Architetture dei sistemi operativi:
Un sistema grande e complesso quale un sistema operativo può essere realizzato in molti modi diversi, Tradizionalmente, il sistema era sviluppato come un unico blocco (sistema **monolitico**) che esegue tutte le funzioni del sistema dalla gestione della CPU all'interfaccia utente. Non vi è alcun ordinamento tra le funzioni del sistema: tutte le funzioni e le strutture dati sono accessibili da qualsiasi punto del kernel. Manutenzione ed espansione sono molto difficili: ad esempio un bug in una funzione potrebbe causare il blocco dell'intero sistema, e trovarlo e risolverlo potrebbe essere difficile poiché il codice è molto integrato. Il vantaggio principale di quest'approccio è che il sistema è compatto, veloce ed efficiente, quindi è un approccio adatto a sistemi semplici.
Un'evoluzione è il sistema a **struttura gerarchica,** che organizza il sistema su livelli funzionali. Una funzione di un certo livello può chiamare solo funzioni di livello inferiore. Non vi è comunque una separazione tra le componenti del SO. Distinguere le dipendenze gerarchiche potrebbe non essere immediato. Manutenzione ed espansione rimangono difficili, ma sono più gestibili di un sistema monolitico.
Un primo approccio modulare è il sistema **stratificato**.
Il sistema è scomposto in un certo numero di livelli (il più basso è l'hw). Ogni livello è un modulo che implementa un componente del SO (gestione CPU, memoria, …), nasconde i propri dettagli implementativi agli altri e può comunicare con il livello sottostante attraverso un'interfaccia ben definita.
Lo sviluppo e il debug del sistema sono facilitati: quando si implementa un nuovo livello, si presume che quelli sottostanti funzionino correttamente, poiché sono già stati testati; e dopo aver implementato e testato anche il nuovo livello, si può passare a un livello superiore in modo analogo.
I vantaggi quindi sono la modularità e la facilità di sviluppo e manutenzione.
Gli svantaggi sono la possibile difficoltà nell'identificare i livelli e l'efficienza limitata: una chiamata a una funzione di un certo livello deve passare attraverso tutti i livelli sopra ad esso, riducendo le prestazioni del sistema.
Ulteriore evoluzione è l'approccio a **microkernel,** che struttura il sistema rimuovendo dal kernel tutte le funzioni non indispensabili, implementandole come servizi a livello utente. Il kernel così creato è molto piccolo e ha solo funzioni minime di gestione processi, memoria e comunicazione. Lo scopo principale del microkernel è di permettere la comunicazione tra programmi client e servizi, oltre che tra i servizi stessi così che possano richiedersi funzioni l'un l'altro. La comunicazione avviene tramite messaggi scambiati dal kernel, quindi c'è un grande sovraccarico di gestione, che incide sull'efficienza del sistema. Questo modello permette la massima separazione tra meccanismi e politiche (è sempre bene separare il "come si fa" dal "cosa si fa"), una facile modificabilità, la massima portabilità (basta reimplementare le poche funzioni del microkernel utilizzate dai servizi per far funzionare il sistema su una macchina diversa) e affidabilità: se un servizio fallisce, il resto del sistema rimane inalterato, e si può riavviare il servizio.
Struttura modulare:
Quest'approccio coinvolge l'uso di tecniche di programmazione a oggetti, ed è il più utilizzato.
Come nell'approccio a microkernel, il kernel possiede solo un insieme minimo di funzioni, ma quì si collega dinamicamente ai moduli che implementano le varie funzioni del sistema al boot e durante l'esecuzione. Come nel modello a strati, le implementazioni di ogni modulo sono nascoste agli altri ma hanno interfacce ben definite per comunicare. La comunicazione è diretta, migliorando così le prestazioni rispetto all'approccio a microkernel (rimangono però limitate). Gli altri vantaggi di quest'implementazione sono gli stessi dei sistemi a microkernel.
Sistema a macchine virtuali (VM):
Il kernel di un sistema a VM si occupa di fornire una copia esatta dell'hardware sottostante ad ogni macchina virtuale eseguita in modalità utente, ognuna delle quali esegue un SO (anche diversi tra loro). In questo modo, si può fornire l'illusione a ogni utente di avere una macchina propria: schedulazione e memoria virtuale si occupano di suddividere tempo di cpu e memoria tra le VM, mentre spooling e file system gli forniscono periferiche e dischi virtuali. Un ulteriore vantaggio è l'isolamento di ogni macchina virtuale dalle altre e dalla macchina reale, che le rende adatte a testare delle modifiche a un

sistema operativo senza causare tempi di inattività nel sistema. Sfortunatamente questo impedisce anche la condivisone diretta di dati e risorse tra le VM. È però possibile condividere un disco virtuale tra le VM, o creare una rete virtuale tra esse che gli consenta di comunicare pur rimanendo completamente isolate.
Lo svantaggio principale di questi sistemi è il calo di prestazioni dovuto allo strato di virtualizzazione: se ad esempio un processo in modalità utente virtuale esegue una richiesta di IO, questa dovrà passare prima al kernel del SO di quella VM, quindi al software di controllo della VM (che dovrà eventualmente interpretare la richiesta), e infine il kernel del sistema VM, in modalità reale fisica, potrà eseguire la richiesta. Questo riduce molto le prestazioni.