Struttura dei Sistemi Operativi

Struttura dei sistemi operativi

- Servizi del sistema operativo
- Interfaccia utente
- Chiamate di sistema
- Programmi di sistema
- Progettazione e realizzazione
- Struttura del sistema operativo
- Macchine virtuali
- Debugging del sistema operativo
- Generazione di sistemi

Introduzione – 1

- I sistemi operativi forniscono l'ambiente in cui si eseguono i programmi
  - Essendo organizzati secondo criteri che possono essere assai diversi, tale può essere anche la loro struttura interna
- La progettazione di un nuovo sistema operativo è un compito complesso
  - ⇨ il tipo di sistema desiderato definisce i criteri di scelta dei metodi e degli algoritmi implementati

Introduzione – 2

- In fase di progettazione, il sistema operativo può essere definito/valutato in base a…
  - …i servizi che esso dovrà fornire
  - …l'interfaccia messa a disposizione di programmatori e utenti
  - …la complessità di realizzazione

Servizi del sistema operativo – 1

- Interfaccia utente — Tutti gli attuali SO sono dotati di un'interfaccia utente, a linea di comando (*Command Line Interface*, CLI) o grafica (*Graphic User Interface*, GUI)
- Esecuzione di programmi — capacità di caricare un programma in memoria ed eseguirlo, eventualmente rilevando, ed opportunamente gestendo, situazioni di errore
- Operazioni di I/O — il SO fornisce ai programmi utente i mezzi per effettuare l'I/O su file o periferica
- Gestione del file system — capacità dei programmi di creare, leggere, scrivere e cancellare file e muoversi nella struttura delle directory

Servizi del sistema operativo – 2

- **Comunicazioni** — scambio di informazioni fra processi in esecuzione sullo stesso elaboratore o su sistemi diversi, connessi via rete
  - Le comunicazioni possono avvenire utilizzando *memoria condivisa* o con *scambio di messaggi*
- **Rilevamento di errori** — il SO deve tenere il sistema di calcolo sotto controllo costante, per rilevare possibili errori, che possono verificarsi nella CPU e nella memoria, nei dispositivi di I/O o durante l'esecuzione di programmi utente
  - Per ciascun tipo di errore, il SO deve prendere le opportune precauzioni per mantenere una modalità operativa corretta e consistente
  - I servizi di debugging possono facilitare notevolmente la programmazione e, in generale, l'interazione con il sistema di calcolo

Servizi del sistema operativo – 3

- Esistono funzioni addizionali atte ad assicurare l'efficienza del sistema (non esplicitamente orientate all'utente)
  - Allocazione di risorse — quando più utenti o più job vengono serviti in concorrenza, le risorse disponibili devono essere allocate equamente ad ognuno di essi
  - Accounting e contabilizzazione dell'uso delle risorse — tener traccia di quali utenti usano quali e quante risorse del sistema (utile per ottimizzare le prestazioni del sistema di calcolo)
  - Protezione e sicurezza — i possessori di informazione memorizzata in un sistema multiutente o distribuito devono essere garantiti da accessi indesiderati ai propri dati; processi concorrenti non devono interferire fra loro
    - Protezione: assicurare che tutti gli accessi alle risorse di sistema siano controllati
    - Sicurezza: si basa sull'obbligo di identificazione tramite *password* e si estende alla difesa dei dispositivi di I/O esterni (modem, adattori di rete, etc.) da accessi illegali

Interfaccia utente CLI

- L'interfaccia utente a linea di comando permette di impartire direttamente comandi al SO (istruzioni di controllo)
  - Talvolta viene implementata direttamente nel kernel, altrimenti attraverso programmi di sistema (UNIX/ Linux)
  - Può essere parzialmente personalizzabile, ovvero il SO può offrire più *shell*, più ambienti diversi, da cui l'utente può impartire le proprie istruzioni al sistema
  - La sua funzione è quella di interpretare ed eseguire la le istruzioni di comando (siano esse istruzioni built–in del SO o nomi di eseguibili utente) – interprete dei comandi

L'interprete dei comandi

- I comandi ricevuti dall'interprete possono essere eseguiti secondo due modalità:
  - Se il codice relativo al comando è parte del codice dell'interprete, si effettua un salto all'opportuna sezione di codice
    - ⇨Poiché ogni comando richiede il proprio segmento di codice, il numero dei comandi implementati determina le dimensioni dell'interprete
  - I comandi vengono implementati per mezzo di programmi di sistema
    - ⇨I programmatori possono aggiungere nuovi comandi al sistema creando nuovi file con il nome appropriato
    - ⇨L'interprete dei comandi non viene modificato e può avere dimensioni ridotte

L'interprete dei comandi in DOS

```
C:\>dir

 Volume in drive C is MS-DOS 5_0
 Volume Serial Number is 446B-2781
 Directory of C:\

COMMAND  COM     47845 11-11-91   5:00a
        1 file(s)      47845 bytes
                    10280960 bytes free

C:\>ver

MS-DOS Version 5.00


C:\>
```

L'interprete dei comandi in Linux – 1

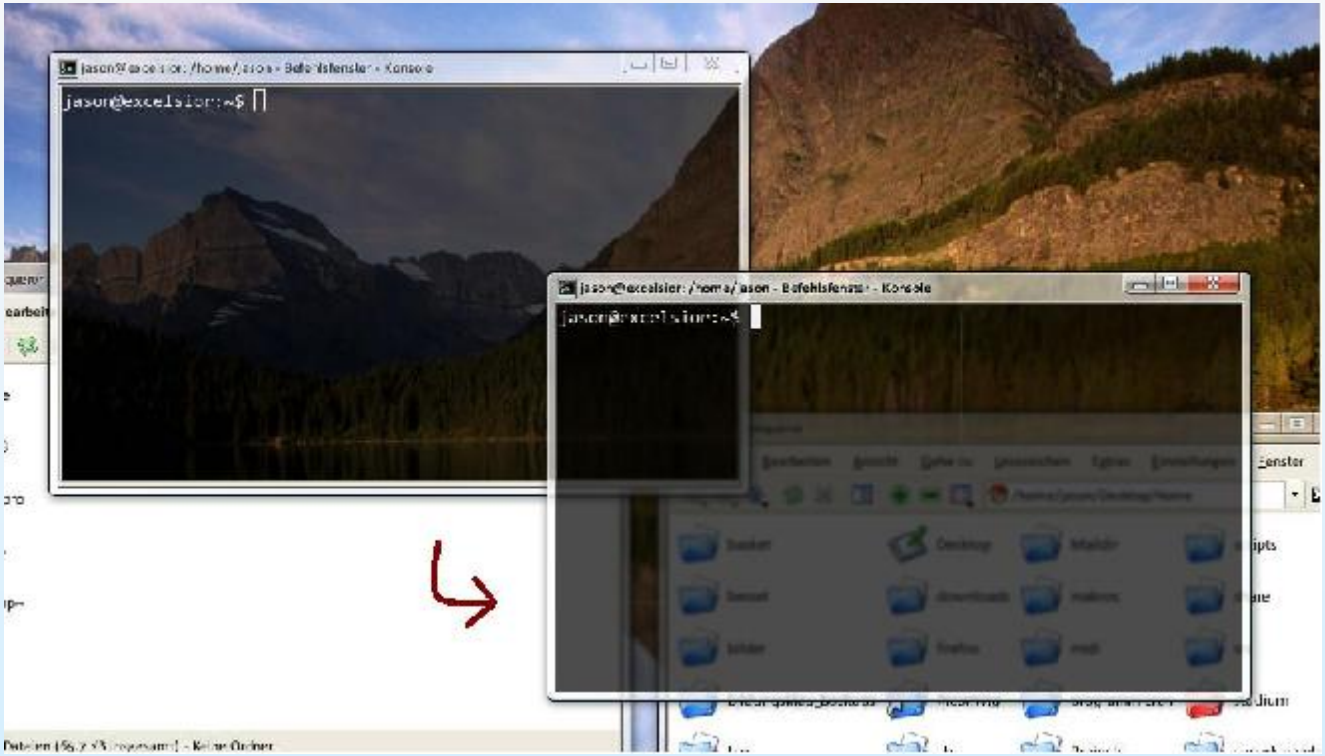
- Esempio
  - A fronte del comando

    `$ rm file.txt`

    l'interprete cerca un file chiamato **`rm`**, generalmente seguendo un percorso standard nel file system (**`usr/bin`**), lo carica in memoria e lo esegue con il parametro **`file.txt`**

  - Esegue la cancellazione – *remove* – del file **`file.txt`**

L'interprete dei comandi in Linux – 2

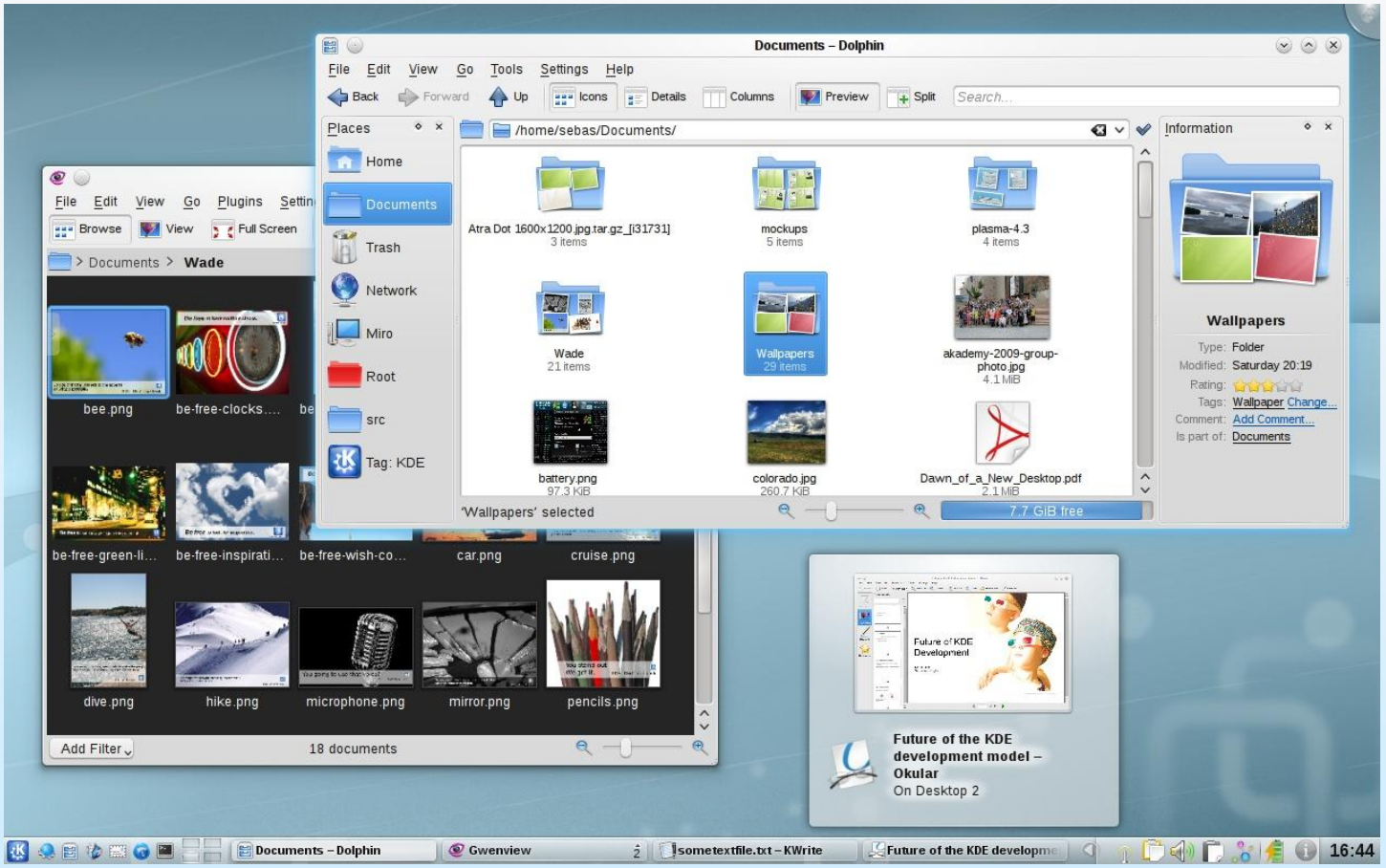


Bash shell (per Bourne Again SHell) è una shell testuale del progetto GNU, ma disponibile anche per alcuni sistemi Microsoft Windows (es. Cygwin)

