1.7 Struttura di un sistema operativo

Esistono sei strutture per quanto riguarda i sistemi operativi: `Sistemi Monolitici`, `Sistemi a livelli/strati`, `Microkernel`, `Sistemi Client-Server`, `Macchine Virtuali` e `Exokernel`

1.7.1 Sistemi Monolitici

> ***DEF:*** Un sistema operativo monolitico è un'architettura del sistema operativo in cui l'intero sistema operativo funziona nello spazio del kernel.

Esistono tre livelli principali nei sistemi operativi monolitici: `Livello applicazione`, `Kernel monolitico` e `Livello hardware`.

In questi sistemi operativi, **ogni applicazione ha il proprio spazio di indirizzi**. Pertanto, le applicazioni sono più sicure. **Il kernel gestisce i servizi del sistema operativo**, che includono il `file system`, `lo scheduler della CPU` e il `gestore della memoria`.

Le applicazioni richiedono servizi dal kernel tramite **chiamate di sistema**.

Quando un'applicazione richiede un servizio, lo spazio degli indirizzi hardware dell'applicazione passa allo spazio degli indirizzi hardware del sistema operativo per eseguirlo. I sistemi operativi monolitici gestiscono un'interfaccia virtuale di alto livello sull'hardware del computer. Inoltre, in questo, è possibile aggiungere driver di dispositivo al kernel come moduli.

Un esempio di sistema che usa il Kernel Monolitoco è **Linux**

1.7.2 Sistemi a livelli/strati

Al contrario,

> ***DEF:*** Un sistema operativo a più livelli è un'architettura del sistema operativo divisa in un numero di livelli, ciascuno dei quali esegue una funzionalità specifica.

Lo scopo dello sviluppo di sistemi operativi a più livelli è evitare i limiti dei sistemi operativi monolitici.

Nei sistemi operativi a più livelli, tutti i livelli esistono separatamente e la modifica in un livello non influisce sugli altri livelli.

Pertanto, è anche più facile creare, mantenere e aggiornare i sistemi operativi a più livelli. Inoltre, il livello più basso gestisce le operazioni relative all'hardware mentre il livello più alto gestisce le applicazioni utente.

Ci sono sei livelli principali nei sistemi operativi a più livelli:

```
Hardware
```

Questo è il livello più basso nell'architettura del sistema operativo. Questo livello **gestisce i dispositivi hardware**.

```
Livello CPU
```

Gestisce le attività di pianificazione e pianifica i processi per la CPU.

```
Gestione della memoria
```

Gestisce la memoria.

Sposta i processi dal disco alla memoria primaria per l'esecuzione e invia i processi eseguiti al disco.

```
Gestione dei processi:
```

Gestisce i processi. Questo livello assegna la CPU per eseguire i processi.

```
Buffer IO
```

Consente agli utenti di **interagire con il sistema e gestisce i buffer per i dispositivi IO**, assicurando che i dispositivi IO funzionino correttamente.

```
Programmi utente
```

Il livello più alto nel sistema operativo a più livelli e **gestisce i programmi utente come elaboratori di testi, browser** ecc.

1.7.3 Microkernel

> ***DEF:*** Un microkernel in un Sistema Operativo è un software o anche un codice che contiene la quantità quasi minima di funzioni e caratteristiche necessarie per implementare un sistema operativo.

