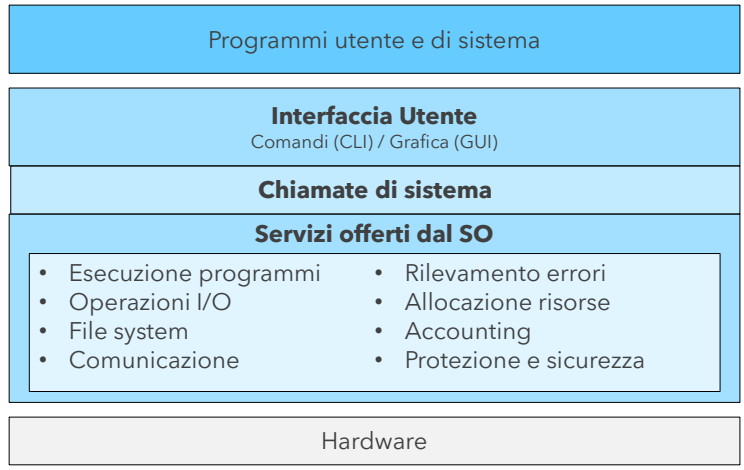
Struttura dei Sistemi Operativi

78

Servizi di un sistema operativo



79

Servizi Sistema Operativo

- Un **insieme dei servizi offerti dal Sistema Operativo** fornisce funzioni utili per l'utente:
  - **Interfaccia Utente** (User interface) – Quasi tutti i sistemi operativi hanno una interfaccia utente (UI)
    - Varia tra *Command-Line* (CLI), *Graphics User Interface* (GUI), *Batch*
  - **Esecuzione dei programmi** – Il sistema deve poter caricare un programma in memoria ed eseguirlo, terminare la sua esecuzione, sia normale che in caso di errore
  - **Operazioni di I/O** – Un programma in esecuzione può aver bisogno di I/O per l'accesso a file o per interagire con un dispositivo.
  - **Accesso a File-system** – Il File-system è di particolare interesse. I programmi hanno bisogno di leggere/scrivere file e directories, crearle e cancellarle, ricercarle, etc.

80

Servizi Sistema Operativo (Cont.)

- Un insieme dei servizi offerti dal Sistema Operativo fornisce funzioni utili per l'utente (Continua):
  - **Comunicazioni** – Processi possono scambiare informazioni, su uno stesso calcolatore o tra calcolatori in rete
    - La comunicazione può essere fatta attraverso memoria condivisa o attraverso invio di messaggi (pacchetti inviati dal SO)
  - **Rilevazione errori** – Il SO deve essere sempre attento a possibili errori
    - Possono avvenire nella CPU e memoria, in dispositivi I/O e in programmi utenti
    - Per ogni tipo di errore il SO dovrebbe effettuare l'azione appropriata per garantire una elaborazione corretta e consistente
    - Supporto per il debug può migliorare notevolmente la possibilità di usare efficientemente il sistema da parte dell'utente e del programmatore

81

Servizi Sistema Operativo (Cont.)

- Esiste un altro insieme di servizi del SO che servono a garantire l'uso efficiente del sistema stesso attraverso la condivisione delle risorse
  - **Allocazione risorse –** Quando più utenti o più job sono in esecuzione contemporaneamente, ognuno deve avere a disposizione delle risorse
    - Molti tipi di risorse - Alcuni come cicli di CPU, memoria primaria, e file storage possono avere una allocazione specifica, altri come dispositivi di I/O possono avere una gestione generale di tipo richiesta/rilascio.
  - **Accounting/Logging –** Per tenere traccia di quali utenti usano, quanto e quali tipi di risorse di sistema
  - **Protezione e sicurezza –** I possessori delle informazioni memorizzate in un sistema multiutente o in rete possono voler controllare l'uso delle informazioni, processi concorrenti non dovrebbero interferire tra loro
    - **Protezione** assicura che tutti gli accessi al sistema sono controllati
    - **Sicurezza** del sistema dall'esterno: richiede l'uso dell'autenticazione degli utenti, si estende alla difesa di dispositivi di I/O esterni dai tentativi di uso non valido.
    - Se un sistema deve essere protetto e sicuro, delle precauzioni devo essere prese in ogni sua parte. Una catena è forte come il suo anello più debole.