Interfaccia utente GUI – 1

- Interfaccia user–friendly che realizza la metafora della scrivania (desktop)
  - Interazione semplice via mouse
  - Le icone rappresentano file, directory, programmi, azioni, etc.
  - I diversi tasti del mouse, posizionato su oggetti differenti, provocano diversi tipi di azione (forniscono informazioni sull'oggetto in questione, eseguono funzioni tipiche dell'oggetto, aprono directory – *folder*, o *cartelle*, nel gergo GUI)

Interfaccia utente GUI – 2



Il desktop di GNU/Linux

Interfaccia utente

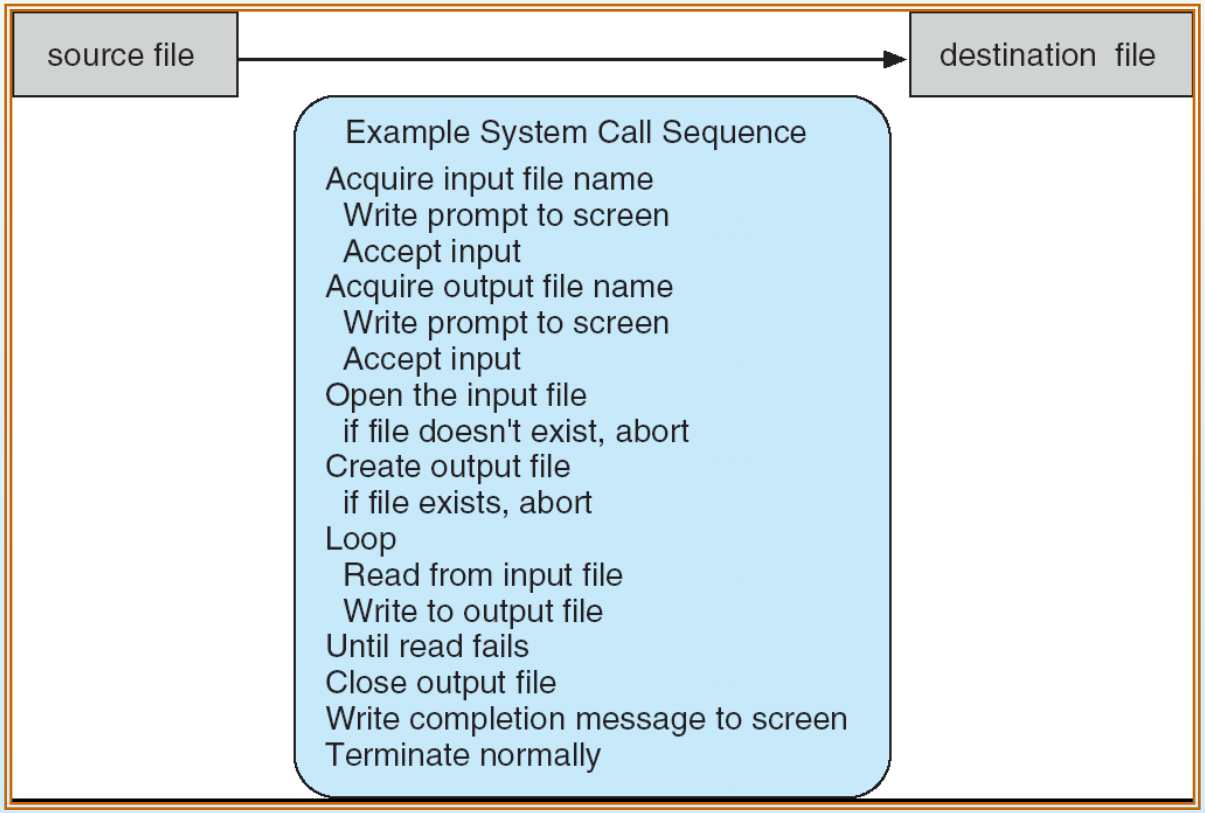
- Molti sistemi operativi attuali includono interfacce sia CLI che GUI
  - Microsoft Windows principalmente basato su una interfaccia grafica, ma dotato anche di una shell di comandi DOS–based (**cmd**)
  - Apple Mac OS X interagisce per mezzo della GUI "Aqua", ma è dotato di un kernel UNIX e mette a disposizione diversi tipi di shell
  - Solaris è tipicamente CLI, con interfaccia GUI opzionale (Java Desktop, KDE)
  - Linux è modulare; si può scegliere tra GUI molto avanzate (KDE, GNOME, etc.) e la CLI

Chiamate di sistema

- Le chiamate al sistema forniscono l'interfaccia fra i processi e i servizi offerti dal SO
- Sono realizzate (invocate) utilizzando linguaggi di alto livello (C o C++)
- Normalmente vengono richiamate dagli applicativi attraverso API (*Application Programming Interface*) piuttosto che per invocazione diretta
- Alcune API molto diffuse sono la Win64 API per Windows, la POSIX API per i sistemi POSIX–based (tutte le versioni di UNIX, Linux, e Mac OS X), e la Java API per la Java Virtual Machine (JVM)
- POSIX: Portable Operating System Interface for UNIX

Esempio di chiamate al sistema

- Sequenza di chiamate al sistema per realizzare la copia di un file in un altro

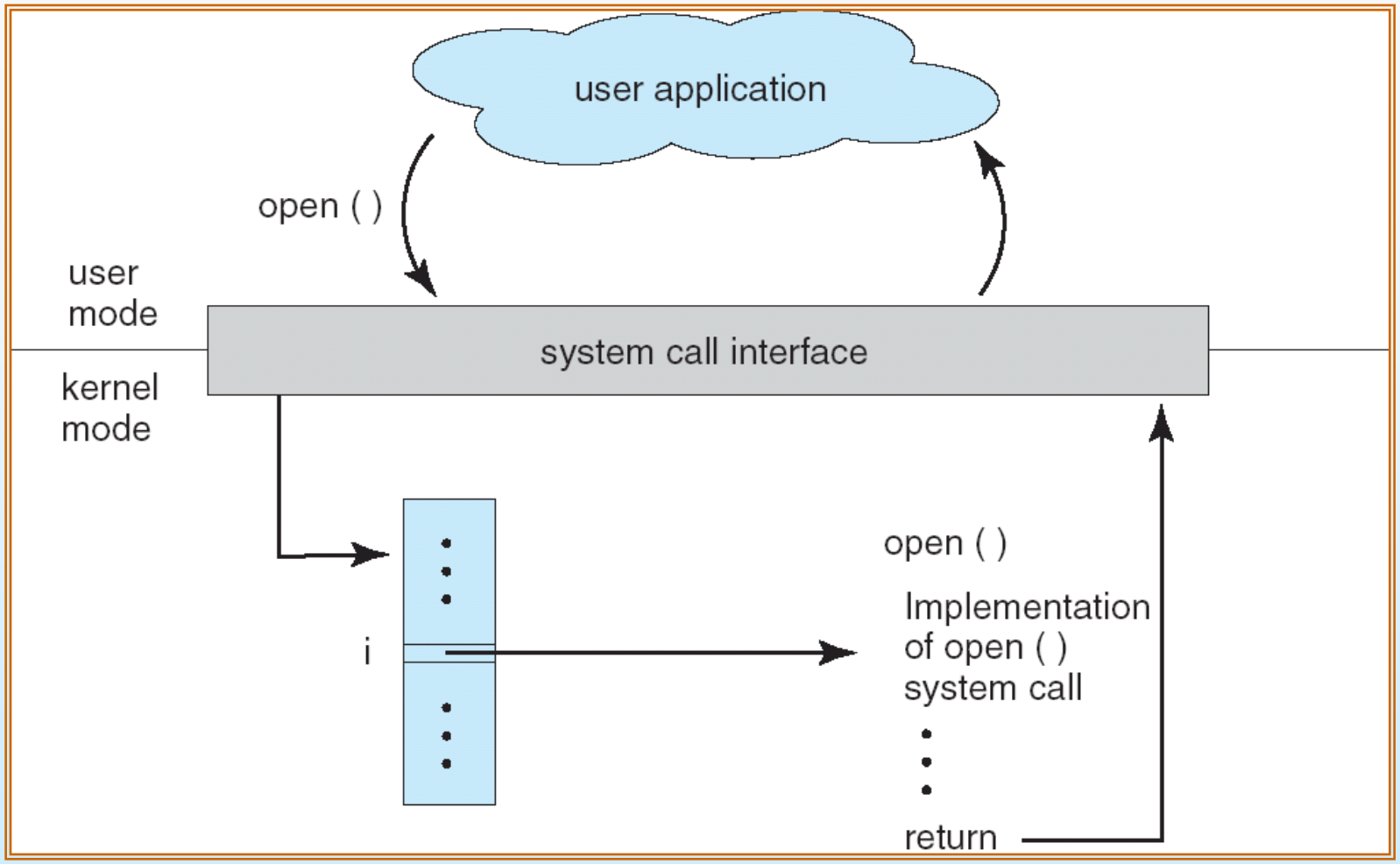


⇨ Migliaia di chiamate al sistema al secondo!

Chiamate di sistema (Cont.)

