

Sistemi Operativi

Docente: Ugo Erra

ugoerr+so@dia.unisa.it



2° LEZIONE

STRUTTURE DEI SISTEMI OPERATIVI

***CORSO DI LAUREA TRIENNALE IN INFORMATICA
UNIVERSITA' DEGLI STUDI DELLA BASILICATA***



Sommario della lezione



- Servizi di un Sistema Operativo
- Interfaccia utente del Sistema Operativo
- System calls
- Tipi di System calls
- Progettazione e realizzazione di un Sistema Operativo
- Struttura di un Sistema Operativo
- Macchine Virtuali

Progettazione di un Sistema Operativo



- La progettazione di un Sistema Operativo è un compito tra i più complessi dell'ingegneria del software
- In base al tipo di sistema desiderato si definiscono i metodi e gli algoritmi necessari
- Un Sistema Operativo può essere visto attraverso tre punti di vista:
 - Utente
 - Programmatore
 - Progettista

Servizi di un Sistema Operativo



- Un Sistema Operativo deve fornire un insieme di funzioni utili agli utenti:
 - **Interfaccia con l'utente**
 - ✦ Tutti i sistemi operativo forniscono una interfaccia utente (*User Interface*) che può essere a riga di comando (*Command Line Interface*) oppure una interfaccia grafica (*Graphics User Interface*)
 - **Esecuzione di un programma**
 - ✦ Il sistema deve essere in grado di caricare in memoria dei programmi, mandarli in esecuzione e terminare in maniera corretta l'esecuzione anche in presenza di errori
 - **Operazioni di I/O**
 - ✦ Un programma in esecuzione necessita di I/O da un dispositivo
 - **Gestione del file system**
 - ✦ Il file system permette ai programmi di leggere e scrivere file e directory, crearle, cancellarle e la gestione dei permessi di lettura e scrittura

Servizi di un Sistema Operativo



- Un Sistema Operativo deve fornire un insieme di funzioni utili agli utenti:
 - **Comunicazioni**
 - ✦ I processi possono scambiarsi informazioni, sullo stesso computer o attraverso una rete tra computer disposti su una rete
 - ✦ La comunicazione può avvenire attraverso una memoria condivisa o attraverso il *message passing* (cioè pacchetti spostati all'interno di S.O.)
 - **Rilevamento d'errori**
 - ✦ Un Sistema Operativo deve essere costantemente capace di rilevare possibili errori
 - ✦ Gli errori possono presentarsi sulla CPU, in memoria, sui dispositivi di I/O o nei programmi utenti
 - ✦ Per ogni tipo di errore, il Sistema Operativo deve intraprendere l'azione opportuna per assicurare una continuazione corretta
 - ✦ Il debug solitamente facilita enormemente l'usabilità del sistema da parte degli utenti e dei programmatori

Servizi di un Sistema Operativo



- Un altro insieme di funzioni è essenziale per assicurare il corretto funzionamento del sistema attraverso le risorse condivise
 - **Assegnazione delle risorse**
 - ✦ In un ambiente con più utenti diversi processi sono in esecuzione e le risorse devono essere assegnate ad ognuno di loro
 - ✦ Le risorse da assegnare possono essere di diverso tipo – cicli di CPU, memoria, spazio su disco, dispositivi di I/O
 - **Contabilizzazione dell'uso delle risorse**
 - ✦ Tenere traccia degli utenti e della quantità di risorse che possono utilizzare
 - **Protezione e sicurezza**
 - ✦ Le informazioni devono poter essere archiviate e gestite secondo politiche di protezione, inoltre i processi non devono poter interferire l'uno con l'altro
 - ✦ Se un sistema deve essere protetto e sicuro, al suo interno devono esistere delle precauzioni ovunque. La forza di una catena è esattamente quella del suo anello più debole

Interfaccia utente del Sistema Operativo



- Vi sono due modi per comunicare con un Sistema Operativo
 - Interprete dei comandi
 - Interfaccia grafica

Interprete dei comandi



- L'interprete dei comandi permette di impartire comandi da eseguire
 - Può essere implementata nel kernel da un programma di sistema
 - Solitamente un utente ha la possibilità di scegliere diversi interpreti di comando chiamate **shell**
 - Una shell fondamentalmente preleva il comando impartito e lo esegue
 - ✦ Il comando può essere implementato internamente alla shell oppure può essere un programma esterno da avviare
 - ✦ Nel caso in cui la shell esegue programmi esterni un eventuale aggiunta di nuovo comandi non comporta la modifica della shell

Interfaccia grafica



- La metafora del desktop rappresenta l'interfaccia verso il sistema
 - Di solito si utilizzando mouse, tastiera, e monitor
 - Le icone rappresentano file, programmi o altro
 - L'interazione avviene usando il mouse sopra gli oggetti del desktop generando opportune azioni da gestire (apri, sposta, esegui, etc...)
 - Inventata alla Xerox PARC...copiata dalla Apple Computer ☺
- Molti sistemi includono sia una CLI sia una GUI
 - Microsoft Windows usa una GUI ma è possibile adoperare anche una CLI
 - Apple MacOS X usa l'interfaccia grafica "Aqua" ma essendo un sistema basato su UNIX è possibile adoperare anche una shell
 - Solaris utilizza prevalentemente una CLI ma è possibile installare anche una GUI (Java Desktop, KDE)

