3° lezione S.O.

Due Date	@March 11, 2025
Multi-select	Kernel VM
Status	Done

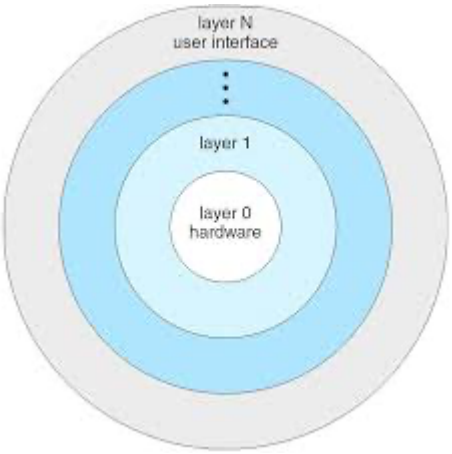
L'idea degli strati nella progettazione dei sistemi operativi

Quando si progetta un sistema operativo, una delle principali sfide è organizzare in modo efficiente la gestione delle numerose funzionalità e risorse. Un approccio strutturato e modulare è quello degli **strati** (o "layered architecture"), che suddivide il sistema in livelli indipendenti, ognuno con un compito specifico.

L'idea di progettare un sistema operativo a strati nasce dall'esigenza di:

- **Semplificare lo sviluppo e la manutenzione**, separando le varie funzionalità in moduli distinti.
- **Facilitare il debugging e il testing**, poiché ogni strato può essere testato singolarmente.
- **Migliorare la scalabilità e l'intercambiabilità**, permettendo la modifica di un livello senza influenzare gli altri.

Come funziona il modello a strati?



Immaginiamo di costruire un sistema operativo partendo da una **base solida**, il primo strato, che fornisce i servizi essenziali. Su di esso si aggiungono ulteriori livelli, ciascuno responsabile di una specifica funzione.

Ad esempio, possiamo strutturare il sistema operativo con i seguenti strati:

1. **Strato hardware** → Interfaccia diretta con il processore, la memoria e i dispositivi.
2. **Gestione dei processi e della CPU** → Allocazione delle risorse e scheduling dei processi.
3. **Gestione della memoria** → Implementazione della memoria virtuale e gestione della RAM.
4. **File system** → Organizzazione e accesso ai dati memorizzati su disco.
5. **Interfaccia utente e applicazioni** → Comunicazione con l'utente e esecuzione dei programmi.

Questo consente anche di eseguire cambiamenti facilmente ad esempio gli algoritmo di scheduling lasciando in alterato tutto il resto

Ogni strato fornisce servizi a quello superiore e si appoggia sulle funzionalità offerte dagli strati inferiori. Ad esempio, un processo che necessita di memoria non interagisce direttamente con l'hardware, ma si affida al livello di gestione della memoria, che può utilizzare il disco per estendere virtualmente la RAM disponibile.

Grazie alla struttura stratificata del sistema operativo e alla virtualizzazione della CPU, le operazioni di **memoria virtuale**(come il paging e lo swapping su disco) possono avvenire in parallelo rispetto ad altri processi, senza bloccare l'esecuzione complessiva del sistema.

Questo è possibile perché ogni processo dispone di una propria **Virtual CPU**, che gli dà l'illusione di avere accesso esclusivo alla CPU fisica. In realtà, il sistema operativo gestisce l'allocazione della CPU tra più processi, consentendo di eseguire altre operazioni mentre, ad esempio, una pagina di memoria viene caricata dal disco.

In questo modo, le procedure di gestione della memoria virtuale possono operare in background, senza interrompere il flusso di esecuzione di altri componenti del sistema. Ciò migliora l'efficienza complessiva, riducendo i tempi di attesa e ottimizzando l'uso delle risorse hardware.

