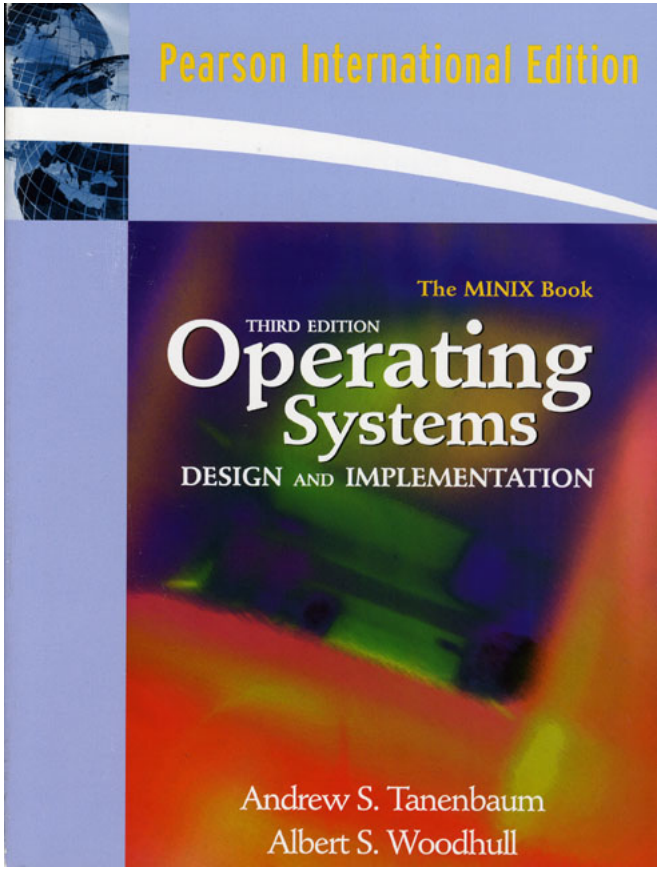
Sistemi Operativi

Lezione 1-2
Introduzione ai Sistemi Operativi

Il corso

- 6+3 ore di lezione settimanali (12 e 18 crediti)
- Le lezioni di laboratorio saranno interposte a quelle di teoria
- Esame:
  - Scritto (domande a risposta multipla) + Orale
  - Prova pratica per la parte di laboratorio
  - Sono previste 2 prove intermedie
- Libro di testo per la parte di teoria: “Operating Systems Design and Implementation”, di A. Tanenbaum e A. Woodhull, III ed.
- Sito ufficiale del corso: http://homes.dico.unimi.it/sisop

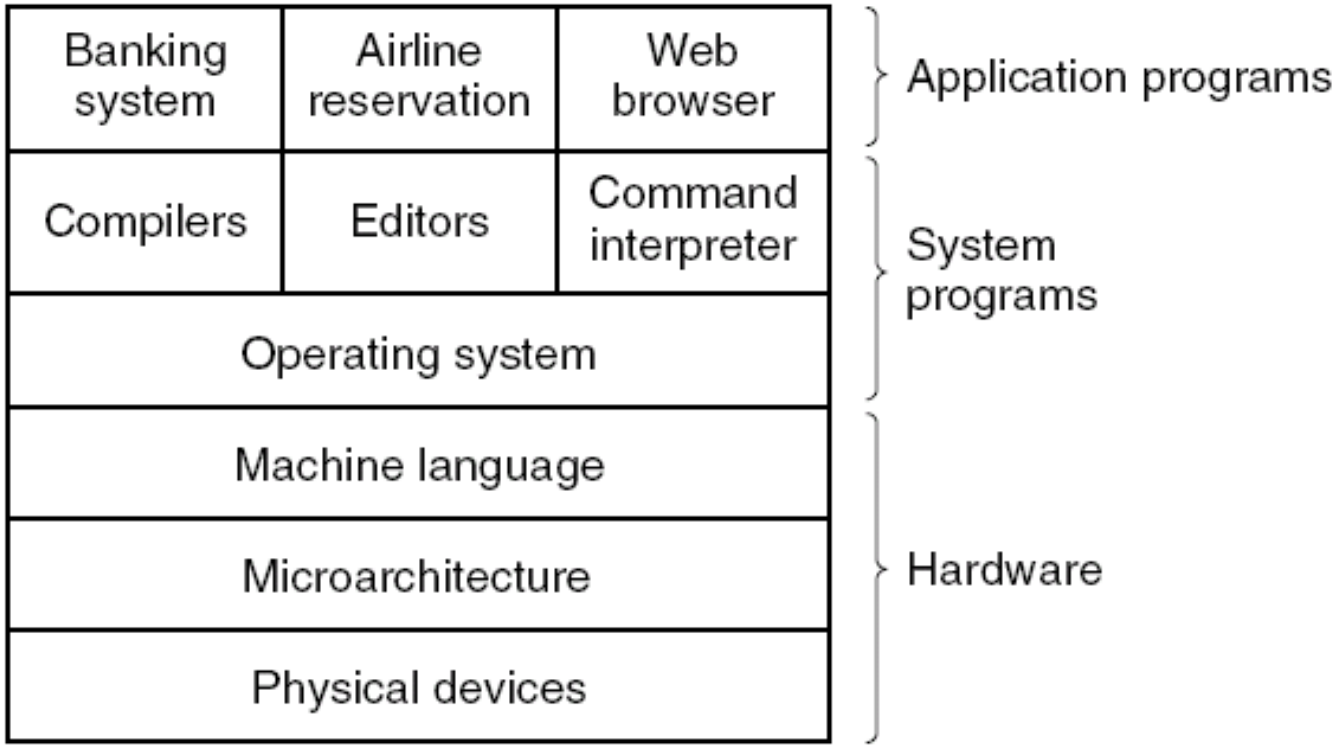
LIBRO DI TESTO



Il sistema operativo

- Il sistema operativo, è un insieme di programmi predisposti per assolvere ai seguenti compiti:
  - Gestire in modo ottimale le risorse di un calcolatore
  - Facilitare a programmatori ed utenti finali l'uso della sottostante macchina hardware

Una gerarchia di Sistema



Kernel mode

- La caratteristica che contraddistingue il sistema operativo dalle altre componenti del software di base è la modalità d'esecuzione con cui opera
- Il sistema operativo, o più precisamente alcune sue componenti, è l'unico programma presente in un calcolatore che opera con il processore in modalità kernel contrariamente agli altri programmi che operano PRINCIPALMENTE in user space