Chiamate di sistema

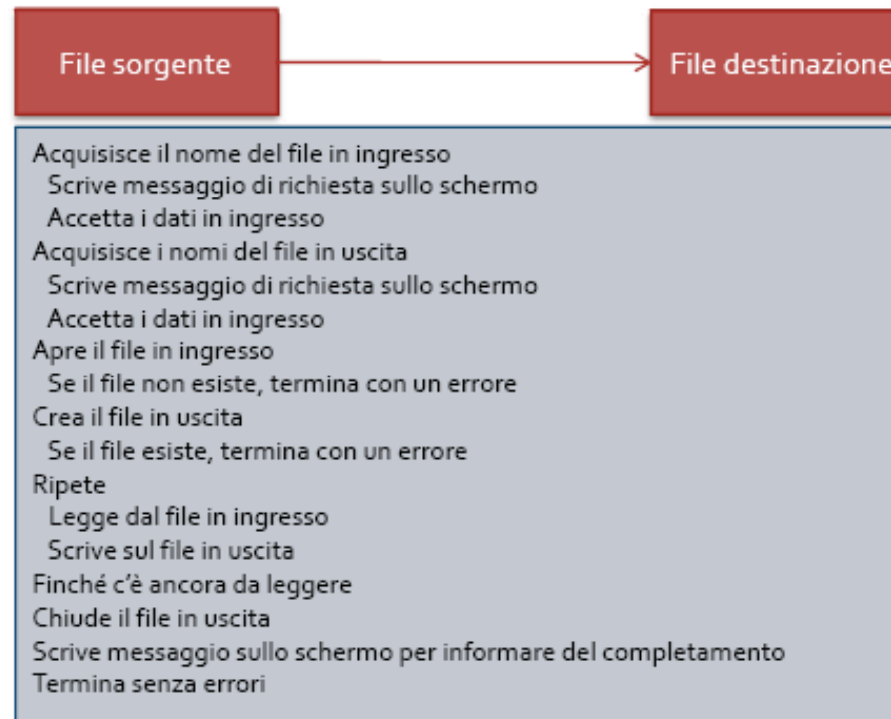


- Le **chiamate di sistema** (o **system call**) rappresentano l'interfaccia ai servizi forniti dal Sistema Operativo
- Tipicamente sono scritte in un linguaggio ad alto livello (C/C++)
- Sono principalmente accessibili attraverso una **API (Application Program Interface)** e quasi mai direttamente anche se è possibile
- Le API più comuni sono:
 - Win32 API per Windows
 - POSIX API per i sistemi POSIX compatibili (UNIX, Linux, MacOS X)
 - Java API per la Java virtual machine
 - Perché utilizzare un API piuttosto che utilizzare direttamente le system call?

Esempio di System Call



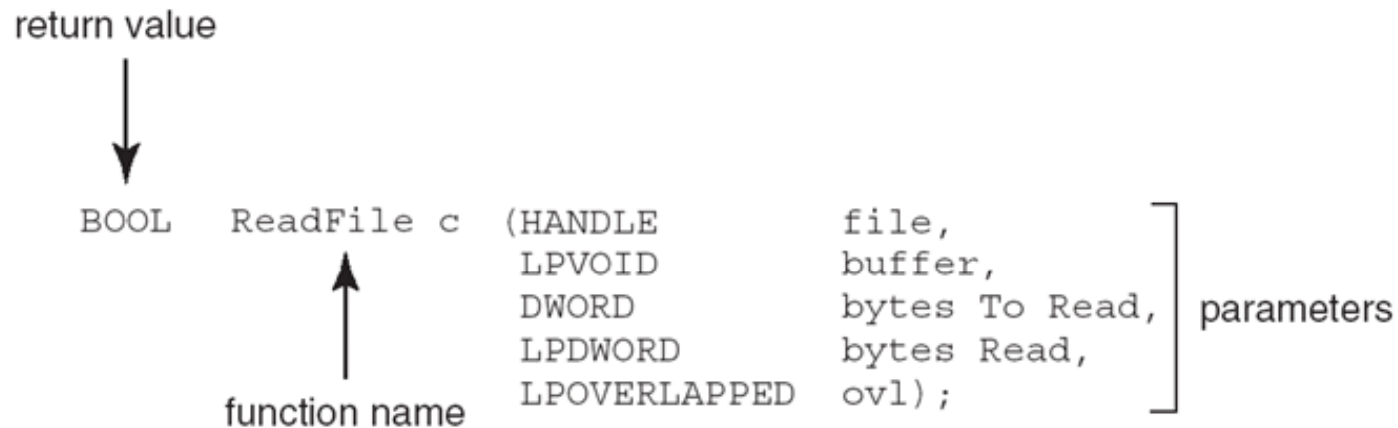
- La sequenza delle system call per poter copiare il contenuto di un file in un altro file



Esempio di una chiamata di sistema



- Consideriamo la funzione Win32 ReadFile
 - Una funzione per leggere dati da un file