- Normalmente, a ciascuna *system call* è associato un numero
  - L'interfaccia delle chiamate al sistema mantiene una tabella indicizzata dal numero di system call, effettua la chiamata e ritorna lo stato del sistema dopo l'esecuzione (ed eventuali valori restituiti)
- L'utente non deve conoscere i dettagli implementativi delle system call: deve conoscere la modalità di utilizzo dell'API (ed eventualmente il compito svolto dalle chiamate al sistema)
  - L'intermediazione della API garantisce la portabilità delle applicazioni
  - Molto spesso una system call viene chiamata tramite una funzione di libreria standard (ad esempio **`stdlibc`**)

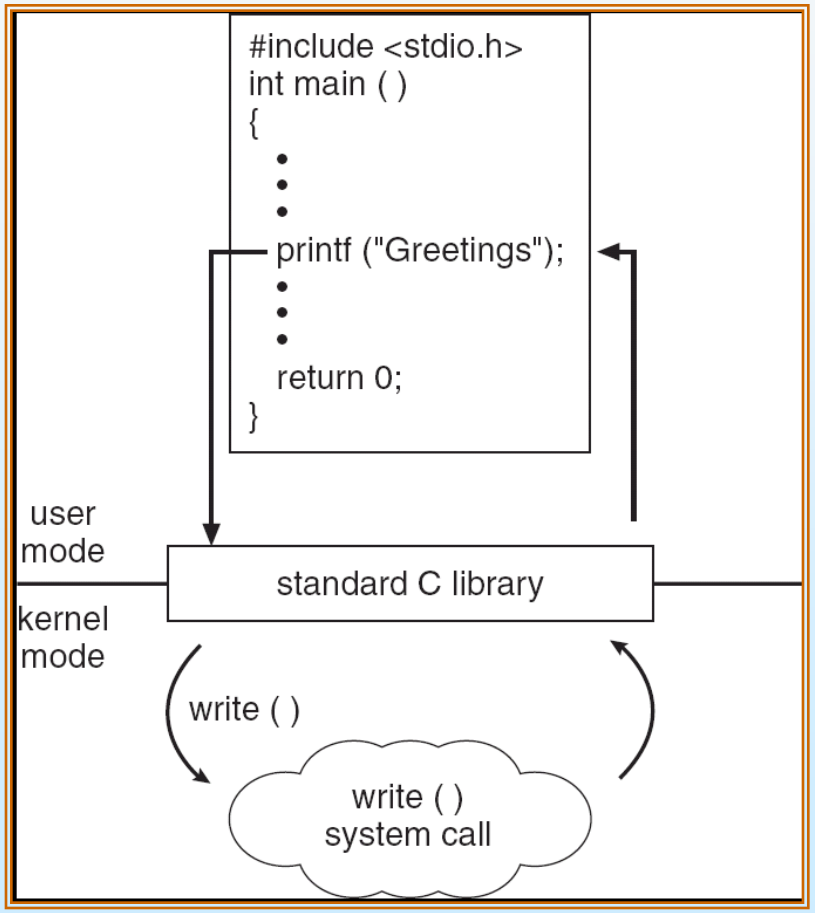
Relazioni API – System call – SO



Gestione della chiamata di sistema **`open()`** invocata da un'applicazione utente

Esempio con la libreria standard C

- Per Linux, la libreria standard del linguaggio C (il *run–time support system*) fornisce una parte dell'API



- Programma C che invoca la funzione di libreria per la stampa *printf()*
- La libreria C intercetta la funzione e invoca la system call *write()*
- La libreria riceve il valore restituito dalla chiamata al sistema e lo passa al programma utente

Un altro esempio…

- Funzione C che copia il contenuto di un file in un altro

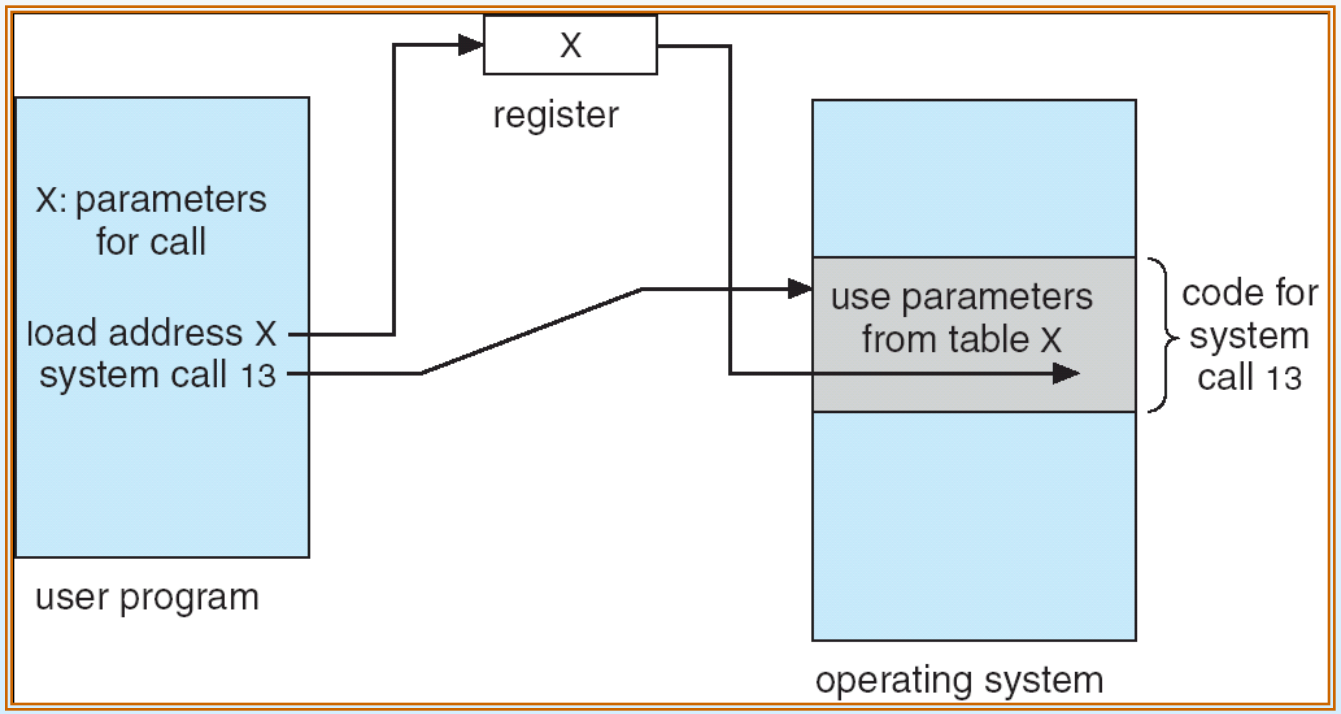
```
#include <stdio.h>
#include <stddef.h>
#define FAIL 0
#define SUCCESS 1

int copy_file(infile, outfile)
char *infile, *outfile;

{
    FILE *fp1, *fp2;
    if ((fp1 = fopen(infile, "rb")) == NULL)
       return FAIL;
    if ((fp2 = fopen(outfile, "wb")) == NULL)
    {
       fclose(fp1);
       return FAIL;
     }
     while (!feof(fp1))
            putc(getc(fp1), fp2);
     fclose(fp1);
     fclose(fp2);
     return SUCCESS;
}
```

- Per eseguire l'I/O, è necessario associare un flusso ad un file o a una periferica
  - ⇨ occorre dichiarare un puntatore alla struttura FILE
- La struttura FILE, definita in stdio.h, è costituita da campi che contengono informazioni quali il nome del file, la modalità di accesso, il puntatore al prossimo carattere nel flusso
- Entrambi i file vengono acceduti in modalità binaria
- La macro *getc()* legge il prossimo carattere dal flusso specificato e sposta l'indicatore di posizione del file avanti di un elemento ad ogni chiamata

Passaggio di parametri alle system call

- Spesso l'informazione necessaria alla chiamata di sistema non si limita al solo nome (o numero di identificazione)
  - Il tipo e la quantità di informazione varia per chiamate diverse e diversi sistemi operativi
- Esistono tre metodi generali per passare parametri al SO
  - Il più semplice: passaggio di parametri nei registri
    - Talvolta, possono essere necessari più parametri dei registri presenti
  - Memorizzazione dei parametri in un blocco in memoria e passaggio dell'indirizzo del blocco come parametro in un registro
    - Approccio seguito da Linux e Solaris
  - *Push* dei parametri nello stack da parte del programma; il SO recupera i parametri con un *pop*
  - Gli ultimi due metodi non pongono limiti al numero ed alla lunghezza dei parametri passati

Passaggio di parametri tramite tabella



Passaggio dei parametri tramite indirizzo del blocco di memoria

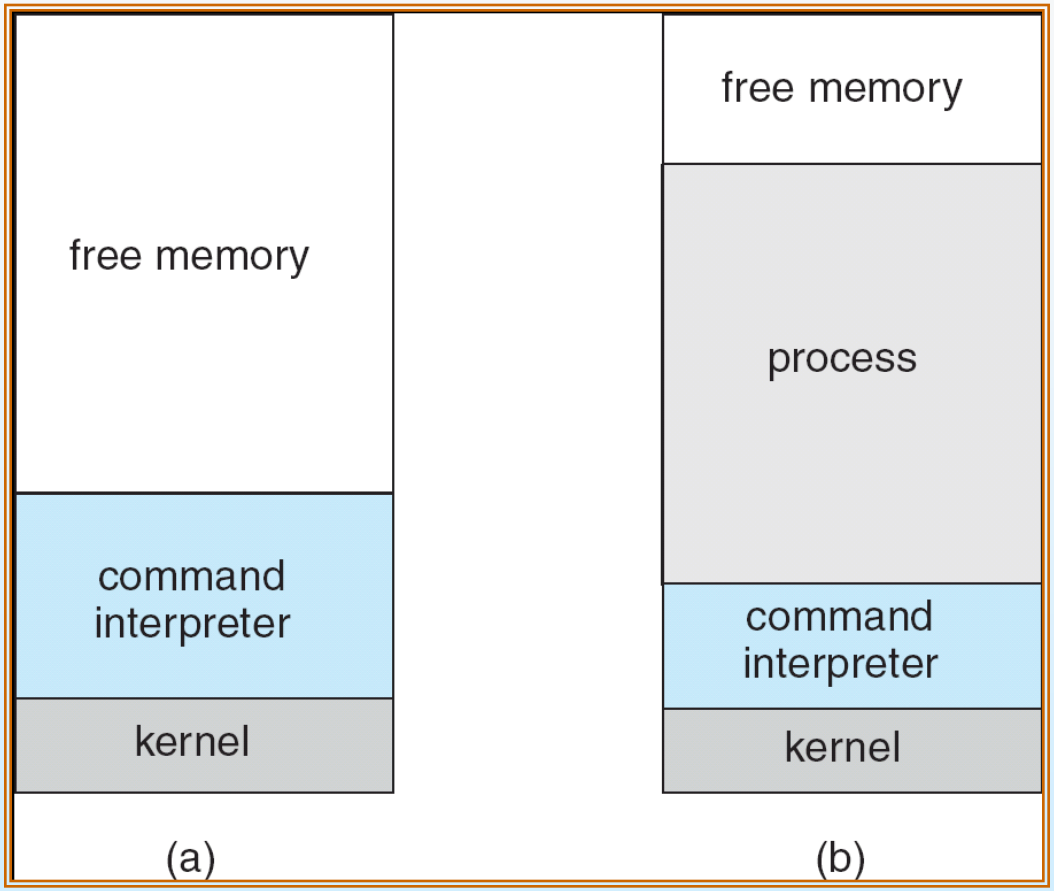
Tipi di chiamate al sistema – 1

- Controllo dei processi
- Gestione dei file
- Gestione dei dispositivi di I/O
- Gestione delle informazioni
- Comunicazione