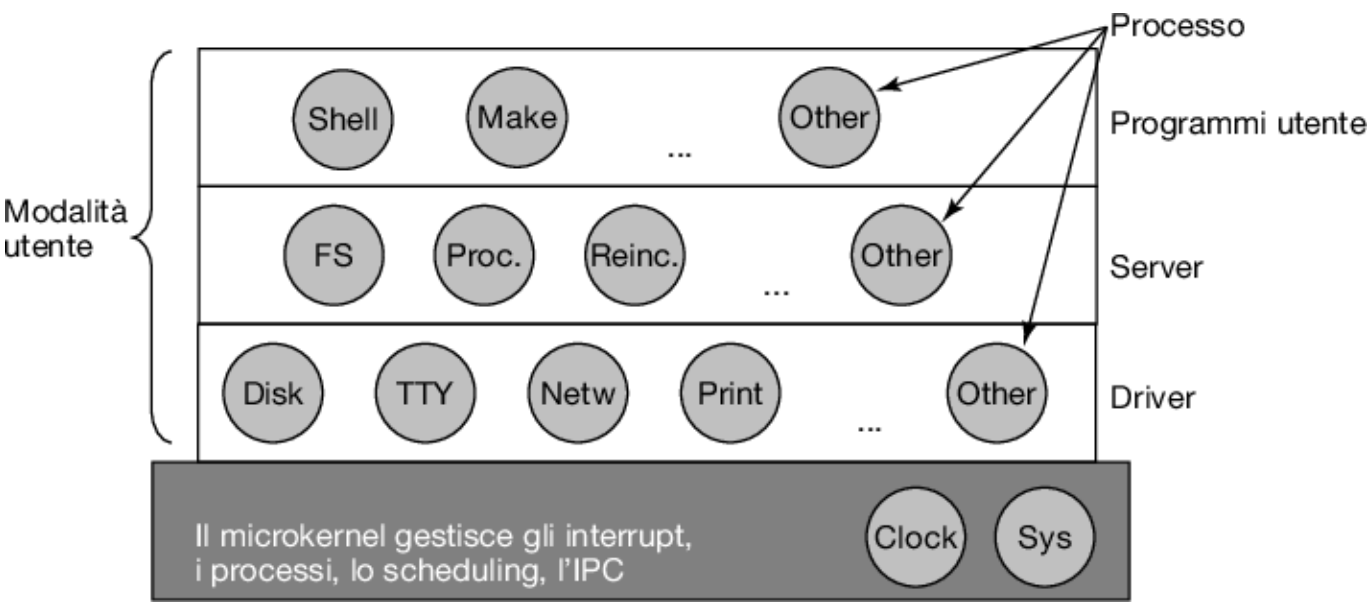
Questa scelta di implementare una quantità quasi minima di funzioni e numero minimo di meccanismi, **nasce dall'idea di differire da un sistema a livelli/strati**.

Tradizionalmente, tutti gli strati andavano nel kernel, ma ciò non è necessario. In realtà, si può fare un grande sforzo per collocare il meno possibile in modalità kernel, dato che errori del kernel possono far cadere immediatamente il sistema.

Quindi l'idea alla base del progetto del microkernel è di **raggiungere un'alta stabilità suddividendo il sistema operativo in** `piccoli moduli` ben defniti, **uno solo** dei quali (microkernel) **verrà eseguito in modalità kernel** e il ***resto come uno dei più normali processi utente*** relativamente deboli.

In particolare, facendo girare ogni driver di dispositivo e file system come processo utente separato

> ***E.g***: Un driver audio potrà far sì che il suono sia storpiatoo non si senta bene, ma non bloccherà il computer, mentre in un kernel monolitico può causare un crash/blocco totale.



Un esempio di Microkernel sono: QNX, MINIX3 e POSIX.

Piccole Note

```
Strato Server
```

Lo `strato Server` compie la maggior parte del lavoro del sistema operativo. Uno o più server **gestiscono il file system, creano il gestore del processo, cancellano e gestiscono processi** e così via.

Vengono inviati brevi messaggi ai server, **richiedendo le chiamate di sistema POSIX**.

```
Reincarnation Server
```

Questo server ha il compito è di **verificare se gli altri server e driver stanno funzionando correttamente**.

Nel caso che ne riscontri uno mal funzionante, **viene immediatamente sostituito** senza alcuna interruzione per l'utente. In questo modo il sistema si fa **automanutenzione e può raggiungere una stabilità notevole**.