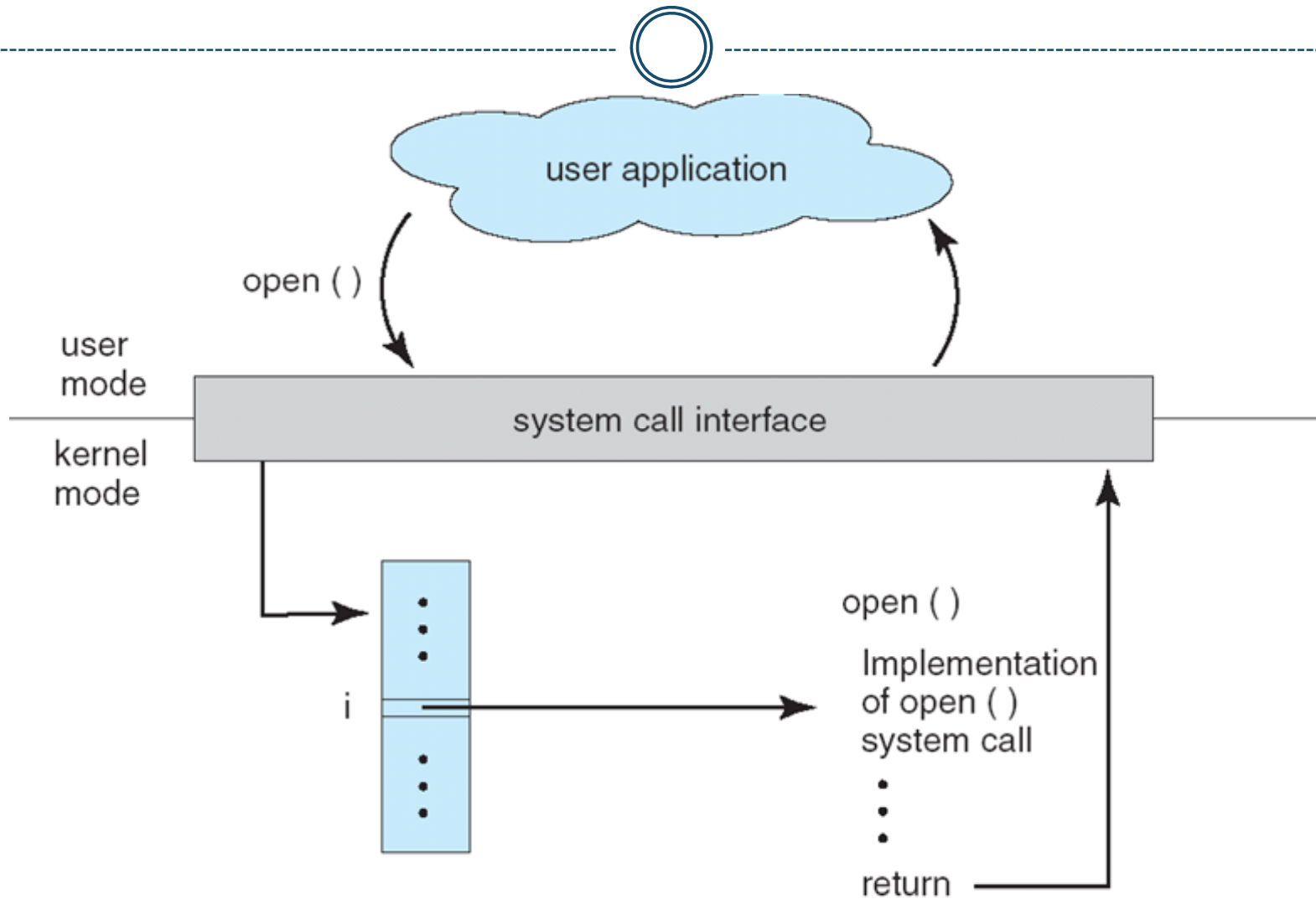
- I parametri passati alla funzione sono:
 - HANDLE file — file da cui leggere
 - LPVOID buffer — buffer da cui leggere i dati provenienti dal file
 - DWORD bytesToRead — numero di byte da trasferire dal file al buffer
 - LPDWORD bytesRead — numero di byte effettivamente letti nell'ultima invocazione
 - LPOVERLAPPED ovl — indicata l'uso dello spooling

In che modo accediamo alle system call



- Tipicamente ad ogni system call è associato un numero
- Il Sistema Operativo mantiene una tabella con tutti gli indici delle system call
- L'interfaccia della system call (cioè la funzione) invoca la chiamata all'interno del kernel del Sistema Operativo e ritorna lo stato della system call e un certo numero di valori
 - Il programmatore non ha bisogno di sapere come è implementata la system call
- Devo solo capire come funziona l'API e cosa restituirà la chiamata
- In questo modo molti dettagli dell'interfaccia del S.O. sono nascosti al programmatore attraverso l'API
- Normalmente al programmatore è fornito un sistema di supporto all'esecuzione (o *run-time support library*) come un insieme di funzioni incluse nel compilatore

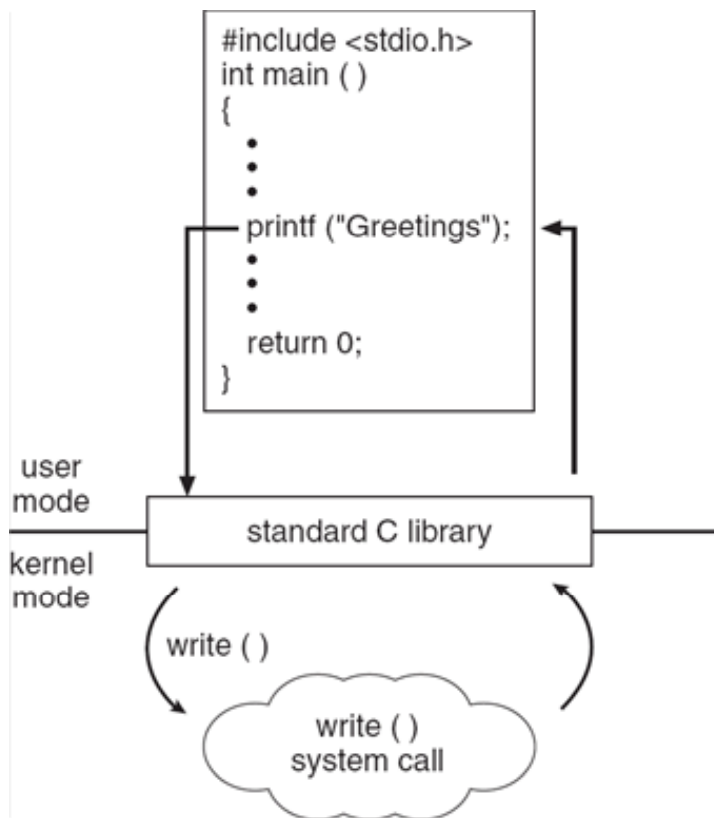
Gestione della chiamata di sistema `open ()`