Tipi di chiamate al sistema – 2

- Controllo dei processi
  - Creazione e arresto di un processo (*fork*, *exit*)
  - Caricamento ed esecuzione (*exec/execve*)
  - Esame ed impostazione degli attributi di un processo (priorità, tempo massimo di esecuzione – *get/set process attributes*)
  - Attesa per il tempo indicato o fino alla segnalazione di un evento (*wait/waitpid*)
  - Assegnazione e rilascio di memoria (*alloc*, *free*)
  - Invio di segnali (*signal, kill*)

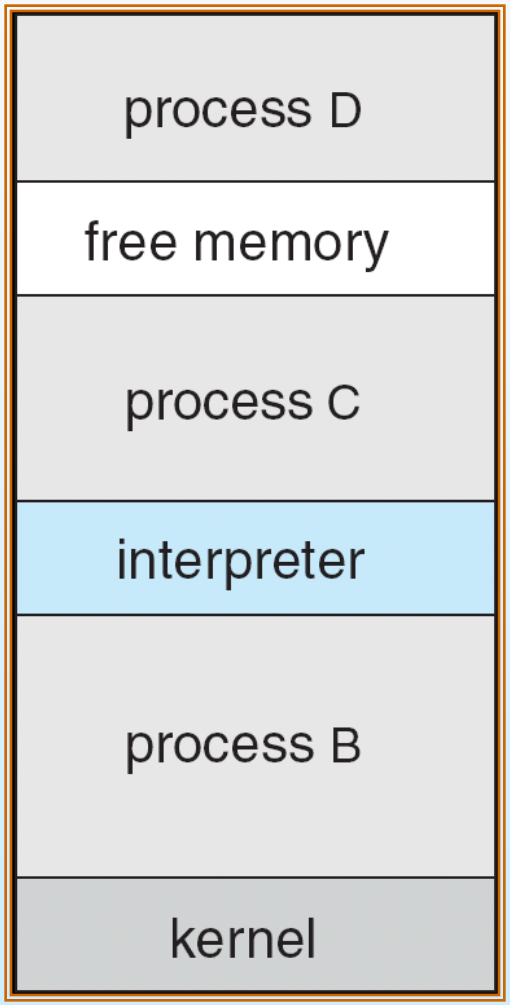
Esecuzione di programmi in MS–DOS



(a) Allo startup del sistema

(b) Durante l'esecuzione di un programma utente

Esecuzione multipla di programmi in FreeBSD



Tipi di chiamate al sistema – 3

- Gestione dei file
  - Creazione e cancellazione di file (*create*, *delete*)
  - Apertura e chiusura di file (*open*, *close*)
  - Lettura, scrittura e posizionamento (*read*, *write*, *seek*)
  - Esame ed impostazione degli attributi di un file (nome, tipo, codici di protezione, informazioni di contabilizzazione – *get/set file attributes*)
- Gestione dei dispositivi di I/O
  - Richiesta e rilascio di un dispositivo (*request*, *release*)
  - Lettura, scrittura e posizionamento
  - Esame ed impostazione degli attributi di un dispositivo (*ioctl*)

Tipi di chiamate al sistema – 4

- Gestione delle informazioni
  - Esame ed impostazione dell'ora e della data (*time*, *date*)
  - Informazioni sul sistema (*who*, *du*)
  - Esame ed impostazione degli attributi dei processi, file e dispositivi (*ps*,*getpid*)
- Comunicazione
  - Creazione e chiusura di una connessione (*open connection*, *close connection*)
  - Invio e ricezione di messaggi (*send*, *receive*)
  - Informazioni sullo stato dei trasferimenti
  - Inserimento ed esclusione di dispositivi remoti
  - Condivisione della memoria (*shm_open*, *mmap*)

Esempio – 1

- Cosa producono in stampa questi codici?

```
main ()
{
    val = 5;
    if(fork())
       wait(&val);
    val++;
    printf("%d\n", val);
    return val;
}
```

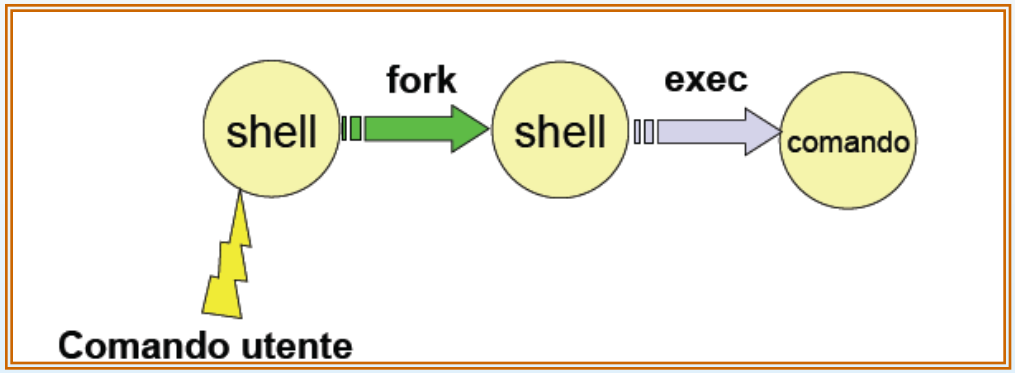
Il processo figlio incrementa il valore di val e lo stampa, quindi lo restituisce al padre (che era in attesa)
Il padre incrementa ancora val e lo stampa
⇨ 6, 7

```
main ()
{
    val = 5;
    if(fork())
       wait(&val);
    else
       return val;
    val++;
    printf("%d\n", val);
    return val;
}
```

Il processo figlio termina immediatamente restituendo il controllo al padre
Il padre incrementa val e lo stampa
⇨ 6

Esempio – 2

- Per ogni comando, la shell genera un processo figlio (una nuova shell) dedicato all'esecuzione del comando:



- Possibilità di due diversi comportamenti
  - Il padre si pone in attesa della terminazione del figlio (esecuzione in foreground) → `$ ls -l pippo`
  - Il padre procede nell'esecuzione concorrentemente al figlio (esecuzione in background) → `$ ls -l pippo &`

Programmi di sistema – 1

- I programmi di sistema forniscono un ambiente conveniente per lo sviluppo e l'esecuzione di programmi utente (semplici interfacce alle system call o programmi complessi)
- Esistono programmi di sistema per…
  - Gestione di file
  - Informazioni di stato
  - Modifica di file
  - Supporto a linguaggi di programmazione
  - Caricamento ed esecuzione di programmi
  - Comunicazioni
  - Realizzazione di programmi applicativi
- L'aspetto del SO per la maggioranza degli utenti è definito dai programmi di sistema, non dalle chiamate di sistema vere e proprie

Programmi di sistema – 2

- Gestione di file – per creare, cancellare, copiare, rinominare, stampare e, genericamente, gestire le operazioni su file e directory
- Informazioni di stato
  - Per ottenere dal sistema informazioni tipo data, spazio di memoria disponibile, spazio disco, numero di utenti abilitati
  - Per ottenere informazioni sulle statistiche di utilizzo del sistema di calcolo (prestazioni, logging, etc.) e per operazioni di debugging
  - Per effettuare operazioni di stampa
  - Per ottenere informazioni sulla configurazione del sistema

Programmi di sistema – 3

- Modifica di file
  - Editori di testo, per creare e modificare file
  - Comandi speciali per cercare informazioni all'interno di file o effettuare trasformazioni sul testo
- Supporto a linguaggi di programmazione – assembler, compilatori e interpreti, debugger
- Caricamento ed esecuzione di programmi – linker, loader, per linguaggio macchina e linguaggi di alto livello
- Comunicazioni – per creare connessioni virtuali tra processi, utenti e sistemi di elaborazione
  - Permettono agli utenti lo scambio di messaggi video e via e–mail, la navigazione in Internet, il login remoto ed il trasferimento di file

Riassumendo…

- I tipi di richieste di servizio al SO variano secondo il livello delle richieste stesse
- Il livello cui appartengono le chiamate di sistema deve offrire le funzioni di base (controllo di processi e memoria e gestione di file e dispositivi)
- Le richieste di livello superiore, soddisfatte dall'interprete dei comandi o dai programmi di sistema, vengono tradotte in una sequenza di chiamate al SO
- Oltre le categorie di richieste di servizio standard, gli errori nei programmi possono considerarsi richieste di servizio implicite

Progettazione del sistema operativo

- La struttura interna dei diversi SO può variare notevolmente…
  - …in dipendenza dall'hardware
  - …e dalle scelte progettuali che, a loro volta, dipendono dallo scopo del sistema operativo e ne influenzano i servizi offerti
- Richieste utente ed obiettivi del SO
  - Richieste utente – il SO deve essere di semplice utilizzo, facile da imparare, affidabile, sicuro e veloce
  - Obiettivi del sistema – il SO deve essere semplice da progettare, facile da realizzare e manutenere, flessibile, affidabile, error–free ed efficiente

Meccanismi e politiche

- Per la progettazione e la realizzazione di un sistema operativo è fondamentale mantenere separati i due concetti di…
  - Politica: Quali sono i compiti e i servizi che il SO dovrà svolgere/fornire? (Es.: scelta di un CPU scheduling)
  - Meccanismi: Come realizzarli? (Es.: timer)
- I meccanismi determinano "come realizzare qualcosa", le politiche definiscono "il qualcosa" da realizzare
  - La separazione fra politiche e meccanismi garantisce la massima flessibilità se le decisioni politiche subiscono cambiamenti nel corso del tempo

Realizzazione del sistema operativo

- Tradizionalmente i SO venivano scritti in linguaggio assembly; attualmente vengono invece sviluppati in linguaggi di alto livello, particolarmente orientati al sistema: C o C++
- Vantaggi
  - Veloci da codificare
  - Codice compatto, di facile comprensione, messa a punto e manutenzione
  - Portabilità
- Possibili svantaggi
  - Potenziale minor efficienza del codice C rispetto all'assembly
    - ⇨Valutazione del sistema ed eventuale riscrittura di piccole porzioni "critiche" di codice (scheduler della CPU, gestore della memoria) in assembly