82

Interfacce utente del SO - CLI

Command Line Interface permette l'inserimento diretto di comandi

- Alcune volte implementato nel kernel, altre volte implementato nei programmi di sistema
- Alcune volte sono disponibili più tipi di CLI – **shells**
- Fondamentalmente riceve un comando dall'utente e lo esegue
  - Alcune volte i comandi sono built-in nell'interprete dei comandi, altre volte sono solo nomi dei programmi
    - Nell'ultimo caso aggiungere nuovi comandi non richiede di modificare la shell

83

Interfacce Utente del SO - GUI

- Interfaccia user-friendly che usa la metafora della scrivania (**desktop)**
  - Basata su mouse, tastiera, e monitor
  - **Icone** rappresentano file, programmi, azioni, ecc.
  - I pulsanti del mouse premuti su oggetti dell'interfaccia provocano varie azioni (danno informazioni, aprono opzioni, eseguono funzioni, aprono directory/cartelle)
  - Inventato a Xerox PARC (1973)
- Molti sistemi includono interfacce CLI e GUI
  - Microsoft Windows ha GUI con CLI "command" shell
  - Apple Mac OS X ha "Aqua" GUI con sotto un kernel UNIX e sono disponibili le shell unix
  - Solaris ha CLI e delle GUI (Java Desktop, KDE)

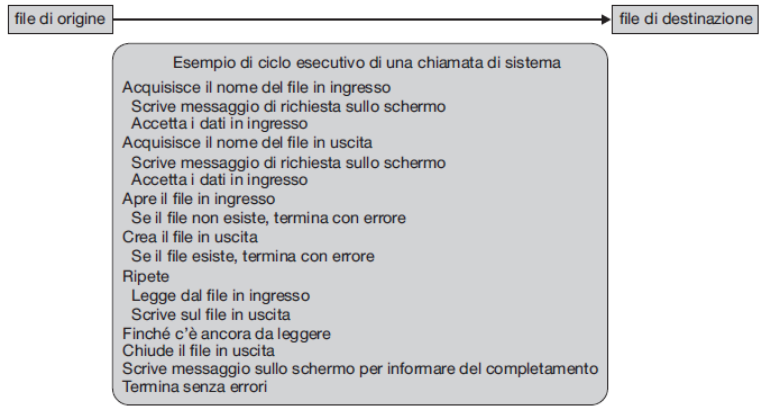
Chiamate di Sistema

- Interfaccia di programmazione verso i servizi forniti dal SO
- Tipicamente scritte in C/C++
- Principalmente usate dai programmi attraverso **Application Program Interface** (API) di alto livello piuttosto che attraverso l'uso diretto della chiamata di sistema
- Le tre API piu' comuni sono: Win32 API per Windows, POSIX API per sistemi POSIX-based (includendo tutte le versioni di UNIX, Linux, e Mac OS X), e Java API per la Java virtual machine (JVM)

(Nota che I nomi delle chiamate di sistema riportate sono generici)

Esempio di Chiamata di Sistema

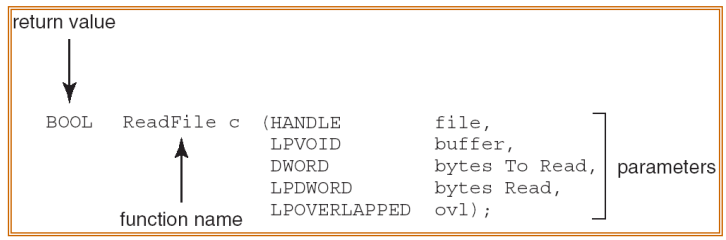
- Sequenza di chiamate di sistema per copiare il contenuto di un file su un altro file



