

# Ripasso Sistemi Operativi

Matteo Franchini

22 agosto 2023

- 1. Introduzione
- 2. Evoluzione dei sistemi operativi
- 3. Appocondimento delle interruzioni
- 4. Servizi e organizzazione del SO
- 5. Virtualizzazione
- 6. Processi

# Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Apporfondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

# Che cos'è un sistema operativo?

## Introduzione

Un programma che funge da intermediario tra l'utente di un computer e l'hardware del computer stesso

## Obiettivi del sistema operativo:

- Eseguire i programmi dell'utente e facilitarne la risoluzione dei problemi
- Rendere il sistema informatico comodo da usare
- Utilizzare l'hardware del computer in modo efficiente

# Definizione di SO

## Introduzione

**Il sistema operativo** è un allocatore di risorse:

- gestisce tutte le risorse
- decide tra richieste in conflitto per un uso efficiente ed equo delle risorse

**Il sistema operativo** è un programma di controllo

- controlla l'esecuzione dei programmi per prevenire gli errori e l'uso improprio del computer

Non c'è una definizione universalmente accettata

- con il termine sistema operativo si intende quell'insieme di programmi che provvedono alla gestione Hw e Sw di un sistema di calcolo
- "*Tutto ciò che viene fornite quando si ordina un SO*"

Una definizione alternativa (**Tanenbaum**):

*un sistema operativo è un programma che controlla le risorse di un calcolatore e fornisce ai suoi utenti un'interfaccia o macchina virtuale più agevole da utilizzare della macchina "nuda"*

*L'unico programma che è sempre in esecuzione sul computer è il kernel (nucleo) del SO.*

Il resto è

- un programma di sistema (fornite con il SO di cui costituisce una parte)
- un programma applicativo

# Sistema Operativo

## Introduzione

Può essere visto come:

### 1. Allocatore di risorse Hw e Sw:

- tempo di CPU, spazio di memoria, dispositivo di I/O, compilatori
- Le risorse devono essere assegnate a programmi specifici secondo determinate politiche

### 2. Programma di controllo: controlla l'esecuzione dei programmi per prevenire errori ed usi impropri del calcolatore

Obiettivi principali del SO:

- rendere più **semplice** l'uso di un sistema di elaborazione
- rendere più **efficiente** l'uso delle risorse del sistema di elaborazione

Il SO è costituito dall'insieme dei programmi (sw o fw) che **rendono praticamente utilizzabile** l'elaboratore agli utenti cercando contemporaneamente di **ottimizzarne le prestazioni**.

- **Visione top-down**: il sistema operativo come una macchina estesa (**astrazione**)
- **Visione bottom-up**: il sistema operativo come un gestore di risorse (**fornisce protezione, risoluzione dei conflitti**)

# Avvio dell'elaboratore

## Introduzione

Vi è un **programma di bootstrap** che normalmente è memorizzato in una **memoria non volatile**, inizializza e verifica il corretto funzionamento dei componenti Hw del sistema. **Carica il kernel del SO e inizia l'esecuzione.**

Nei PC più vecchi c'era il **BIOS**, ora invece è stato sostituito con il **UEFI**.

# Proprietà fondamentali di un SO

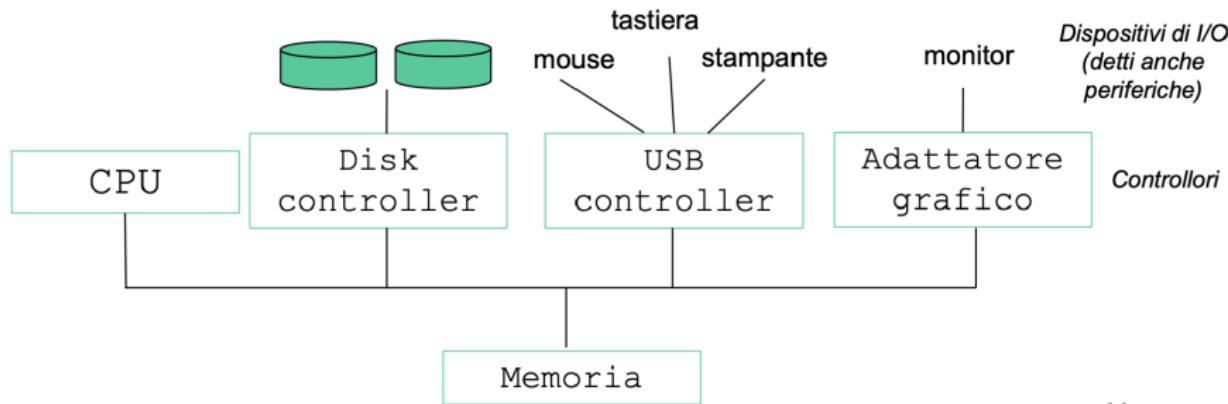
## Introduzione

- **Affidabilità**
- **Efficienza**
- **Sicurezza**

# Organizzazione di un elaboratore

## Introduzione

Uno o più CPU si connettono mediante un **bus condiviso**, vi è l'esecuzione concorrente delle CPU e dei dispositivi che competono per i cicli di memoria



14

# Funzionamento di un elaboratore

## Introduzione

- I dispositivi di I/O e la CPU possono operare simultaneamente
- Ogni controller di dispositivo è responsabile di un particolare dispositivo
- Ogni controllore di dispositivo ha un buffer locale
- La CPU sposta i dati da/alla memoria principale a/dai buffer locali
- L'I/O è dal dispositivo al buffer locale del controllore
- Il controllore del dispositivo informa la CPU di aver terminato la sua operazione causando un interrupt

# Funzioni comuni degli interrupt

## Introduzione

La gestione degli interrupt deve salvare l'indirizzo dell'istruzione interrotta

## Eccezione

Una trap o eccezione è un'interruzione generata dal Sw e causata da un errore o da una richiesta al SO da parte di un programma

Un sistema operativo è guidato dagli interrupt

# Gestione degli interrupt

## Introduzione

Il sistema operativo preserva lo stato della CPU memorizzando i registri e il contatore del programma

# Struttura dell'I/O

## Introduzione

1. Dopo l'avvio dell'I/O, il controllo ritorna al programma utente solo al completamento dell'I/O
  - Un'istruzione *wait* mette a riposo la CPU fino al prossimo interrupt
  - Ciclo di attesa
  - È in sospeso al massimo una richiesta di I/O alla volta, nessuna elaborazione simultanea di I/O
2. Dopo l'avvio dell'I/O, il controllo torna al programma utente senza attendere il completamento dell'I/O
  - **System call:** richiesta al SO per consentire al programma di attendere il completamento dell'I/O
  - La tabella di stato dei dispositivi contiene una voce in cui è indicato l'**indirizzo** e lo **stato**
  - Il SO **indicizza la tabella dei dispositivi di I/O** per determinare lo stato del dispositivo e modificare la voce della tabella per includere interrupt

# Funzioni specifiche di gestione

## Introduzione

### Gestione della memoria centrale

- caricare in memoria programmi e dati
- evitare interferenze fra programmi diversi
- assegnare la memoria in base a criteri di efficienza
- minimizzare i trasferimenti tra memoria centrale e memoria di massa

## Gestione della memoria secondaria

- consentire l'accesso all'informazione in base alla sua organizzazione logica anziché fisica
- controllare i diritti di accesso ai file da parte degli utenti
- consentire creazione, modifica, cancellazione dei file

## Gestione dei dispositivi periferici

- mascherare al programmatore la complessità delle operazioni di I/O
- effettuare controlli sul corretto funzionamento delle operazioni
- risolvere conflitti nell'utilizzo di una stessa periferica da parte di più programmi

## Gestione dei processi

- decidere quale programma userà il processore (**scheduling**) in base a criteri di corretto funzionamento e di efficienza
- verificare che i programmi rilascino il processore entro il tempo stabilito

# Funzioni di un SO

## Introduzione

- definizione e gestione dell'interfaccia utente
- gestione dei job
- gestione delle risorse di sistema
- ausili per la messa a punto dei programmi
- ausili per la gestione dei dati - file system

# Struttura della memoria

## Introduzione

### Memoria principale

solo supporti di memoria di grandi dimensioni a cui la CPU può accedere direttamente

### Memoria secondaria

estensione della memoria principale che **fornisce un'ampia capacità di memorizzazione non volatile**

- Dischi rigidi
- A stato solido

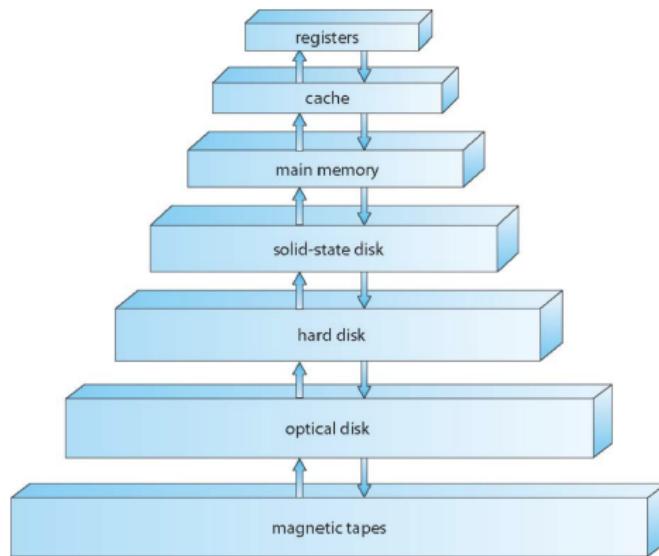
# Gerarchia di memoria

## Introduzione

**Sistemi di storage organizzati in una gerarchia.**

**Caching:** copiare le informazioni in un sistema di archiviazione più veloce

**Driver del dispositivo:** per ogni controller di dispositivo per gestire l'I/O



# Caching

## Introduzione

Le informazioni in uso vengono **copiate temporaneamente da una memoria più lenta a una più veloce**

La memoria più veloce (**cache**) viene controllata per prima per determinare se le **informazioni sono lì**, in caso contrario, i dati vengono copiati nella cache e utilizzati lì.