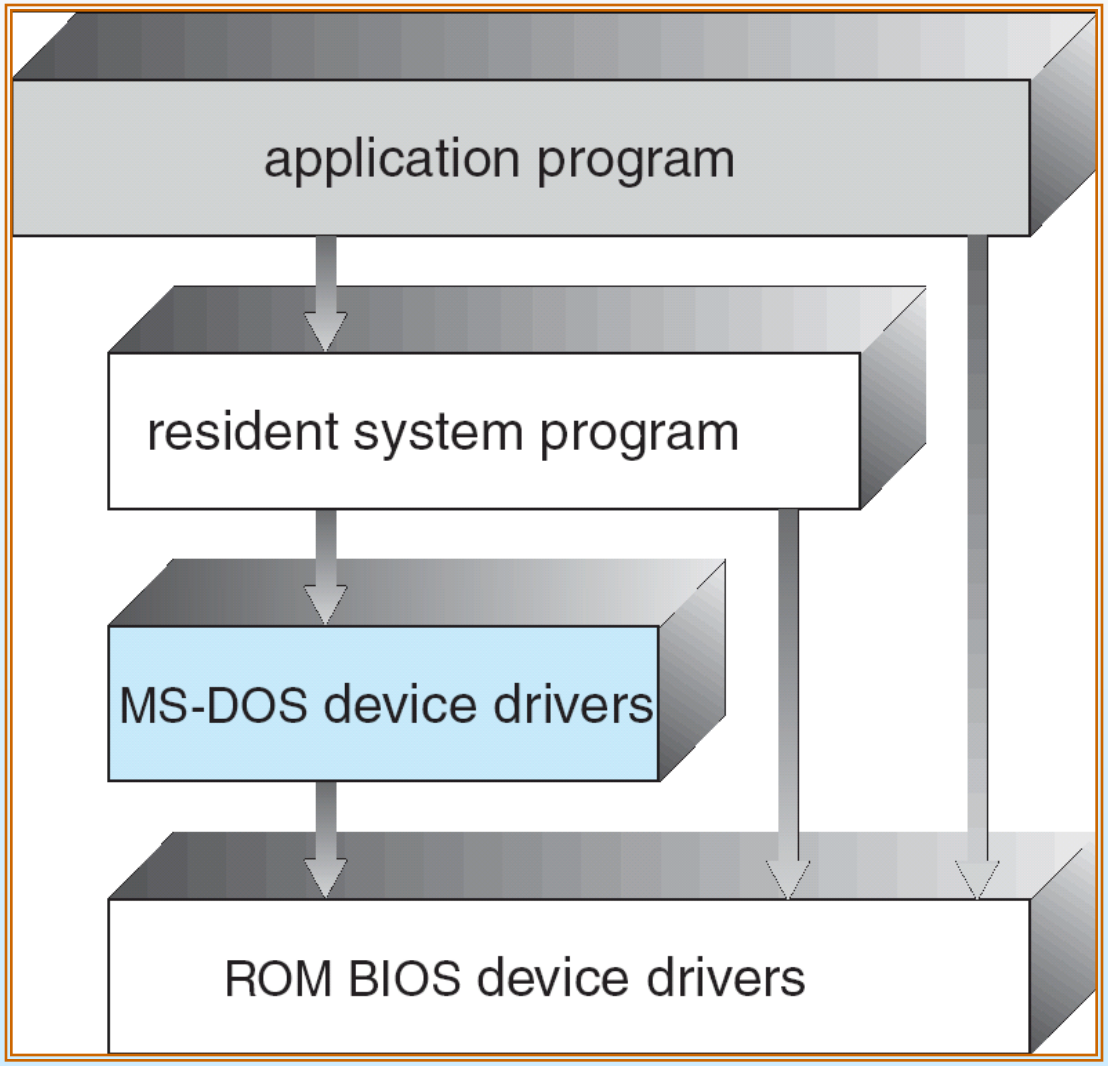
Struttura del sistema operativo

- Sistemi storici: monolitici
  - Le funzioni di gestione delle risorse sono realizzate nel nucleo e l'intero sistema operativo tende a identificarsi col nucleo
- Attualmente: suddivisione in piccole componenti, ciascuna delle quali deve essere un modulo ben definito del sistema, con interfacce e funzioni chiaramente stabilite in fase di progettazione

SO con struttura semplice

- MS–DOS — scritto per fornire il maggior numero di funzionalità utilizzando la minor quantità di spazio possibile:
  - Non è suddiviso in moduli
  - Sebbene MS–DOS abbia una qualche struttura, le sue interfacce e livelli di funzionalità non sono ben separati
    - Le applicazioni accedono direttamente alle routine di sistema per l'I/O (ROM BIOS)
    - Vulnerabilità agli errori ed agli "attacchi" dei programmi utente
  - Intel 8088, per cui MS–DOS fu progettato, non offre duplice modo di funzionamento e protezione hardware ⇨ impossibile proteggere hardware/SO dai programmi utente

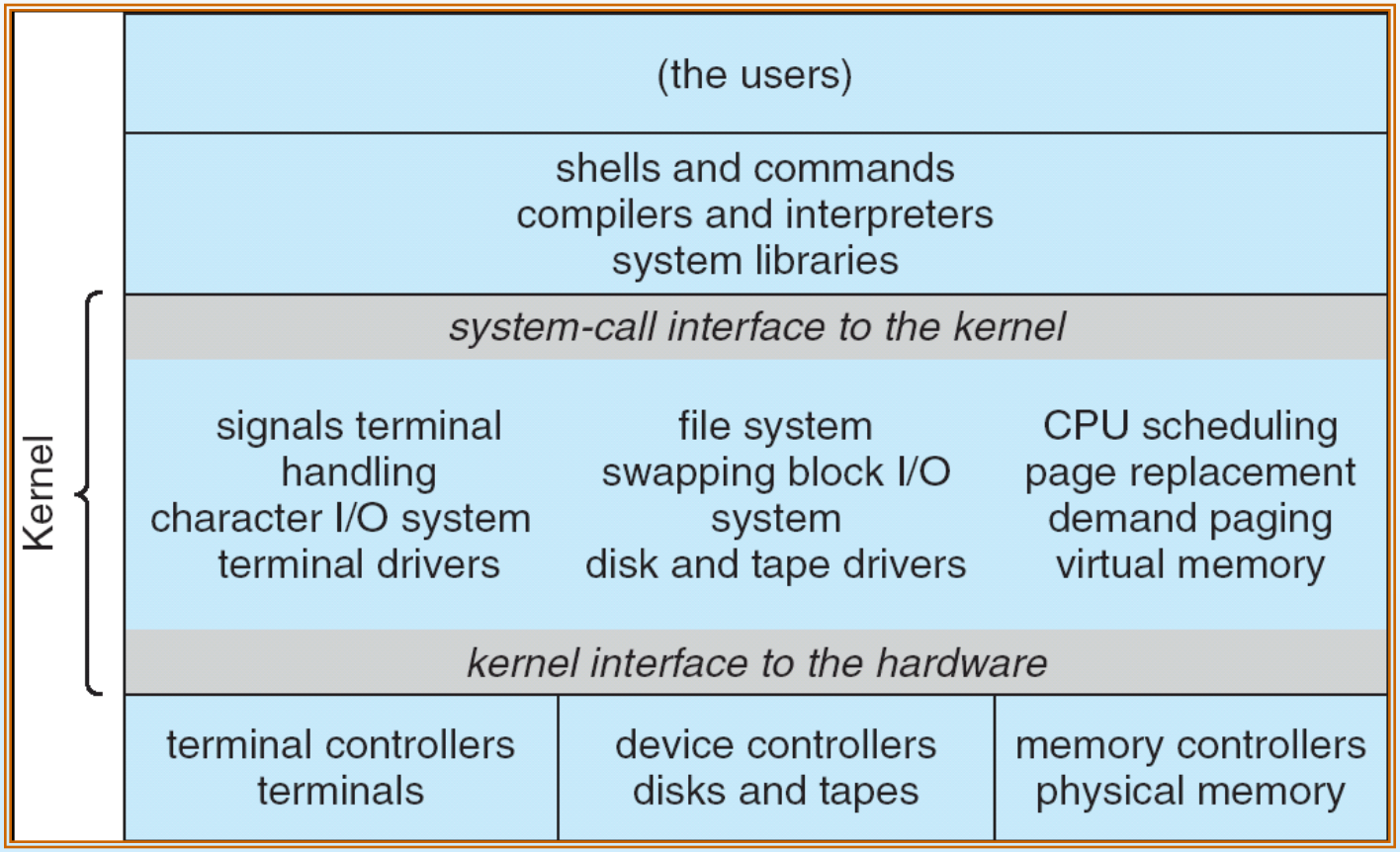
Struttura degli strati di MS–DOS



UNIX

- UNIX — a causa delle limitate funzionalità hardware disponibili all'epoca della realizzazione, il sistema operativo originale aveva una struttura scarsamente stratificata
- UNIX è costituito di due parti separate:
  - I programmi di sistema
  - Il kernel:
    - È formato da tutto ciò che si trova sotto l'interfaccia delle chiamate di sistema e sopra l'hardware
    - Fornisce il file system, lo scheduling della CPU, la gestione della memoria ⇨ un gran numero di funzioni per un solo livello!

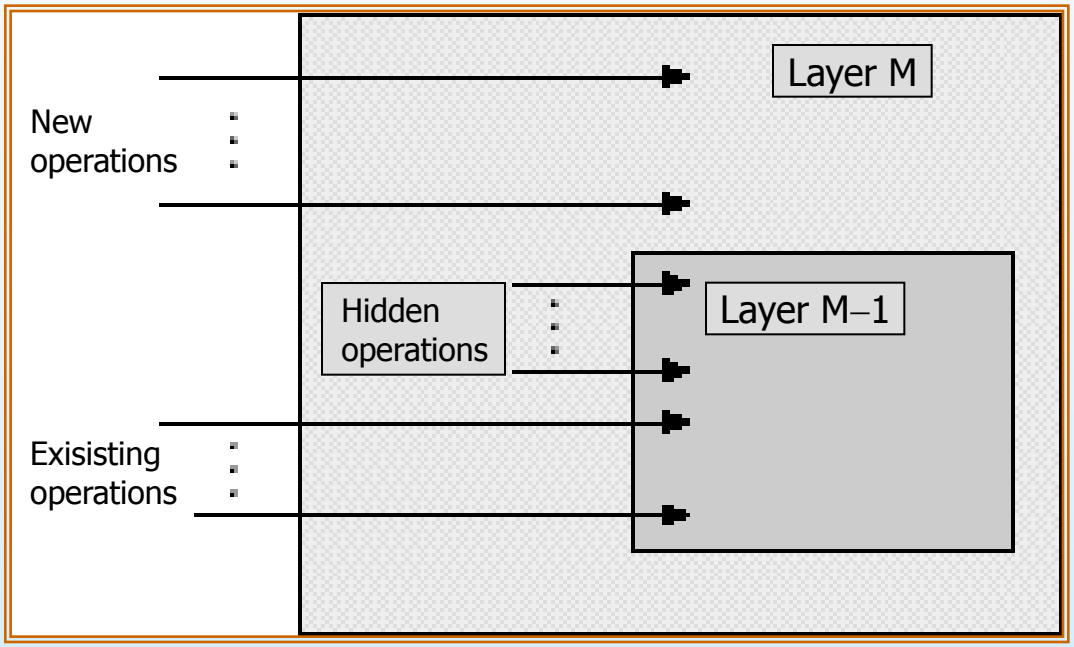
Struttura del sistema UNIX



Approccio stratificato – 1

- In presenza di hardware appropriato, i SO possono assumere architettura modulare, per meglio garantire il controllo sulle applicazioni
- Il SO è suddiviso in un certo numero di strati (livelli), ciascuno costruito sopra gli strati inferiori
  - Il livello più basso (strato 0) è l'hardware, il più alto (strato N) è l'interfaccia utente
- L'architettura degli strati è tale che ciascuno strato impiega esclusivamente funzioni (operazioni) e servizi di strati di livello inferiore (usati come black–box)
  - Incapsulamento delle informazioni