Gestione di `write()` della libreria standard del C



- Un programma C che invoca la funzione `printf()`, chiamerà a sua volta la system call `write()`

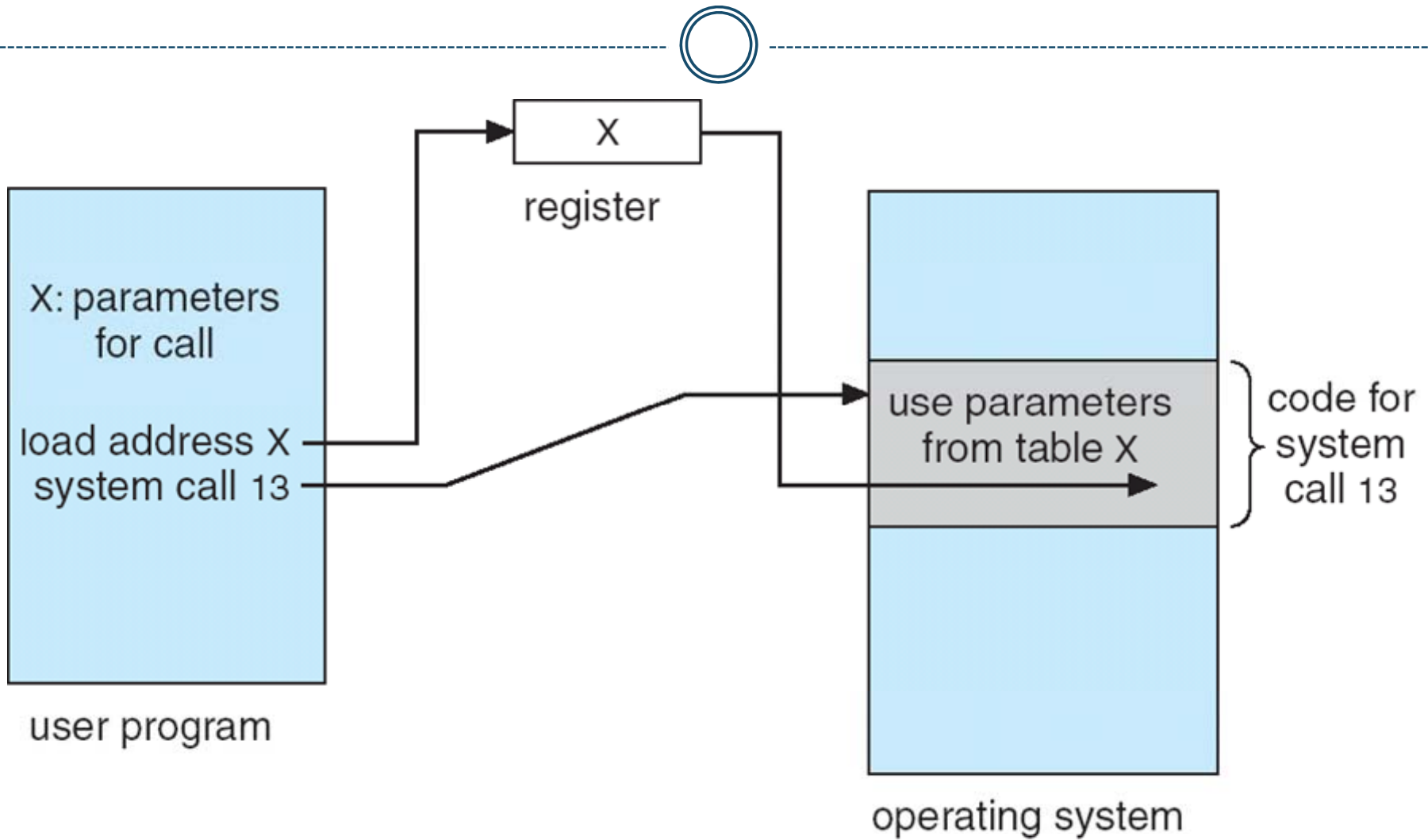


Passaggio dei parametri nelle system call



- Per poter chiamare una system call abbiamo bisogno di passargli un insieme di parametri
- E' possibile adoperare tre metodi per il passaggio dei parametri
 - Passare tutti i parametri nei registri
 - ✦ In alcuni casi ci potrebbero essere più parametri che registri
 - I parametri sono memorizzati in un blocco in memoria, l'indirizzo del blocco è passato come parametro in un registro
 - ✦ Approccio utilizzato da Linux e Solaris
 - ✦ Non limita il numero dei parametri che è possibile passare
 - I parametri sono passati attraverso lo stack, inserendoli (*push*) ed estraendoli (*pop*) dallo stack attraverso il sistema operativo

Passaggio dei parametri per tabella



Categoria di chiamate di sistema



- Le chiamate di sistema sono classificabili in cinque categorie:
 - Controllo dei processi
 - Gestione dei file
 - Gestione dei dispositivi
 - Gestione delle informazioni
 - Comunicazioni

Controllo dei processi



- Terminazione normale ed anormale
- Caricamento, esecuzione
- Creazione ed arresto di un processo
- Esame ed impostazioni degli attributi di un processo
- Attesa per il tempo indicato
- Attesa e segnalazione di un evento
- Assegnazione e rilascio della memoria

Gestione dei file



- Creazione e cancellazione dei file
- Apertura e chiusura
- Lettura, scrittura e posizionamento
- Esame ed impostazione degli attributi di un file

Gestione dei dispositivi



- Richiesta e rilascio di un dispositivo
- Lettura, scrittura e posizionamento
- Esame ed impostazione degli attributi di un dispositivo
- Inserimento logico ed esclusione logica di un dispositivo