86

Esempio di Standard API

- Consideriamo la funzione *ReadFile()*
- Fa parte delle Win32 API– una funzione per leggere da un file



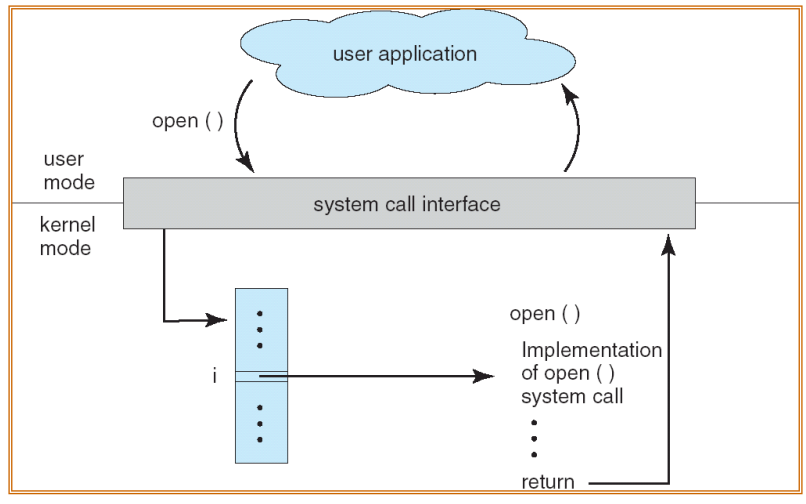
- Parametri:
  - HANDLE file – riferisce al file da leggere
  - LPVOID buffer – un buffer dove saranno scritti i dati letti dal file
  - DWORD bytesToRead – il numero di bytes da leggere da file
  - LPDWORD bytesRead – il numero di bytes letti nell'ultima operazione di lettura
  - LPOVERLAPPED ovl – indica se deve usare overlapped I/O (asincrono)

87

Implementazione delle Chiamate di Sistema

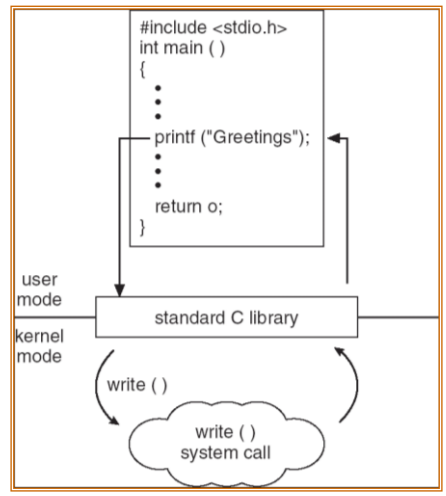
- Tipicamente **ogni system call è identificata da un numero**
  - L'interfaccia per le System-call mantiene una tabella indicizzata in base a questi numeri
- L'interfaccia delle chiamate di sistema invoca la chiamata di sistema a livello di kernel e ritorna lo stato della chiamata e ogni valore di ritorno
- Il chiamante non ha necessità di sapere come la chiamata di sistema è implementata
  - Deve solo seguire la API e capire cosa il SO farà come risultato della chiamata
  - I dettagli della interfaccia sono nascosti al programmatore dalle API
    - Gestite da librerie di supporto a run-time (insieme di funzioni in librerie incluse con il compilatore)

API – Chiamata di Sistema – SO



Esempio Liberia Standard C

- Programma C che invoca printf() funzione di libreria, che chiama la write() system call



