Struttura di un Sistema Operativo

Esistono diverse strutture di Sistemi Operativi divisibili in:

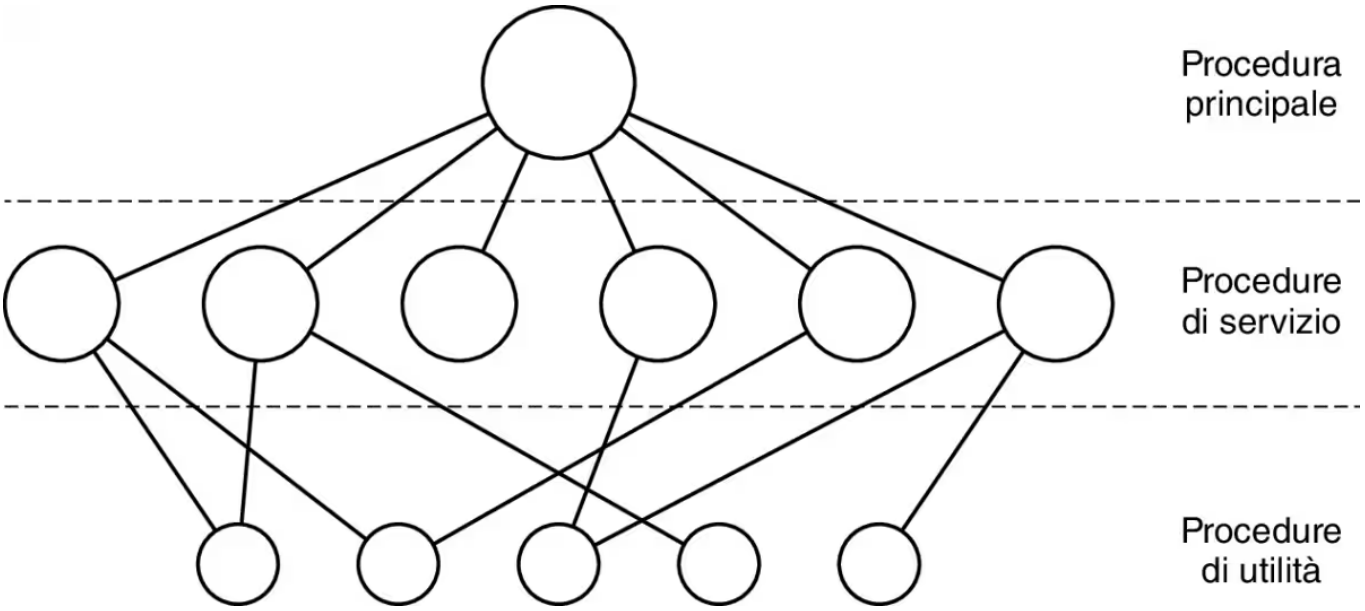
- **Sistemi monolitici**
- **Sistemi stratificati**
- **Microkernel**
- **Sistemi Client-Server**
- **Macchine virtuali**
- **Exokernel e Unikernel**

Sistemi monolitici

Organizzazione più comune nel quale il SO viene eseguito come singolo programma in modalità kernel. Il SO è come una raccolta di procedure tutte linkate all'interno di un solo grande programma eseguibile.

In questo sistema ogni procedura è libera di chiamarne altre, il che risulta efficiente, ma nella situazione in cui una proceudre dovesse andare in crash l'interno sistema sarebbe a rischio.

Per costruire il programma oggetto del SO bisogna prima compilare tutte le procedure e poi unirle utilizzando il linker di sistema e **non esiste alcun occultamento di informazioni**.
Esistono anche dei sistemi monolitici in cui è possibile avere una struttura:



Questi livelli chiamano rispettivamente:

- **La procedura di servizio richiesta**

- **Le chiamate di sistema**
- **Aiutano le procedure di servizio**

> **Molti SO supportano anche esetesioni caricabili (come i driver), sono componenti caricati a richiesta e in UNIX si chiamano librerie condivise in windows DDL.**

Sistemi a livelli

L'organizzazione stratificata è una generalizzazione del sistema monolitico.
Il primo sistema a implementare questa idea era il THE, che si basava su sei livelli implementati uno sopra l'altro che gestivano:

- 0 - L'allocazione del processo e multiprogrammazione.
- 1 - La memoria.
- 2 - Comunicazione tra operatore e processo.
- 3 - Input/Output.
- 4 - Programmi utente.
- 5 - Il luogo dove era posizionato l'operatore.

Questa idea è stata ripresa da MULTICS che la rappresentava con degli **anelli concentrici**, quelli più interni erano più privilegiati di quelli esterni.
Quando un'anello esterno voleva chiamare una procedura interna doveva fare l'equivalente di una chiamata di sistema.