Il software di base

- Il SO è una componente del sw di base di un sistema
- I compiti che il software di base deve assolvere sono:
  - Abilitare l'uso del computer e delle sue componenti ad un utente
  - Gestire le risorse del sistema
  - Facilitare l'uso delle stesse ai programmatori di applicazioni
- Esempi di sw di base:
  - Compilatori e interpreti
  - DBMS
  - Sistemi operativi
  - Sistemi operativi di rete

Il software applicativo

- I compiti che il software applicativo deve assolvere sono:
  - Soddisfare le varie esigenze degli utenti finali (utilizzatori) in merito all'uso del calcolatore nelle loro attività
- Esempi di sw applicativo:
  - Word processor, foglio elettronico
  - Contabilità, Fatturazione
  - WWW, Posta elettronica, News

Evoluzione dei sistemi operativi

Le generazioni

- Generazione 1 (1945 – 55): Valvole modello open shop, applicazioni scientifiche
- Generazione 2 (1955 – 65): Transistor, sistemi batch, applicazioni scientifiche e prime applicazioni commerciali
- Generazione 3 (1965 – 80): ICs, sistemi multiprogrammati e timesharing, applicazioni commerciali
- Generazione 4 (1980 – Present): VLSI, applicazioni personali

Unità di misura

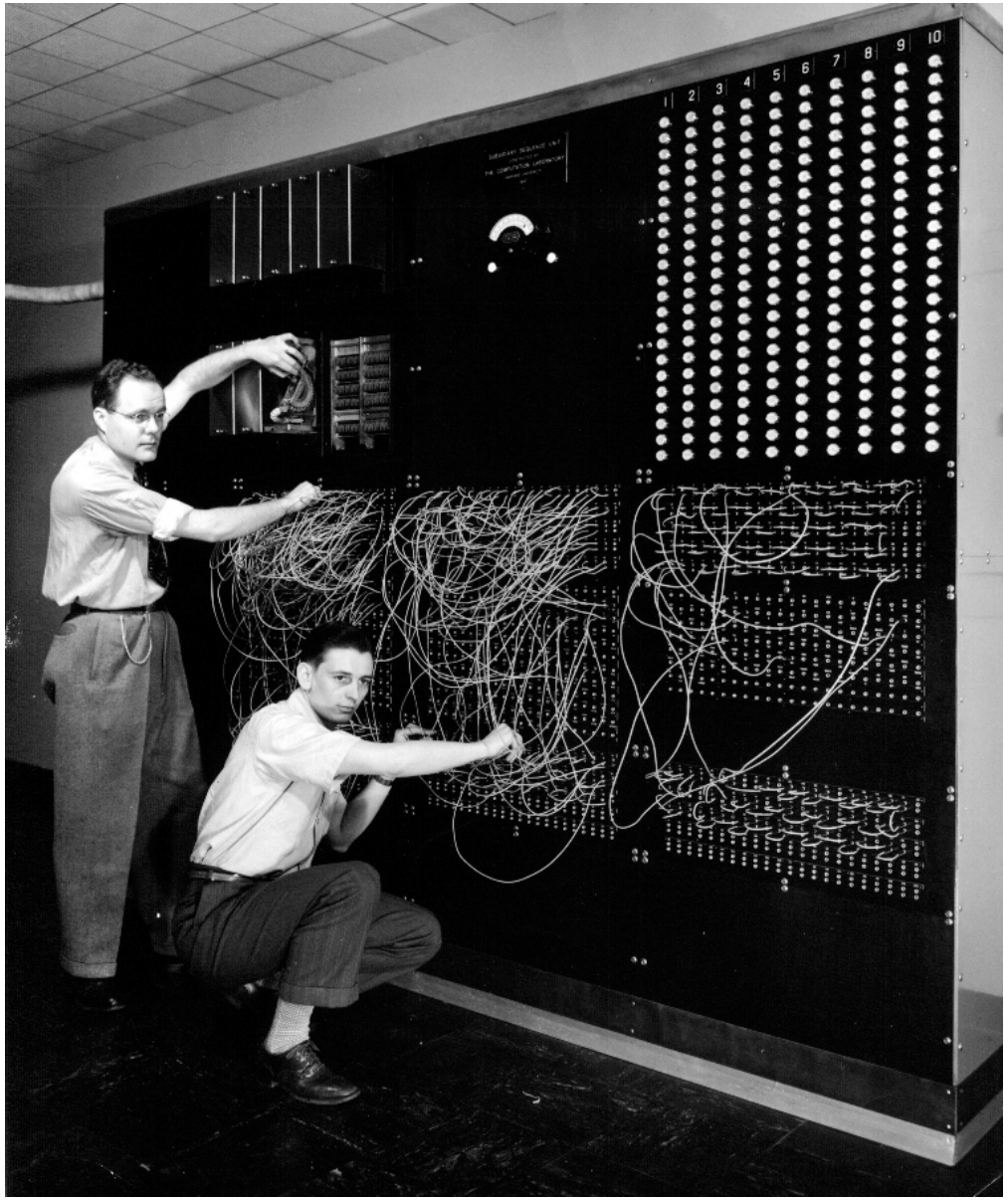
Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^{3}	1,000	Kilo
10^{-6}	0.000001	micro	10^{6}	1,000,000	Mega
10^{-9}	0.000000001	nano	10^{9}	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta

Premesse

- Tempo di turnaround: intervallo di tempo che intercorre tra l'istante di tempo in cui si sottopone un processo (job) al sistema e l'istante di tempo in cui termina l'esecuzione del job
- Execution time: quantità di tempo utilizzata dalla CPU per eseguire un determinato processo
- Utilizzo CPU in un intervallo di tempo T: execution time/T
- Troughput in un intervallo T: numero di processi eseguiti in T

Premesse

- Risorse usate da un'applicazione tipica
  - Input time manuale 15 min
  - Input time automatico 0.3 min
  - Output time 0.5 min
  - Execution time 1 min



1944-Harvard Mark-1

Conceived by Harvard professor Howard Aiken, and designed and built by IBM, the Harvard Mark-1 was a room-sized, relay-based calculator.

The machine had a fifty-foot long camshaft that synchronized the machine's thousands of component parts.

The Mark-1 was used to produce mathematical tables but was soon superseded by stored program computers.