Approccio stratificato – 2



Approccio stratificato – 3

- Vantaggio
  - Semplicità di realizzazione e messa a punto (che viene attuata strato per strato)
- Svantaggi
  - Difficoltà nella definizione appropriata dei diversi strati, poiché ogni strato può sfruttare esclusivamente le funzionalità degli strati su cui poggia
  - Tempi lunghi di attraversamento degli strati (passaggio di dati) per portare a termine l'esecuzione di una system call

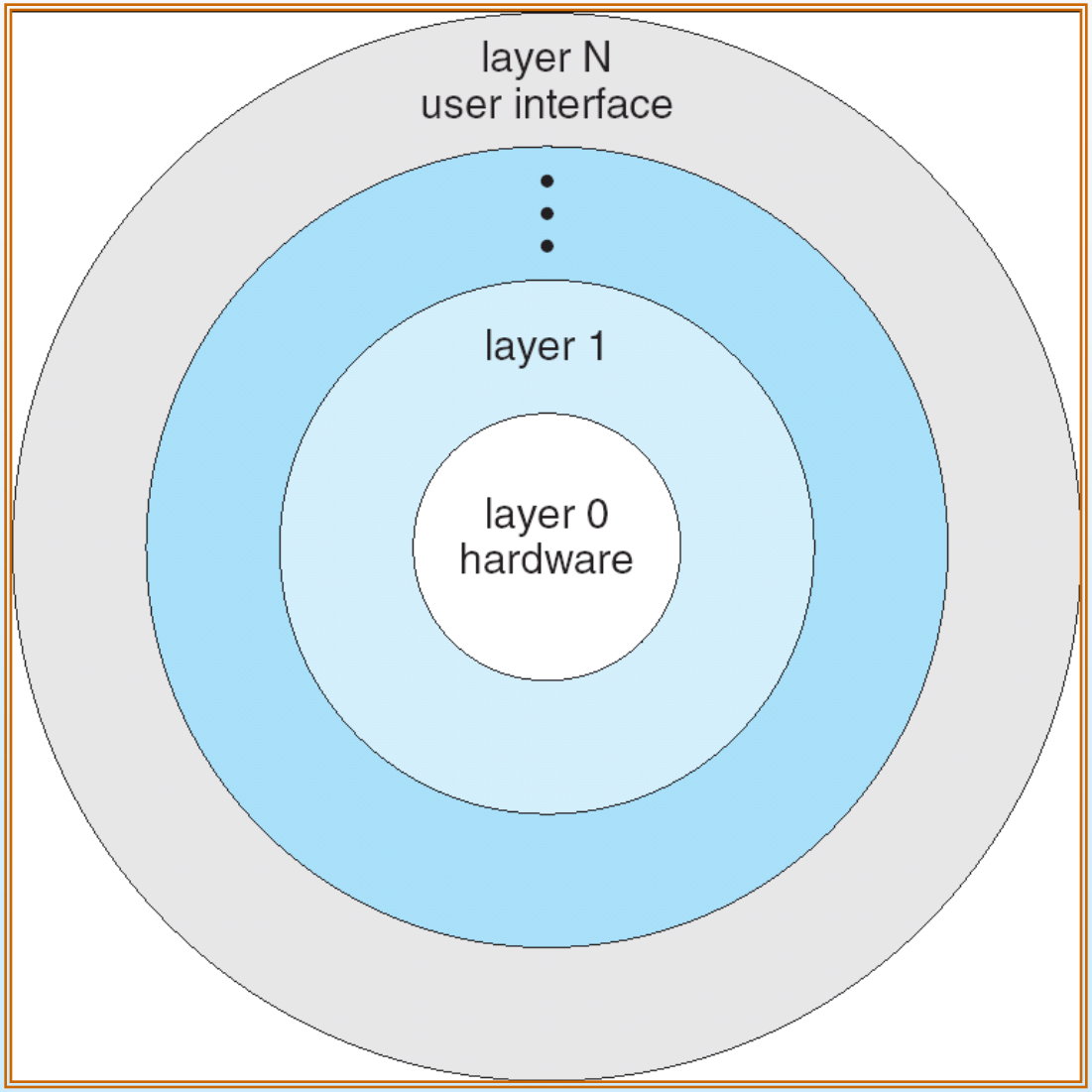
Approccio stratificato – 4

- Esempio 1 – Difficoltà di definizione degli strati
  - Il driver della memoria ausiliaria (*backing store*) dovrebbe trovarsi sopra lo scheduler della CPU, perché può accadere che il driver debba attendere un'istruzione di I/O e, in questo periodo, la CPU viene sottoposta a scheduling
  - Lo scheduler della CPU deve mantenere più informazioni sui processi attivi di quante ne possono essere contenute in memoria: deve fare uso del driver della memoria ausiliaria

Approccio stratificato – 5

- Esempio 2 – Scarsa efficienza del SO
  - Per eseguire un'operazione di I/O, un programma utente invoca una system call che è intercettata dallo strato di I/O…
  - …che esegue una chiamata allo strato di gestione della memoria…
  - …che richiama lo strato di scheduling della CPU…
  - …che la passa all'opportuno dispositivo di I/O

Sistema operativo stratificato



Struttura dei sistemi microkernel – 1

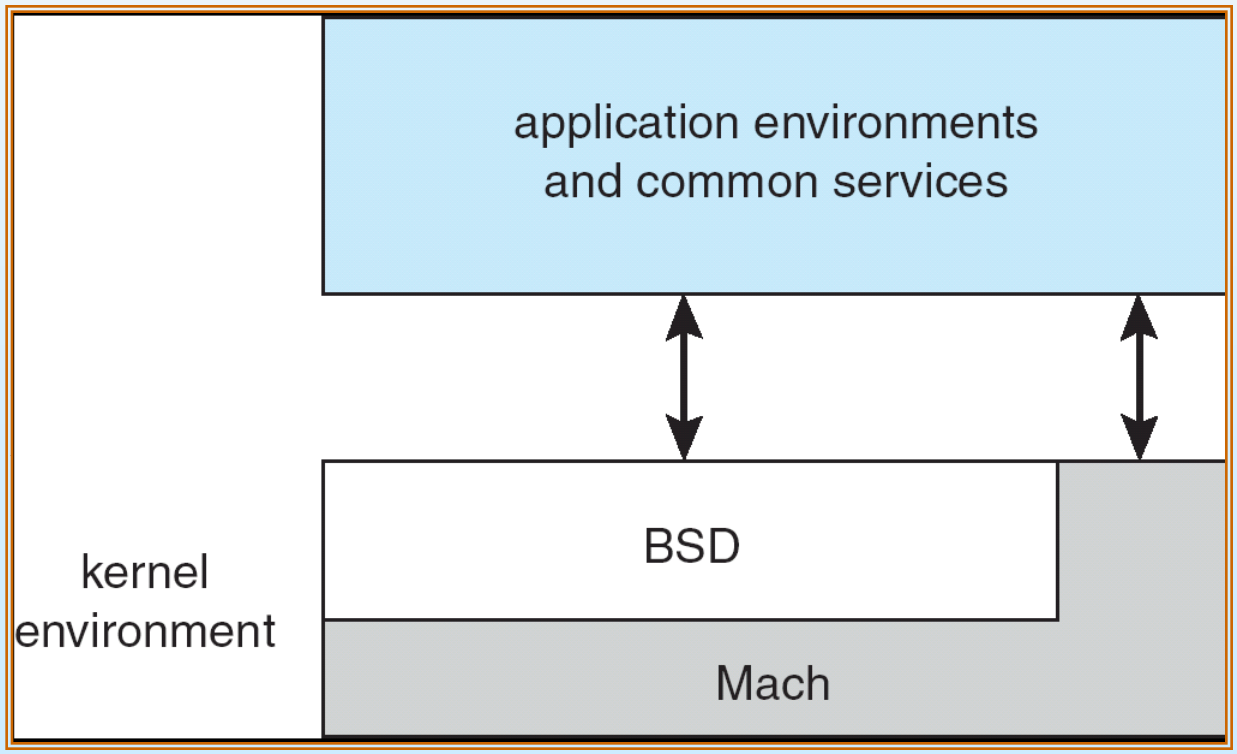
- Quasi tutte le funzionalità del kernel sono spostate nello spazio utente
- Un microkernel offre i servizi minimi di gestione dei processi, della memoria e di comunicazione
  - Scopo principale: fornire funzioni di comunicazione fra programmi client e servizi (implementati esternamente)
  - Le comunicazioni hanno luogo tra moduli utente mediante scambio di messaggi (mediati dal kernel)
  - Esempi: prime versioni di Windows NT, Mach, Tru64, GNU Hurd

Struttura dei sistemi microkernel – 2

- Vantaggi
  - Funzionalità del sistema più semplici da estendere: i nuovi servizi sono programmi di sistema che si eseguono nello spazio utente e non comportano modifiche al kernel
  - Facilità di modifica del kernel
  - Sistema più facile da portare su nuove architetture
  - Più sicuro e affidabile (meno codice viene eseguito in modo kernel)
- Svantaggi
  - Possibile decadimento delle prestazioni a causa dell'overhead di comunicazione fra spazio utente e spazio kernel