Gestione delle informazioni



- Esame ed impostazione dell'ora e della data
- Esame ed impostazione dei dati del sistema
- Esame ed impostazione degli attributi dei processi, file e dispositivi

Comunicazione



- Creazione e chiusura di una connessione
- Invio e ricezione dei messaggi
- Informazioni sullo stato di un trasferimento
- Inserimento ed esclusione di dispositivi remoti

Programmi di sistema



- Un sistema operativo fornisce un ambiente conveniente per lo sviluppo e l'esecuzione dei programmi. Normalmente possono essere divisi in:
 - Gestione dei file
 - Informazioni di stato
 - Modifica dei file
 - Ambienti d'ausilio alla programmazione
 - Caricamento ed esecuzione dei programmi
 - Comunicazioni
- La maggior parte degli utenti normalmente ha una visione del sistema operativo attraverso le applicazioni più che attraverso le chiamate di sistema
 - Ma ogni applicazione utilizza in fondo sempre le stesse chiamate di sistema

Progettazione e realizzazione di un S.O. - 1



- Non esiste un metodo completo per la progettazione e la realizzazione di Sistema Operativo, ma alcuni approcci si sono dimostrati efficaci
- La struttura interna di un Sistema Operativo può variare ampiamente
- Normalmente si parte definendo gli scopi e le specifiche
 - Considerando anche l'hardware ed il tipo di sistema a disposizione
- Possono essere imposti due tipi di obiettivi:
 - Utenti
 - ✦ Gli utenti vogliono un sistema operativo che sia semplice da usare, da imparare, affidabile, sicuro e veloce
 - Sistema
 - ✦ Il Sistema Operativo deve essere semplice da progettare, da implementare e da mantenere, flessibile, affidabile, privo di errori ed efficiente

Progettazione e realizzazione di un S.O. - 2



- Due importanti principi da nella progettazione e nella realizzazione di un Sistema Operativo separare sono:
Meccanismi: Come lo facciamo?
Criteri: Cosa deve essere fatto?
- Un meccanismo determina come eseguire qualche cosa, un criterio decide che cosa si debba fare
 - Questa separazione dei criteri dai meccanismo è un principio importante, la massima flessibilità di progettazione la possiamo ottenere solo se i criteri possono essere cambiati nel futuro

Esempio - 1



- Supponiamo di voler terminare i processi che entrano in un loop infinito
- Possiamo adoperare un **timer variabile che ad ogni impulso** decrementa un contatore
- Il Sistema Operativo assegna per ogni processo un valore ad un contatore
- Quando il contatore assume valore negativo il processo viene terminato
 - Il Sistema Operativo decide che il processo ha superato il tempo di esecuzione previsto per quel processo
- Il timer è un *meccanismo* che garantisce che nessun processo possa essere eseguito troppo a lungo sulla CPU
- La quantità di tempo da impostare riguarda i *criteri*

Esempio - 2



- Nell'assegnazione delle risorse le domande a cui dobbiamo rispondere sono:
 - Come (*meccanismo*) facciamo ad assegnare correttamente le risorse ad ogni processo?
 - Che cosa (*criterio*) dobbiamo fare per poter assegnare correttamente le risorse ad ogni processo?

Struttura del Sistema Operativo



- Un Sistema Operativo può essere progettato strutturandolo in diversi modi, tra i più comuni abbiamo
 - Struttura semplice
 - Metodo stratificato
 - Microkernel

Struttura semplice

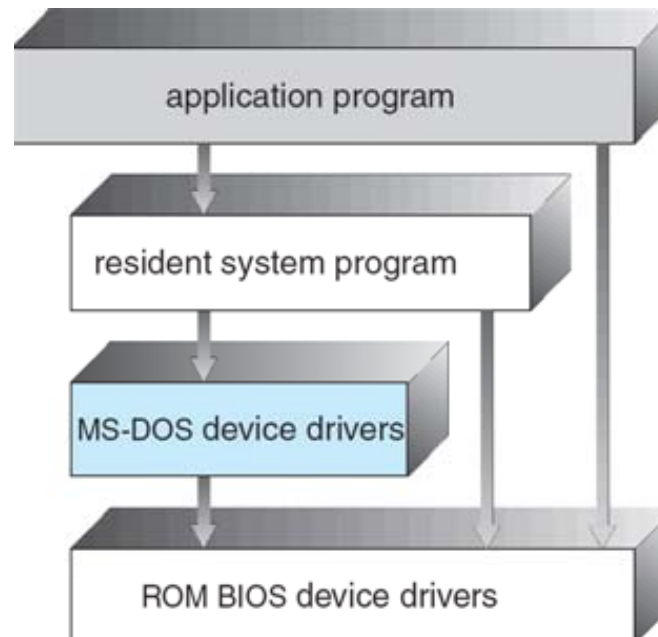


- I primi sistemi operativi erano limitati dal punto di vista dell'hardware
 - Ad esempio i microprocessori 8086 non avevano la modalità kernel quindi gli utenti potevano facilmente mettere in crisi il sistema o bloccare il sistema
- Lo scopo principale era quello di fornire quante più funzioni possibili nel modo più semplice e diretto cercando di mantenere la complessità del sistema al minimo
- Il concetto di system call era molto limitato e molte operazioni erano svolte accedendo direttamente all'hardware

MS-DOS



- Il Sistema Operativo MS-DOS fu progettato per fornire il massimo delle funzionalità nel minor spazio possibile
 - Non era diviso in moduli
 - Le interfacce e i livelli di funzionalità non erano ben separate