Vantaggi del modello a strati

- **Maggiore organizzazione** – La suddivisione in livelli aiuta a mantenere il codice più leggibile e modulare.
- **Facilità di testing** – È possibile verificare la correttezza di ogni strato isolatamente.
- **Modificabilità** – Un miglioramento a un livello (ad esempio, un nuovo algoritmo di scheduling) può essere implementato senza dover riscrivere l'intero sistema.
- **Isolamento e sicurezza** – Ogni strato può limitare l'accesso diretto alle risorse critiche, riducendo i rischi di errori o vulnerabilità.

Svantaggi

Nonostante i vantaggi, il modello a strati presenta alcune difficoltà:

▼ **Definizione degli strati**

Non sempre è chiaro quali funzionalità debbano appartenere a ciascun livello.

▼ **L'overhead di comunicazione**

- Un possibile svantaggio della struttura a strati è l'**overhead di comunicazione** tra i livelli. Ogni volta che un'operazione richiede l'intervento di più strati, si aggiunge un certo **costo computazionale** dovuto al passaggio di informazioni e alle chiamate tra moduli separati.

Ad esempio, un'applicazione che deve accedere a un file non interagisce direttamente con il disco, ma passa attraverso vari strati:

1. L’applicazione richiede il file al **livello dell’interfaccia utente**.
2. La richiesta viene inoltrata al **file system**, che verifica la disponibilità dei dati.
3. Se necessario, il file system chiede al **gestore della memoria** di caricare la pagina richiesta in RAM.
4. Se la pagina non è già in memoria, il gestore della memoria deve recuperarla dal **disco**, utilizzando i servizi dello **strato hardware**.

Ogni passaggio comporta un’ulteriore elaborazione e quindi un potenziale **rallentamento** delle operazioni. Questo **overhead**, però, è il compromesso necessario per avere un sistema organizzato, modulare e più facile da gestire e aggiornare.

Il fatto che la virtual memory sta ad un livello sopra può divetare un problema perché le strutture dati di gestione dei processi sono molto ingombranti.

▼ **Rigidità**

Una suddivisione troppo rigida potrebbe rendere difficile l’ottimizzazione delle prestazioni.

Dal modello a strati al modello ad anelli nei sistemi operativi

Abbiamo discusso il modello **a strati**, che organizza il sistema operativo in livelli separati, ognuno con compiti specifici. Tuttavia, una variante storica di questo approccio è il **modello ad anelli**, utilizzato nei sistemi operativi come Unix e nei suoi predecessori.

La differenza tra il modello a strati e il modello ad anelli

Concettualmente, i due modelli sono simili:

- Ogni livello (o anello) **offre servizi** allo strato superiore e si appoggia su quello inferiore.
- Le **strutture dati di uno strato sono incapsulate**, cioè **non accessibili direttamente** dagli strati superiori, garantendo modularità e possibilità di sostituire algoritmi e strutture interne senza impatti sugli altri livelli.

La differenza principale sta nel modo in cui il modello ad anelli **garantisce protezione tra i livelli**: mentre il modello a strati è una separazione **logica e progettuale**, il modello ad anelli introduce una **protezione hardware tra gli strati**, impedendo a un livello superiore di accedere direttamente a risorse riservate a un livello inferiore.

Il problema del modello a strati: la mancanza di protezione hardware

Nei sistemi basati esclusivamente sulla separazione logica tra strati, si presenta un problema fondamentale

Cosa succede se c'è un bug in uno degli strati?

Se un errore software porta a un accesso a una memoria sbagliata, uno strato potrebbe modificare dati appartenenti a un altro livello, causando corruzione del sistema. **L'architettura a strati da sola non fornisce protezione** da questo tipo di problemi, poiché i limiti tra gli strati sono imposti **solo dalla progettazione software**, non da vincoli hardware.

Nel migliore dei casi, il sistema crasha subito. Nel peggiore, l'errore corrompe i dati e si manifesta più avanti, rendendo difficile diagnosticare il problema.

Il modello ad anelli e la protezione hardware

Per mitigare questo problema, alcuni sistemi operativi hanno adottato un modello **multi-strato basato su anelli di protezione**. Questo modello è simile al precedente ma introduce una **separazione hardware tra i livelli**, impedendo a un livello superiore di accedere direttamente a risorse privilegiate.