La **cache è più piccola della memoria di cui si fa il caching**, la gestione della cache è un importante problema di progettazione.

# Struttura di accesso diretto alla memoria

## Introduzione

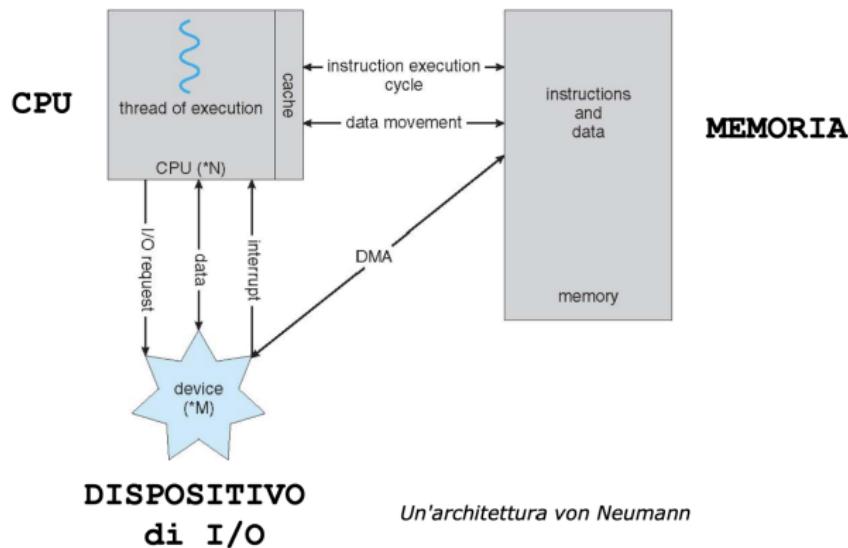
### DMA (Direct Memory Access)

Utilizzata per dispositivi di I/O ad alta velocità in grado di trasmettere informazioni a velocità prossime a quelle della memoria.

Il controller del dispositivo trasferisce blocchi di dati dalla memoria tampone direttamente alla memoria principale senza l'intervento della CPU

# Funzionamento PC moderno

## Introduzione



# Architettura del sistema informatico

## Introduzione

L'uso e l'importanza dei sistemi multiprocessore è in crescita.

I vantaggi includono:

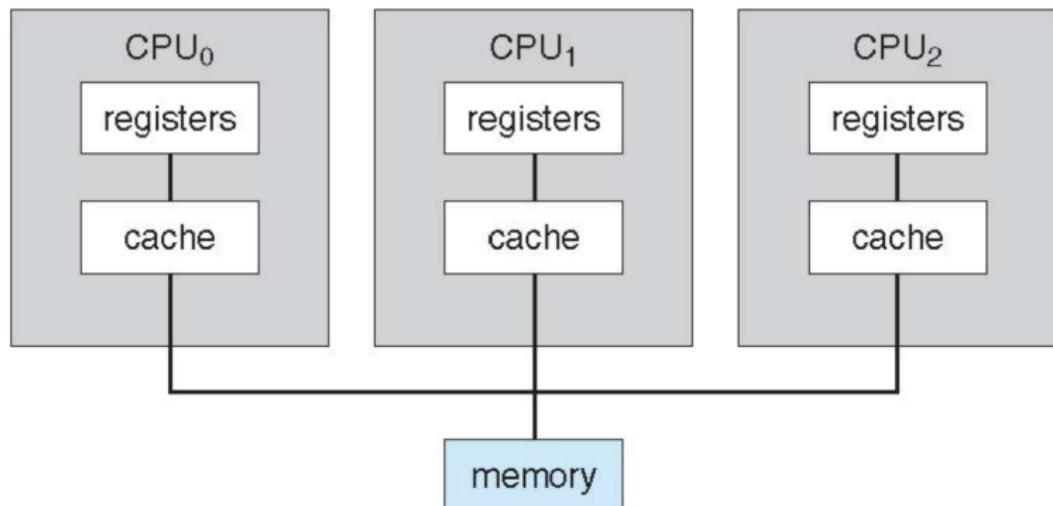
1. **Aumento del throughput**
2. **Economia di scala**
3. Maggiore affidabilità: *graceful degradation o tolleranza ai guasti*

Ci sono due tipi di multiprocesso:

1. **Multiprocesso asimmetrico**: a ogni processore
2. **Multiprocesso simmetrico**: ogni processore esegue tutti i compiti

# Architettura del multiprocessore simmetrico

## Introduzione



# Sistemi cluster

## Introduzione

Sono come i sistemi multiprocessore, ma con **più sistemi che lavorano insieme**, di solito condividono lo **storage** tramite una **rete SAN (Storage Area Network)**.

### Tipi di cluster

- **Clustering asimmetrico:** prevede una macchina in modalità **hot-standby**
- **Clustering simmetrico:** prevede più nodi che eseguono applicazioni, monitorandosi a vicenda

Alcuni cluster sono destinati al calcolo ad alte prestazioni (HPC), altri hanno un **gestore di lock distribuito (DLM)** per evitare operazioni in conflitto.

# Multiprogrammazione e multitasking

## Introduzione

### Multiprogrammazione (sistema batch)

- Un singolo utente non può tenere occupati CPU e dispositivi di I/O in ogni momento
- La **multiprogrammazione organizza i lavori** (codice e dati) in modo che la CPU ne abbia sempre uno da eseguire
- Un sottoinsieme di lavori totali nel sistema viene tenuto in memoria
- Quando deve aspettare, il sistema operativo **passa a un altro lavoro**

## Multitasking

Il **timesharing** è un'estensione logica in cui la **CPU passa ai lavori con una frequenza tale da consentire agli utenti di interagire con ciascun lavoro mentre è in esecuzione**

- Il tempo di risposta deve essere  $< 1$  secondo
- Ogni utente ha almeno un programma in esecuzione
- Se diversi lavori sono pronti per essere eseguiti contemporaneamente abbiamo la **schedulazione della CPU**
- La **memoria virtuale** consente l'esecuzione di processi non completamente in memoria

# Operazioni del sistema operativo

## Introduzione

Guidato dalle interruzioni (Hw e Sw):

- Interruzione hardware da parte di uno dei dispositivi
- Interruzione software (eccezione o trap): errore, richiesta di servizio, altri problemi

Il funzionamento in **doppia modalità** permette al sistema operativo di proteggere se stesso e gli altri componenti del sistema: **modalità utente e modalità kernel**.

Alcune istruzioni designate come *privilegiate* sono eseguibili solo in **modalità kernel**.

# Transizione dalla modalità utente a quella kernel

## Introduzione

C'è un **timer per prevenire il loop infinito**.

Il timer è impostato per interrompere il computer dopo un certo periodo di tempo e conserva un **contatore che viene decrementato dall'orologio fisico**.

Quando il **contatore è zero viene generato un interrupt**.

# Gestione del processo

## Introduzione

### Processo

Un processo è un **programma in esecuzione**. È un'unità di lavoro all'interno del sistema. Il programma è un'**entità passiva**, il processo è un'**entità attiva**.

Un **processo a singolo thread** ha un contatore di programma che specifica la posizione della prossima istruzione da eseguire.

Un **processo multi-thread** ha un contatore di programma per ogni thread.

# Attività di gestione dei processi

## Introduzione

Il sistema operativo è responsabile delle seguenti attività relative alla gestione dei processi:

- Creazione e cancellazione di processi utente e di sistema
- Sospendere e riprendere i processi
- Meccanismi di sincronizzazione dei processi
- Meccanismi di comunicazione tra i processi
- Meccanismi per la gestione dei deadlock

# Gestione della memoria di massa

## Introduzione

Generalmente i dischi sono utilizzati per memorizzare dati che non stanno nella memoria principale o dati che devono essere tenuti per un periodo di tempo "lungo".

Attività del sistema operativo

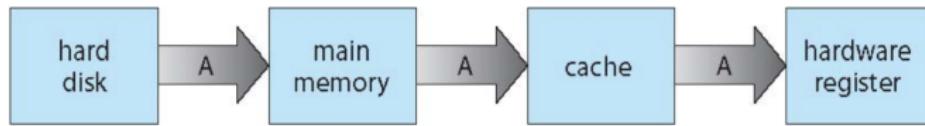
- Gestione dello spazio libero
- Allocazione dello spazio di archiviazione
- Pianificazione dei dischi

# Migrazione del dato "A" da un disco ad un registro

## Introduzione

Gli ambienti multitasking devono fare attenzione a utilizzare il valore più recente, indipendentemente dalla sua posizione nella gerarchia di memorizzazione.