1946-February, the public got its first glimpse of the ENIAC, a machine built by John Mauchly and J. Presper Eckert that improved by 1,000 times on the speed of its contemporaries.
Start of project:

1943Completed:

1946Programmed:plug board and switches

Speed:5,000 operations per second

Input/output:cards, lights, switches, plugs

Floor space:1,000 square feet

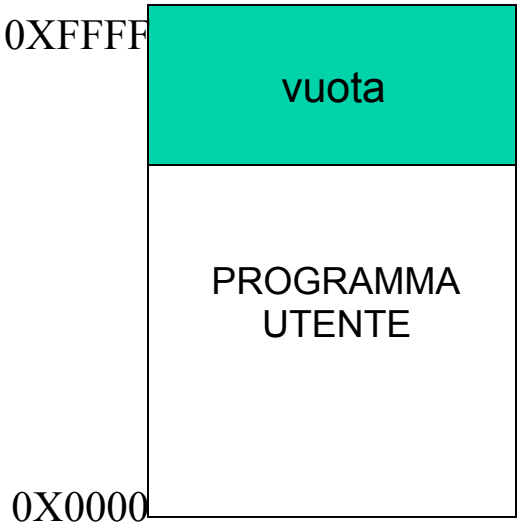
Project leaders:John Mauchly and J. Presper Eckert.

Open Shop

- Gli utenti accedevano a turno al calcolatore dove caricavano ed eseguivano i loro programmi
- Sistema poco efficiente
  - Un utente ogni 20 min circa
  - Utilizzo CPU 1/20 = 5%

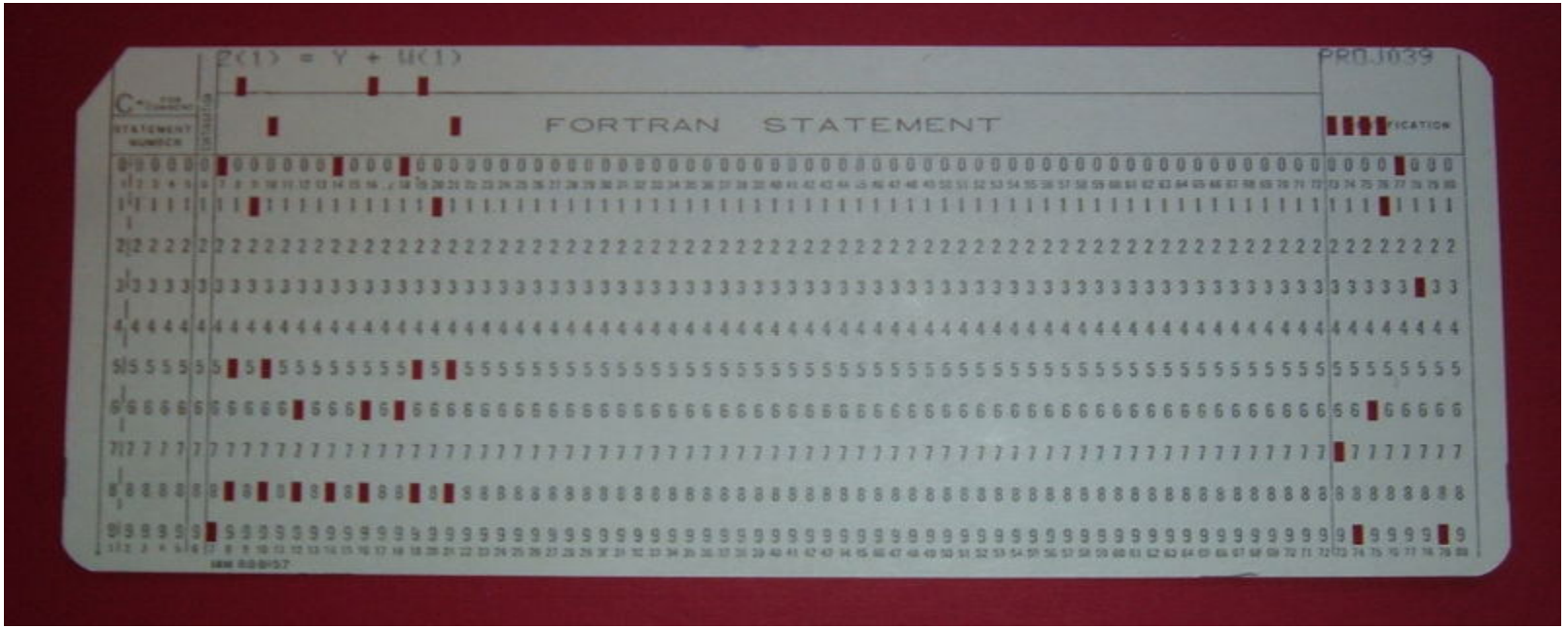
I Generazione

- In questo caso la memoria del calcolatore era completamente a disposizione dell'utente finale ed appariva in questo modo:



II Generazione

- Un primo miglioramento derivò dall'eliminazione dei tempi di input di dati e programmi con l'introduzione delle schede perforate
- L'utente preparava off-line il suo programma su schede perforate
- Accedeva al calcolatore a cui faceva leggere le schede, il programma veniva eseguito ed i risultati stampati



Punch Cards

The IBM card format, designed in 1928, had rectangular holes, 80 columns with 12 punch locations each, one character to each column. Card size was exactly 187.325 by 82.55 mm.