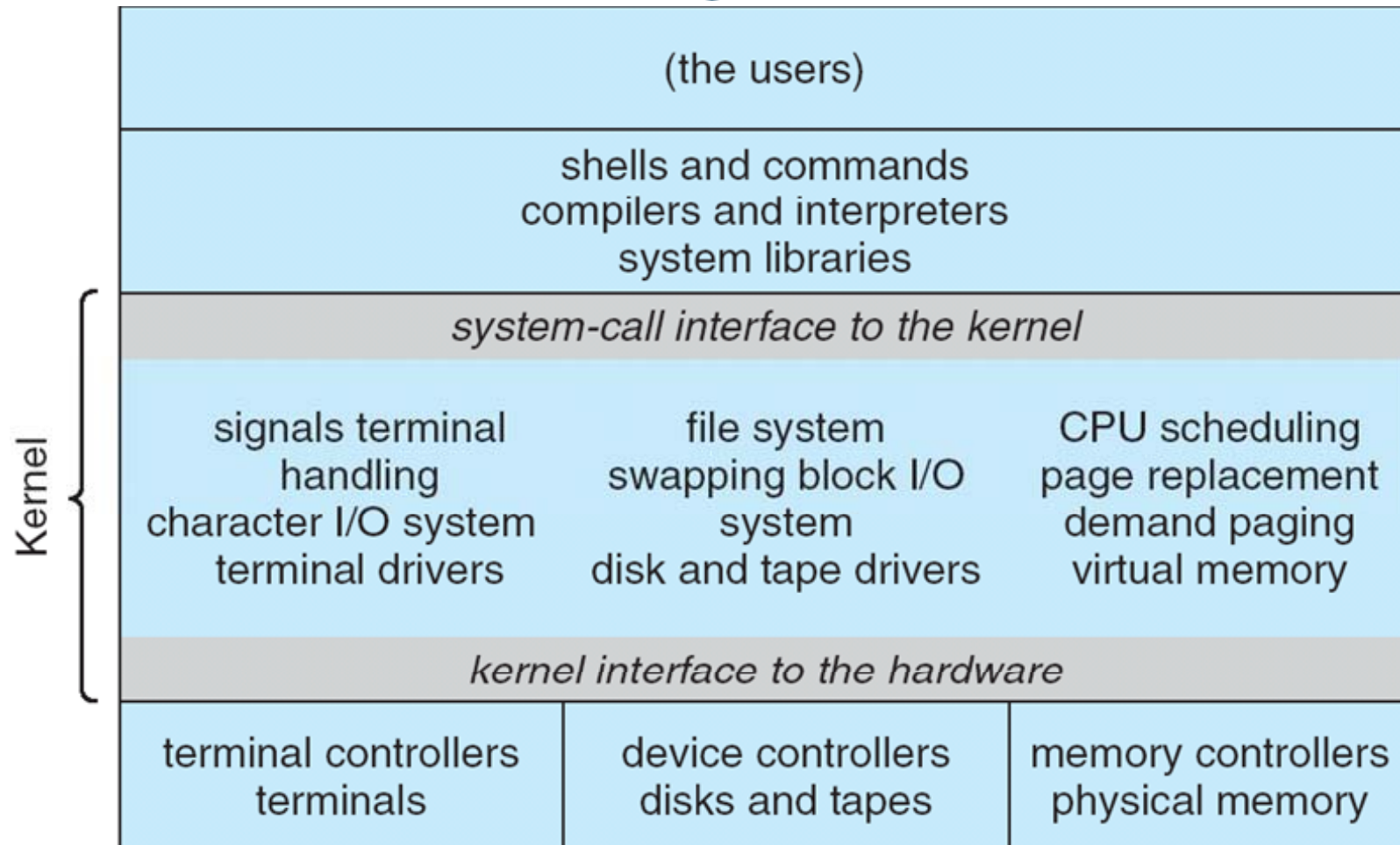


UNIX



- Le prime versioni di UNIX erano limitate dall'hardware, questo implicava una struttura limitata
- UNIX consisteva due parti separati:
 - I programmi di sistema
 - Il kernel
 - ✦ Consisteva di tutti quello al di sotto delle chiamate di sistema e sopra lo strato fisico
 - ✦ Forniva i servizi per il file system, lo scheduling della CPU, gestione della memoria, etc...

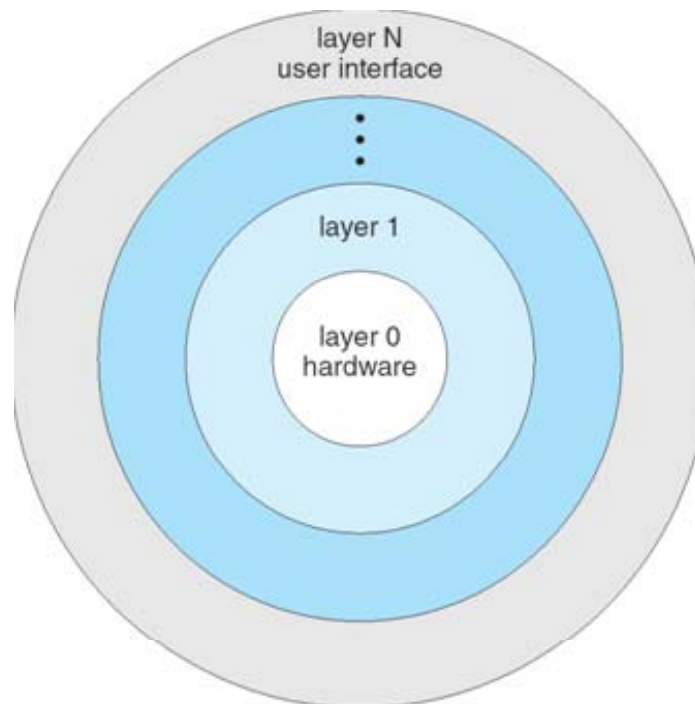
Struttura del sistema UNIX



Metodo stratificato



- Il Sistema Operativo è diviso in un certo numero di strati
 - Lo strato più basso è l'hardware, lo strato più alto è l'interfaccia utente
- Attraverso la modularità ogni strato è progettato in modo tale da utilizzare le funzioni ed i servizi degli strati inferiori