Gli ambienti multiprocessore devono garantire la coerenza della cache in Hw, in modo che tutte le CPU abbiano il valore più recente nella loro cache



# Sottosistema I/O

## Introduzione

Uno degli scopi del sistema operativo è quello di **nascondere all'utente le peculiarità dei dispositivi Hw.**

Il sottosistema I/O è responsabile di:

- Gestione della memoria dell'I/O comprende il buffering, il caching in una memoria più veloce, lo spooling
- Interfaccia generale dispositivo-driver
- Driver per dispositivi Hw specifici

# Protezione e sicurezza

## Introduzione

### Protezione

Qualsiasi meccanismo di controllo dell'accesso dei processi o degli utenti alle risorse definite dal SO

### Sicurezza

Difesa del sistema da attacchi interni ed esterni

I sistemi in genere distinguono innanzitutto fra gli utenti, per determinare chi può fare cosa:

- Le identità degli utenti (**ID utente**)
- L'ID utente viene poi associato a tutti i file e processi di quell'utente per determinare il controllo degli accessi
- L'identificativo di gruppo (**ID di gruppo**) consente di definire un insieme di utenti e di gestire i controlli, quindi anche di associarlo a ciascun processo, file o altro
- L'**escalation dei privilegi** consente all'utente di passare ad un ID effettivo con maggiori diritti

# Ambienti di elaborazione - Tradizionali

## Introduzione

I **portali** forniscono accesso web ai sistemi interni. I **computer in rete** sono come terminali web.

I computer mobili si collegano tramite **reti wireless**

# Ambienti informatici - Mobile

## Introduzione

Permette nuovi tipi di applicazioni come la realtà aumentata.  
Utilizza reti wireless IEEE 802.11 o reti dati cellulari per la connettività

# Ambienti di elaborazione - Distribuiti

## Introduzione

Collezione di **sistemi separati**, eventualmente eterogenei, collegati in rete tra loro.

La **rete** è un percorso di comunicazione, il **TCP/IP** è il più comune:

- Rete Locale (LAN)
- Rete geografica (WAN)
- Rete metropolitana (MAN)
- Rete personale (PAN)

# Ambienti informatici - Virtualizzazione

## Introduzione

### Emulazione

Utilizzata quando il tipo di CPU di origine è diverso da quello di destinazione.

Questo metodo è generalmente più lento.

Quando il linguaggio del computer non viene compilato in codice nativo siamo di fronte all'**interpretazione**

### Virtualizzazione

Sistema operativo compilato in modo nativo per la CPU, che esegue sistemi operativi **guest** anch'essi compilati in codice nativo

# Ambienti di elaborazione - Cloud Computing

## Introduzione

Fornisce elaborazione, storage e persino applicazioni come servizio attraverso una rete.

Estensione logica della virtualizzazione perché utilizza la virtualizzazione come base per le sue funzionalità.

Alcuni tipi:

- **Cloud pubblico:** disponibile via internet a chiunque sia disposto a pagare
- **Cloud privato:** gestito da un'azienda per uso proprio
- **Cloud ibrido:** include componenti di cloud pubblico e privato
- **SaaS (Software as a Service):** uno o più applicazioni disponibili via internet
- **PaaS (Platform as a Service):** stack di software pronto per l'uso di applicazioni via internet
- **IaaS (Infrastructure as a Service):** server o storage disponibili via internet

# Aree di applicazione di un SO

## Introduzione

- **Sistemi di tipo generale**
- **Sistemi in tempo reale**
  - applicazioni per il controllo di processo e di apparati fisici
  - applicazioni interattive, interrogazione di basi di dati, query web

# Ambienti di elaborazione - Sistemi embedded in tempo reale

## Introduzione

I sistemi **embedded in tempo reale** sono la forma più diffusa di computer.

Il sistema operativo in tempo reale ha vincoli temporali fissi ben definiti

- L'elaborazione deve essere eseguita entro i vincoli
- Operazione corretta solo se i vincoli sono rispettati

# Sistemi operativi open source

## Introduzione

Sistemi operativi resi disponibili in formato di codice sorgente piuttosto che solo binari closed-source.

Avviato dalla **Free Software Foundation (FSF)**, che ha una licenza pubblica GNU (GPL).

Esempi: GNU/LINUX

# Sistemi operativi open source

## Introduzione

### Sistemi aperti

- Realizzati e mantenuti da comunità di sviluppatori volontari
- Movimento per il software "open source"
- Le applicazioni beneficiano della piena conoscenza del SO, il SO evolve nel tempo e migliora grazie alla sua trasparenza
- Esempio canonico: Linux
- Interfaccia utente e **look and feel** sono modificabili

# Utenti del SO

## Introduzione

- **Utenti finali del sistema:** per essi il sistema operativo è trasparente
- **Programmatori applicativi:** utilizzano i servizi del SO per la realizzazione e l'esecuzione dei loro programmi
- **Programmatori di sistema:** aggiornano e modificano i programmi del SO per adeguarli a nuove necessità del sistema o degli utenti applicativi
- **Operatori:** controllano il funzionamento e rispondono alle richieste di intervento da parte del sistema
- **Amministratore del sistema:** stabilisce le politiche di gestione del sistema e ne cura l'osservanza

# Tipi di SO

## Introduzione

### Sistemi proprietari

- Progettati da costruttori al fine di sfruttare in modo ottimale le risorse di ciascun tipo di macchina
- Programmi utente e applicazioni si interfacciano al SO in modo diverso tra le diverse famiglie di sistemi
- IBM: OS/360 370, VM, MVS

### Sistemi standard

- Progettati da aziende software o da grandi utenti per consentire lo sviluppo di applicazioni portabili su sistemi diversi
- Interfaccia di programmazione con cui le applicazioni interagiscono con il SO rimane costante nelle diverse versioni
- Esempi: UNIX, MS-DOS

# Proprietà fondamentali di un SO

## Introduzione

- **Architettura:** com'è organizzato?
- **Condivisione:** quali risorse vengono condivise?
- **Efficienza:** come massimizzare l'utilizzo delle risorse disponibili?
- **Affidabilità/tolleranza ai guasti**
- **Estensibilità**
- **Protezione e sicurezza**
- **Conformità a standard**

# Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Apporfondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

# Stadi evolutivi e modalità d'uso dei sistemi

## Evoluzione dei sistemi operativi

- Sistema isolato ('50-'60): batch
- Sistema centralizzato ('60-'70): remote job entry, teleprocessing, time sharing
- Sistemi decentrati ('70-): minicalcolatore, controllo real-time, data logging
- Sistemi distribuiti ('80-): multiprocessori, reti locali

# Client-Server

## Evoluzione dei sistemi operativi

I dispositivi **client** sono utilizzati dagli utenti, molti sistemi sono **server** che rispondono alle richieste di servizio dei client offrendo: servizi di calcolo o gestione (DBMS), servizi di gestione dei file (NFS), altri servizi

# Sistema di elaborazione distribuito

## Evoluzione dei sistemi operativi

È costituito da nodi tra loro collegati in ciascuno dei quali sono presenti capacità di: **elaborazione, memorizzazione, comunicazione**.

### Vantaggi:

- Tolleranza al guasto: un guasto non provoca l'arresto del sistema, ma solo una riduzione delle prestazioni
- Prestazioni: l'elaborazione di norma è effettuata nel posto stesso di utilizzazione, si ha un miglioramento delle prestazioni
- Condivisione: la capacità di elaborazione, i programmi ed i dati esistenti nell'intero sistema sono patrimonio comune di tutti gli utenti

# Evoluzione dei SO

## Evoluzione dei sistemi operativi

- **Primi calcolatori:** privi di SO, problemi di complessità di operazioni, inefficienza
- **Prima generazione ('50-'60):** virtualizzazione dell'I/O, librerie di controllo dei device, problemi di debug, nasce il **sistema operativo** come stratificazione successiva di funzioni volte ad aumentare l'efficienza e la semplicità d'uso della macchina
- **Seconda generazione ('60-'65):** indipendenza tra programmi e dispositivi usati, parallelizzazione degli utenti tramite **multiprogrammazione e time sharing**
- **Terza generazione ('65-'75):** SO unico per una famiglia di elaboratori, risorse virtuali, sistemi multifunzione

- **Quarta generazione ('75-'85)**: sistemi a macchine virtuali, sistemi multiprocessore e distribuiti
- **Quinta generazione ('85-'95)**: elaboratori personali, reti locali, avvio di internet
- **Sesta generazione ('95-'05)**: elaborazione distribuita, peer to peer, servizi online

# Tecniche di gestione di un sistema di calcolo

## Evoluzione dei sistemi operativi

### Monoprogrammazione

Gestisce in **modo sequenziale** nel tempo i diversi programmi. L'inizio di un programma avviene solamente dopo il completamente del programma precedente.

**Tutte le risorse sono dedicate ad un solo programma.**

Bassa utilizzazione delle risorse

$$\text{uso CPU} = \frac{T_p}{T_t}$$

$T_p$  = tempo dedicato dalla CPU all'esecuzione del programma

$T_t$  = tempo totale di permanenza nel sistema del programma

## Sistema multiprogrammato

Carica in memoria e gestisce **simultaneamente più programmi indipendenti**, nel senso che ciascuno di essi può iniziare o proseguire l'elaborazione prima che un altro sia terminato.  
Le risorse risultano meglio utilizzate in quanto **riducono i tempi morti**

# Gestione Batch

## Evoluzione dei sistemi operativi

Significa raggruppare i lavori o programmi in **lotti** per conseguire una **maggior utilizzazione delle risorse**, cioè un *throughput* più elevato.

Nel caso di multiprogrammazione il SO deve provvedere algoritmi per la scelta di quell'insieme di programmi che, in esecuzione contemporanea, massimizza il throughput.

La **gestione batch** può essere **locale** (unità centrale direttamente collegata ai dispositivi di I/O) o **remota** (è presente una trasmissione dei job e dei risultati ed eventualmente una memorizzazione intermedia).

# Time sharing

## Evoluzione dei sistemi operativi

L'elaboratore serve **simultaneamente** una pluralità di utenti, dotati di terminali, dedicando a ciascuno di essi tutte le risorse del sistema per **quanti fissati di tempo**.