1.7.4 Client-Server

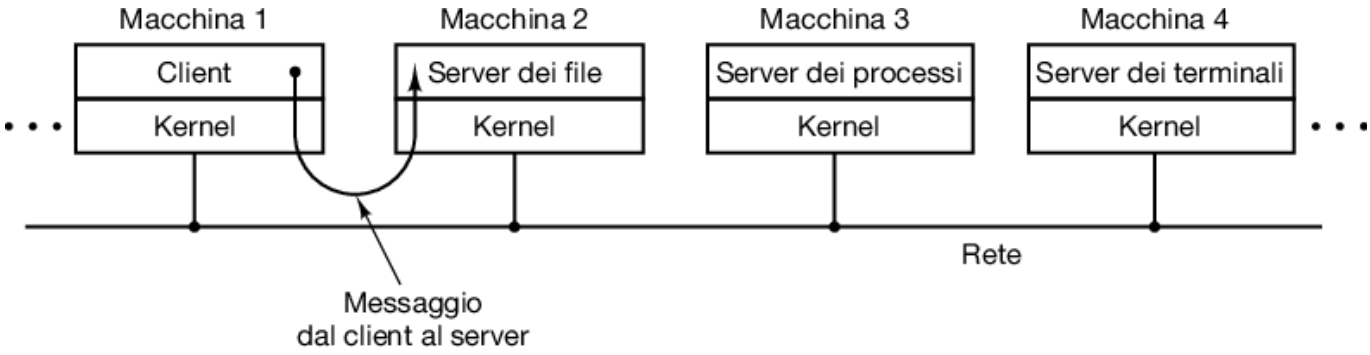
Una leggera variazione rispetto all'idea del microkernel è distinguere due classi di processi, `i server`, *ognuno dei quali mette a disposizione alcuni servizi*, e i `client`, *che li utilizzano*.

Questo modello è conosciuto come **modello client-server**.

Speso il livello più basso è un microkernel, ma non è necessario. L'aspetto principale è la presenza di processi client e processi server.

Per ottenere un servizio, **un processo client costruisce un messaggio indicando la propria richiesta e lo invia al servizio appropriato (server)**. **Il servizio quindi esegue il lavoro e rimanda indietro la risposta**. Se client e server sono eseguiti sulla stessa macchina sono possibili alcune ottimizzazioni, ma concettualmente si tratta del passaggio di un messaggio.

Questo modello client-server offre un'astrazione utilizzabile per una macchina singola o per una rete di macchine, dato che al client non fa differenza se il server è locale o in rete.



1.7.5 Macchine Virtuali

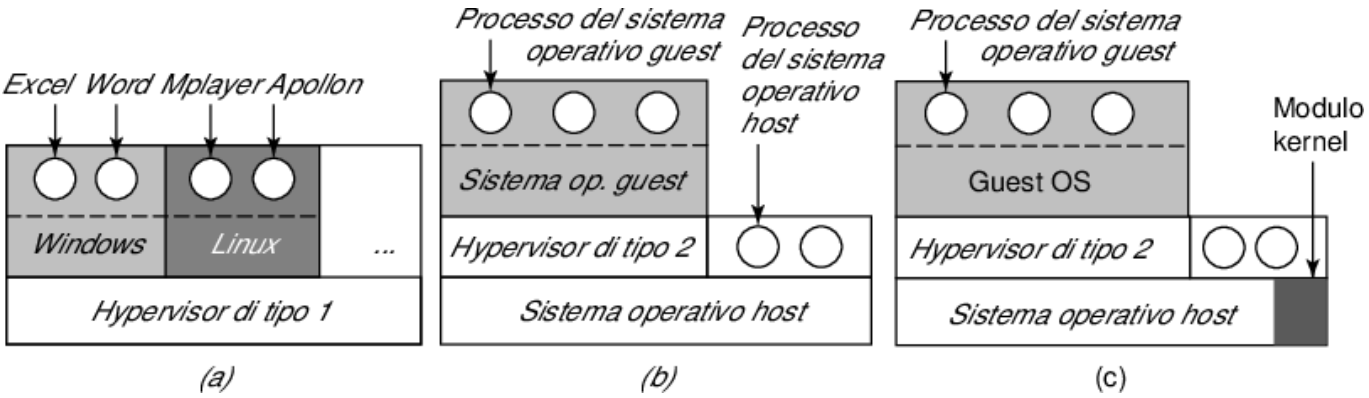
Nel passato molte aziende hanno tradizionalmente tenuto i loro server di posta, server web, server FTP e altri server su computer separati, talvolta con sistemi operativi diversi.

Una soluzione a quessto è la `Virtualizzazione`.

Tramite la virtualizzazione **si possono tenere tutti questi servizi/S.O sulla stessa macchina**, *senza che il blocco di un server/S.O determini il crollo degli altri*.

