi hanno una mailbox condivisa
- Una connessione può essere associata a più di due processi (non binaria)
- Fra ogni coppia di processi comunicanti possono esserci più connessioni
- La connessione può essere unidirezionale o bidirezionale

Operating Systems: IPC

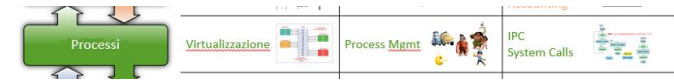
Gestione delle Code di Messaggi: Comunicazione Indiretta



- Una mailbox può appartenere ad un processo o al SO
- In ogni caso esiste sempre un processo che ha il diritto di proprietà
- Se appartiene al processo allora risiede nel suo spazio di indirizzi
 - Viene deallocata alla terminazione del processo
 - Il proprietario è l'unico che può ricevere
 - Tutti gli altri sono **utenti** che inviano messaggi

Operating Systems: IPC

Gestione delle Code di Messaggi: Comunicazione Indiretta



- Per le mailbox appartenenti al SO, il SO stesso mette a disposizione delle chiamate di sistema per:
 - Creare mailbox
 - Cancellare mailbox
 - Inviare e ricevere messaggi
 - Cedere o concedere il diritto di proprietà sulla mailbox
- Il processo creatore ha il **diritto di proprietà**
 - Inizialmente è l'unico che può ricevere
- La concessione del diritto di proprietà può portare ad avere più riceventi

- Il passaggio di messaggi può essere bloccante (sincrono) oppure non bloccante (asincrono)

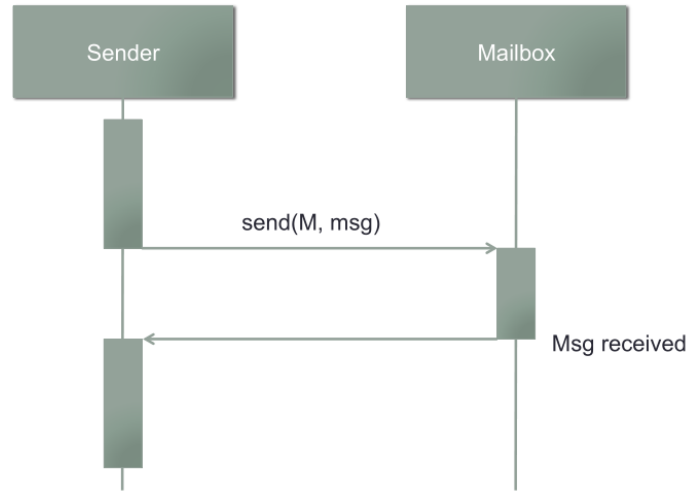
RENDEZVOUS

- **Invio bloccante:** il processo che invia viene bloccato finché il messaggio viene ricevuto dal processo che riceve o dalla mailbox
- **Ricezione bloccante:** il ricevente si blocca sin quando un messaggio non è disponibile
- **Invio non bloccante:** il processo che invia manda il messaggio e riprende l'attività
- **Ricezione non bloccante:** il ricevente acquisisce un messaggio o valido o nullo