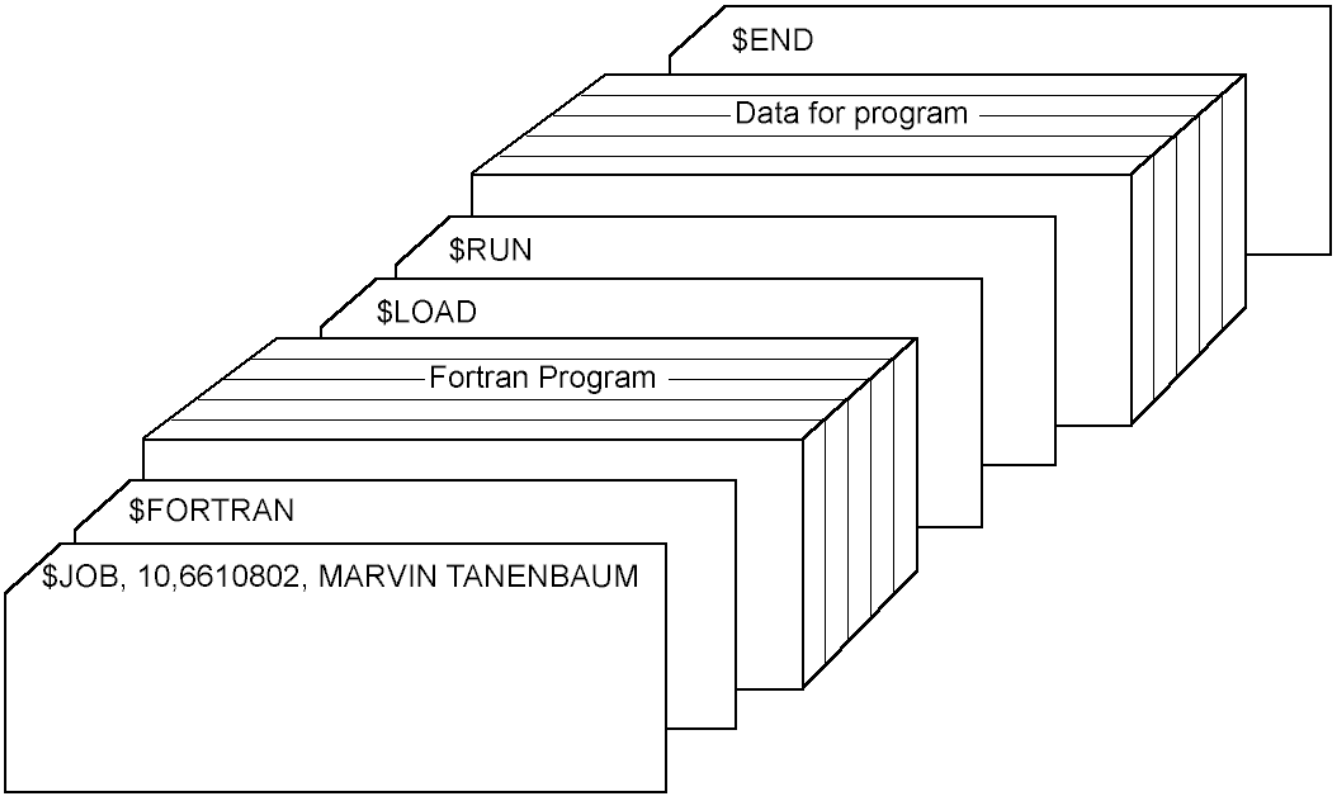
The top two positions of a column were called zone punches, 12 (top) and 11. These often encoded plus and minus signs. The lower ten positions represented (from top to bottom) the digits 0 through 9. Originally only numeric information was coded, with 1 or 2 punches per column: digits (digit [0-9]) and signs (zone [12,11]).

Later, codes were introduced for upper-case letters and special characters. A column with 2 punches (zone [12,11,0] + digit [1-9]) was a letter; 3 punches (zone [12,11,0] + digit [2-4] + 8) was a special character.

IBM Keypunch Machine 026

http://www.tietokonemuseo.saunalahti.fi/eng/late_eng.htm

II Generazione



II Generazione

- L'avvento dei lettori di schede incominciò a richiedere la presenza in memoria centrale di un programma che fosse in grado di:
  - Gestire l'unità periferica (CR)
  - Leggere le schede
  - Interpretare ed eseguire i comandi richiesti (Job Control Language)
  - Copiare il contenuto delle schede in memoria
  - A lettura terminata "lanciare" l'esecuzione del programma

Profilo di esecuzione

- Il profilo temporale dell'esecuzione di un insieme di programmi in un sistema di II generazione è

- Nel caso di applicazioni standard, il throughput del sistema è 33 job/h, con un uso della CPU di 33/60 = 55%

Batch

- Nasce l'esigenza di ottimizzare l'uso dei sistemi di calcolo e di eliminare dal "ciclo produttivo" le unità di I/O
- Le unità di I/O che fino a quel momento erano state collegate ai calcolatori, card reader e stampanti, vengono sostituite da unità a nastri molto più veloci
- Si modifica completamente lo schema di accesso ad un calcolatore

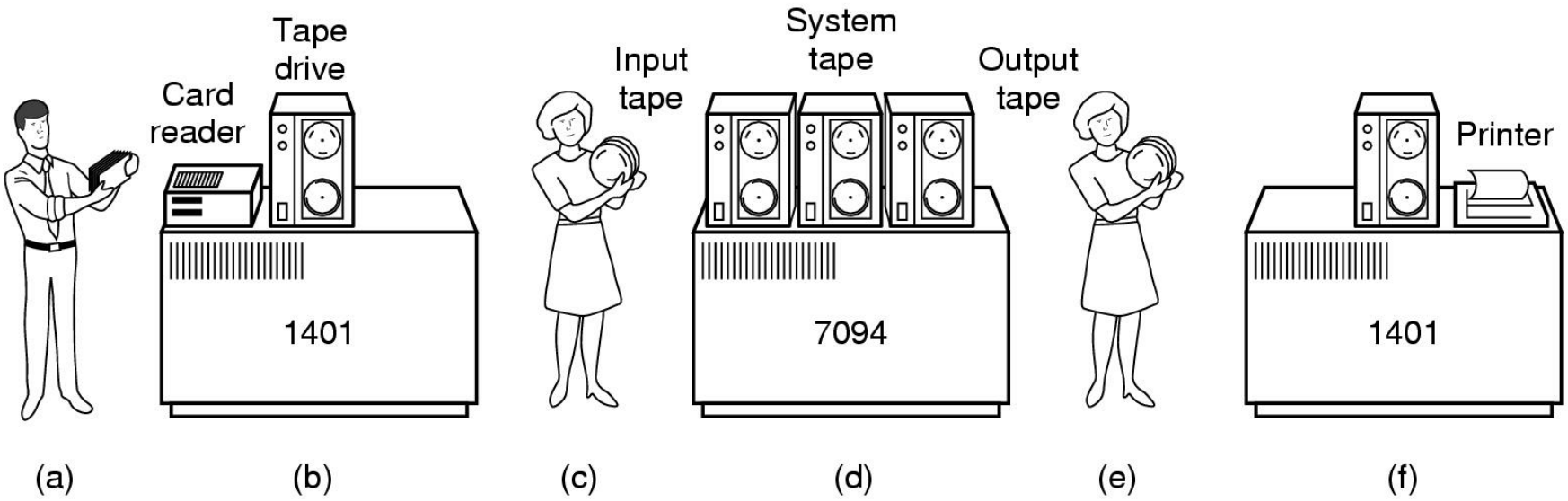
Magnetic Tapes

Magnetic tape has been used to store digital information since at least 1951 when Presper Eckert and John Mauchly used it in the UNIVAC I computer. Their tape was made of metal, but later tapes have been made mostly of plastic.

Some users of punched cards were initially reluctant to use tape because they could no longer see their data. IBM's invention of the "vacuum channel" tape drive and improved magnetic materials resulted in reliable large-capacity tapes, which even reluctant customers eventually adopted.