> ***E.g***: Nell'hosting Web al giorno d'oggi vengono **usate piccole Macchine Virtuali installate su Server potenti**, **così da avere tantissimi utenti (clienti) connessi ad una macchina**, portando **meno consumi, spazio fisico usato e scalabità**. I clienti vedranno solo la macchina virtuale, che opera come un computer normale.

Questo tipo di esempio è rappresentato nella figura (a)



Un grande problema della Virtualizzazione era lo sviluppo del software.

Per permettere che **il software della macchina virtuale fosse eseguito su un computer**, **la sua CPU doveva essere virtualizzabile.**

Quando un **sistema operativo in esecuzione su una macchina virtuale** (in modalità utente) **esegue un'istruzione privilegiata, come fare un I/O**, è essenziale che **l'hardware esegua un trap verso l'hypervisor**, in modo che l'istruzione possa essere emulata via software.

Su alcune CPU, tipo i Pentium, i tentativi di eseguire istruzioni privilegiate in modalità utente sono ignorati. Cioè rende impossibile disporre di macchine virtuali su questo hardware, tranne alcuni tentativi riusciti che avevano terribili problemi di performance.

Ai giorni nostri, oltre a `VMware e a Xen`, fra gli hypervisor più conosciuti troviamo oggi `KVM` **(Linux),** `VirtualBox` **(Oracle) e** `Hyper-V` **(Microsoft)**.

Alcuni progetti di ricerca inizialmente migliorarono le prestazioni rispetto agli interpreti come Bochs traducendo blocchi di codice al volo, memorizzandoli in una cache interna e quindi riutilizzandoli se dovevano essere nuovamente eseguiti. Le prestazioni venivano in questo modo sensibilmente migliorate,

Ma nonostante questi miglioramenti, queste soluzioni non erano ancora abbastanza veloci da poter essere utilizzati in ambiente commerciale, dove invece le prestazioni sono fondamentali.

Il passaggio successivo per migliorare le prestazioni fu l'aggiunta di un modulo kernel che eseguisse il grosso del lavoro.

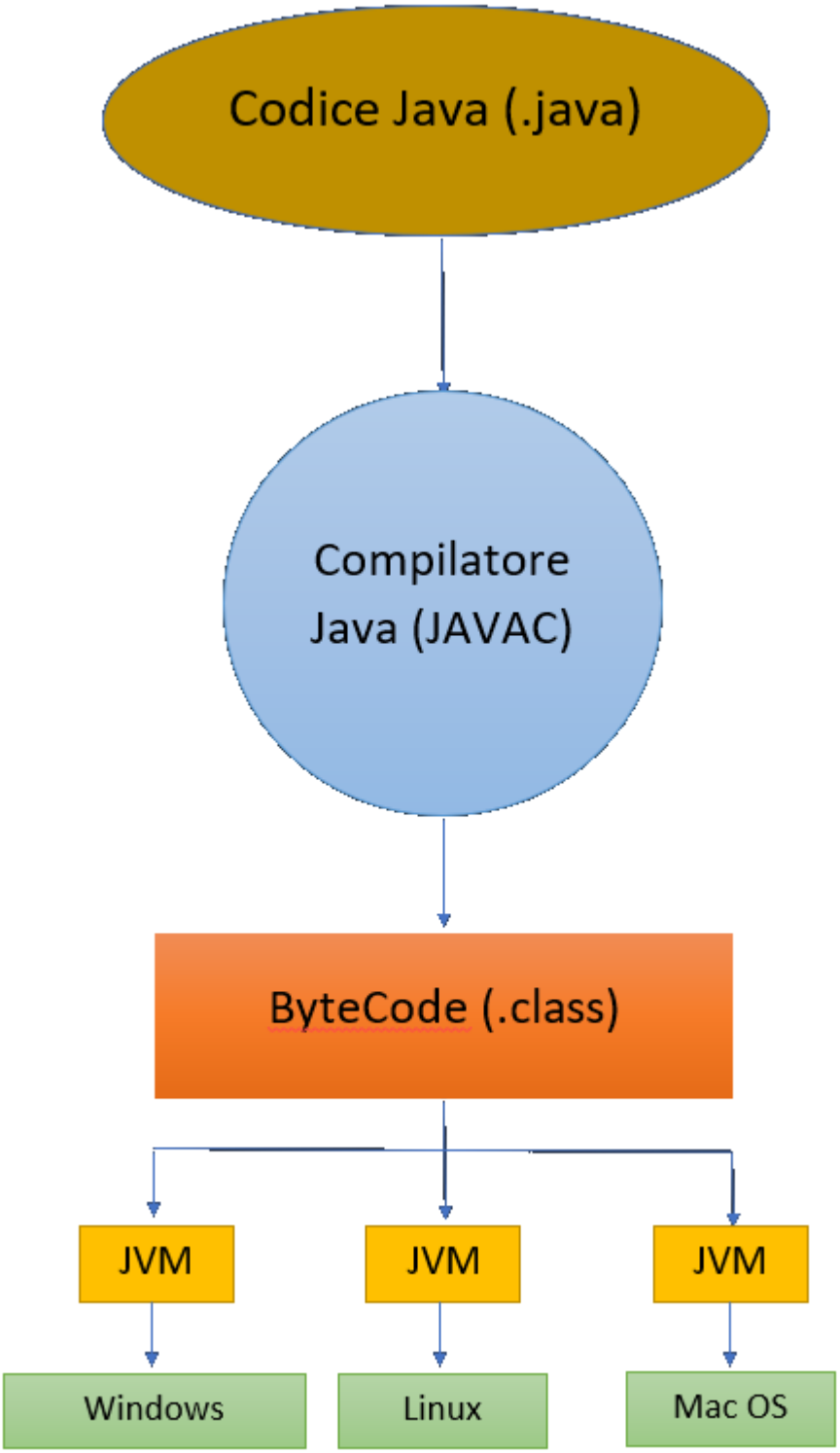
In pratica, oggi tutti gli hypervisor in commercio, come VMware Workstation, utilizzano una strategia ibrida. Sono chiamati `hypervisor di tipo 2`.