Migliora i **tempi di risposta** ma peggiora l'utilizzazione delle risorse.  
Normalmente una modalità di gestione **time-sharing** è adottata nei **sistemi conversazionali**, in cui più utenti contemporaneamente "colloquiano" con il sistema

# Spooling

## Evoluzione dei sistemi operativi

Per accrescere la velocità di esecuzione dei programmi conviene utilizzare la **memoria a disco per simulare i dispositivi di I/O**: il disco viene usato come un buffer di grosse dimensioni a cui accedono sia il lettore di schede che la CPU.

I trasferimenti lettore di schema-memoria di massa (**spool-in**) e memoria di massa-stampante (**spool out**) sono effettuati da appositi programmi detti di **spooling**.

# Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Apporfondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

# Interruzioni

## Appofondimento delle interruzioni

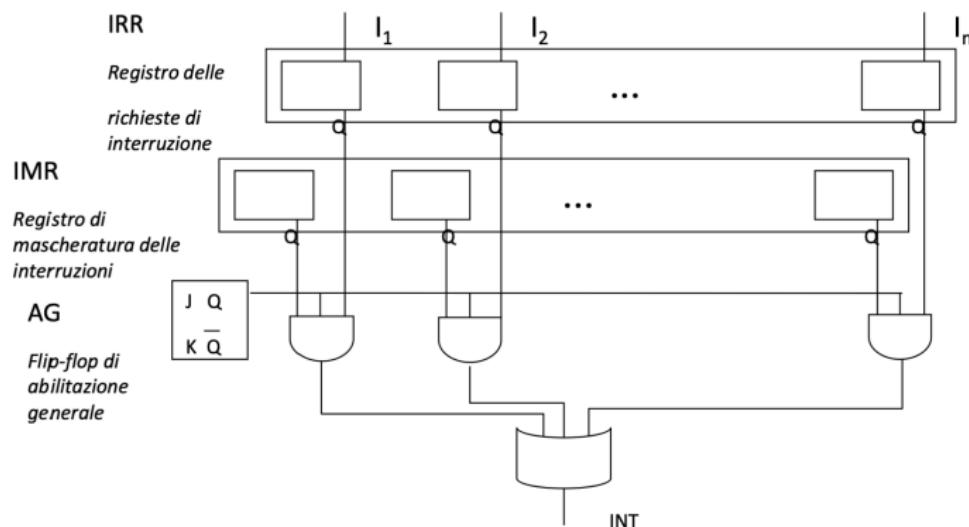
1. **Interruzioni hardware (asincrone)**: per terminazione di un trasferimento dati o per condizione di errore rilevata dalle o nelle periferiche
2. **Interruzioni software (sincrone)**: eccezioni o trap (es. divisione per zero), programmate o supervisory call (per utilizzare funzioni del SO)

# Sistema delle interruzioni

## Appofondimento delle interruzioni

A ciascuna causa di interruzione è associata un'azione che verrà effettuata dal programma di risposta alle interruzioni.

Una causa **non produce di per se un'interruzione** ma solo una **richiesta di interruzione**. Affinché la richiesta di interruzione segua effettivamente un'interruzione è necessario che **la causa di interruzione sia abilitata**.



# Sistema delle interruzioni

## Appofondimento delle interruzioni

L'interruzione "i" si manifesta se:

- $AG = 1$  (il sistema di interruzione è abilitato)
- $IRRi = 1$  (si è presentata la causa di interruzione "li")
- $IMRi = 1$  (l'interruzione "i" è abilitata nel registro di maschera)

# Processo delle interruzioni

## Appofondimento delle interruzioni

Al verificarsi di un'interruzione occorre

- 1. salvare tutte le informazioni necessarie per la ripresa del programma interrotto**

l'Hw provvede in generale a salvare PC e PSW, mentre i registri generali vengono tipicamente salvati in software. È necessario che il salvataggio dei registri avvenga ad interrupt disabilitati.

- 2. individuare la causa dell'interruzione**

l'Hw può trasferire il controllo sempre al medesimo programma di risposta, che provvederà quindi ad individuare la causa dell'interruzione tramite *skip chain*

- 3. eseguire le azioni richieste (servizio dell'interruzione)**

Durante il servizio dell'interruzione il sistema delle interruzioni può essere riabilitato (AG), eventualmente selettivamente (IMR)

- 4. ripristinare lo stato del programma interrotto e riavviato**

Il ripristino dei registri deve avvenire ad interrupt disabilitati.

Un'apposita istruzione di ritorno dell'interrupt (RTI) ripristina i registri.

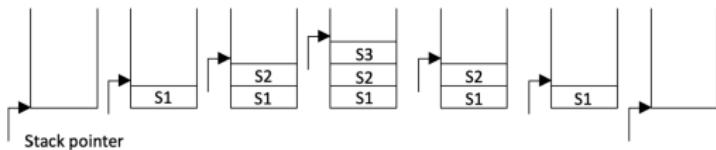
# Livelli di priorità

## Appofondimento delle interruzioni

I diversi tipi di fonti di interruzione possono suggerire una gestione con diversi livelli di priorità, se all'atto del ritorno esistono più richieste pendenti si serve di quella di livello più elevato.

# Salvataggio dello stato tramite stack

## Approfondimento delle interruzioni



S1: stato del processore salvato all'arrivo della prima interruzione

S2: stato del processore relativo all'esecuzione del programma di risposta della prima interruzione salvato all'arrivo della seconda interruzione

S3: stato del processore relativo all'esecuzione del programma di risposta della seconda interruzione salvato all'arrivo della terza interruzione

Programma di risposta ad interruzione di livello L:

- salvo lo stato del processore nello stack e modifica SP
- abilita interruzioni di livello  $K > L$
- esegue programma di risposta
- ripristina lo stato del processore modificando SP

# Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Apporfondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

# Servizi del sistema operativo

## Servizi e organizzazione del SO

Forniscono un ambiente per l'esecuzione di programmi e servizi a programmi e utenti:

- **Interfaccia utente**
- **Esecuzione del programma**
- **Operazioni di I/O**
- **Manipolazione del file system**
- **Comunicazioni**
- **Rilevamento di errori**

Permette anche il funzionamento efficiente del sistema stesso attraverso la condivisione delle risorse

- **Allocazione delle risorse:** quando più utenti o più job vengono eseguiti simultaneamente
- **Contabilità:** per tenere traccia di quali utenti utilizzano la quantità e il tipo di risorsa del computer
- **Protezione e sicurezza**

# Interfaccia utente del SO - CLI

## Servizi e organizzazione del SO

La CLI consente l'immissione diretta di comandi, a volte implementato nel **kernel**, a volta da un **programma di sistema**.  
A volte sono implementate più versioni: **shell**

# Interfaccia utente del SO - GUI

## Servizi e organizzazione del SO

Interfaccia desktop con metafora facile da usare.

Molti sistemi includono sia **CLI che GUI**:

- Microsoft Windows è un'interfaccia GUI con una shell di comando CLI
- Apple Mac OS X è una GUI "Aqua" con kernel UNIX

# Chiamate di sistema (SysCall)

## Servizi e organizzazione del SO

Interfaccia di programmazione per i servizi forniti dal SO, tipicamente scritta in C o C++.

I programmi vi accedono tramite un'**interfaccia di programmazione delle applicazioni (API)** di alto livello.

Le **API** più comuni sono **Win32 API**, **POSIX API** (per tutte le versioni di UNIX)

# Esempio di API standard

## Servizi e organizzazione del SO

Come API standard, si consideri la funzione

`read()`

si ottiene dalla pagina *man* invocando il comando

`man read`

# Implementazione della chiamata di sistema

## Servizi e organizzazione del SO

In genere, a ciascuna SysCall è associato un numero (l'interfaccia delle chiamate di sistema mantiene una tabella indicizzata in base a questi numeri).

L'interfaccia SysCall invoca la chiamata di sistema prevista nel kernel del SO e restituisce lo stato della chiamata di sistema e gli eventuali valori di ritorno.

Il chiamante non deve sapere nulla di come viene implementata la SysCall.

# Passaggio dei parametri della SysCall

## Servizi e organizzazione del SO

Spesso sono necessarie più informazioni della semplice identità della chiamata di sistema desiderata.

Tre metodi generali utilizzati per passare parametri al SO

- Passarli nei registri (più semplice)
- Parametri memorizzati in un blocco, o tabella, in memoria
- Parametri messi sullo stack dal programma

# Tipi di System Call

## Servizi e organizzazione del SO

- **Controllo del processo**
  - creare un processo, terminare un processo
  - caricare, eseguire
  - allocare e liberare memoria
  - debugger per determinare i bug, esecuzione a passo singolo
  - blocchi per gestire l'accesso ai dati condivisi
- **Gestione dei file**
- **Gestione dei dispositivi**
  - richiedere un dispositivo, rilasciare un dispositivo
- leggere, scrivere, riposizionare
- **Manutenzione delle informazioni**
- **Comunicazione**
  - inviare, ricevere messaggi nel **modello a scambio di messaggi**
  - creare e accedere a regioni di memoria nel **modello a memoria condivisa**
- **Protezione**

# Esempio: MS-DOS

## Servizi e organizzazione del SO

- Lavoro a task singolo
- Shell invocata all'avvio del sistema
- Metodo semplice per eseguire un programma
- Singolo spazio di memoria
- Uscita dal programma → viene ricaricato lo shell

# Esempio: FreeBSD

## Servizi e organizzazione del SO

- Variante UNIX
- Multitasking
- Lo shell esegue la chiamata di sistema *fork()* per creare un processo
- Il processo esce con
  - *codice* = 0 - nessun errore
  - *codice* > 0 - codice errore

# Servizi di sistema

## Servizi e organizzazione del SO

I programmi di sistema forniscono un comodo **ambiente per lo sviluppo e l'esecuzione dei programmi**. Possono essere suddivisi in:

- Manipolazione dei file
- Informazioni di stato a volte memorizzate in un file
- Supporto ai linguaggi di programmazione
- Caricamento ed esecuzione dei programmi
- Comunicazioni
- Servizi di background: avvio a tempo di boot, servizi come il controllo del disco
- Programmi applicativi

# Linker e loader

## Servizi e organizzazione del SO

Il **linker** combina i file oggetto in singoli file eseguibili e il **loader** li carica in memoria

# Programmi e SO

## Servizi e organizzazione del SO

**Application Binary Interface (ABI)** è un'architettura equivalente all'API che definisce come differenti componenti del codice possono interfacciarsi ad un dato sistema operativo o ad una data architettura

# Struttura del sistema operativo

## Servizi e organizzazione del SO

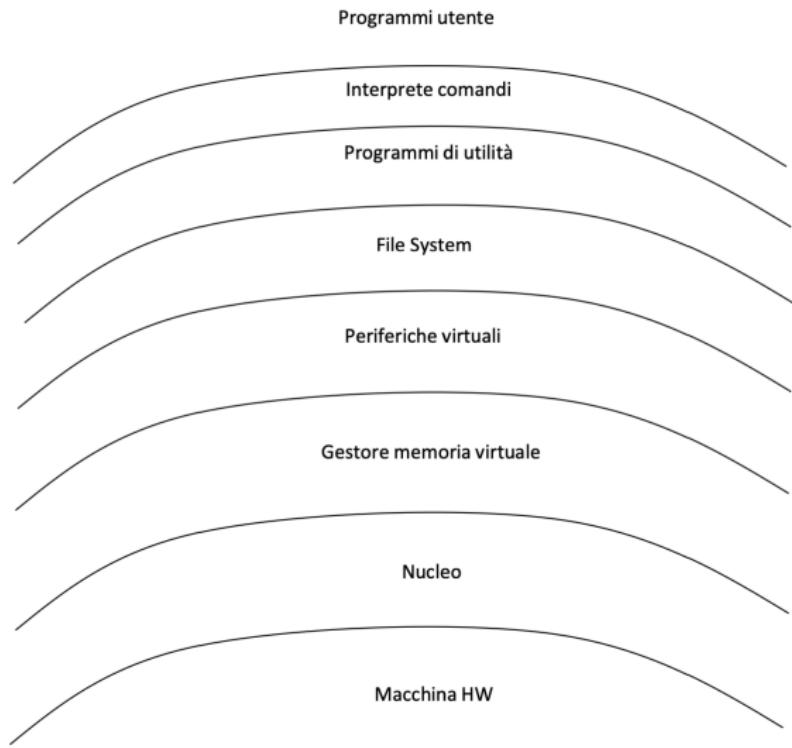
**Sistema a livelli:** il livello più **interno** è l'Hw, quello più **esterno** è l'**interfaccia utente**

Ci sono vari modi per strutturare un SO:

- Struttura semplice **MS-DOS**
- Più complessa **UNIX**
- A strati **un'astrazione**
- Microkernel **Mach**

# Struttura a strati

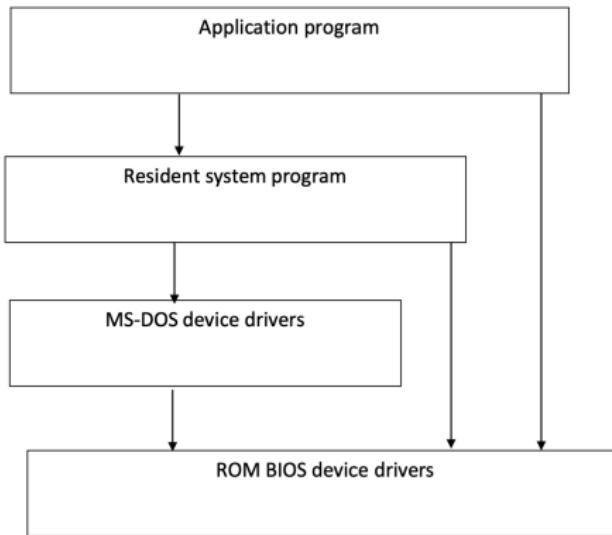
## Servizi e organizzazione del SO



# Struttura a livelli di MS-DOS

## Servizi e organizzazione del SO

- MS-DOS - scritto per fornire la maggior parte delle funzionalità nel minor spazio possibile
  - Non suddiviso in moduli
  - Sebbene MS-DOS abbia una certa struttura, le sue interfacce e i suoi livelli di funzionalità non sono ben separati



# Struttura non semplice - UNIX

## Servizi e organizzazione del SO

UNIX - limitato dalla funzionalità Hw, il sistema operativo UNIX originale aveva una struttura limitata.

UNIX è composto da due parti separabili: **programmi di sistema, kernel** (ovvero tutto quello che trova sotto l'interfaccia delle chiamate di sistema e al di sopra dell'Hw fisico)

## Struttura monolitica

(Users)			
Shells and commands Compilers and interpreters System libraries			
<i>System-call interface to the kernel</i>			
Signals Terminal handling Character I/O system Terminal drivers			
File system swapping block I/O system disk and tape drivers	CPU scheduling Page replacement Demand paging Virtual memory		
<i>Kernel interface to the hardware</i>			
Terminal controllers terminals	Device controllers Disk and tapes	memory controllers physical memory	

*Con i moduli del kernel caricabili dinamicamente, supportati dai moderni SO incluso Linux, si può avere un kernel è costituito da un insieme di componenti di base, integrati da funzionalità aggiunte dinamicamente all'avvio o in esecuzione per mezzo di moduli.*

113

Sposta il più possibile dal kernel allo spazio comune.

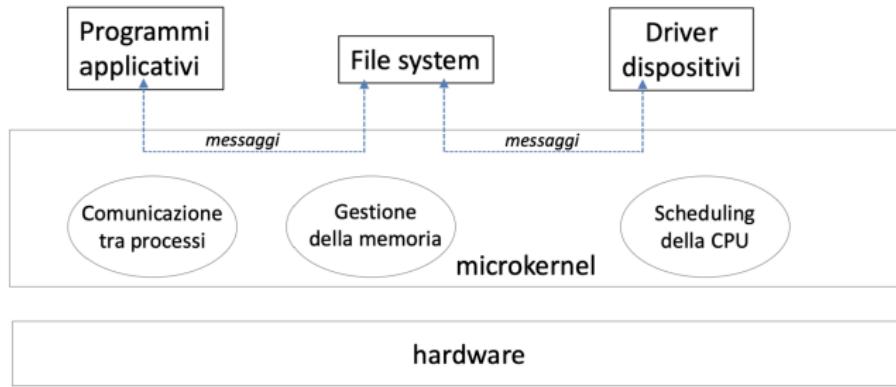
Esempio di **microkernel per Mach**: la comunicazione tra i moduli utente avviene tramite il passaggio di messaggi

**Vantaggi**: estensione più semplice del SO, più affidabile, più sicuro

**Svantaggi**: sovraccarico di prestazioni della comunicazione tra spazio utente e spazio kernel

# Architettura a microkernel

## Servizi e organizzazione del SO



*Esempi di microkernel: Darwin (componente kernel dei SO dei sistemi operativi macOS e iOS) contiene anche il microkernel Mach, QNX Neutrino alla base del SO real-time QNX per sistemi embedded*

# Moduli

## Servizi e organizzazione del SO

Molti sistemi operativi moderni implementano moduli del kernel caricabili (approccio orientato agli oggetti, ogni componente centrale è separato, ciascuno parla con gli altri attraverso interfacce note).

Nel complesso è simile ai livelli ma con una maggiore flessibilità

La maggior parte dei SO moderni non sono un unico modello puro, l'ibrido combina più approcci per rispondere alle esigenze delle prestazioni. I kernel di Linux sono nello spazio degli indirizzi del kernel, quindi monolitici, ma anche modulari per il caricamento dinamico delle funzionalità.

Windows per lo più monolitico, più microkernel per le diverse personalità dei sottosistemi.

Sistema operativo mobile di Apple, strutturato su Mac OS X con funzionalità aggiuntive.

API Cocoa Touch Objective-C per lo sviluppo delle applicazioni.

Livelli di servizi multimediali per grafica, video e audio.

Sviluppato principalmente da Google ed ha uno stack simile a quello di iOS (open source).

Basato sul kernel Linux ma modificato, l'ambiente di runtime include un set di librerie di base e la macchina virtuale Dalvik

# Debug del sistema operativo

## Servizi e organizzazione del SO

**Il debug consiste nel trovare e correggere gli errori o i bug.**

I SO generano file di log contenenti informazioni sugli errori.

**Il fallimento di un'app può generare un file di core dump che cattura la memoria del processo.**

**Il fallimento del SO può generare un file di crash dump contenente la memoria del kernel**

Lo strumento **DTrace** in Solaris, FreeBSD e MacOS X consente la strumentazione live sui sistemi di produzione.

**Le sonde si attivano quando il codice viene eseguito all'interno di un provider**, catturando i dati di stato e inviandoli ai consumatori di tali sonde.

# Generazione del sistema operativo

## Servizi e organizzazione del SO

- I sistemi operativi sono progettati per funzionare su una qualsiasi classe di macchine
- Il programma **SYGEN** ottiene informazioni sulla configurazione specifica del sistema Hw

# Avvio del sistema

## Servizi e organizzazione del SO

Quando l'alimentazione viene inizializzata sul sistema, l'esecuzione inizia da una posizione di memoria fissa.

Il SO deve essere reso disponibile all'Hw in modo che quest'ultimo possa avviarlo.