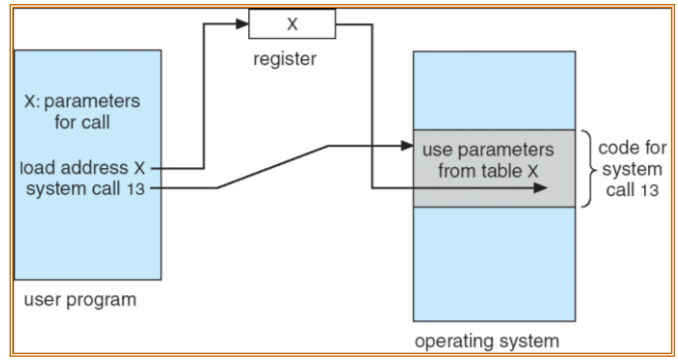
90

Passaggio dei Parametri a Chiamata di Sistema

- Tre metodi generali sono usati per passare parametri a SO
  - Piu' semplice: passare i **parametri nei registri della CPU**
    - In alcuni casi si possono avere piu' parametri che registri
    - Usato da Linux
  - Parametri **memorizzati in un blocco o tabella in memoria**, l'indirizzo del blocco viene passato come parametro in un registro
    - Usato da Solaris
  - Parametri *pushed* sullo *stack* dal programma e *popped* dallo stack dal SO
  - Metodi che usano il blocco e lo stack non limitano il numero e la lunghezza dei parametri passati.

91

Passaggio parametri con Tabella



92

Tipi di Chiamate di Sistema

- Controllo processi
- Gestione File
- Gestione Dispositivi
- Comunicazioni

93

Linux system calls

- **Le system call** sono chiamate tramite istruzione **INT 80h** (interrupt sincrona chiamata esplicitamente) e registro ***EAX contiene il numero della system call*** da eseguire, gli altri registri conterranno altri parametri necessari alla system call
- per un elenco delle system call si veda https://chromium.googlesource.com/chromiumos/docs/+/HEAD/constants/syscalls.md
- **INT 80h** passa dalla modalità utente alla modalità kernel, su processori intel a 32 e 64 bit viene usata istruzione specifica *sysenter/sysexit* e *syscall/sysret* (più veloci della INT)

Linux system calls

- Il kernel nella procedura associata all' interruzione **80h** (ISR) può controllare se la chiamata è valida e usando una tabella interna al kernel può chiamare la funzione associata alla syscall indicata da registro EAX. Al termine in EAX è presente il risultato, se <0 *fallimento*, >=0 *successo*
- *Esempio:*

```
MOV EAX, 04h      // write syscall
MOV EBX, 1        // standard output
MOV ECX, buffer   // puntatore al buffer di caratteri da stampare
MOV EDX, count    // numero di caratteri del buffer da stampare
INT 80h           // esegue la syscall
```

Programmi di Sistema

- I programmi di sistema forniscono un ambiente per lo sviluppo di programmi e per la loro esecuzione. Possono essere divisi in programmi per:
  - La manipolazione di file
  - Ottenere informazioni sullo stato
  - La modifica di file
  - Il supporto di linguaggi di programmazione
  - Il caricamento di programmi e loro esecuzione
  - Le comunicazioni
  - Programmi applicativi
- Per la maggior parte degli utenti un sistema operativo è definito dai programmi di sistema disponibili, non dalle chiamate di sistema fornite.

Programmi di Sistema

- Forniscono un ambiente per lo sviluppo ed esecuzione dei programmi
  - Alcuni sono semplicemente delle interfacce per le chiamate di sistema; altri sono considerevolmente più complessi.
- **Gestione file** – Crea, cancella, copia, rinomina, stampa, lista e generalmente manipola file e directory.
- **Informazione sullo stato**
  - Alcuni chiedono al sistema informazioni – data e ora, quantità di memoria disponibile, uso spazio disco, numero di utenti
  - Altri forniscono informazioni dettagliate per l'analisi della performance, per il logging, e per il debug
  - Tipicamente questi programmi stampano l'output su terminale o altri dispositivi di output
  - Alcuni sistemi implementano un registro – usato per memorizzare e reperire informazioni sulla configurazione

Programmi di sistema (continua)

- **Modifica di file**
  - Editor di testo per creare e modificare file
  - Comandi speciali per cercare nei file o per effettuare trasformazioni del testo
- **Supporto per i linguaggi di programmazione** – Compilatori, assemblatori, debugger e interpreti
- **Caricamento programmi ed esecuzione** - Absolute loaders, relocatable loaders, linkage editors, e overlay-loaders, programmi per il debug per linguaggi di alto livello e per linguaggio macchina
- **Comunicazione** – Fornisce meccanismi per creare connessioni virtuali tra processi, utenti e calcolatori
  - Permettere agli utenti di scambiarsi messaggi, navigare pagine web, inviare e-mail, collegarsi in modo remoto, trasferire file tra macchine diverse.