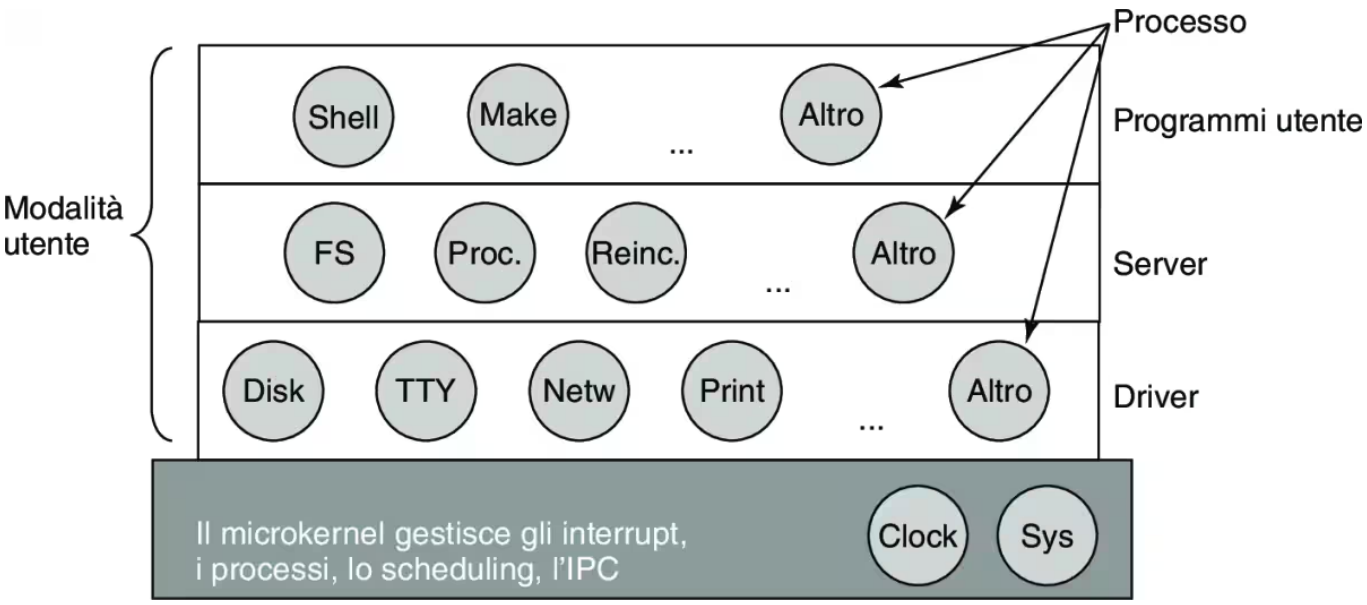
Caratteristiche di questi sistemi:

- **kernel centralizzato**
- **Interconnessione tra i componenti**
- **Scalabilità**

Microkernel

L'idea generale del microkernel è opposta ai sitemi monolitici, dobbiamo infatti inserire il **minor numero di processi nella modalità kernel** dato che gli errori del kernel possono bloccare l'intero sistema.

Quindi si vuole accingere ad un'alta stabilità creando tanti moduli ben definiti, dei quali solo il microkernel eseguito in modalità kernel.



Fuori dal kernel abbiamo tre livelli di processi, tutti in modalità utente:

- **Driver** che non hanno accesso alle funzioni di I/O.
- **Server** che compiono la maggior parte del lavoro del Sistema Operativo.
- **Programmi utente**

Il livello dei server gestisce:

- File system
- creano il gestore del processo

Reincarnation server

Server il cui compito è di verificare se gli altri server e driver funzionano.

Quindi in questa struttura tutti i driver e server hanno esattamente il potere di compiere il loro lavoro e niente di più, il che limita enormemente il danno che un componente può fare.

POLA:

Il principio su cui si basa il sistema è detto POLA (**Principle of Least Authority**).

Inoltre i processi comunicano attraverso il passaggio di messaggi.

> ⚠ **Il passaggio di messaggi è più lento di una chiamata di funzione (kernel monolitico).**

Modello client-server

Leggera variazione del modello precedente, ci sono due classi di processo:

- **Client**: Sfruttano i servizi.
- **Server**: Mettono a disposizione servizi.

Di conseguenza ci sono processi server e processi client.

La comunicazione avviene tramite scambio di messaggi per ottenere un servizio un processo client deve costruire un messaggio da mandare al server che offre un determinato servizio, questo deve poi eseguire il lavoro e inviare una risposta.

> ✎ **Non è necessario che i processi si trovino sulla stessa macchina, poiché non è necessario che i client sappiano se i messaggi sono gestiti localmente o no.**

Macchine virtuali

Permettono l'esecuzione di più sistemi operativi su un unico hardware fisico, simulando un'ambiente separato per ciascuno.
Ha origine con il sistema **VM/370** in grado di creare ambienti virtuali identici all'hardware fisico capaci di eseguire diversi SO contemporaneamente.

> ✎ **Vantaggi:**
>
> 1)**Isolamento**
> 2)**Flessibilità**: Eseguire diversi sistemi contemporaneamente
> 3)**Gestione semplificata delle risorse**

Vennero inizialmente inventate per separare la multiprogrammazione dalla macchina estesa.