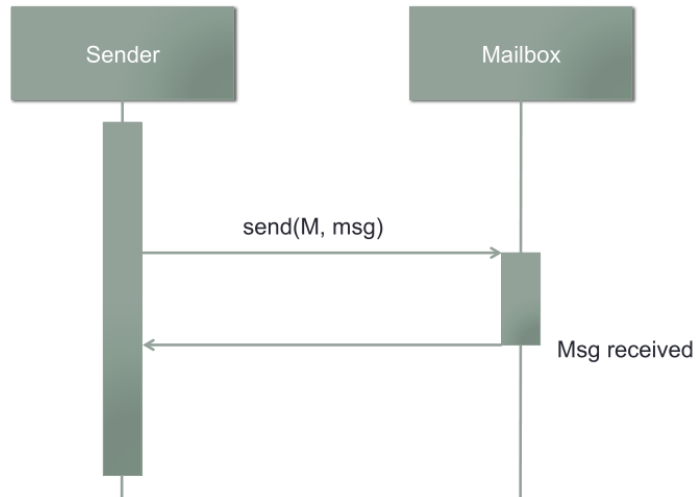
Operating Systems: IPC

Gestione delle Code di Messaggi: Sincronizzazione - Invio

Invio Bloccante



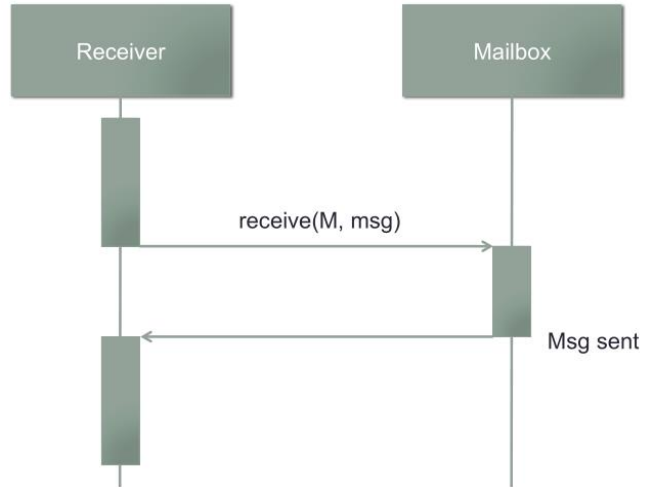
Invio non-Bloccante



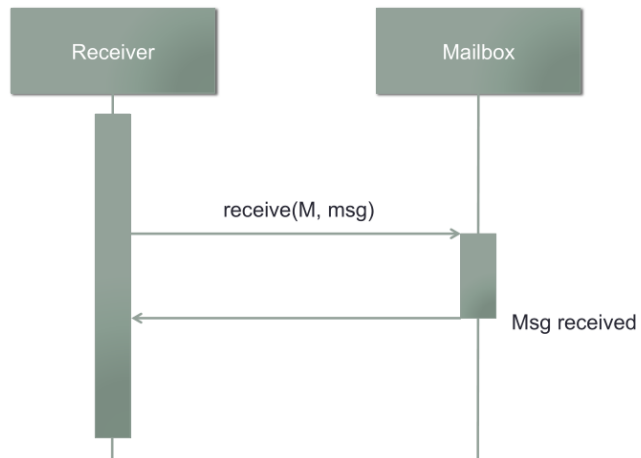
Operating Systems: IPC

Gestione delle Code di Messaggi: Sincronizzazione - Ricezione

Ricezione Bloccante



Ricezione non-Bloccante



Operating Systems: IPC

Gestione delle Code di Messaggi: Buffer

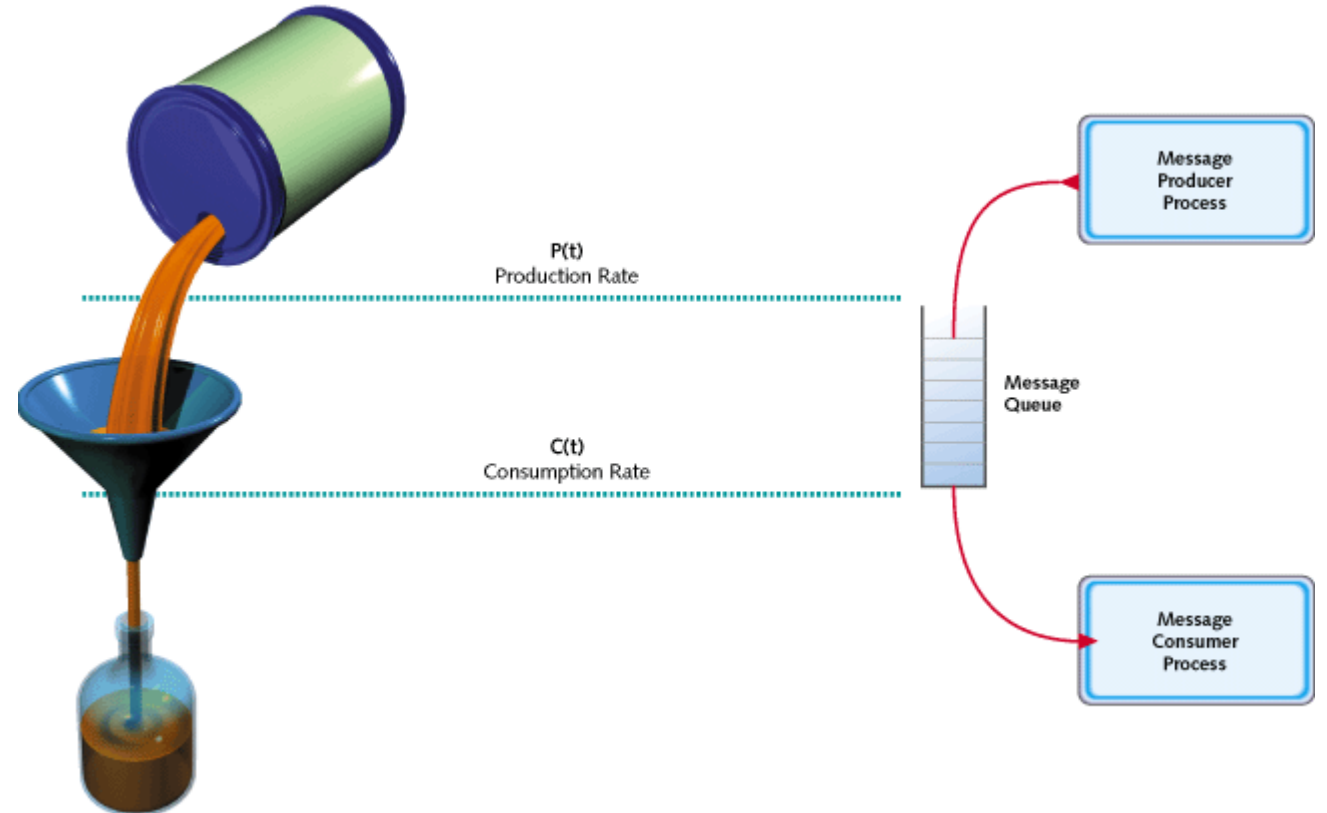
I messaggi scambiati tra processi risiedono in code temporanee (sia comunicazione diretta che indiretta)

- Tre modi per implementare le code:

1. **Capacità zero (no Buffer)** – Il mittente deve bloccarsi finché il destinatario riceve il messaggio (rendezvous)

2. **Capacità limitata (Buffer)** – lunghezza finita di n messaggi. Il mittente deve bloccarsi se la coda è piena

3. **Capacità illimitata (Buffer)** – lunghezza infinita. Il mittente non si blocca mai



Per l'ordinamento delle code dei messaggi nella mailbox sono usate le stesse politiche per la gestione dei processi in attesa

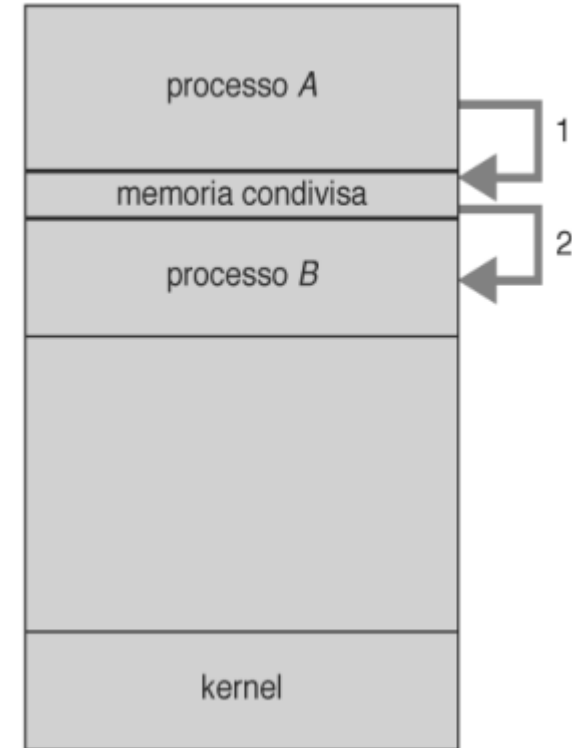
- First In, First Out (FIFO)
- Priorità
- Scadenza

Operating Systems: IPC

IPC: Implementazioni a Memoria Condivisa

Pipe

Memoria Condivisa



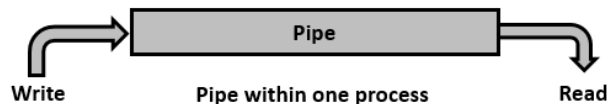
Operating Systems: IPC

Gestione delle Pipe

Operating Systems: Obiettivi Funzioni Servizi

IPC: Pipe (datagram FIFO) in Unix

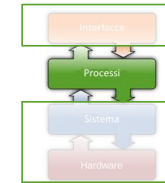
Pipe is one-way communication
Flusso (stream) Unidirezionale



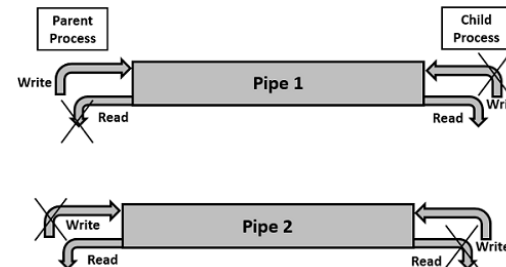
```
rishabh@rishabh:~/GFG$ ls -l | more
total 28
drwxrwxr-x 2 rishabh rishabh 4096 Jan 29 21:11 demo1
-rw-rw-r-- 1 rishabh rishabh  26 Jan 25 23:03 demo1.txt
drwxrwxr-x 2 rishabh rishabh 4096 Jan 29 21:11 demo2
-rw-rw-r-- 1 rishabh rishabh   0 Jan 25 23:04 demo2.txt
drwxrwxr-x 2 rishabh rishabh 4096 Jan 29 21:11 demo3
-rw-rw-r-- 1 rishabh rishabh   0 Jan 25 23:04 demo.txt
-rw-rw-r-- 1 rishabh rishabh 123 Jan 26 16:02 sample1.txt
-rw-rw-r-- 1 rishabh rishabh  44 Jan 26 15:52 sample2.txt
-rw-rw-r-- 1 rishabh rishabh   0 Jan 26 00:12 sample3.txt
-rw-rw-r-- 1 rishabh rishabh  26 Jan 25 23:03 sample.txt
```

Two-way Communication Using Pipes
(combinando 2 pipe)

Permettere a 2 processi di scambiarsi informazioni fra loro.