98

Progettazione e implementazione di Sistemi Operativi

- Progettazione e implementazione di un sistema operativo è un problema difficile, ma alcuni approcci si sono dimostrati di successo
- La struttura interna di sistemi operativi differenti può variare moltissimo
- Si inizia definendo gli obiettivi (goals) e le caratteristiche del sistema
- Dipende dall'hardware scelto e dal tipo di sistema (a lotti, in time-sharing, mono/multiutente, realtime o uso generale)
- *User* goals e *System* goals
  - User goals – il sistema operativo dovrebbe essere facile da usare, facile da imparare, affidabile, sicuro e veloce
  - System goals – il sistema operativo dovrebbe essere facile da progettare, implementare e manutenere, così come essere flessibile, affidabile e efficiente

99

Progettazione e implementazione di Sistemi Operativi (continua)

- Un principio importante è di separare

  Criteri o politiche: Cosa sarà fatto?
  Meccanismi: Come sarà fatto?
- I meccanismi determinano come fare qualcosa, i criteri decidono cosa sarà fatto
  - La separazione tra criteri e meccanismi è un principio molto importante permette la massima flessibilità se i criteri cambiano nel tempo
- Esempio:
  - Il *Timer* è un ***meccanismo***
  - La decisione di quanto tempo assegnare ad un processo riguarda un ***criterio***
- Per l'assegnazione delle risorse sono importanti i criteri

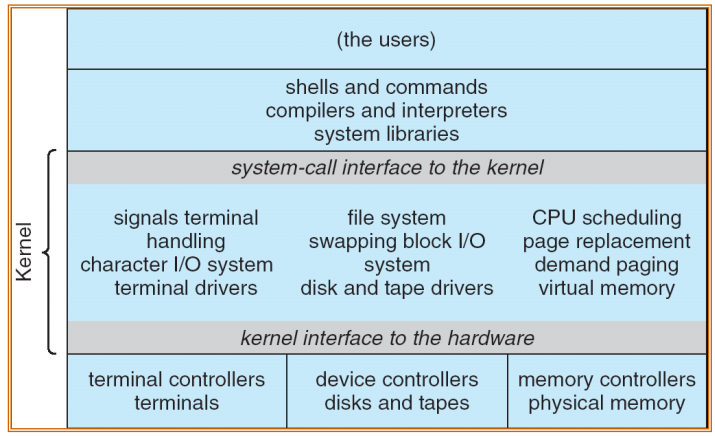
100

Struttura monolitica

- Approccio più semplice
- Il kernel è un unico file binario ed ha un unico spazio di indirizzamento
- Al boot viene caricato e messo in esecuzione
- Unix e Linux usavano questo approccio
- **Problema:** device driver integrati nel kernel, difficile supportare nuovi device, implica ricompilare tutto il kernel e fare reboot del sistema.