Tape data formats vary widely, and bit density has increased dramatically. Tape has now been largely replaced by hard disks for secondary storage, but it is still used for backing up of data.

Batch



1964 - IBM announced the System/360, a family of six mutually compatible computers and 40 peripherals that could work together.

The initial investment of $5 billion was quickly returned as orders for the system climbed to 1,000 per month within two years.

At the time IBM released the System/360, the company was making a transition from discrete transistors to integrated circuits, and its major source of revenue moved from punched-card equipment to electronic computer systems

Profilo di esecuzione

- Il profilo temporale dell'esecuzione di un insieme di programmi in questo caso può essere così schematizzato

- Nel caso di applicazioni standard, il throughput del sistema è 55 job/h, con un uso della CPU di 55/60 = 91%

Batch

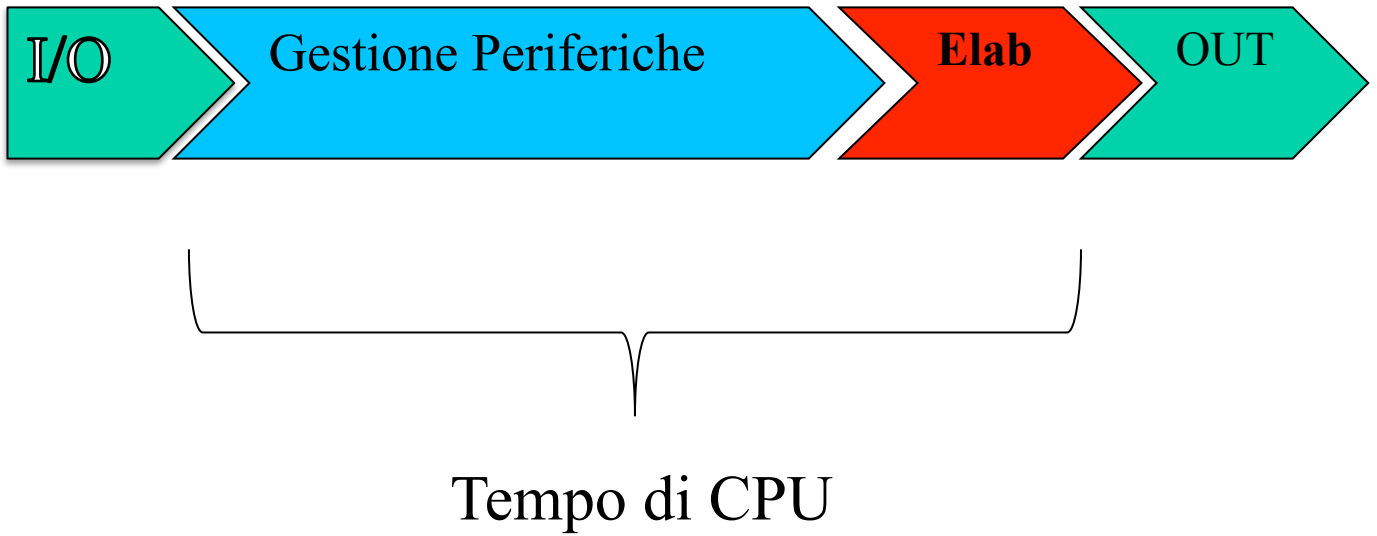
- Il tempo di risposta era diventato estremamente elevato
- L'utente doveva attendere la fine del batch per poter accedere ai risultati della propria elaborazione
- Nella maggioranza dei casi il tempo di risposta si aggirava intorno alle 24 ore

Batch

- Un ulteriore elemento di criticità presente nei sistemi batch era costituito dalla modifica delle applicazioni “tipo”
  - Sino alla fine degli anni ’50 il calcolatore era usato principalmente per computazioni di tipo scientifico che richiedevano quindi un uso intensivo della CPU (CPU bound)
  - Dalla fine degli anni ’50 il calcolatore incominciò ad essere utilizzato in applicazioni commerciali, caratterizzate da un forte uso di I/O

Criticità Batch

- Si assisteva quindi al seguente fenomeno



III Generazione: Multiprogrammazione

- Applicazioni tipicamente commerciali o molto interattive
- Nascono come risposta al problema di un miglior sfruttamento della CPU
- SO di riferimento: IBM OS/360

1972 – PDP 11/40 Minicomputer

DEC developed the PDP-11 as a family of 16-bit minicomputers that could grow with customers as their computing needs increased.

From the first PDP- 11/20 in 1970 through the PDP-11/94 in 1990, DEC produced a variety of compatible machines and sold over 500,000.

Much like the 12-bit PDP-8 before it, DEC further integrated the PDP-11 family until in 1982 it had placed one of its largest models, the PDP-11/70, on two large-scale integrated circuits.

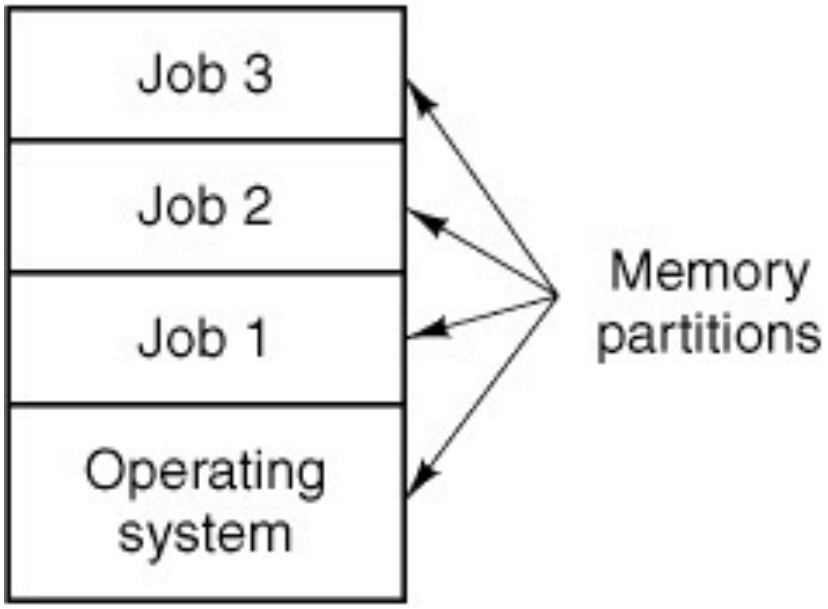
Since the PDP-8 and PDP-11 systems were subject to export restrictions, Soviet bloc computer companies commonly cloned the systems and packaged them under different names.