Il comune **bootstrap loader** permette di selezionare il kernel da più dischi, versioni, opzioni del kernel.

# Struttura del SO

## Servizi e organizzazione del SO

La stratificazione più opportuna può risultare non evidente, dipende dall'evoluzione tecnologica dell'Hw.

**Sistema a macchine virtuali** (VM IBM): usando lo scheduling della CPU e la tecnica della memoria virtuale si possono creare macchine virtuali, una per ogni processo

Realizzazione in linguaggi ad alto livello.

**Nucleo o kernel**: mette a disposizione le SysCall ai programmi di sistema ed applicativi

# Nucleo di un SO

## Servizi e organizzazione del SO

Fornisce un meccanismo per la creazione e la distruzione dei processi.  
Provvede allo **scheduling della CPU**, alla **gestione della memoria** e dei  
**dispositivi di I/O**.

Fornisce strumenti per la sincronizzazione

# Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Appofondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

# Macchine virtuali

## Virtualizzazione

Creano un'illusione di processi multipli, ciascuno in esecuzione sul suo processore privato e con la propria memoria virtuale privata, messa a disposizione dal proprio kernel del SO, che può essere diverso per processi diversi

# Virtualizzazione

## Virtualizzazione

Dato un sistema caratterizzato da un insieme di risorse, virtualizzare il sistema significa presentare all'utilizzatore una visione delle risorse del sistema diversa da quella reale.

**Obiettivo:** disaccoppiare il comportamento delle risorse Hw e Sw di un sistema di elaborazione, così come viste dall'utente, dalla loro realizzazione fisica

# Esempi di virtualizzazione

## Virtualizzazione

**Astrazione:** in generale un oggetto astratto (risorse virtuale) è la rappresentazione semplificata di un oggetto (risorsa fisica)

**Linguaggio di programmazione:** la capacità di portare lo stesso programma su architetture diverse è possibile grazie alla **definizione di un VM** in grado di **interpretare ed eseguire ogni istruzione del linguaggio**

**Virtualizzazione a livello di processo:** i sistemi multitasking permettono la contemporanea esecuzione di più processi, ognuno dei quali dispone di una VM dedicata

# Sistemi operativi per la virtualizzazione

## Virtualizzazione

La macchina fisica viene trasformata in  $n$  interfacce (VM), ognuna delle quali è una replica della macchina fisica.

Su ogni macchina virtuale è possibile installare ed eseguire un sistema operativo: **VM monitor**

# Virtualizzazione di sistema

## Virtualizzazione

Il disaccoppiamento è realizzato da un componente chiamato VM monitor il cui compito è consentire la condivisione da parte di più macchine virtuali di una singola piattaforma Hw

Il VMM è il mediatore unico nelle interazioni tra le macchine virtuali e l'Hw sottostante, che garantisce

- **Isolamento tra le VM**
- **Stabilità del sistema**

# VMM di sistema vs VMM ospitato

## Virtualizzazione

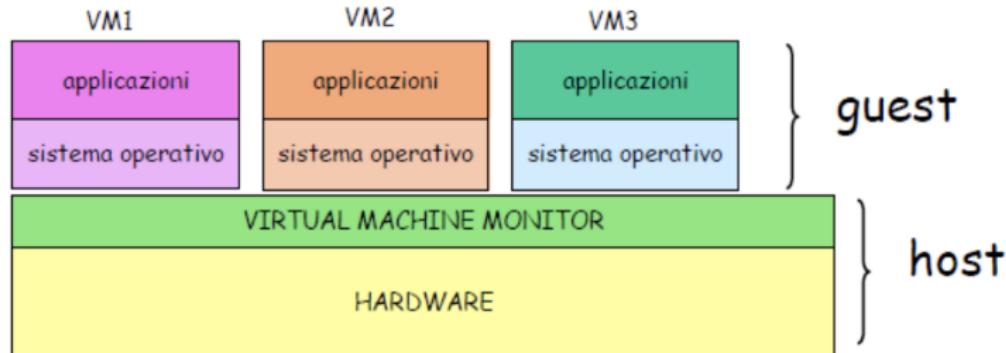
**VMM di sistema:** le funzionalità di virtualizzazione vengono integrate in un SO leggero, costituendo un unico sistema posto direttamente sopra l'Hw dell'elaboratore.

È necessario corredare il VMM di tutti i driver necessari per pilotare le periferiche

# VMM di sistema

## Virtualizzazione

- **Host:** piattaforma di base sulla quale si realizzano macchine virtuali
- **Guest:** la VM



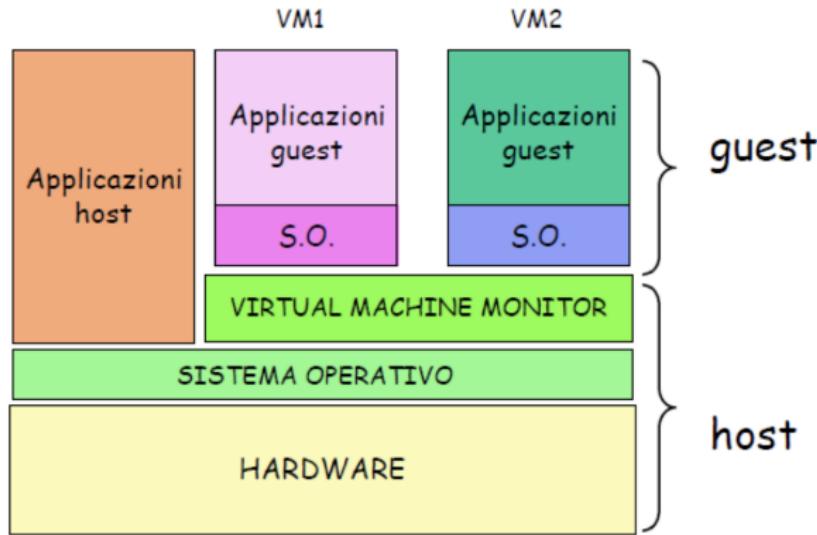
## VMM di Sistema

# VMM ospitato

## Virtualizzazione

Il VMM viene installato come un'app sopra un sistema operativo esistente, che opera nello spazio utente e accede all'Hw tramite le SysCall del SO su cui viene installato.

Prodotti: User Mode Linux, VMWare, Microsoft Virtual Server



# Emulazione vs Virtualizzazione

## Virtualizzazione

- **Emulazione**

- eseguire app (o SO) compilate per un'architettura su un'altra
- uno strato sw che emula le funzionalità dell'architettura

- **Virtualizzazione**

- definizione di contesti di esecuzione multipli su di un singolo processore, partizionando le risorse

# Vantaggi della virtualizzazione

## Virtualizzazione

**Uno o più SO sulla stessa macchina fisica  
Isolamento dell'ambiente di esecuzione**

# Vantaggi della virtualizzazione

## Virtualizzazione

**Consolidamento Hw:** possibilità di concentrare più macchine su un'unica architettura Hw per un utilizzo efficiente dell'Hw

**Gestione facilitata della macchine:** migrazione a caldo di macchine virtuali tra macchine fisiche, ovvero la possibilità di manutenzione Hw senza interrompere i servizi forniti dalle macchine virtuali, disaster recovery

# Virtualizzazione a livello del SO

## Virtualizzazione

Una modalità di virtualizzazione in cui il kernel consente l'esistenza di molteplici istanze isolate nello spazio utente.

Consentono di migliorare la sicurezza, l'indipendenza dall'Hw e la gestione delle risorse.

Rispetto alla **virtualizzazione piena (basata su VMM)** l'overhead è **inferiore ma è anche inferiore la flessibilità**

# Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Appofondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

# Concetto di processo

## Processi

Un SO esegue una serie di programmi:

- Sistema batch - lavori (job)
- Sistema time-sharing - programmi o attività dell'utente

## Processo

È un **programma in esecuzione**, l'esecuzione del processo deve procedere in modo sequenziale

## Parti multiple

- Codice del programma
- Attività corrente
- Stack che contiene dati temporanei
- Sezione dati
- Heap che contiene la memoria allocata

# Algoritmo, Programma, Processo

## Processi

### Algoritmo

Procedimento logico che deve essere eseguito per risolvere il problema in esame

### Programma

Descrizione dell'algoritmo tramite un opportuno formalismo (linguaggio di programmazione) che rende possibile l'esecuzione dell'algoritmo da parte di un particolare elaboratore

### Processo (sequenziale)

Sequenza di eventi cui dà luogo un elaboratore quando opera sotto il controllo di un particolare programma

# Stato del processo

## Processi

Per tenere traccia e gestire correttamente i programmi in memoria principale, il SO deve associare esplicitamente ad essi un concetto di **stato del processo**

- **In esecuzione**
- **Pronto per l'esecuzione se dispone di tutte le risorse e le condizioni logiche per eseguire ma non dispone del processore**
- **In attesa se non dispone delle risorse e delle condizioni logiche per essere eseguito**

Durante l'esecuzione, un processo cambia stato

- **Nuovo**: il processo viene creato
- **Running**: le istruzioni vengono eseguite
- **Waiting**: il processo è in attesa di qualche evento
- **Ready**: il processo è in attesa di essere assegnato ad un processore
- **Terminated**: il processo ha terminato l'esecuzione

# Gestione dei processi

## Processi

La possibilità che la CPU venga commutata in un qualsiasi istante da un processo ad un altro rende indispensabile ad ogni commutazione salvare tutte le informazioni contenute nei registri della CPU e relative al processo che è stato sospeso

### Descrittore di processo o PCB

Area di memoria, mantenuta all'interno dell'area protetta del SO, associata al processo e contenente tutte le informazioni proprie del processo

# Process Control Block (PCB)

## Processi

### Informazioni associate a ciascun processo

- Stato del processo
- Program counter
- Registri della CPU
- Informazioni sulla programmazione della CPU
- Informazioni sulla gestione della memoria
- Informazioni di accounting: CPU utilizzata, tempo di clock
- Informazioni sullo stato dell'I/O

Si consideri la **possibilità di avere più contatori di programma per processo**, quindi più **thread di controllo**. Deve quindi esserci una memoria per i dettagli dei thread, quindi dei contatori multipli nel PCB

# Scheduling dei processi

## Processi

Massimizzare l'uso della CPU, passare rapidamente i processi alla CPU per la condivisione del tempo.