101

Struttura di UNIX



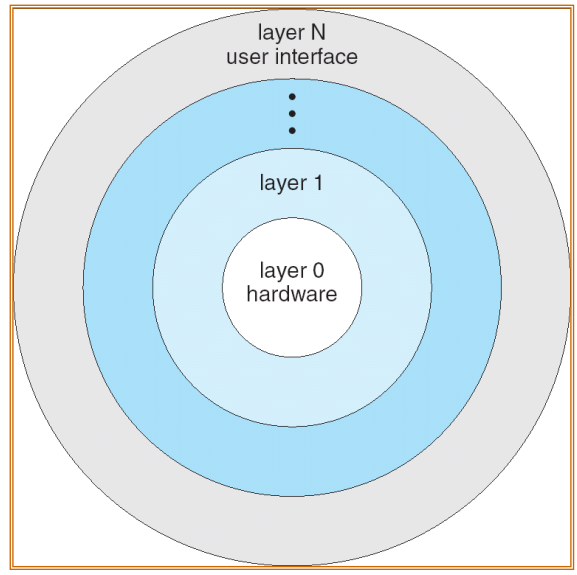
102

Approccio stratificato

- Il sistema operativo è diviso in un numero di strati (layer), ognuno costruito sulla base degli stati inferiori. Lo stato più basso (layer 0), è l'hardware; lo stato più alto (layer N) è la interfaccia utente.
- Con la modularità, gli strati sono selezionati in modo che ognuno usi funzioni e servizi dei soli strati di livello inferiore
- Facile da verificare e testare
- Difficile definire bene gli strati
- Può introdurre inefficienze dovute all'attraversamento dei vari strati in una chiamata.

103

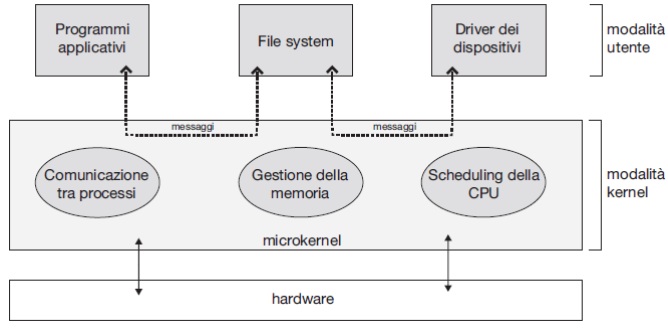
Sistema Opertivo Stratificato



104

Struttura a Microkernel

- Sposta il più possibile dal kernel nello spazio "utente"
- Il kernel si occupa solo di: **gestione processi**, **gestione memoria** e **comunicazione tra processi**
- La comunicazione tra i moduli utente usa l'invio di messaggi

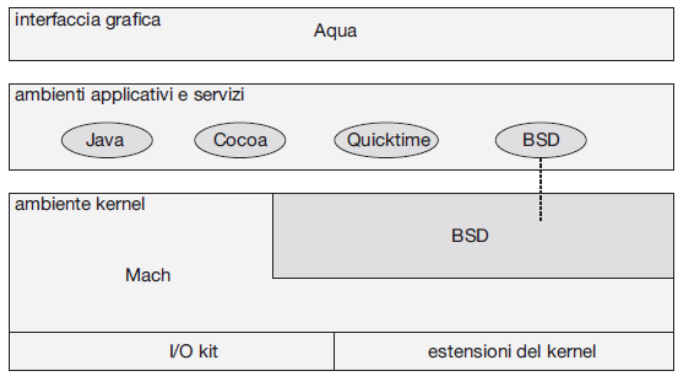


105

Struttura a Microkernel

- Benefici:
  - Più facile da estendere
  - Più facile portare il sistema operativo su nuove architetture hardware
  - Più affidabile (meno codice in esecuzione in modo kernel)
  - Più sicuro
- Demeriti:
  - Diminuzione di performance dovuto alla comunicazione tra spazio utente e kernel