Struttura di MAC OS X

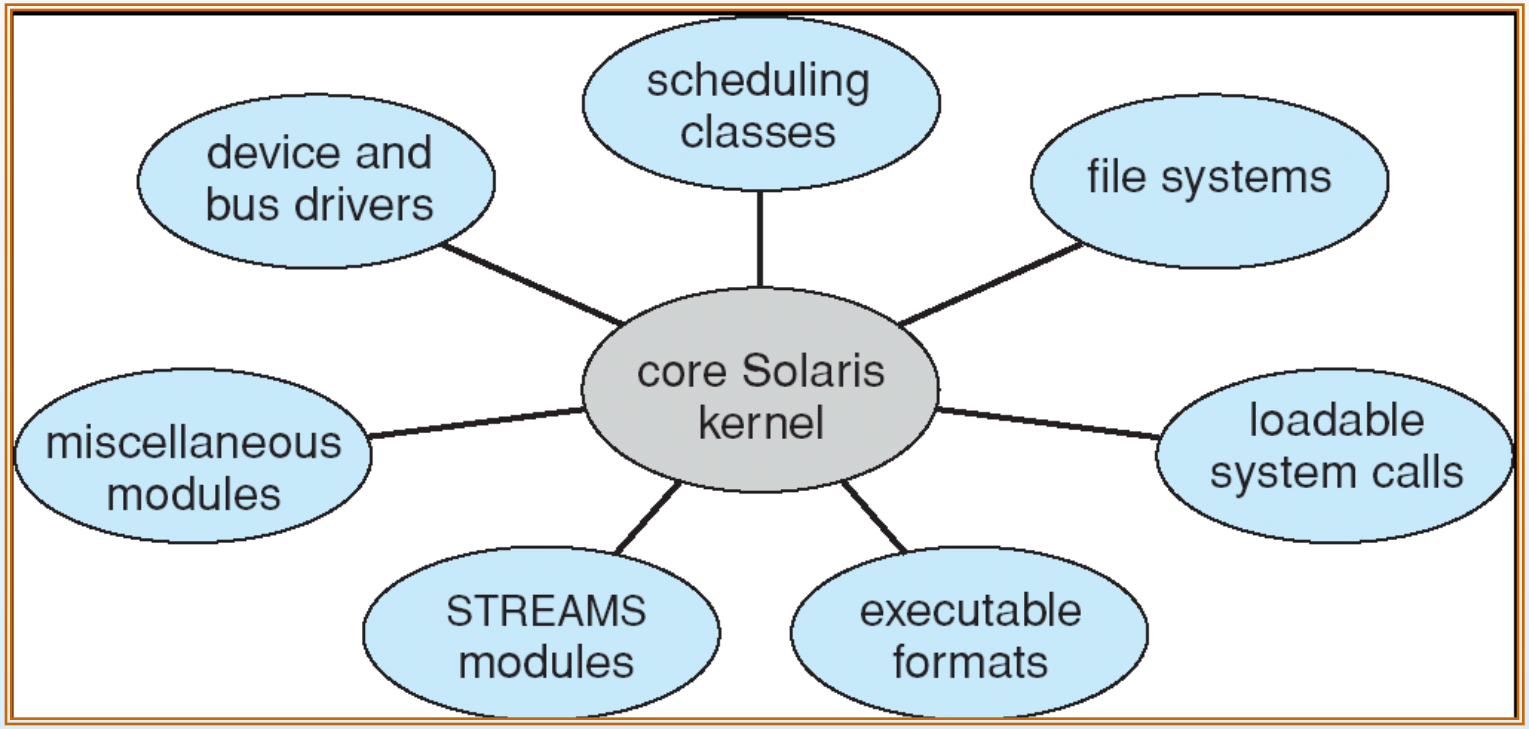
- Il microkernel Mach gestisce la memoria, le chiamate di procedura remote (RPC), la comunicazione fra processi (IPC) e lo scheduling dei thread
- Il kernel BSD mette a disposizione una CLI, i servizi legati al file system ed alla rete e la API POSIX



Kernel modulari

- In molti degli attuali SO il nucleo è realizzato in maniera modulare
  - Ciascun modulo implementa una componente base del kernel, con interfacce e funzioni definite con precisione
  - Ciascun modulo colloquia con gli altri mediante l'interfaccia comune
  - Ciascun modulo può essere o meno caricato in memoria come parte del kernel, secondo le esigenze (caricamento dinamico dei moduli, all'avvio o a run–time)
- L'architettura a moduli è simile all'architettura a strati, ma garantisce SO più flessibili (ogni modulo può invocare funzionalità da qualsiasi altro modulo): più facili da manutenere ed evolvere

Approccio modulare di Solaris



- L'organizzazione modulare lascia la possibilità al kernel di fornire i servizi essenziali, ma permette anche di implementare dinamicamente servizi aggiuntivi, specifici per il particolare sistema di calcolo

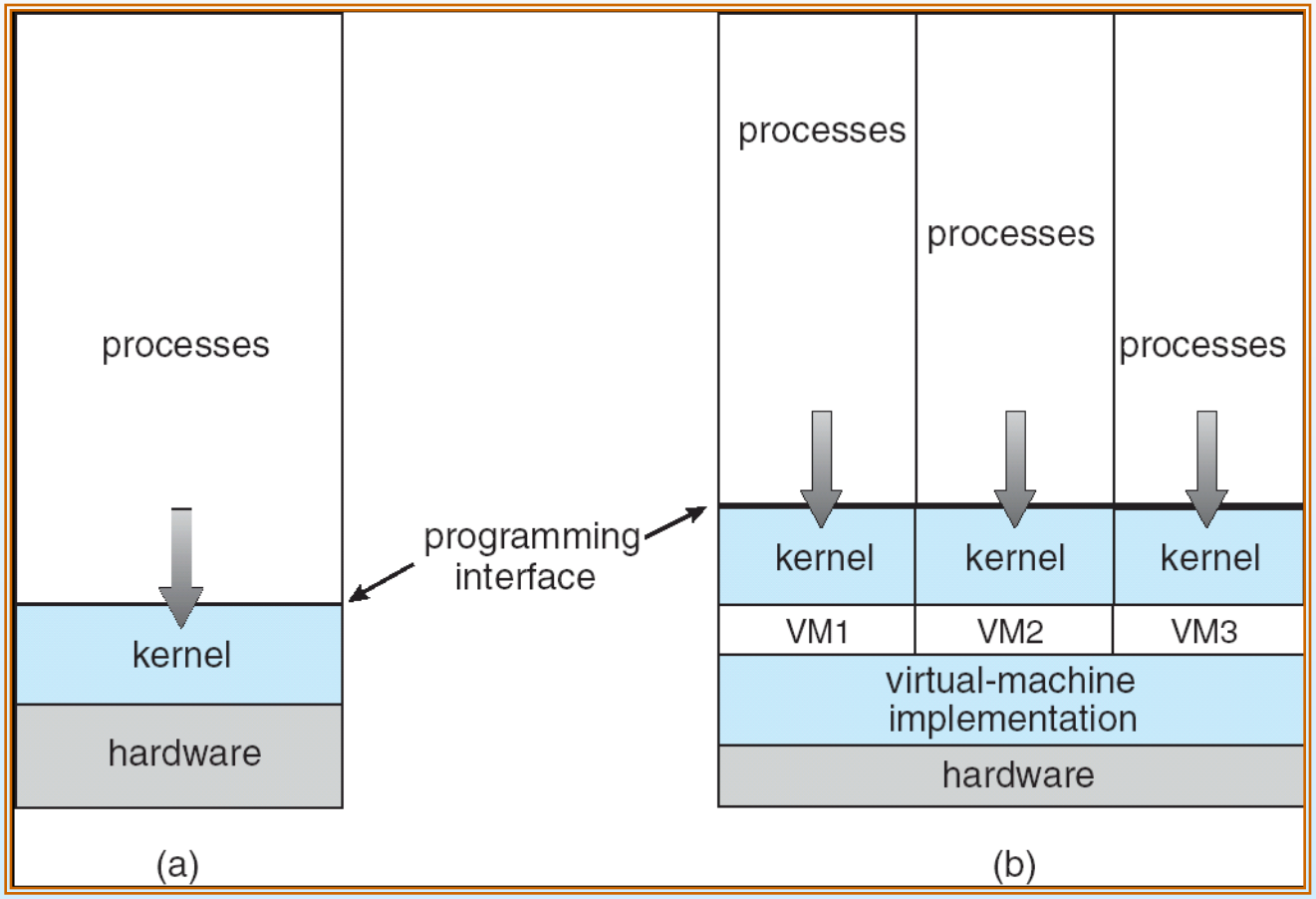
Macchine virtuali – 1

- La macchina virtuale porta l'approccio stratificato alle sue estreme conseguenze logiche
- Una macchina virtuale realizza un'interfaccia indistinguibile dalla macchina fisica sottostante: ogni processo ospite può usufruire di una copia virtuale di un calcolatore
  - Solitamente il processo ospite è un sistema operativo
- Le risorse del computer fisico vengono condivise dall'hypervisor in modo che ciascuna macchina virtuale sembri possedere il proprio processore, la propria memoria e i propri dispositivi
- Sia l'hardware virtuale che il sistema operativo (ospite) vengono eseguiti in un ambiente (strato) isolato

Macchine virtuali – 2

- Le risorse del computer fisico vengono condivise in modo da creare le macchine virtuali
- L'hypervisor può essere parte di un SO host (per esempio un modulo) o un micro–kernel che genera le macchine virtuali (VM)
- L'hypervisor può sfruttare caratteristiche specifiche del processore (hardware virtualization)
  - Lo scheduling della CPU può creare l'illusione che ogni utente abbia un proprio processore
  - Lo spooling e il file system possono fornire dispositivi di I/O virtuali (per esempio, stampanti)
  - Lo spazio disco può essere "suddiviso" per creare dischi virtuali
  - Un normale terminale utente in time–sharing funziona come console per l'operatore della macchina virtuale