Lo **scheduler dei processi** seleziona tra i processi disponibili per la successiva esecuzione sulla CPU

Mantiene le code di scheduling dei processi

- **Ready queue:** insieme di tutti i processi del SO che risiedono nella memoria principale
- **Wait queues:** insieme dei processi in attesa di un evento

# Commutazione di contesto

## Processi

Quando la CPU passa a un altro processo, il sistema deve salvare lo stato del vecchio processo e caricare lo stato salvato per il nuovo processo tramite un context switch.

**Contesto di un processo rappresentato nel PCB.**

Il tempo di commutazione del contesto è un **overhead**; il sistema non svolge alcun lavoro utile durante la commutazione.

Il tempo necessario per lo switch dipende dal supporto Hw

# Multitasking nei sistemi mobili

## Processi

A causa della superficie dello schermo e dei limiti dell'interfaccia utente, iOS prevede un:

- singolo processo in **foreground** primo piano, controllato dalla GUI
- processi multipli in **background**

# Tipi di scheduler

## Processi

- **Scheduler a breve termine (o della CPU)**: seleziona quale processo deve essere eseguito successivamente e alloca la CPU, a volte è l'unico scheduler del sistema
- **Scheduler a lungo termine (o job scheduler)**: seleziona quali processi devono essere messi in coda di attesa
- I processi possono essere descritti come:
  - processo I/O-bound: passa più tempo a fare I/O che calcoli
  - Processo CPU-bound: passa più tempo a eseguire i calcoli

# Creazione del process

## Processi

I processi padre (**parent**) creano processi figli (**children**) che a loro volta creano altri processi, formando un albero di processi.

In generale i processi vengono identificati e gestiti tramite un **identificatore di processi (PID)**.

### Opzioni di condivisione delle risorse:

- padre e figli condividono tutte le risorse
- i figli condividono un sottoinsieme delle risorse del padre
- padre e figlio non condividono alcuna risorsa

### Opzioni di esecuzione

- il genitore e i figli eseguono simultaneamente
- il genitore attende che i figli terminino

## Spazio degli indirizzi:

- i figli duplicano quello del genitore
- il figlio ha un programma caricato al suo interno

## Esempi UNIX

- la chiamata di sistema  
`fork()`  
crea un nuovo processo
- La chiamata di sistema  
`exec()`  
è usata dopo una  
`fork()`  
per sostituire lo spazio di memoria del processo con un nuovo  
programma

# Terminazione del processo

## Processi

Il processo esegue l'ultima istruzione e poi chiede al SO di eliminarlo utilizzando la chiamata di sistema

`exit()`

Il genitore può terminare l'esecuzione dei processi figli:

- il processo figlio ha superato le risorse allocate
- il compito assegnato al figlio non è più necessario
- il processo padre sta terminando e il SO non consente a un figlio di continuare se il suo genitore termina

Alcuni sistemi operativi non consentono ai figli di esistere se il loro genitore è terminato. Se un processo termine, **anche tutti i suoi figli devono essere terminati**.

In UNIX il processo genitore può attendere la terminazione di un processo figlio utilizzando la chiamata di sistema

`wait()`

Se nessun genitore è in attesa (non ha invocato `wait()`), il processo è uno **zombie**.

Se il genitore è terminato senza invocare `wait()`, il processo è **orfano**

# Architettura multiprocesso - Browser Chrome

## Processi

Molti browser web venivano eseguiti come singoli processi, il **browser chrome** è multiprocesso con 3 diversi tipi di processi:

- Il processo del **browser** gestisce l'interfaccia utente
- Il processo **render** esegue il rendering delle pagine web
- Processo dei plug-in per ogni tipo di plug-in

# Comunicazione tra processi

## Processi

I processi all'interno di un sistema possono essere **indipendenti** o **cooperanti**.

I processi **cooperanti** necessitano di **comunicazione interprocesso (IPC)**  
Due **modelli** di IPC:

- Memoria condivisa
- Passaggio di messaggi

# Modelli di interazione tra processi

## Processi

Modello ad ambiente globale o modello a memoria comune, modello ad ambiente locale o modello a scambio di messaggi.

Tipi di interazione tra processi:

- **Competizione**
- **Cooperazione**

# Modello ad ambiente globale

## Processi

Il sistema è visto come un insieme di processi e oggetti (risorse)

Il modello ad ambiente globale rappresenta la natura astrazione di un sistema multiprogrammazione costituito da uno o più processori che hanno accesso ad una memoria comune.

Ad ogni processore può essere eventualmente associata una **memoria privata**, ma ogni interazione avviene tramite oggetti contenuti nella **memoria comune**

# Modello a scambio di messaggi

## Processi

Il sistema è visto come un insieme di processi ciascuno operante in un ambiente locale non accessibile direttamente a nessun altro processo. Ogni forma di interazione tra processi (**comunicazione, sincronizzazione**) avviene tramite scambio di messaggi  
Modello a scambi di messaggi rappresenta la naturale astrazione di un sistema privo di memoria comune, in cui a ciascun processore è associata una memoria privata

# Processi cooperanti

## Processi

Un processo indipendente non può influenzare o essere influenzato dall'esecuzione di un altro processo

Processi cooperanti possono influenzare o essere influenzati dall'esecuzione di un altro processo.

Vantaggi della cooperazione tra processi

- Condivisione delle informazioni
- Velocità di calcolo
- Modularità
- Convenienza

# Problema produttore-consumatore

## Processi

Paradigma dei processi cooperanti, il processo produttore produce informazioni che vengono consumate da un processo consumatore

- **Unbounded-buffer** non pone limiti pratici alla dimensione del buffer
- **Bounded-buffer** presuppone l'esistenza di una dimensione fissa del buffer

# Comunicazione tra processi (IPC) - Memoria condivisa

## Processi

Un'area di memoria condivisa tra i processi che desiderano comunicare. La comunicazione è sotto il controllo dei processi degli utenti e non del SO. Il problema principale è fornire un meccanismo che permetta ai processi utente di sincronizzare le loro azioni quando accedono alla memoria condivisa.

# Comunicazione tra processi - Scambio di messaggi

## Processi

Meccanismo che consente ai processi di comunicare e sincronizzare le loro azioni

Sistema di messaggi - i processi comunicano tra loro senza ricorrere a variabili condivise.

La struttura IPC fornisce due operazioni:

- **send(messaggio)**
- **receive(messaggio)**

## Implementazione del collegamento di comunicazione

- **Fisico**

- Memoria condivisa
- Bus hardware
- rete

- **Logico**

- Diretto o indiretto
- Sincrono o asincorno
- Buffering automatico o esplicito

# Costrutti linguistici per il modello a scambio di messaggi

## Processi

### Classificazione

- **Designazione** dei processi sorgente e destinatario di ogni comunicazione
  - designazione diretta o esplicita
    - simmetrica
    - asimmetrica
  - designazione indiretta o globale
    - mailbox
    - porte
- **Tipo di sincronizzazione** tra i processi comunicanti
  - sincrona
  - asincrona

# Primitive con designazione esplicita

## Processi

**send** <expression\_list> **to** <destination\_designator>

**receive** <variable\_list> **from** <source\_designator>

- L'esecuzione della *send* determina il contenuto del messaggio mediante la valutazione delle espressioni in <expression\_list>.
- <destination\_designator> dà al programmatore il controllo su dove inviare il messaggio.
- L'esecuzione della *receive* determina l'assegnamento dei valori contenuti nel messaggio alle variabili in <variable\_list>, e la successiva distruzione del messaggio.
- <source\_designator> dà al programmatore il controllo sull'origine dei messaggi.

# Designazione esplicita

## Processi

- La coppia (*<destination\_designator>*, *<source\_designator>*) definisce un *canale di comunicazione*.

- **Schema simmetrico**

I processi si nominano *esplicitamente* (*direct naming*) l'un l'altro:

*send message to P2*  
*receive message from P1*

- P1 invia un messaggio che può essere ricevuto solo da P2
- P2 riceve un messaggio che può provenire solo da P1
- Semplice da realizzare e da utilizzare: un processo può controllare in maniera selettiva gli intervalli di tempo in cui riceve messaggi dagli altri processi.
- E' usato nei modelli del tipo *pipeline*: collezione di processi concorrenti in cui l'output di un processo costituisce l'input di un altro. Il sistema è concepito in termini di flusso di informazione.

## Direct naming: schema asimmetrico

### Processi

Il mittente nomina **esplicitamente** il destinatario, mentre questi, al contrario, non esprime il nome del processo con cui desidera comunicare. La **designazione asimmetrica** facilita l'organizzazione dell'interazione tra processi secondo il paradigma **client-server**, in cui un processo gestore di una risorsa (server) riceve richieste da più processi client.

# Modello client-server

## Processi

Corrisponde all'uso di un **processo come gestore di una risorsa**

**Schema da-molti-a-uno:** i processi client inviano richieste non ad un particolare server, ma ad uno qualunque scelto tra un insieme di **server equivalenti**.