Memory Type:Core
Speed:1.25 MHz +
Memory Size:56K+
Cost:$20,000 +
Memory Width:(16-bit)

Multiprogrammazione

- I sistemi multiprogrammati godono delle seguenti caratteristiche:
  - Consentono la coesistenza CONTEMPORANEA in memoria centrale di due o più programmi
  - Adottano meccanismi che consentono di svincolare l'attività della CPU da quella delle periferiche di I/O (vedi slide successive)
  - Ottimizzano l'uso della CPU

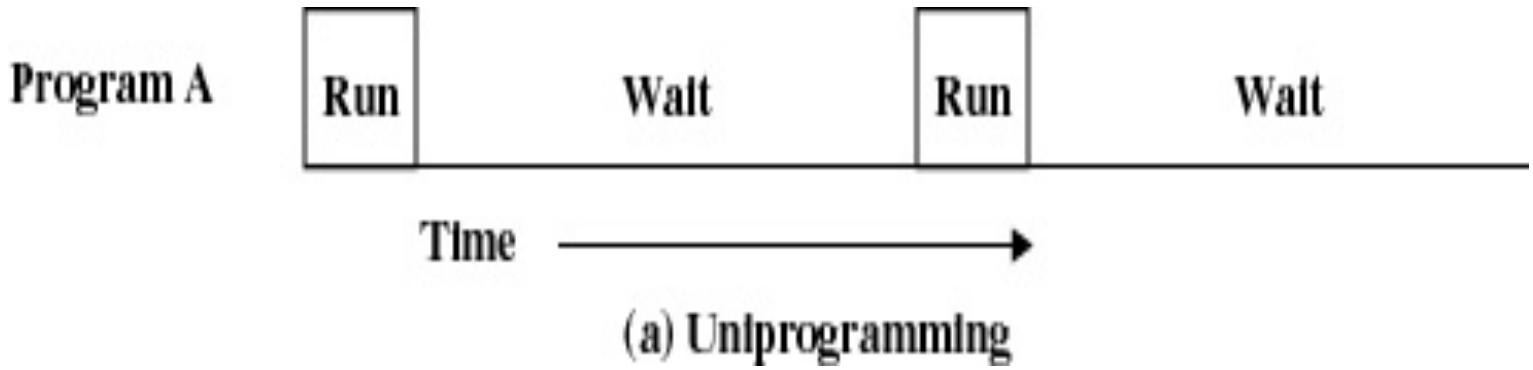
Multiprogrammazione



Multiprogrammazione

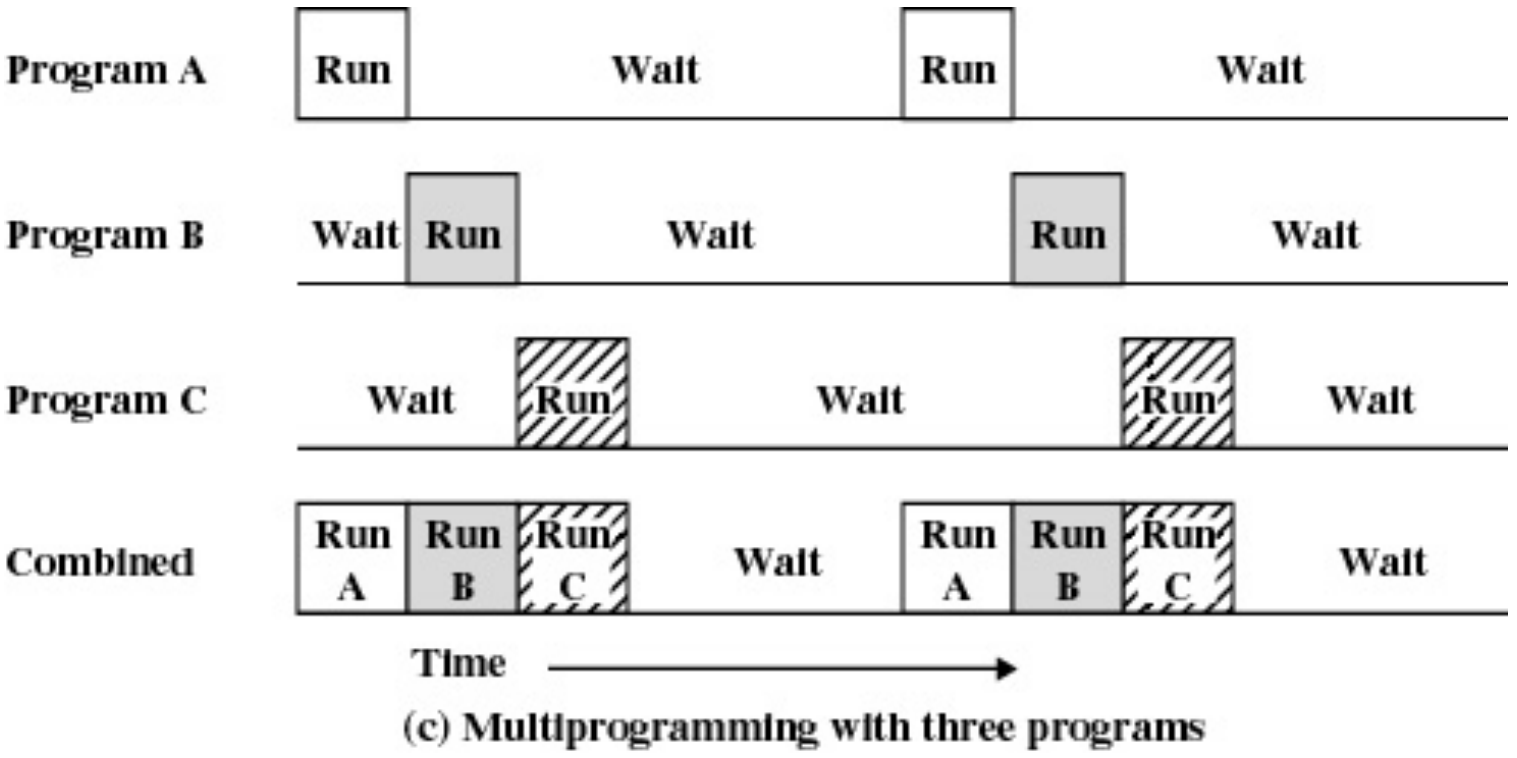
- Principio di funzionamento:
  - Viene scelto un programma P1 da eseguire, tra quelli presenti in memoria
  - Quando P1 richiede lo svolgimento di un'operazione di I/O, la stessa viene demandata alla periferica, il programma P1 viene momentaneamente sospeso, sarà ripreso successivamente
  - La CPU procede nell'esecuzione di un altro programma