L'idea è rappresentata da una struttura ad **anelli concentrici**:

- L'**anello più interno** (Ring 0) contiene il **nucleo del sistema operativo (kernel)**, con accesso completo all'hardware.
- Gli **anelli intermedi** contengono componenti meno privilegiati del sistema, come i driver di dispositivo.
- Gli **anelli più esterni** eseguono le applicazioni utente, con il minor livello di privilegio.

Ogni anello può invocare i servizi dell'anello immediatamente inferiore tramite interfacce ben definite, ma **non può accedere direttamente a memoria o**

strutture dati degli anelli più interni.

Questa struttura consente al sistema operativo di **contenere gli errori**:

- Se un'applicazione in esecuzione in un anello esterno ha un bug, non può modificare direttamente i dati del kernel.
- Se un driver di dispositivo ha un errore, il danno è limitato al livello del driver e non compromette l'intero sistema.

Il supporto hardware al modello ad anelli

Il modello ad anelli ha richiesto un supporto specifico a livello di CPU. Ad esempio, i processori **x86** di Intel implementano **quattro livelli di privilegio (Ring 0 a Ring 3)** per garantire che il codice utente non possa interferire con il kernel. Tuttavia, i sistemi moderni spesso utilizzano solo due livelli principali:

- **Ring 0** per il kernel e i driver più critici.
- **Ring 3** per le applicazioni utente.

Questa separazione, insieme alle funzionalità moderne di gestione della memoria (come la **Memory Protection Unit** e la **Virtual Memory**), fornisce una sicurezza molto più robusta rispetto al modello a strati tradizionale.

Dal modello ad anelli al concetto di microkernel

Il concetto di **microkernel** nasce come risposta alla complessità e ai limiti dei kernel monolitici. Nei sistemi operativi tradizionali, il kernel include tutte le funzionalità principali, dalla gestione dei processi al file system e ai driver di dispositivo. Questo approccio, sebbene efficiente in termini di prestazioni, rende il sistema **meno modulare**, più difficile da mantenere e più vulnerabile ai bug, poiché ogni componente è strettamente integrato con il resto del kernel.

Il **microkernel**, invece, riduce al minimo le funzionalità presenti nel kernel, mantenendo solo quelle essenziali, come la gestione della CPU, della memoria di base e la comunicazione tra processi. **Tutte le altre funzionalità vengono spostate in servizi esterni**, che operano nello **spazio utente** anziché nello **spazio kernel**. In questo modo, il kernel rimane snello, mentre le funzionalità aggiuntive (file system, driver, networking, gestione della stampa, ecc.) vengono gestite separatamente.

Gestione dei Processi e Struttura del Microkernel

Il microkernel si occupa di gestire processi e memoria, operando a un livello più basso rispetto agli altri componenti del sistema operativo. Il suo compito principale è **mantenere il controllo sulle risorse di base**, delegando invece le funzionalità avanzate ai servizi utente.

Un aspetto cruciale del microkernel è la **comunicazione tra i processi** (IPC - Inter-Process Communication). Poiché molti componenti del sistema non operano direttamente nello spazio kernel, devono poter **scambiarsi messaggi** in modo sicuro ed efficiente. Questo approccio consente una maggiore modularità e isolamento, ma introduce un **overhead** che può rallentare il sistema.

Servizi e Struttura Modulare

In un'architettura microkernel, i servizi di sistema non fanno parte del kernel stesso, ma operano come **processi separati**in modalità utente. Questo porta a un sistema più sicuro e stabile:

- Se un **servizio si blocca**, non compromette l'intero sistema operativo.
- È possibile aggiornare o sostituire un servizio senza dover modificare il kernel.
- La separazione tra kernel e servizi limita i danni di eventuali vulnerabilità, aumentando la sicurezza.