La macchina virtuale Java

Un'altra area in cui le macchine virtuali sono utilizzate, ma in modo un po' diverso, è l'esecuzione di programmi `Java`.

Quando Sun Microsystem inventò il linguaggio di programmazione Java, c**reò anche una macchina virtuale** (cioè un'architettura di computer) chiamata `JVM` (***J**ava **V**irtual **M**achine*).

Il `compilatore Java` produce codice per la `JVM`, poi generalmente eseguito da un interprete software `JVM`. Il vantaggio di tale metodo è che il `codice JVM` **(bytecode) può essere spedito tramite Internet su qualunque computer che disponga di un interprete JVM ed eseguito lì**.



1.7.6 Exokernel

Piuttosto che clonare la macchina reale, come avviene con le macchine virtuali, un'altra strategia è partizionarla, assegnando quindi a ogni utente un sottoinsieme delle risorse.

> ***E.g:*** Una macchina virtuale potrebbe così prendersi i blocchi del disco da `0` a `1023`, la successiva i blocchi da `1024` a `2047` e così via.

Il livello alla base, che gira in modalità kernel, è un programma chiamato **exokernel**.

Il suo compito è quello di allocare risorse alle macchine virtuali e poi controllare i tentativi di impiego per essere certo che nessuna macchina provi a usare risorse di un'altra. **Ogni macchina virtuale** a livello utente e**segue il suo sistema operativo personale**, a parte il fatto che **ognuna è limitata a usare le sole risorse che ha richiesto e che le sono state allocate**.

Il vantaggio dello schema dell'exokernel è che risparmia uno strato di corrispondenza. *Negli altri progetti ogni macchina virtuale pensa di avere il proprio disco, con blocchi che vanno da `0` a un `massimo`, così il monitor della macchina virtuale deve mantenere delle tabelle per rimappare gli indirizzi del disco (e di tutte le altre risorse)*.

Con l'exokernel, questo rimappaggio non è più necessario. **L'exokernel necessita solo di tenere traccia di quale sia la macchina virtuale a cui è stata assegnata una certa risorsa**. Questo metodo **ha ancora il vantaggio di tenere separata la multiprogrammazione dal codice del sistema operativo utente** (nello spazio utente), **ma a un costo inferiore**, dato che il c**ompito dell'exokernel è far sì che tutte le macchine virtuali siano tra loro indipendenti**.