Mono programmazione



(a) Uniprogramming

Multiprogrammazione



(c) Multiprogramming with three programs

Interrupt

- Per realizzare questa modalità ci si è rifatti ad un uso intensivo dei segnali di interrupt
- L'interrupt è un segnale elettrico inviato da un dispositivo esterno, al microprocessore più precisamente al controller
- L'interrupt consente al processore di interrompere le attività in corso e di eseguirne altre

III Gen.: Multiprogrammazione

- Affinché lo schema illustrato funzioni, è necessario disporre di:
  - Routine di gestione degli interrupt
  - Moduli per la gestione dei programmi sospesi e di quelli pronti all'esecuzione
  - Moduli per la gestione delle periferiche
    - più processi possono richiedere l'uso della stessa risorsa
  - Moduli per la gestione della memoria
- Tutte queste funzionalità sono accorpate nel sistema operativo

Time Sharing

- Sistemi altamente interattivi
- Ogni utente lavora interattivamente e riceve periodicamente attenzione dal calcolatore
- L'impressione che ne ottiene è di avere a disposizione un sistema dedicato
- Un sistema time-sharing non deve necessariamente essere multiprogrammato
- CTS5 → Multics → Unix

Batch vs. Time sharing

	Batch Multiprogramming	**Time Sharing**
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	language commands provided with the job Job control	Commands entered at the terminal

Real-time

- Sistemi che devono garantire l'esecuzione di certe operazioni entro un limite di tempo prefissato
- La nozione di real-time è legata all'ambito applicativo
- Sistemi hard real-time
  - i vincoli di tempo non possono essere violati pena gravi danni al sistema e all'ambiente
- Sistemi soft real-time
  - il mancato rispetto dei vincoli di tempo porta ad un degrado delle prestazioni del sistema

IV Generazione

- Sistemi operativi più facili da utilizzare
  - Installabili da utenti poco esperti
  - Utilizzabili da ogni classe di utente
- Nasce la nozione di sistema user-friendly
  - Molta enfasi viene posta sulle interfacce utente, grazie soprattutto al lavoro della Apple che introduce le interfacce a finestre

IV Generazione

- I principali sistemi operativi di IV generazione sono:
  - UNIX
    - molto diffuso su workstation nelle varie versioni: HP-UX, SOLARIS, AIX, ULTRIX
    - in ambiente PC: LINUX, FreeBSD, OpendBSD
  - MS-DOS → W95 → WNT → W2000 → XP → VISTA
    - Nato per PC IBM e compatibili, che usavano il processore Intel 8088
    - Con WNT Microsoft ha iniziato ad operare sul mercato delle WKS

IV Generazione

- I sistemi operativi di IV generazione hanno dovuto, per primi, fare i conti con una nuova risorsa: LA RETE
- A partire dalla metà degli anni ’80, lo sviluppo di protocolli per reti locali prima e per reti geografiche immediatamente dopo ha favorito lo sviluppo delle reti di calcolatori

IV Generazione

- Per poter operare in una rete, un calcolatore deve essere predisposto del necessario
  - HW
    - cavi e schede di rete
  - SW
    - protocolli per la comunicazione con gli altri host collegati in rete
- I protocolli più diffusi
  - Ethernet per le reti locali
  - TCP-UDP/IP per le reti geografiche

IV Generazione

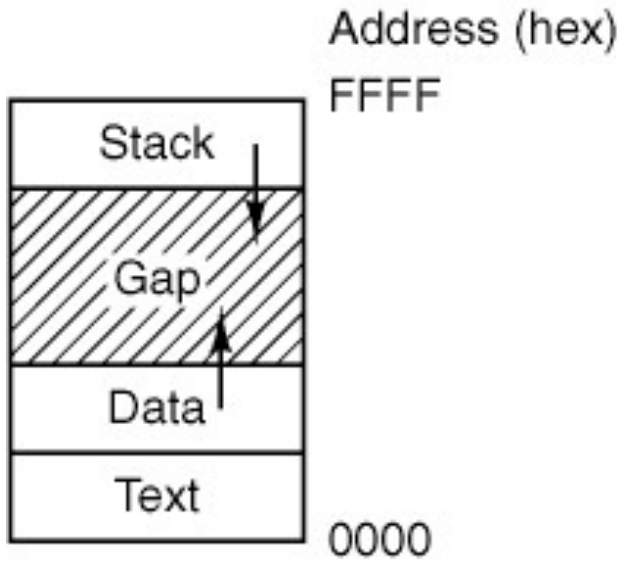
- Si assiste ad una continua evoluzione verso l'usabilità del sistema, che richiede interfacce sempre più vicine all'uomo
- Vi è una continua evoluzione di unità periferiche e sistemi di interconnessione (bluetooth, wi.fi, digital camera, biometric devices) che il sistema operativo deve essere in grado di inglobare e gestire
- La complessità del SO sta crescendo

Le funzionalità di un SO

- L' insieme dei programmi che compongono un SO svolgono le seguenti funzioni:
  - Inizializzazione del sistema
  - Gestione dei processi
  - Gestione della Memoria
  - Gestione delle periferiche I/O (tra cui gli apparati di comunicazione)
  - Gestione dei file
  - Protezione
  - Interprete dei comandi

Processi

- Un processo è una qualunque attività svolta dal sistema operativo ed è sempre riconducibile all'esecuzione di un programma
- Un processo è caratterizzato da tre componenti: data, testo, stack



Processi

- Il sistema operativo è responsabile per lo svolgimento delle seguenti attività inerenti la gestione dei processi:
  - Creazione e cancellazione dei processi
  - Sospensione e “risveglio” dei processi
  - Predisposizione di meccanismi per:
    - Sincronizzazione tra processi
    - Comunicazione tra processi

Gestione della Memoria

- *La memoria centrale* è un enorme array di byte ciascuno con i propri indirizzi usato per la memorizzazione temporanea di dati e programmi accessibili dal CPU e dispositivi di I/O
- Le attività di memory management svolte dal SO sono
  - Tenere traccia delle porzioni di memoria usate e di chi le sta usando
  - Decidere quali attività svolgere quando si liberano porzioni di memoria
  - Allocare e deallocare spazi di memoria quando richiesto

Secondary-Storage Management

- Siccome la memoria centrale è piccola e volatile, nel computer è previsto un sistema di memoria secondaria per preservare le informazioni da elaborare
- I dischi sono la periferica principale usata a questo scopo
- Nell'ambito del disk management il SO provvede a:
  - Allocare spazi di disco ai processi che ne fanno richiesta
  - Gestire gli spazi liberi
  - Gestire le richieste di accesso al disco da parte dei processi