Alcuni esempi di **servizi separati** in un sistema microkernel sono:

- **File system** – Gestisce la lettura e scrittura su disco.
- **Networking** – Gestisce la comunicazione in rete.
- **Driver hardware** – Interagiscono con periferiche come stampanti e schede di rete.
- **Gestione della stampa** – Controlla l'invio di documenti alla stampante senza interferire con altri processi.

Questa modularità consente di **attivare o disattivare funzionalità specifiche**, adattando il sistema a diversi scenari d'uso.

Vantaggi dell'Approccio Microkernel

L'adozione di un'architettura microkernel offre diversi vantaggi:

1. **Maggiore sicurezza** – Riducendo il codice eseguito in modalità kernel, si limita la superficie d'attacco per eventuali exploit.
2. **Stabilità migliorata** – Un crash di un servizio utente non compromette l'intero sistema.
3. **Maggiore modularità** – È possibile aggiornare e sostituire i singoli servizi senza dover ricompilare o modificare il kernel.
4. **Facilità di portabilità** – Il kernel essenziale può essere utilizzato su diverse piattaforme hardware, mentre i servizi vengono adattati separatamente.

Un ulteriore vantaggio è la **gestione degli errori**: se un servizio si blocca, il sistema può rilevarlo e **riavviarlo automaticamente** senza interrompere il resto delle operazioni. Questo meccanismo, chiamato **reincarnazione dei servizi**, garantisce maggiore affidabilità.

Tuttavia, il microkernel introduce un **costo in termini di prestazioni**: la necessità di **scambiare messaggi** tra il microkernel e i servizi introduce **overhead**, rallentando alcune operazioni rispetto ai kernel monolitici, dove i componenti sono direttamente integrati.

Che cosa è l'overhead?

L'**overhead** in un sistema operativo basato su microkernel è il costo aggiuntivo derivante dalla necessità di gestire le comunicazioni tra i diversi componenti del sistema. Poiché un microkernel mantiene solo le funzionalità essenziali (gestione della CPU, memoria e IPC), tutto il resto – come il file system, i driver di dispositivo e i servizi di rete – viene eseguito come processi separati nello spazio utente.

A differenza di un **kernel monolitico**, dove i componenti comunicano direttamente all'interno dello stesso spazio di memoria, un microkernel **richiede scambi di messaggi tra i processi** per eseguire operazioni comuni.

Overhead della comunicazione IPC (Inter-Process Communication)

- Nei microkernel, i servizi operano come processi separati e devono scambiarsi informazioni utilizzando meccanismi di **messaggistica**.
- Ogni volta che un processo utente deve accedere a una risorsa (ad esempio, leggere un file), non può farlo direttamente: deve inviare una richiesta al servizio appropriato (ad esempio, il file system), che poi risponde con i dati richiesti.

- Questo continuo scambio di messaggi introduce **ritardi**, poiché ogni messaggio implica passaggi di contesto e operazioni di copia dei dati tra processi.

Cosa succede se si ha un bug nel microkernel?

Nel caso di un **microkernel puro**, se si verifica un **bug in un servizio**, il sistema può gestire l'errore in modo isolato. Poiché i servizi funzionano in **spazio utente** anziché nello spazio kernel, un crash di un singolo servizio **non compromette l'intero sistema operativo**.

Grazie a questa separazione, il sistema può **rilevare il malfunzionamento** e riavviare automaticamente il servizio interessato senza dover riavviare l'intero sistema. Questo meccanismo viene chiamato **reincarnazione dei servizi** ed è uno dei punti di forza dei microkernel in termini di stabilità e affidabilità.

Tuttavia, se il bug è nel **microkernel stesso**, l'intero sistema potrebbe bloccarsi, poiché il kernel gestisce funzionalità critiche come la gestione della memoria e la comunicazione tra i processi. Fortunatamente, essendo il microkernel **molto piccolo e con meno codice rispetto a un kernel monolitico**, la probabilità di bug critici è più bassa.