È difficile realizzazione con designazione diretta. Richiede di passare ad una **designazione indiretta o globale (mailbox)**

# Modello client-server e naming

## Processi

Il **direct naming** è in generale poco adatto al modello client-server.  
In presenza di più **client** la *receive* di un server dovrebbe consentire la ricezione di un messaggio da un qualsiasi client.  
In presenza di più **server equivalenti** la *send* di un clinet dovrebbe produrre un messaggio che possa essere ricevuto da un qualsiasi server.  
Occorre uno schema più sofisticato per la definizione dei canali di comunicazione: **designazione globale o indiretta** che fa uso di nomi globali detti **mailbox**

# Uso delle mailbox

## Processi

La **mailbox** consente in modo immediato la **programmazione delle interazioni client-server** anche nel caso da-molti-a-molti.

I client eseguono una *send* sulla mailbox associata al servizio, i server una *receive*.

La realizzazione delle mailbox in ambiente distribuito presenta problemi di **natura realizzativa**. Il supporto a tempo di esecuzione del linguaggio deve garantire che:

- un messaggio di richiesta indirizzato ad una mailbox è inviato a tutti i processi che possono eseguire una *receive* su di essa
- non appena il messaggio è ricevuto da un processo, esso deve diventare indisponibile per tutti gli altri server

Sono mailbox il cui nome può comparire solamente in un processo come *<source-designator>* in uno statement di *receive*.

Un processo può selezionare i messaggi che desidera ricevere attraverso l'uso di **porte distinte**

# Designazione (naming)

## Processi

- **Direct naming simmetrico:** comunicazione *one to one*
- **Direct naming asimmetrico e port naming:** comunicazione *many to one*
- **Global naming:** comunicazione *many to many*

### Naming statico:

- Impedisce ad un programma di comunicare tramite canali non noti a tempo di compilazione
- il potenziale accesso di un programma ad un canale deve essere assicurato fin dall'inizio

### Naming dinamico

- uno schema statico di base di designazione dei canali viene arricchito mediante variabili per la designazione di sorgente o destinazione

# Sincronizzazione - send asincrona

## Processi

Il processo mittente **continua la sua esecuzione** immediatamente dopo che il messaggio è stato inviato.

Il messaggio ricevuto contiene informazioni che non possono essere associate allo stato attuale del mittente

L'interazione viene definita come **scambio di messaggi asincrono**.

In analogia con il meccanismo semaforico, la **send asincrona** è caratterizzata da:

- **flessibilità d'uso**
- **carenza espressiva**

Richiede, a livello realizzativo, un buffer di capacità limitata. Si può ovviare **modificandone la semantica**

1. un processo mittente si blocca qualora la coda dei messaggi sia piena
2. la primitiva send, in caso di coda piena, solleva un'**eccezione** che viene notificata al processo mittente

# Sincronizzazione - send sincrona

## Processi

Il processo mittente si blocca in attesa che il messaggio sia stato ricevuto.  
Un messaggio ricevuto contiene informazioni corrispondenti allo **stato attuale del processo mittente**.

L'invio di un messaggio costituisce un **punto di sincronizzazione** sia per il mittente che per il destinatario.

L'interazione viene definita come **scambio di messaggi sincrono**

## Send di tipo "chiamata di procedura remota"

### Processi

"Rendez-vous" esteso: il processo mittente **rimane in attesa** fino a che il ricevimento non ha terminato di svolgere l'azione richiesta.

Un processo client "chiama" una procedura eseguita da un processo server su una macchina potenzialmente remota.

Il nome della procedura remota identifica un **processo in caso di direct naming**.

I programmi risultano più facilmente verificabili grazie alla localizzazione dei vincoli di sincronizzazione

## Receive

### Processi

Normalmente è bloccante se non vi sono messaggi sul canale. Costituisce un **punto di sincronizzazione** per il processo ricevente.

Si ricorre ad una primitiva che verifica lo stato del canale e restituisce un messaggio se esso è presente, ovvero un'indicazione di canale vuoto (**receive non bloccante**).

Inconveniente: per l'attesa di messaggi da specifici canali occore fare uso di cicli di attesa attiva

# Chiamata di procedura remota (RPC)

## Processi

Consente di esprimere a più alto livello e in maniera più sintetica le interazioni di tipo client-server.

Service è il nome di un canale:

- se la designazione è diretta, service indica il processo servitore
- se la designazione è indiretta (porte o mailbox), service indica il tipo di servizio richiesto

Specifica lato server:

1. come procedura dichiarate separatamente
2. come statement collocato in un punto qualunque di un processo

# RPC: specifica lato server

## Processi

La procedura remota viene dichiarata come una procedura in un linguaggio sequenziale e realizzata come un processo server che attende la ricezione di un messaggio, esegue il corpo e trasmette un messaggio di risposta.

Può essere realizzata come un **singolo processo** che esegue le richieste una alla volta in modo sequenziale, oppure con la creazione di un **nuovo processo per ogni chiamata**. Le varie istanze sono eseguite concorrentemente, e potranno eventualmente doversi sincronizzare tra loro. La RPC è uno **statement**, e come tale può essere collocato in un punto qualunque del processo server.

L'esecuzione delle **accept** sospende il servitore fino all'arrivo di un messaggio corrispondente alla **call** di servizio.

L'esecuzione del corpo può fare uso dei valori dei parametri e di tutte le variabili accessibili dallo scope dello statement.

Al termine viene trasmesso il messaggio di risposta al processo chiamante, dopo di che il processo server **continua la propria esecuzione**.

# Uso delle accept

## Processi

Quando il lato server è specificato con una **accept**, la RPC viene chiamata *extended rendez-vous*: client e server si "incontrano" per la durata dell'esecuzione del corpo della accept per poi proseguire separatamente.

### Vantaggi:

- il server può fornire più tipi di servizi
- il server può decidere quando servire le call dei client
- le istruzioni di accept possono essere alternate o innestate
- vi possono essere più accept di chiamate allo stesso servizio con diverso body

# Uso delle RPC

## Processi

Lo schema di comunicazione realizzato dal meccanismo della chiamata a procedura remota è di tipo asimmetrico e da molto ad uno.

L'accoppiamento tra una chiamata priva di parametri ed una accept priva di corpo rappresenta la trasmissione ed il relativo riconoscimento di un segnale di sincronizzazione.

Problemi della RPC:

- interazioni non client-server
- perdita di messaggi nelle architetture distribuite

# Esempi di sistemi IPC - Mach

## Processi

La comunicazione di Mach è basata su messaggi

- Anche le chiamate di sistema sono messaggi
- Ogni task riceve due mailbox alla creazione: Kernel e Notify
- Le mailbox necessarie per la comunicazione sono create tramite `port_allocate()`
- L'invio e la ricezione sono flessibili, ad esempio quattro opzioni se la casella di posta è piena:
  - Attendere indefinitamente
  - Attendere al massimo n millisecondi
  - Restituire immediatamente
  - Mettere temporaneamente in cache un messaggio

# Esempi di sistemi IPC - Windows

## Processi

Centralità del **message passing** tramite una struttura avanzata di chiamata di procedura locale (LPC).

Funziona solo tra processi sullo stesso sistema, utilizza porte per stabilire e mantenere i canali di comunicazione.

La comunicazione funziona come segue:

- Il client apre un handle dell'oggetto porta di connessione del sottosistema
- Il cliente invia una richiesta di connessione
- Il server crea due porte di comunicazione private e restituisce al cliente l'handle di una di esse
- Il client e il server utilizzano l'handle della porta corrispondente per inviare messaggi o callback e per ascoltare le risposte

# Comunicazioni nei sistemi client-server

## Processi

- Socket
- Chiamate di procedura remota
- Pipe
- Invocazione di metodi remoti (Java)

### Socket

Una **socket** è definita come un **endpoint** per la comunicazione.

Concatenazione di indirizzo IP e porta - numero incluso all'inizio del pacchetto di messaggi per differenziare i servizi di rete su un host.

La comunicazione consiste in una coppia di socket.

Tutte le porte inferiori a 1024 sono ben conosciute, utilizzate per servizi standard.

Indirizzo IP speciale 127.0.0.1 (loopback) per riferirsi al sistema su cui il processo è in esecuzione

# Chiamate di procedura remota

## Processi

### RPC

La RPC astrae le chiamate di procedura tra processi su sistemi di rete

**Stub - proxy lato client per la procedura effettiva sul server.**

Lo stub sul lato client individua il server e fa il marshalling dei parametri.

Lo stub lato server riceve questo messaggio, scompatta i parametri ed esegue la procedura sul server

Rappresentazione dei dati gestita tramite il formato XDL per tenere conto delle diverse architetture (**Big endian, little endian**).

La comunicazione remota ha più scenari di fallimento rispetto a quella locale.

Il sistema operativo fornisce tipicamente un servizio di rendez-vous (o mathmaker) per connettere client e server

Agisce come un condotto che consente a due processi di comunicare  
**Le pipe ordinarie non sono accessibili dall'esterno del processo che le ha create.** In generale, un processo padre crea una pipe e la usa per comunicare con un processo figlio che ha creato.  
FIFO: è possibile accedervi senza una relazione genitore-figlio.

**Le pipe ordinarie consentono la comunicazione in stile produttore-consumatore standard.**

Il produttore scrive su un'estremità e il consumatore legge dall'altra parte.  
Le pipe ordinarie sono quindi unidirezionali, richiedono una relazione genitore-figlio tra i processi comunicati.

# Pipe con nome: FIFO

## Processi

Le FIFO sono più potenti delle pipe ordinarie, la comunicazione è bidirezionale.

Non è necessaria alcuna relazione genitore-figlio tra i processi comunicanti. Diversi processi possono utilizzare le pipe con nome per comunicare