Macchine virtuali – 3



(a) Sistema semplice (b) Macchina virtuale

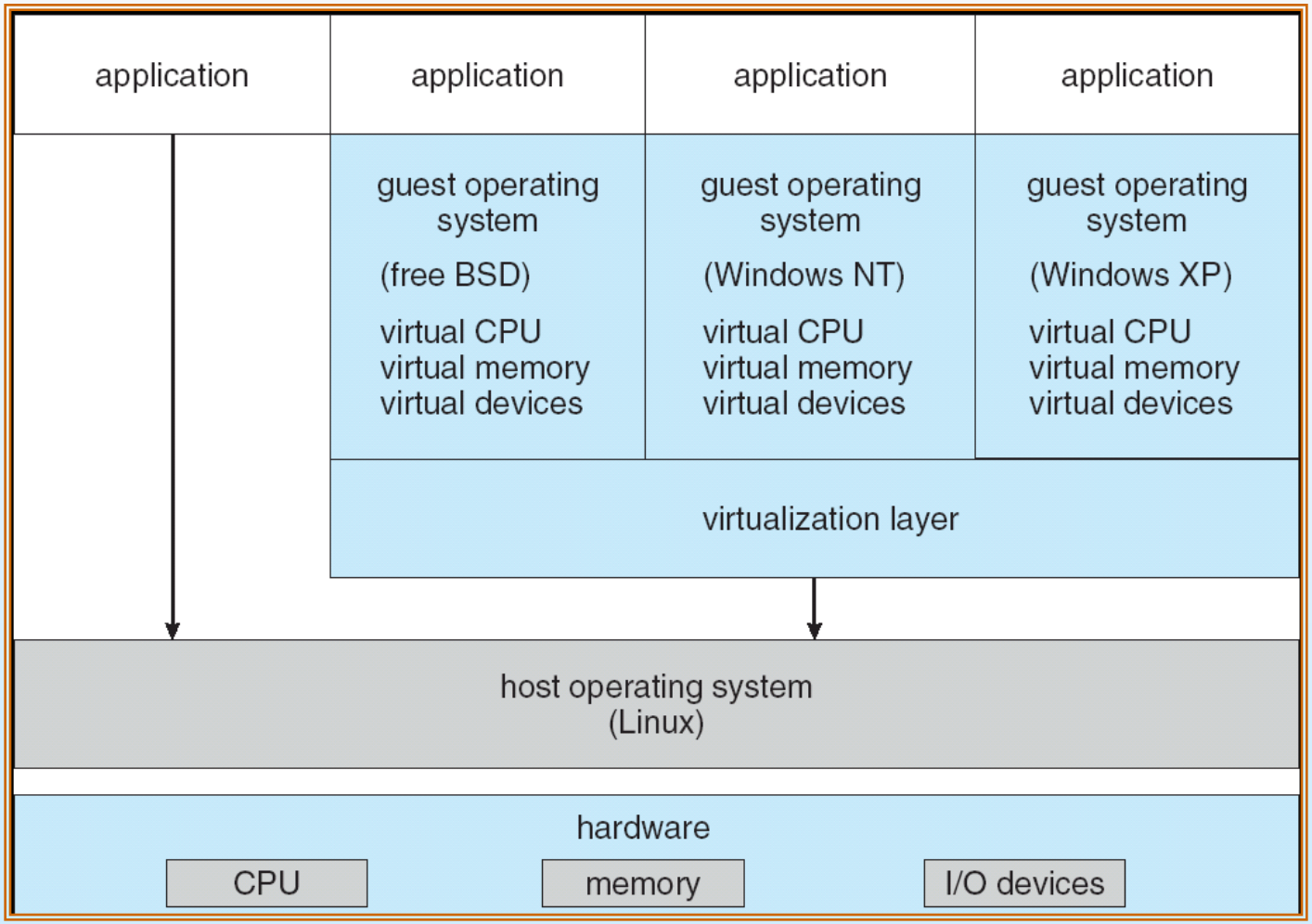
Macchine virtuali – 4

- Il concetto di macchina virtuale fornisce una protezione completa delle risorse di sistema (hardware e SO ospitante), dato che ciascuna macchina virtuale è isolata da tutte le altre
  - Questo isolamento, tuttavia, non permette, in generale, una condivisione diretta delle risorse
  - In Linux, condivisione possibile con *Virtio*
- Per la condivisione di risorse…
  - Condivisione di un volume del file system
  - Rete di macchine virtuali: ogni macchina virtuale può inviare/ricevere informazione sulla rete privata virtuale, modellata come una rete fisica, ma realizzata via software
  - La memoria inutilizzata viene ceduta ad altre VM

Macchine virtuali – 5

- Un sistema con macchine virtuali è un mezzo perfetto per la ricerca e lo sviluppo di sistemi operativi
  - Lo sviluppo del SO è effettuato sulla macchina virtuale, invece che sulla macchina fisica, così da non interferire con il normale funzionamento del sistema
- Tra i vantaggi dell'utilizzo di macchine virtuali vi è inoltre il fatto di poter offrire contemporaneamente ed efficientemente a più utenti diversi ambienti operativi separati
- Il concetto di macchina virtuale è difficile da implementare per il notevole sforzo richiesto per fornire un duplicato esatto della macchina fisica
- Le più diffuse sono Kvm, Xen, VMware, Virtualbox, VirtualPC, Parallels

Architettura VMware



Simulazione

- Sistema ospitante con una propria architettura, sistema ospite compilato per un'architettura diversa
- Esecuzione dei programmi su un emulatore in grado di tradurre le istruzioni del sistema ospite in istruzioni del sistema ospitante
  - Difficoltà nella realizzazione dell'emulatore
  - Possibilità di incrementare la vita dei programmi e mezzo per studiare vecchie architetture di sistema
  - Decadimento delle prestazioni: le istruzioni emulate vengono eseguite molto più lentamente delle istruzioni native

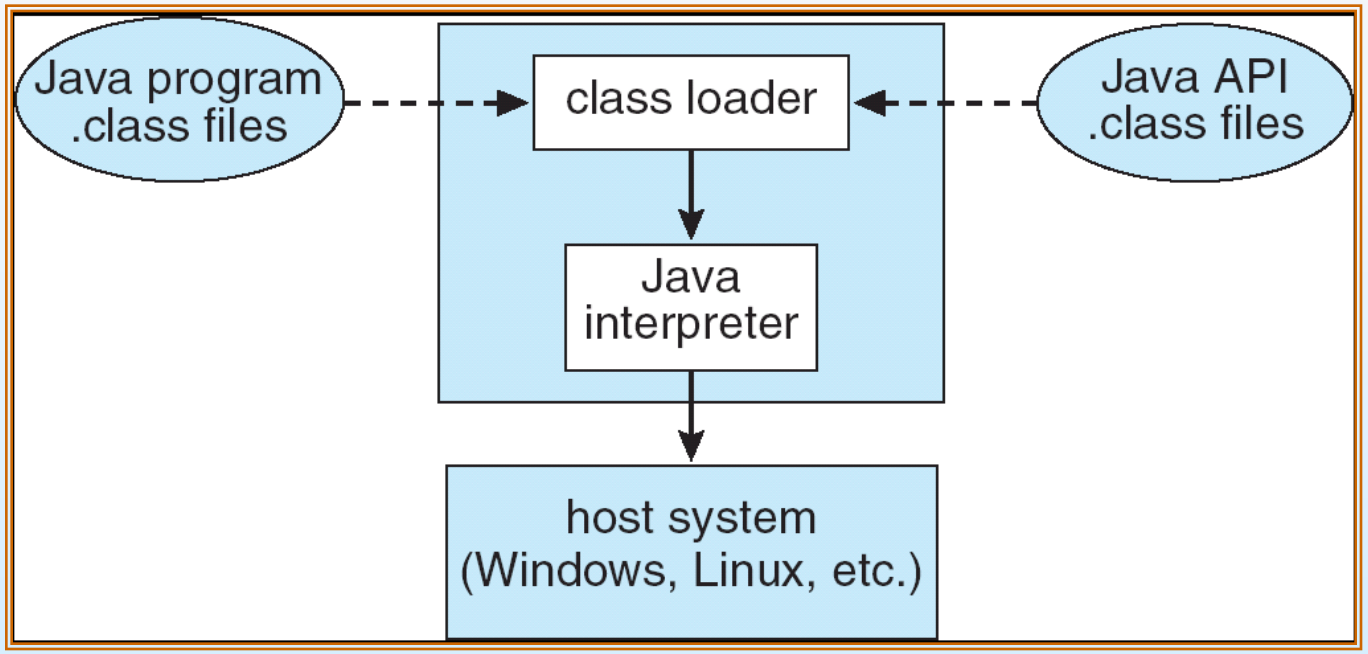
Java Virtual Machine – 1

- Il linguaggio di programmazione Java (1995) è un linguaggio orientato agli oggetti che, oltre a fornire una vasta libreria API, permette la definizione di una macchina virtuale
- Gli oggetti si specificano con il costrutto *class* e un programma consiste di una o più classi
  - Per ognuna, il compilatore produce un file (.class) contenente il *bytecode* – il linguaggio macchina della *Java Virtual Machine* (JVM), indipendente dall'hardware sottostante e che viene eseguito sulla macchina virtuale
- I bytecode sono controllati per verificare la presenza di istruzioni che possono compromettere la sicurezza della macchina

Java Virtual Machine – 2

- La JVM rende possibile lo sviluppo di programmi indipendenti dall'architettura e offre ai programmi scritti in Java un'astrazione uniforme del sistema
- I programmi Java sono però più lenti dei corrispondenti programmi scritti in C (*dynamic binary translation*)
- JVM disponibili per Windows, Linux, Mac OS X, Symbian e plugin presenti per vari browser, che a loro volta sono eseguiti su più piattaforme
- La disponibilità di implementazioni della JVM per diversi ambienti operativi è la chiave della portabilità di Java, proclamata nello slogan "*write once, run everywhere*"
- La macchina virtuale realizza infatti un ambiente di esecuzione omogeneo, che nasconde al software Java (e quindi al programmatore) qualsiasi specificità del SO (e dell'hardware) sottostante

Java Virtual Machine – 3



Debugging del sistema operativo

- Il debugging è l'attività di individuazione e risoluzione di errori nel sistema, i cosiddetti bachi *(bugs*)
- Anche i problemi che condizionano le prestazioni sono considerati bachi, quindi il debugging include anche il *performance tuning*, che ha lo scopo di eliminare i colli di bottiglia del sistema di calcolo
- Legge di Kerningham

  "Il debugging è due volte più difficile rispetto alla stesura del codice. Di conseguenza, chi scrive il codice nella maniera più intelligente possibile non è, per definizione, abbastanza intelligente per eseguire il debugging."

Analisi dei guasti

- Un guasto nel kernel viene chiamato crash
  - Come avviene per i processi utente, l'informazione riguardante l'errore viene salvata in un file di log, mentre lo stato della memoria viene salvato in un'immagine su memoria di massa (crash dump)
  - Tecniche più complesse per la natura delle attività svolte dal kernel
    - Il salvataggio del crash dump su file potrebbe essere rischioso se il kernel è in stato inconsistente
    - Il dump viene salvato in un'aria di memoria dedicata e da lì recuperato per non rischiare di compromettere il file system

Analisi delle prestazioni

- Esecuzione di codice che effettui misurazioni sul comportamento del sistema e salvi i dati su un file di log
- Analisi dei dati salvati nel file di log, che descrive tutti gli eventi di rilievo, per identificare ostacoli ed inefficienze
  - Il contenuto del file di log può essere utilizzato come input per simulazioni del comportamento del SO, nel tentativo di migliorarne le prestazioni
- In alternativa: introduzione, all'interno del SO, di strumenti interattivi che permettano ad amministratore ed utenti di monitorare il sistema (es., istruzione **`top`** di UNIX: mostra le risorse di sistema impiegate ed un elenco ordinato dei principali processi che le utilizzano)

Generazione del sistema operativo – 1

- I sistemi operativi sono progettati per essere eseguiti su una qualunque macchina di una certa classe; il sistema deve però essere configurato per ciascun particolare sistema di calcolo
- Per generare un sistema è necessario usare un programma speciale che può…
  - leggere da un file o richiedere all'operatore le informazioni riguardanti la configurazione specifica del sistema o, alternativamente,…
  - esplorare il sistema di calcolo per determinarne i componenti

Generazione del sistema operativo – 2

- Informazioni necessarie
  - Tipo di CPU impiegate e opzioni installate
  - Tipo di formattazione del disco di avvio (es. numero di partizioni)
  - Quantità di memoria disponibile
  - Dispositivi disponibili (tipo, numero del dispositivo, indirizzo fisico, numero del segnale di interruzione)
  - Scelta delle politiche (numero e dimensione delle aree di memoria per I/O, swapping, etc., algoritmi di sceduling, numero massimo di processi sostenibili)

Avvio del sistema operativo

- *Booting* — Fase di inizializzazione del computer realizzata tramite caricamento del kernel in memoria centrale
- Il *bootstrap loader* è un programma memorizzato in ROM (firmware) in grado (eventualmente caricando il *bootstrap* dal blocco di avvio) di localizzare il kernel, caricarlo in memoria ed iniziare la sua esecuzione
- Tutto il codice di avviamento residente su disco ed il SO stesso possono essere facilmente modificati
- Un disco che contenga una partizione di avvio è chiamato disco di sistema