Esempi microkernel

Sebbene il **modello microkernel puro** non sia stato adottato su larga scala, alcuni sistemi operativi hanno implementato questo approccio con successo:

- **Mach** Usato come base per alcuni sistemi macOS, ha introdotto il concetto di microkernel con meccanismi di IPC avanzati.
- **QNX** Adottato nei sistemi embedded, dove stabilità e modularità sono essenziali.
- **MINIX** Un sistema operativo accademico basato su microkernel, noto per aver ispirato parte dello sviluppo di Linux.

Tuttavia, la maggior parte dei sistemi operativi moderni, come **Linux** e **Windows**, ha adottato un **approccio ibrido**, combinando elementi del microkernel con un'architettura più monolitica. Questo permette di ottenere un compromesso tra **modularità e prestazioni**, riducendo l'overhead della comunicazione tra i servizi.

Windows NT, Microkernel e il Passaggio ai Kernel Modulari

Windows 98 e l'Arrivo di Windows NT

Inizialmente, Windows 98 e le versioni precedenti di Windows erano basate su un'architettura monolitica, pensata per l'uso consumer. Tuttavia, Microsoft decise di sviluppare Windows NT, un sistema più robusto, con un'architettura completamente nuova e pensata per ambienti server.

Windows NT adottava un'architettura **microkernel**, il che significava che molte funzionalità del sistema operativo (come la gestione della grafica e dei driver) non erano integrate direttamente nel kernel ma giravano come servizi separati in modalità utente. Questo design prometteva maggiore stabilità e sicurezza, perché un crash di un singolo servizio non avrebbe compromesso l'intero sistema.

Il Problema delle Prestazioni

Quando Microsoft decise di portare Windows NT anche nel mondo consumer, emerse un problema: il modello microkernel, sebbene teoricamente più robusto, introduceva **un collo di bottiglia**.

Il problema principale riguardava la gestione grafica: nel design originale, anche il sottosistema grafico era un servizio separato, il che significava che ogni operazione grafica richiedeva una continua comunicazione tra il kernel e il servizio grafico tramite **messaggi IPC**. Questo introduceva una latenza significativa e riduceva le prestazioni, specialmente in applicazioni interattive come giochi e programmi multimediali.

Dal Microkernel a un Modello Ibrido

Per migliorare le prestazioni, Microsoft dovette **reintegrare nel kernel alcune parti che inizialmente erano state pensate come servizi separati**. La grafica, ad esempio, venne spostata all'interno del kernel per ridurre l'overhead della comunicazione IPC.

Questo significava che Windows NT non era più un vero microkernel, ma un **kernel ibrido**, che cercava di mantenere la modularità del microkernel pur incorporando componenti critici direttamente nel kernel per migliorare le prestazioni.

Kernel Modulari

Un kernel modulare è un kernel che mantiene una struttura monolitica, ma con la possibilità di **caricare e scaricare moduli dinamicamente**. Un modulo è una piccola componente del sistema operativo che può essere caricata solo quando serve, senza dover ricompilare l'intero kernel.

Ad esempio, un modulo potrebbe essere:

- Un **file system** (es. FAT, NTFS, ext4)
- Un **protocollo di rete**
- Un **driver per una periferica**

Vantaggi dei Kernel Modulari

- **Flessibilità**: I moduli possono essere aggiunti o rimossi senza dover ricompilare il kernel.
- **Efficienza**: A differenza del microkernel, i moduli vengono eseguiti direttamente nel kernel, senza bisogno di passare messaggi tra processi separati.
- **Struttura organizzata**: Ogni modulo ha un’interfaccia ben definita, simile ai concetti della **programmazione a oggetti**, dove i dettagli di implementazione sono nascosti dietro un'interfaccia pubblica.
- Rigidità di ogni modulo

Esempi di Kernel Modulari

Uno dei più noti esempi di kernel modulare è **Linux**, che adotta un modello monolitico ma con il supporto ai moduli caricabili dinamicamente.
Anche **Solaris** e alcune versioni di **MacOS** hanno integrato idee simili.