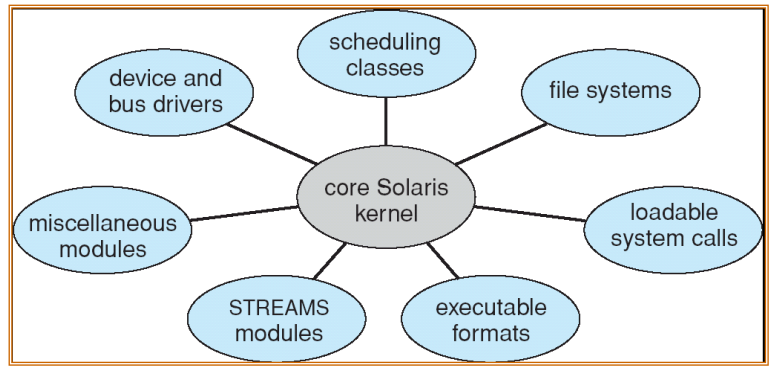
macOS Structure



Moduli

- La maggior parte dei moderni sistemi operativi implementano i moduli kernel
  - Usa un approccio object oriented
  - Ogni componente base è separato
  - Ogni modulo comunica con gli altri moduli attraverso interfacce ben definite
  - Ogni modulo viene caricato in base alle necessità nel kernel
- Approccio simile alla stratificazione ma più flessibile

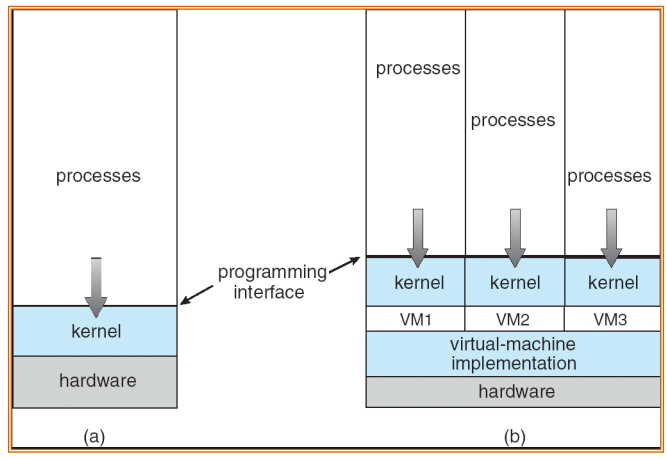
Approccio Modulare di Solaris



Macchine Virtuali

- **Una macchina virtuale fornisce una interfaccia identica all'hardware sottostante**
- Il sistema operativo crea l'illusione di disporre di più macchine ognuna in esecuzione sul suo processore e con la propria memoria (virtuale)
- Le risorse del computer fisico sono condivise per creare le macchine virtuali
  - Lo scheduling della CPU crea l'illusione che ogni VM abbia il proprio processore
  - Il file system può fornire uno spazio di memorizzazione per ogni macchina virtuale
- Nate per mainframe IBM nel 1972 (IBM VM/370), oggi ritornate in uso comune (VMware)
- Utilizzate nei datacenter per utilizzare al meglio le risorse di calcolo

Macchine Virtuali (Continua)



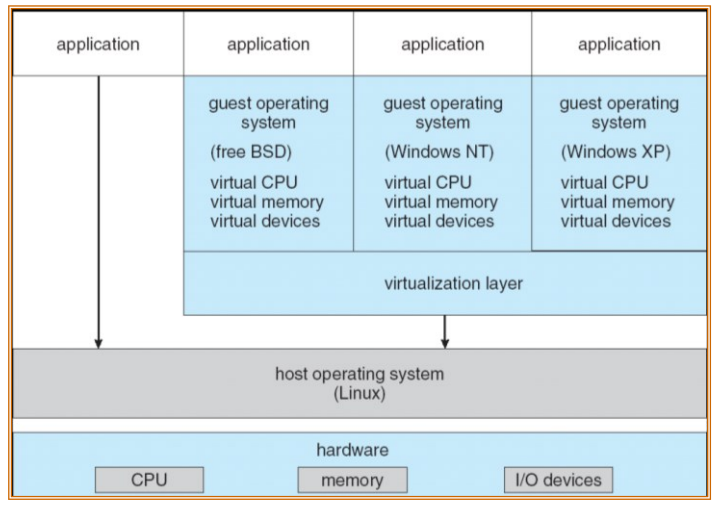
(a) macchina non-virtuale (b) macchina virtuale