Esempio di pipe tra comandi shell



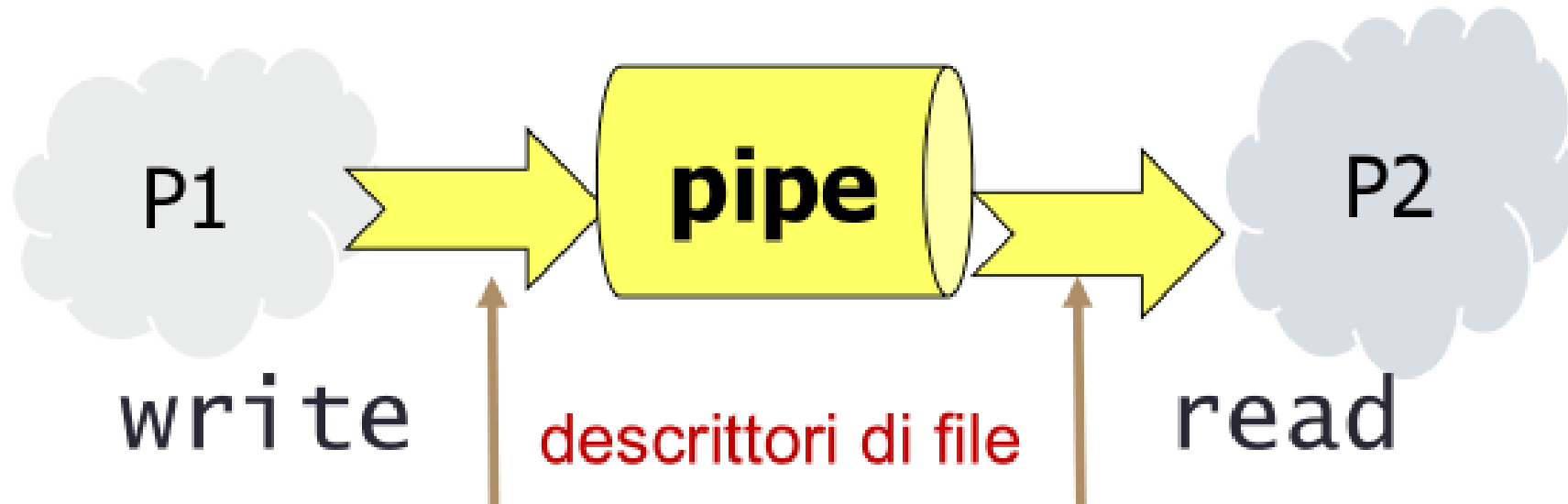
Una pipe è un tipo speciale di file condiviso, riconducibili al modello “memoria condivisa”

Una pipe è un tipo speciale di file condiviso. Due tipi

- Convenzionali (o anonime)
- Named pipe

Comunicazione tra due processi secondo la modalità del «produttore e consumatore»

- Il produttore scrive da un'estremità (write-end)
- Il consumatore legge dall'altra estremità (read-end)
- Comunicazione uni-direzionale



Implementata come file in memoria centrale con scrittura solo in aggiunta e lettura unica solo sequenziale

- Deve esistere una relazione padre-figlio tra i due processi per condividere i descrittori di file
- In Unix può essere creata con la chiamata `pipe(int fd[])`
- `fd` è il descrittore del file, `fd[0]` scrittura, `fd[1]` lettura
- Lettura e scrittura tramite le chiamate `read()` e `write()`

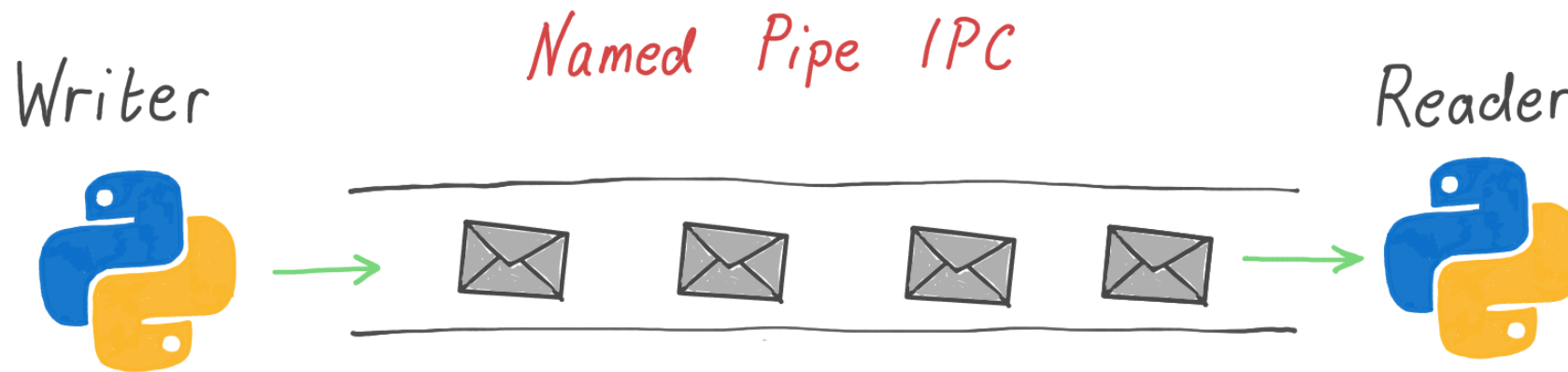


Operating Systems: IPC

Gestione delle Pipe: Named Pipe 1/2

Implementata come file referenziabile tramite File System

- Bidirezionali
- Relazione parentela padre-figlio non necessaria
- Comunicazione fra più di due processi
- Continuano ad esistere anche dopo che i processi comunicanti terminano



Unix:

- Dette FIFO e create con `mkfifo()`, ma sono normali file del file system
- Half-duplex (i dati viaggiano in un'unica direzione alla volta)
- I programmi comunicanti devono risiedere nella stessa macchina (in alternativa, occorrono i socket)
- Dati byte-oriented

Win32:

- Meccanismo più ricco: `createNamedPipe()` (nuova pipe) e `ConnectNamedPipe()` (pipe già esistente)
- Full-duplex (i dati viaggiano contemporaneamente in entrambe le direzioni)
- I programmi comunicanti possono risiedere in macchine diverse (ambiente client/server)
- Dati byte-oriented e message-oriented

Operating Systems: IPC

Gestione della Memoria Condivisa 1/2



Operating Systems: Obiettivi Funzioni Servizi

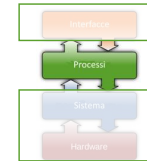
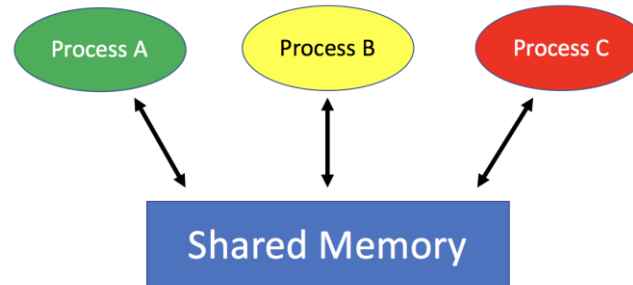
IPC: Shared Memory

Shared Memory: memoria condivisa fra processi

SHMMAX: massima dimensione (in Byte) di un singolo segmento di memoria condivisa

SHMMNI: numero massimo di segmenti di memoria condivisa presenti sul Sistema

SHMALL: numero totale di pagine di memoria condivisa



Alloca porzioni di memoria condivisa fra processi.



I processi comunicanti colloquiano attraverso un'area di memoria condivisa

- Residente nello spazio degli indirizzi del processo che la alloca
- Gli altri processi annettono questa area al loro spazio degli indirizzi
- Tipicamente i processi non possono accedere alla memoria degli altri → I processi comunicanti devono quindi stabilire un “accordo”

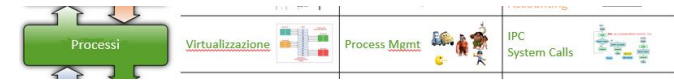
Il SO **non controlla**

- Tipo e allocazione dei dati
- Accesso concorrente alla memoria



Operating Systems: IPC

Gestione della Memoria Condivisa 2/2



Caratteristiche:

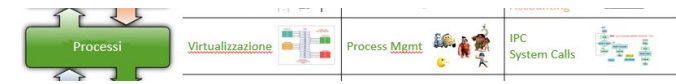
- Condivisione variabili globali
- Condivisione buffer di comunicazione (vedi producer-consumer)

Problemi:

- Identificazione dei processi comunicanti (comunicazione diretta)
- Consistenza degli accessi
- Lettura e scrittura sono incompatibili tra loro → Richiede **sincronizzazione dei processi** per accesso in **mutua esclusione**

Operating Systems: IPC

Gestione della Memoria Condivisa: Produttore/Consumatore



Modello molto comune nei processi cooperanti:

- un processo produttore genera informazioni
- che sono utilizzate da un processo consumatore

Un oggetto temporaneo in memoria (buffer) può essere riempito dal produttore e svuotato dal consumatore

1. **buffer illimitato** (unbounded-buffer): non c'è un limite teorico alla dimensione del buffer

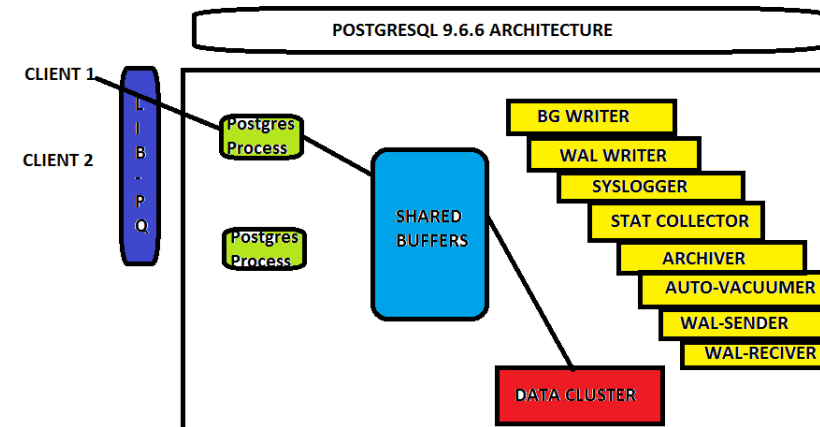
- Il consumatore deve aspettare se il buffer è vuoto
- Il produttore può sempre produrre

2. **buffer limitato** (bounded-buffer): la dimensione del buffer è fissata

- Il consumatore deve aspettare se il buffer è vuoto
- Il produttore deve aspettare se il buffer è pieno

Il buffer può essere:

- fornito dal SO tramite l'uso di funzionalità di comunicazione tra processi InterProcess Communication (IPC)
- oppure essere esplicitamente scritto dal programmatore dell'applicazione con l'uso di memoria condivisa



System Call

Comunicazione con il Kernel

Operating Systems: System Call

Generalità

Le syscall() sono il **modo** in cui un processo entra nel kernel per eseguire alcune attività che richiedono privilegi più alti.

I programmi utilizzano le chiamate di sistema per eseguire una serie di operazioni quali: creazione di processi, operazioni di I/O di rete e di file e molto altro.