L'Importanza della Modularità nei Sistemi Operativi

Caricamento Dinamico dei Moduli

Uno dei grandi vantaggi dei **kernel modulari** è la possibilità di **caricare solo i moduli necessari**. Questo significa che il sistema non ha bisogno di mantenere

attivi tutti i moduli contemporaneamente, riducendo il consumo di memoria e migliorando l'efficienza.

Esempio

Uno dei grandi vantaggi dei **kernel modulari** è la possibilità di **caricare solo i moduli necessari**. Questo significa che il sistema non ha bisogno di mantenere attivi tutti i moduli contemporaneamente, riducendo il consumo di memoria e migliorando l'efficienza.

Ad esempio, se un sistema non ha bisogno di supportare un determinato **file system** o **driver hardware**, il modulo corrispondente non verrà caricato, evitando sprechi di risorse. Quando invece il modulo diventa necessario, può essere **caricato su richiesta (on demand)**, integrandosi dinamicamente nel sistema operativo.

Svantaggi

Se si ha un bug si va ad ibnficaire un po tutto perché si ha un esecuzione totale in modalità kernel.

Moduli ed Estensioni nei Microkernel

Anche nei sistemi basati su **microkernel**, esiste un concetto simile a quello dei moduli: le **estensioni**.

A differenza di un servizio utente tradizionale (che gira in uno spazio separato e comunica con il kernel tramite messaggi), un'estensione può essere **integrata direttamente nel kernel** per migliorare le prestazioni. Questo avviene per alcuni componenti critici che, se eseguiti in modalità utente, introdurrebbero un **overhead eccessivo** dovuto al cambio di contesto e alla comunicazione via messaggi.

In pratica, i sistemi basati su microkernel hanno trovato un compromesso:

- Alcuni servizi meno critici vengono eseguiti in **spazio utente**, migliorando la stabilità.
- Altri servizi più importanti vengono **integrati nel kernel**, migliorando l'efficienza.

MacOS e la Compatibilità con Unix

Un esempio interessante è **macOS**, che pur avendo un kernel basato su **microkernel (XNU)**, mantiene la compatibilità con lo **standard POSIX**, fondamentale per eseguire applicazioni Unix.

Per ottenere questa compatibilità, macOS utilizza uno **strato software** che intercetta le chiamate di sistema Unix e le adatta alla sua architettura interna. Questo permette ai programmi scritti per Unix di funzionare anche su macOS, senza modifiche significative.

Anche **Linux**, pur essendo un kernel monolitico modulare, ha adottato alcune idee dal mondo dei microkernel. Ad esempio, alcuni servizi di sistema meno critici, come la gestione delle stampanti, sono stati **spostati fuori dal kernel** e girano come processi utente. Questo migliora la stabilità del sistema, evitando che un errore in questi servizi possa compromettere il kernel.

Come Vengono Implementati i Driver nei Kernel Modulari

Driver con Moduli

Nei kernel modulari, i **driver hardware** sono trattati come moduli caricabili. Un driver non è altro che un modulo che permette al sistema operativo di comunicare con un dispositivo specifico, come:

- **Schede grafiche**
- **Controller di archiviazione**
- **Dispositivi di rete**

Il **vantaggio** di questo approccio è che un nuovo driver può essere aggiunto senza dover modificare l'intero kernel.

Quando un modulo driver viene caricato, esso fornisce un'interfaccia standard che il sistema operativo utilizza per comunicare con il dispositivo. Questo significa che, indipendentemente dall'hardware specifico, il kernel può interagire con tutti i dispositivi della stessa categoria attraverso un'API comune.

Gestione Dinamica dei Driver

Nel momento in cui un modulo driver viene caricato, il sistema è in grado di gestire **tutti i dispositivi** appartenenti a quella categoria. Se, ad esempio, si

collega una nuova scheda di rete compatibile con un driver già caricato, il sistema sarà immediatamente in grado di riconoscerla e utilizzarla senza bisogno di riavviare.