Macchine Virtuali (Continua)

- Le macchine virtuali forniscono la completa protezione delle risorse del sistema, infatti **ogni macchina virtuale è isolata dalle altre macchine virtuali**. L'isolamento, comunque, non permette la condivisione diretta di risorse.
- Una macchina virtuale è il modo perfetto per fare ricerca e sviluppo sui sistemi operativi. Lo sviluppo viene fatto su una macchina virtuale invece che sulla macchina fisica in modo da non disturbare il normale uso del sistema.
- La virtualizzazione di una macchina è difficile da implementare dovuto allo sforzo necessario a fornire un duplicato esatto della macchina fisica sottostante e a fare questo in modo efficiente
- **Problema:** eseguire codice kernel (del SO ospite) in modo utente (nel SO ospitante)

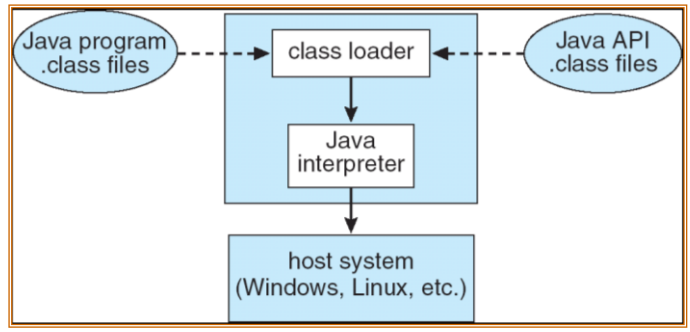
112

Architettura VMware



113

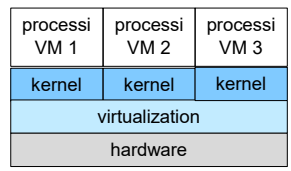
Macchina Virtuale Java



- Più corretto chiamarla **Macchina Astratta**, ha un proprio linguaggio macchina (bytecode) eseguito su più SO host
- Usato per realizzare applicazioni multipiattaforma

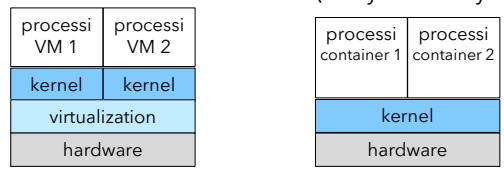
Container Applicativi

- **Problema:** ogni VM in esecuzione su un host implica uso di memoria e CPU per il kernel del sistema operativo e per i processi del SO, questo limita il numero di VM che possono essere eseguite su un host. Inoltre avviare una VM non è immediato.
- Molto spesso tutti i sistemi operativi usati nelle VM sono identici (es Linux) e sono utilizzabili solo tramite la rete (forniscono servizi in cloud), non hanno una interfaccia GUI ma solo una CLI (per amministrazione).
- E' ciò che accade nei datacenter di google, facebook, etc.



Container Applicativi

- L'idea è di avere **un solo kernel che raggruppa i processi in dei «container»** ad ognuno dei quali sono associati:
  - un proprio file system,
  - una propria interfaccia di rete virtuale
- il kernel isola i vari processi che fanno parte del container dagli altri processi degli altri container
- Vantaggi:
  - Minor occupazione di memoria e CPU e overhead di virtualizzazione
  - Avvio di un nuovo container molto veloce rispetto ad avvio di una VM
  - Più facile installare un container (filesystem a layer)



116

Windows 10 WSL

- **Windows Subsystem for Linux**, permette esecuzione di file eseguibili binari linux su windows
- Va installato esplicitamente
  - https://docs.microsoft.com/it-it/windows/wsl/install-win10
- WSL1 (2016)
  - Implementato come interfaccia, traduce le chiamate di sistema linux in chiamate di sistema di windows
  - Problemi con alcune chiamate
- WSL2 (2019)
  - Usa un vero kernel linux in esecuzione in una VM Hyper-V (il sistema di virtualizzazione di microsoft)

117