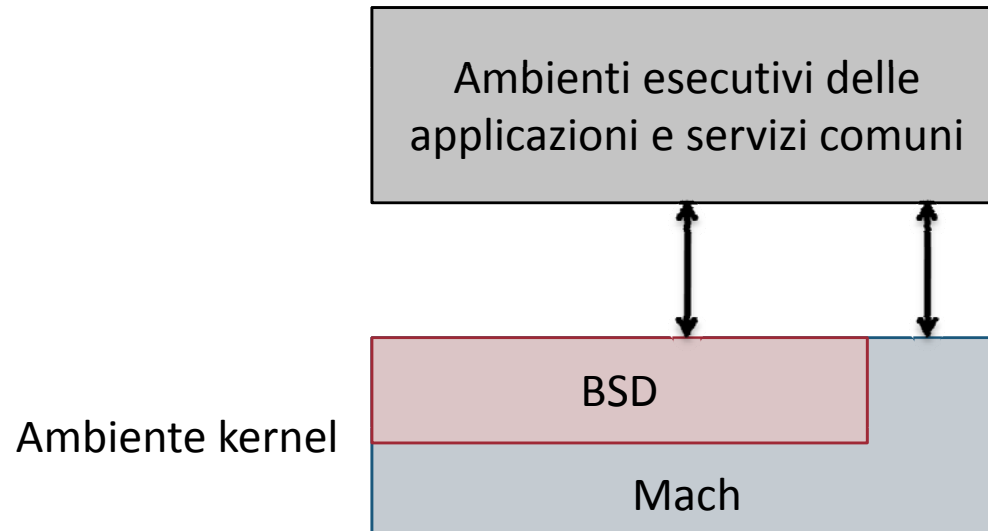


Microkernel



- Nella struttura a **microkernel** molte funzioni sono tirate fuori dal kernel
- La comunicazione avviene attraverso il message passing fornito da moduli di comunicazione
- Vantaggi:
 - Estendibile
 - I nuovi servizi si aggiungono allo spazio utente
 - Porting verso nuove architetture
 - Maggiore affidabilità e sicurezza
 - Se un servizio è compromesso lo sarà all'interno dello spazio utente quindi il sistema continua funzionare
- Svantaggi
 - Cali di prestazioni indotti dal sovraccarico della comunicazione

MacOS X

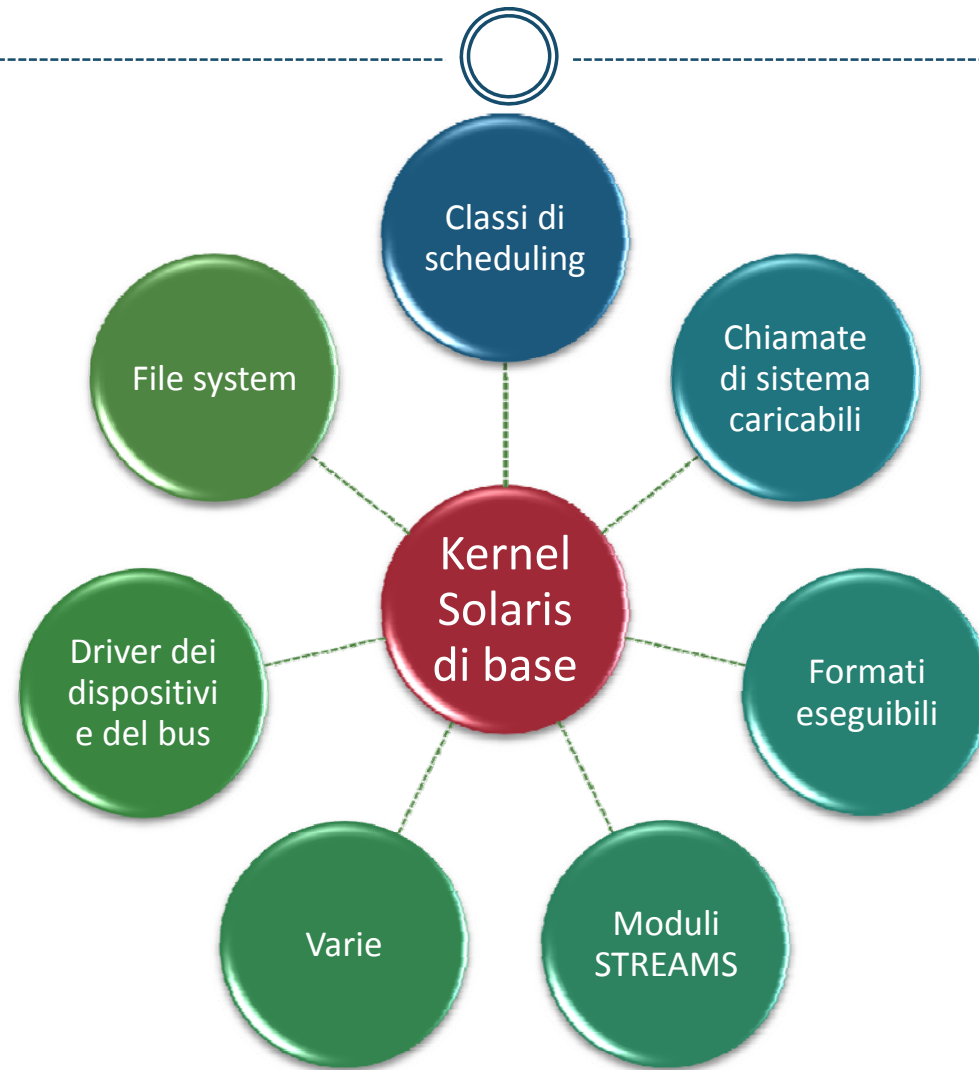


Moduli



- I Sistemi Operativi moderni utilizzano un approccio ibrido implementando dei moduli del kernel
 - Si utilizza un approccio orientato agli oggetti
 - I moduli sono separati
 - Ogni modulo comunica con un altro modulo attraverso una interfaccia nota
 - Ogni modulo è caricato dal kernel quando è necessario
- In definitiva, simile al metodo stratificato ma più flessibile