Gestione dell'Accesso ai Dispositivi nei Microkernel

Nel modello **microkernel**, l'accesso ai dispositivi hardware è regolato in modo rigoroso per garantire sicurezza e stabilità. Poiché i servizi di gestione dell'hardware non operano in modalità kernel, il microkernel deve implementare un sistema di **controllo degli accessi** basato su meccanismi di autorizzazione.

Assegnazione delle Risorse ai Processi

Quando un processo ha bisogno di interagire con un dispositivo, **non può accedere direttamente alle porte di input/output (I/O) del dispositivo**. Questo perché **la gestione dell'hardware è un'operazione privilegiata**, e lasciare che qualsiasi processo possa accedere liberamente alle periferiche rappresenterebbe un rischio per la sicurezza e la stabilità del sistema.

Invece, il **microkernel assegna esplicitamente le risorse** a uno specifico processo, concedendogli il permesso di interagire con un determinato dispositivo. Questo significa che:

- Solo **quel processo** ha il diritto di comunicare con il dispositivo assegnato.
- Altri processi **non vedono né possono accedere** a quel dispositivo.

Mappatura della Memoria e Interazione con i Dispositivi

Quando il microkernel concede a un processo l'accesso a un dispositivo, gli fornisce una **mappatura della memoria** che gli permette di interagire con i registri del dispositivo stesso. Tuttavia, il processo **non ha un accesso diretto all'hardware**:

- Il processo sa che, accedendo a specifici indirizzi di memoria, sta interagendo con il dispositivo.
- Questo avviene attraverso un meccanismo di **delega controllata**, che permette al processo di eseguire solo le operazioni consentite dal microkernel.

Isolamento e Protezione del Sistema

Poiché l'hardware è gestito tramite servizi in spazio utente, i processi **non possono interferire direttamente con altri dispositivi**. Questo evita che un processo malfunzionante o compromesso possa **danneggiare altre parti del sistema** o **causare instabilità globale**.

Tuttavia, se un processo che gestisce un dispositivo presenta un bug o un comportamento anomalo, potrebbe:

- **Bloccare il funzionamento di quel dispositivo** (ad esempio, se il driver della scheda video smette di rispondere, lo schermo potrebbe non aggiornarsi).
- **Generare anomalie nel sistema**, come ritardi o errori nella gestione di altre periferiche se la comunicazione con il microkernel è inefficiente.

Grazie all'architettura del microkernel, però, questi problemi rimangono **circoscritti al singolo servizio**: il resto del sistema continua a funzionare normalmente, e il microkernel può **riavviare il servizio guasto senza dover riavviare l'intero sistema operativo**.

VM(Virtual Machine)

Introduzione alla Virtualizzazione e al Ruolo dell'Hardware

La virtualizzazione riguarda la creazione di ambienti virtuali, in cui risorse hardware come la CPU e la memoria sono simulate tramite software. Questo processo ci permette di far sembrare di avere più risorse di quelle realmente disponibili. Un esempio comune è la creazione di **macchine virtuali** (VM) che operano all'interno di un sistema fisico. Questo approccio ha numerosi vantaggi pratici e teorici, come il miglior utilizzo delle risorse e l'isolamento tra diversi ambienti virtuali.

Virtualizzazione: Come Funziona?

concetto centrale della virtualizzazione è che, partendo da un singolo server fisico, possiamo creare più **macchine virtuali** che operano come se fossero sistemi autonomi. Ogni macchina virtuale può avere il proprio sistema operativo, e questi sistemi operativi possono essere diversi tra loro. Ad

esempio, possiamo eseguire **Windows dentro Linux**, **Linux dentro macOS**, e così via. Questo approccio è vantaggioso per testare software su diverse piattaforme senza necessitare di hardware separato per ciascun sistema operativo.

Un altro aspetto importante è l'**isolamento**. Ogni macchina virtuale funziona in modo indipendente, anche se tutte condividono la stessa risorsa hardware. Ciò significa che, se una macchina virtuale si blocca o viene compromessa, le altre non ne risentono.