Linux: elenco delle chiamate di sistema alla pagina man per syscalls(2):
<https://man7.org/linux/man-pages/man2/syscalls.2.html>

System call	Kernel	Notes
_llseek(2)	1.2	
_newselect(2)	2.0	
_sysctl(2)	2.0	Removed in 5.5
accept(2)	2.0	See notes on socketcall(2)
accept4(2)	2.6.28	
access(2)	1.0	
acct(2)	1.0	
add_key(2)	2.6.10	
adjtimex(2)	1.0	
alarm(2)	1.0	
alloc_hugepages(2)	2.5.36	Removed in 2.5.44
arc_gettls(2)	3.9	ARC only
arc_settls(2)	3.9	ARC only
arc_usr_cmxchg(2)	4.9	ARC only
arch_prctl(2)	2.6	x86_64, x86 since 4.12
atomic_barrier(2)	2.6.34	m68k only
atomic_cmxchg_32(2)	2.6.34	m68k only
bdflush(2)	1.2	Deprecated (does nothing) since 2.6
bind(2)	2.0	See notes on socketcall(2)
bpf(2)	3.18	
brk(2)	1.0	
breakpoint(2)	2.2	ARM OABI only, defined with __ARM_NR prefix
cacheflush(2)	1.2	Not on x86
capget(2)	2.2	
capset(2)	2.2	
chdir(2)	1.0	
chmod(2)	1.0	
chown(2)	2.2	See chown(2) for version details
chown32(2)	2.4	
chroot(2)	1.0	
clock_adjtime(2)	2.6.39	
clock_getres(2)	2.6	
clock_gettime(2)	2.6	
clock_nanosleep(2)	2.6	
clock_settime(2)	2.6	
clone2(2)	2.4	IA-64 only
clone(2)	1.0	
clone3(2)	5.3	
close(2)	1.0	
close_range(2)	5.9	
connect(2)	2.0	See notes on socketcall(2)
copy_file_range(2)	4.5	
creat(2)	1.0	
create_module(2)	1.0	Removed in 2.6

System call	Kernel	Notes
delete_module(2)	1.0	
dup(2)	1.0	
dup2(2)	1.0	
dup3(2)	2.6.27	
opoll_create(2)	2.6	
opoll_create1(2)	2.6.27	
opoll_ctl(2)	2.6	
opoll_pwait(2)	2.6.19	
opoll_pwait2(2)	5.11	
opoll_wait(2)	2.6	
eventfd(2)	2.6.22	
eventfd2(2)	2.6.27	
execv(2)	2.0	SPARC/SPARC64 only, for compatibility with SunOS
execve(2)	1.0	
execveat(2)	3.19	
exit(2)	1.0	
exit_group(2)	2.6	
faccessat(2)	2.6.16	
faccessat2(2)	5.8	
fadvise64(2)	2.6	
fadvise64_64(2)	2.6	
fallocate(2)	2.6.23	
fanotify_init(2)	2.6.37	
fanotify_mark(2)	2.6.37	
fchdir(2)	1.0	
fchmod(2)	1.0	
fchmodat(2)	2.6.16	
fchown(2)	1.0	
fchown32(2)	2.4	
fchownat(2)	2.6.16	
fcntl(2)	1.0	
fcntl64(2)	2.4	
fdatasync(2)	2.0	
fgetxattr(2)	2.6; 2.4.18	
finit_module(2)	3.8	
flistxattr(2)	2.6; 2.4.18	
flock(2)	2.0	
fork(2)	1.0	
free_hugepages(2)	2.5.36	Removed in 2.5.44
fronvexattr(2)	2.6; 2.4.18	
fsconfig(2)	5.2	
fsetxattr(2)	2.6; 2.4.18	
fsmount(2)	5.2	
fsopen(2)	5.2	
fspick(2)	5.2	
fstat(2)	1.0	
fstat64(2)	2.4	
fstatat64(2)	2.6.16	
fstatfs(2)	1.0	
fstatfs64(2)	2.6	
fsync(2)	1.0	
ftruncate(2)	1.0	
ftruncate64(2)	2.4	
futex(2)	2.6	
futimesat(2)	2.6.16	

Operating Systems: System Call

Esecuzione del Kernel



il codice del kernel viene eseguito in due possibili contesti:

1. **Processo** (syscall) in modalità kernel-space per conto di un processo specifico (i.e. non cambia il PID)
2. **Interrupt** in modalità kernel-space non vincolato ad alcun processo → **Context Switch**

La CPU esegue il codice dell'area utente nello spazio utente o il codice del kernel in uno dei due contesti sopra menzionati.

Context Switch: cambiare il processo corrente; è ciò che accade quando cambia il PID corrente (quando lo scheduler anticipa un processo); Nelle righe seguenti vorrei sottolineare che una syscall avviene senza questo cambio di contesto.