Moduli caricabili di Solaris

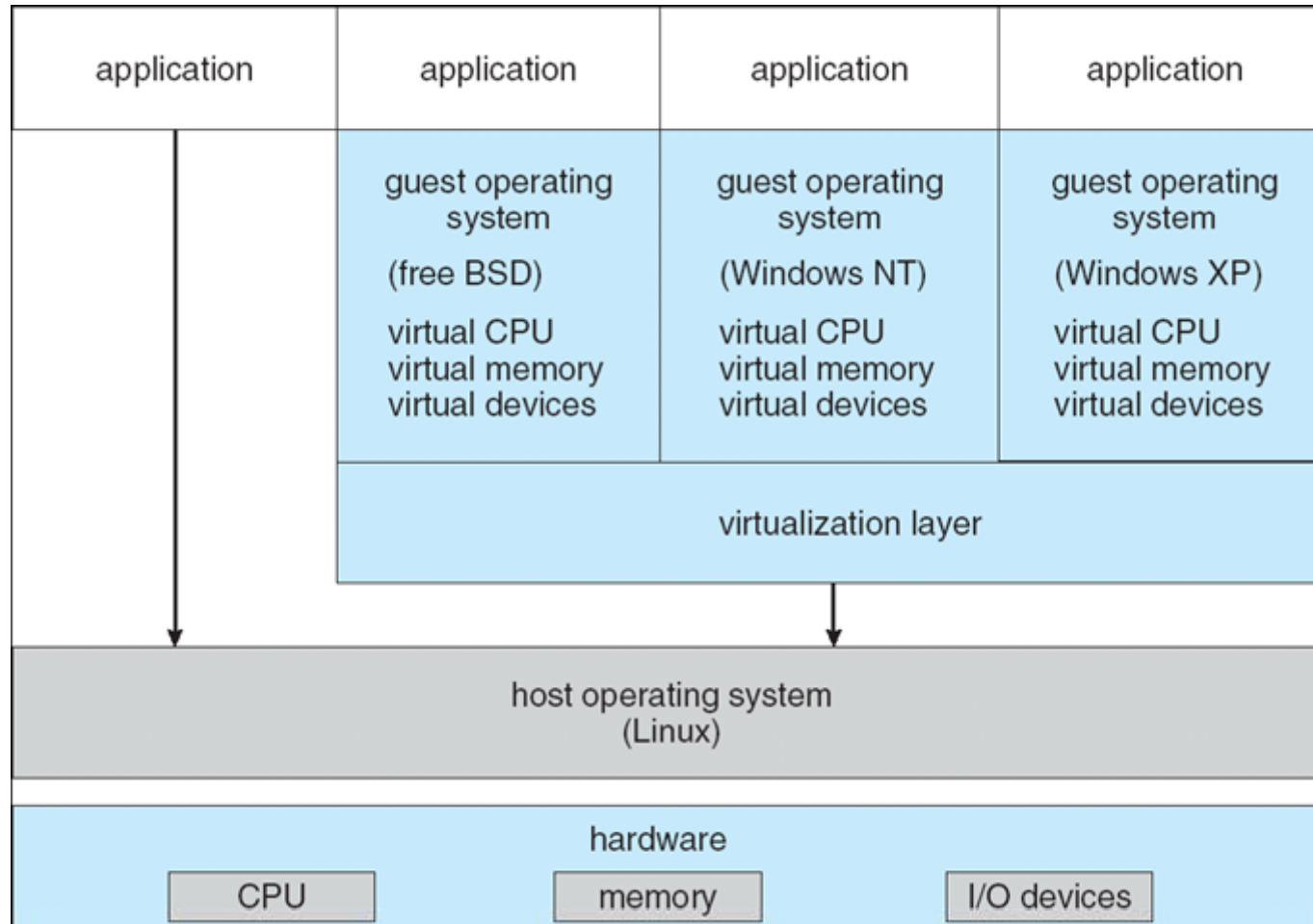


Macchine Virtuali



- Una **macchina virtuale** porta l'approccio basato sulla stratificazione ad una logica conclusione. Trattare l'hardware ed il Sistema Operativo come se fosse tutto hardware
- Una macchina virtuale fornisce una interfaccia identica all'hardware sottostante
- Il Sistema Operativo crea uno più hardware virtuali su cui è possibile installare un sistema operativo

Architettura VMware



Macchine Virtuali



- Le macchine virtuali forniscono la completa protezione delle risorse del sistema poiché ogni macchina virtuale è isolata dalle altre
 - Quindi non c'è una condivisione diretta delle risorse
- Le macchine virtuali sono adoperate per lo sviluppo e la ricerca
 - Lo sviluppo può essere fatto su una macchina virtuale piuttosto che su una reale in modo tale da non interferire con le normali operazioni del sistema
- La maggiore difficoltà nella implementazione di una macchina virtuale è dovuta alla necessità di fornire un duplicato esatto della macchina sottostante