I Vantaggi della Virtualizzazione

▼ **Flessibilità e Compatibilità**

Un vantaggio evidente della virtualizzazione è la possibilità di far girare più sistemi operativi su una singola macchina fisica. Gli sviluppatori, ad esempio, possono testare software progettato per diversi sistemi operativi senza la necessità di più macchine fisiche. Questo è particolarmente utile per applicazioni multiplattaforma, come quelle che devono funzionare su Windows, Linux e macOS.

▼ **Isolamento e Sicurezza**

Le macchine virtuali sono separate tra loro, il che crea un livello di **isolamento** che è utile per la sicurezza. Se un sistema operativo all'interno di una macchina virtuale viene compromesso, gli altri sistemi operativi e l'hardware principale restano intatti. Questo isolamento è un'importante difesa contro malware e attacchi informatici. Inoltre, le VM possono essere facilmente gestite, spostate, o ripristinate senza interferire con altre macchine virtuali.

▼ **Ottimizzazione delle Risorse**

Tradizionalmente, un'azienda avrebbe dovuto dedicare server fisici separati a ciascun servizio, come il sito web, la posta elettronica e il database. Con la virtualizzazione, più servizi possono essere eseguiti sulla stessa macchina fisica, migliorando l'efficienza delle risorse e riducendo i costi.

▼ **Facilità di Backup e Recupero**

La virtualizzazione semplifica anche la gestione dei backup. Le macchine virtuali possono essere **clonate** o salvate in **snapshot**, permettendo un rapido recupero in caso di guasti o attacchi informatici. Inoltre, la

virtualizzazione consente di migrare facilmente le macchine virtuali tra server fisici senza interruzioni significative.

L'Isolamento e la Sicurezza Informatica

Una delle principali ragioni per cui le macchine virtuali sono utilizzate è l'**isolamento** che offrono. Ogni macchina virtuale è indipendente dalle altre, e questo crea un ambiente sicuro per l'esecuzione di applicazioni o servizi. Immagina di avere un server che ospita diversi servizi critici, come un server web, un database e un servizio di posta elettronica. In passato, ciascuno di questi servizi avrebbe richiesto un server fisico separato.

Con la virtualizzazione, possiamo eseguire ciascun servizio su una macchina virtuale separata, anche se fisicamente si trovano sullo stesso hardware. In questo modo, se un attaccante compromette uno dei servizi, ad esempio il server web, gli altri servizi restano sicuri e isolati. Questo tipo di separazione rende molto più difficile per un hacker ottenere l'accesso all'intero sistema.

Sicurezza e Protezione da Attacchi

I problemi di **sicurezza informatica** sono una delle principali preoccupazioni quando si gestiscono servizi esposti su Internet, come i siti web aziendali o i server di posta elettronica. Gli hacker cercano spesso di sfruttare **vulnerabilità** nei software per accedere ai dati sensibili o per compromettere i sistemi.

Nel caso in cui un attaccante sfrutti una vulnerabilità in un servizio, la virtualizzazione aiuta a limitare i danni. Infatti, ogni servizio è isolato all'interno di una macchina virtuale, quindi, se una macchina viene compromessa, le altre restano sicure. Ad esempio, se un hacker riesce a infettare una macchina virtuale che gestisce il sito web, non avrà accesso alle altre macchine virtuali che gestiscono il database o la posta elettronica.

BUG nei S.O.

Un altro rischio riguarda i bug nel sistema operativo o nel kernel. Questi bug possono essere sfruttati per ottenere privilegi superiori (ad esempio, passando da un utente normale a root). Se un sistema non è aggiornato, un attaccante può sfruttare queste vulnerabilità per compromettere la sicurezza.

Virtualizzazione e container come soluzioni moderne di isolamento

In passato, per proteggere i servizi, venivano usate macchine fisiche separate. Oggi, la virtualizzazione e l'uso di container (come Docker) permettono di isolare i servizi in modo efficiente, riducendo la necessità di hardware fisico e migliorando la sicurezza senza compromettere le performance.

Hypervisor L1-L2