Operating Systems: System Call

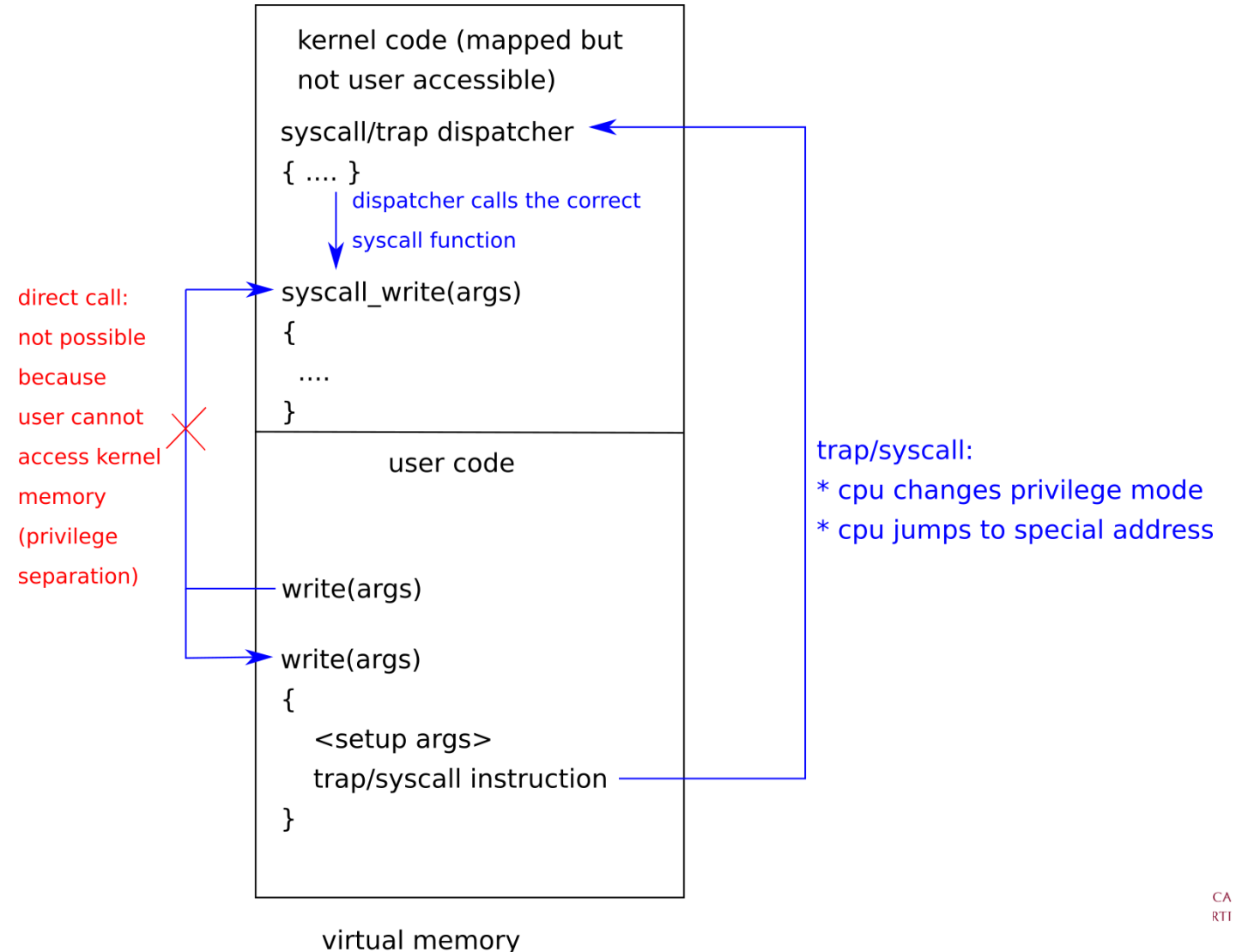
Context Privilege Switch

Una syscall() non richiede necessariamente un cambio di contesto, in generale. Piuttosto un **cambio di privilegi**.

La memoria del kernel (**Privilegio: Sistema**) è mappata in ogni memoria di processo (**Privilegio: Utente**) ma questo **non può accedervi**.

Una syscall() necessita di un cambiamento di privilegio, non di processo.

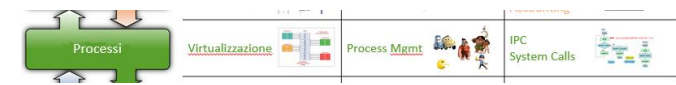
→ **micro-kernel**, caso particolare: syscall() può comportare un **context switch**: i **driver** si trovano in un **processo diverso**.



Operating Systems: System Call

Chiamata di sistema in Linux

Esistono diversi modi in cui i programmi utente possono effettuare `syscall()`.