> ✎ **Cosa Significa Separare la Multiprogrammazione dalla Macchina Estesa**
>
> **Multiprogrammazione**: Significa far girare più programmi nello stesso momento. Ma se uno di questi programmi ha un problema, può influenzare gli altri.
>
> **Macchina Estesa**: È un modo per utilizzare al meglio l'hardware di un computer, cercando di fare più cose contemporaneamente, mancanza di isolamento tra i processi poteva

portare a inefficienze.

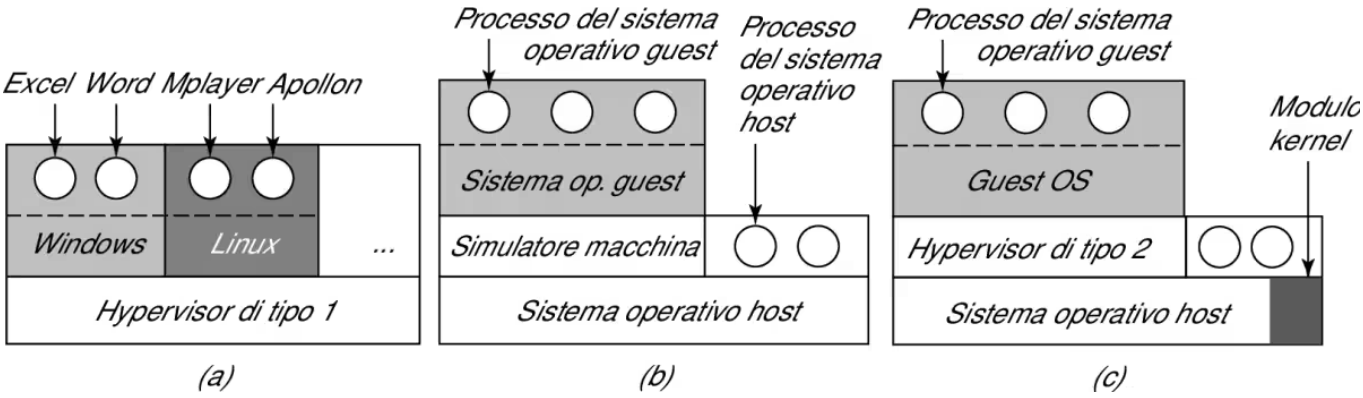
la VM/370 era composta da un **monitor delle macchine virtuali** che girava sul hardware e realizzava la multiprogrammazione fornendo diverse macchine virtuali a livello successivo, queste sono la copia *esatta* dell'hardware.
Poiché ogni macchina virtuale è identica allìhardware reale ognuna può eseguire il suo SO direttamente sull'hardware.
Uno dei sistemi più usati era il **CMS** per gli utenti in timesharing, inoltre le interfacce di chiamata di sistema erano indipendenti dal SO.

> **Il successore moderno del VM/370 è il z/VM usato per far girare diversi sistemi Linux in mabienti ad alta intensità di dati.**

Con il passare degli anni sono state sviluppate diverse tecnologie come quella della **virtualizzazione** che consente di creare versioni virtuali di risorse fisiche, sono composte da un software, **Hypervisor** che ha il compito di emulare l'hardware.

Esistono due tipi di MV:

- **Tipo 1**: Hypervisor eseguito direttamente sull'hardware della macchina.
- **Tipo 2**: Hypervisor ospitato nel SO.



> **In teoria, nell tipo 2, non ci sono specializzazioni da parte del sistema sottostante, in pratica esistono dei moduli del kernel che ottimizano il processo di simulazione.**

Oltre alla virtualizzazione si possono eseguire più istanze di un sistema operativo su un'unica macchina tramite **contenitori**.
Eseguono solitamente solo la parte utente e condividono il kernel de SO host, sono molto leggeri perché contengono solo una parte del sistema operativo.

Contenitore: insieme di uno o più processi isolati dal resto del sistema.

> **Svantaggi:**
>
> 1)Se host e container hanno diversi SO, allora non si può eseguire.
> 2)Non esiste un partizionamento rigido delle risorse.
> 3)I container sono isolati solo a livello di processo.

Una volta finito di usare il container si può liberare lasciando spazio libero sul disco.

Exokernel e Unikernel

L'idea dietro l'**exokernel** è di separare il controllo delle risorse dalla macchina estesa.
Sono molto simili agli hypervisor con la differenza che non emulano l'hardware e fornisce una condivisione sicura delle risorse solo a basso livello.
Ogni MV a livello utente esegue solo il suo SO ed è limitata ad usare le risorse assegnate, questa **struttura elimina la necessità di mappature complesse** perchè dobbiamo concentrarci solo su quale MV ha accesso a quali risorse.

Gli **unikernel** sono sistemi minimi basati sul **LibOS** progettati per eseguire solo una applicazione su una MV, quindi contengono funzionalità solo per il funzionamento dell'applicazione specifica.
Sono molto effiicienti perché non richiedono protezione da parte del sistema operativo.