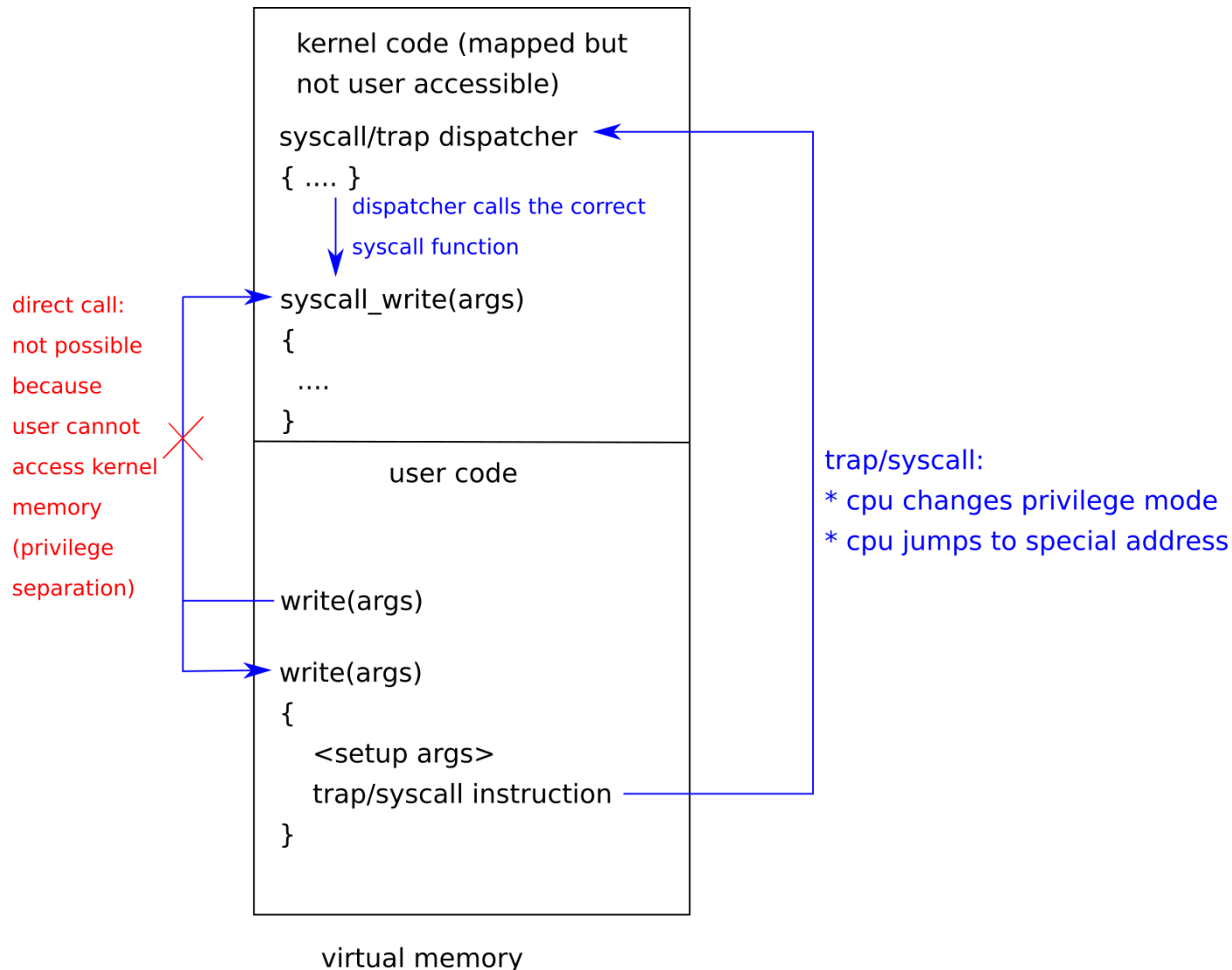
- 1. Legacy System Call:** generare un interrupt software, tramite l'istruzione assembler `int`. Il kernel Linux registra, su architettura Intel, un gestore di interrupt chiamato `ia32_syscall` per il number di interrupt: 128 (0x80). → **Context Switch**
- 2. Fast System Call:** composto da due istruzioni con supporto hardware. Uno per entrare nel kernel e uno per uscire. Le istruzioni di basso livello per effettuare una chiamata di sistema variano a seconda delle architetture della CPU. Es. riguardo la CPU Intel, i metodi sono descritti nella documentazione "Fast System Call".
- 3. Virtual System Call:** Linux supporta **vDSO** (virtual Dynamic Shared Object) per implementare alcune chiamate di sistema interamente nello spazio utente. Queste chiamate NON richiedono l'esecuzione con privilegio Sistema → **no Privilege Switch**.
- 4. Wrapper:** Per molte chiamate di sistema, **glibc** necessita semplicemente di una funzione wrapper in cui sposta gli argomenti nei registri appropriati e quindi esegue le istruzioni `syscall` o `int $0x80`, oppure chiama `__kernel_vsycall`. Lo fa utilizzando una serie di tabelle definite in file di testo che vengono elaborati con script e codice C di output.

Glibc: In qualità di sviluppatore di applicazioni, in genere, non è necessario pensare a come viene eseguita esattamente una chiamata di sistema. È sufficiente includere il file di intestazione appropriato ed effettuare la chiamata come se fosse una normale funzione.

Per maggiori informazioni: <https://blog.packagecloud.io/eng/2016/04/05/the-definitive-guide-to-linux-system-calls>



Operating Systems: System Call Library Wrapper



glibc fornisce il codice wrapper che:

- astrae dal codice sottostante
- organizza gli argomenti da passato
- entra nel kernel.

Spesso la funzione glibc wrapper è piuttosto sottile, fa poco lavoro:

- copiare gli argomenti nei registri giusti;
- invocare la chiamata di sistema;
- impostare `errno` in modo appropriato dopo che la chiamata di sistema è tornata.

(stessi passaggi eseguiti da `syscall(2)`, che possono essere utilizzati per invocare chiamate di sistema per le quali non è fornita alcuna funzione wrapper.)