Gestione I/O

- Un sistema di calcolo è composto da dispositivi di I/O tra loro molto diversi
- In questo ambito il SO deve predisporre
  - Le primitive per il trasferimento dei dati da e verso le periferiche
  - Ottimizzare l'acquisizione ed il trasferimento dei dati
  - Provvedere alla gestione di tutte le periferiche connesse

File Management

- Un *file* è una collezione di dati opportunamente organizzati
- In questo ambito sono di competenza del file system le seguenti attività:
  - Creazione e cancellazione di file
  - Creazione e cancellazione di directory
  - Primitive per la manipolazione di file e directory
  - Protezione di file e directory
  - Trasferimento di file da e verso dispositivi di memoria secondaria

Protezioni

- *Con Protezioni* si intende l'insieme dei meccanismi per controllare chi, in termini di utenti e processi, accede alle risorse del computer e con che modalità (lettura, scrittura, modifica, ecc.)
- I meccanismi di protezione devono:
  - Distinguere tra uso autorizzato e non autorizzato di risorse
  - Specificare i controlli da imporre nell'accesso alle risorse
  - Fornire i meccanismi per mettere in atto detti meccanismi

Comunicare con il SO

- Esistono due modalità attraverso le quali un utente può richiedere l'intervento del Sistema operativo:
  - Modalità interattiva: attraverso la shell
  - Modalità programmata: attraverso le syscall

Shell

- Ogni utente di un computer può interagire direttamente con il sottostante SO fornendogli dei comandi da eseguire (crea un file, esegui un programma, ecc.)
- Questi comandi, prima di essere eseguiti devono essere opportunamente interpretati
- Il programma che interpreta i comandi, non è propriamente una componente del SO opera infatti in user-mode e si chiama SHELL

Astrazione

- Che cos'è l'astrazione?
  - La descrizione di un processo o un artefatto in cui sono evidenziati gli aspetti più rilevanti e utili dell'oggetto in questione, eliminando o nascondendo una serie di dettagli ritenuti irrilevanti per l'uso a cui l'oggetto è destinato
- A cosa serve?
  - A creare, comprendere e gestire sistemi complessi

Astrazione

- Il risultato ottenuto:
  - La descrizione di un sistema complesso attraverso una serie di funzioni (interfaccia) che mettono in evidenza le funzionalità e le informazioni di interesse, nascondendo le informazioni ritenute poco utili
- L'output di un processo di astrazione è ovviamente dipendente dallo scopo a cui è destinato l'oggetto dell'astrazione
- Livello di astrazione: un punto in cui si inserisce un'astrazione nella descrizione di un sistema complesso

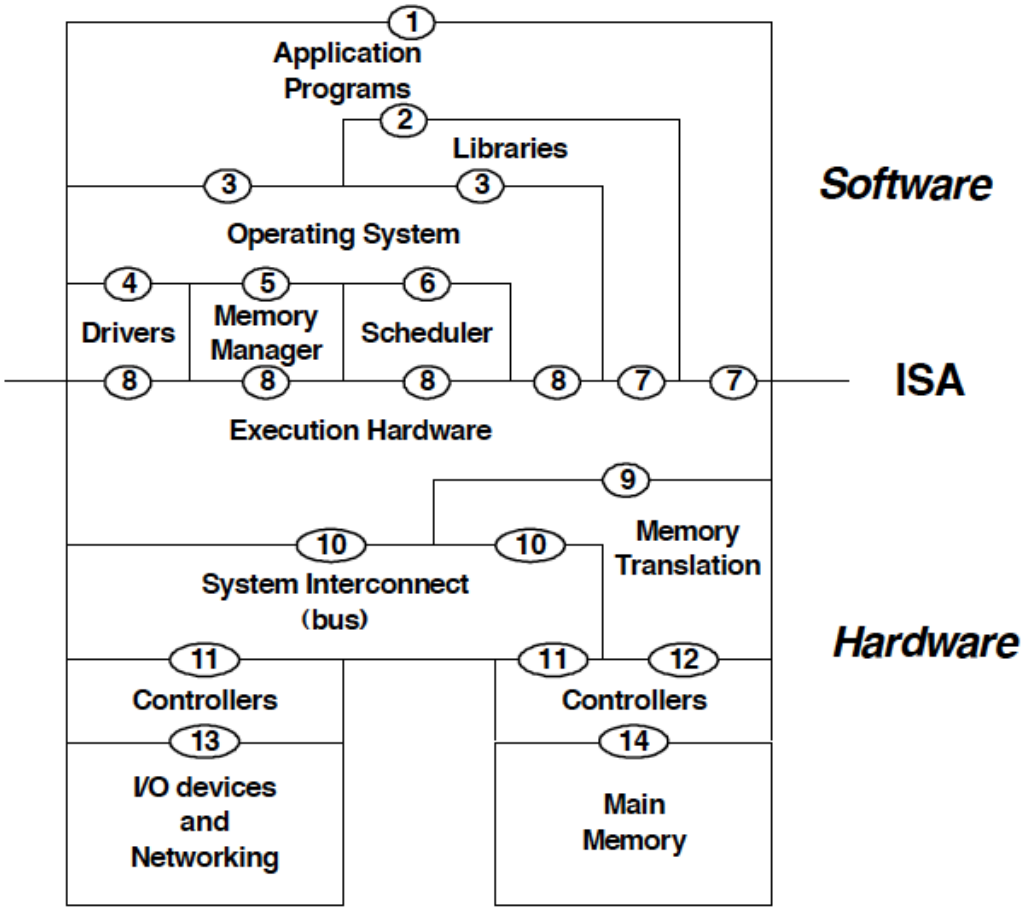


Figure 4. Computer system architectures; Implementation layers communicate vertically via the shown interfaces. This view of architecture is styled after one given by Glenford Myers (Myers 1982).

Syscall

- Il sistema operativo è gestore esclusivo di tutte le risorse di un sistema al fine di:
  - Garantire una gestione ottimale delle stesse
  - Evitare collisioni
  - Fornire modalità semplificate per il loro uso
- … quindi, chiunque debba usare una risorsa deve rivolgersi al sistema operativo:
  - Come?

L'interfaccia delle System Call

- L'interfaccia delle system call è la descrizione dell'insieme delle system call supportate da un OS
- Le system call possono essere ragruppate in termini di funzionalità svolte
  - Process control
  - File management
  - Device management
  - Information management
  - Communication management
  - Time Management

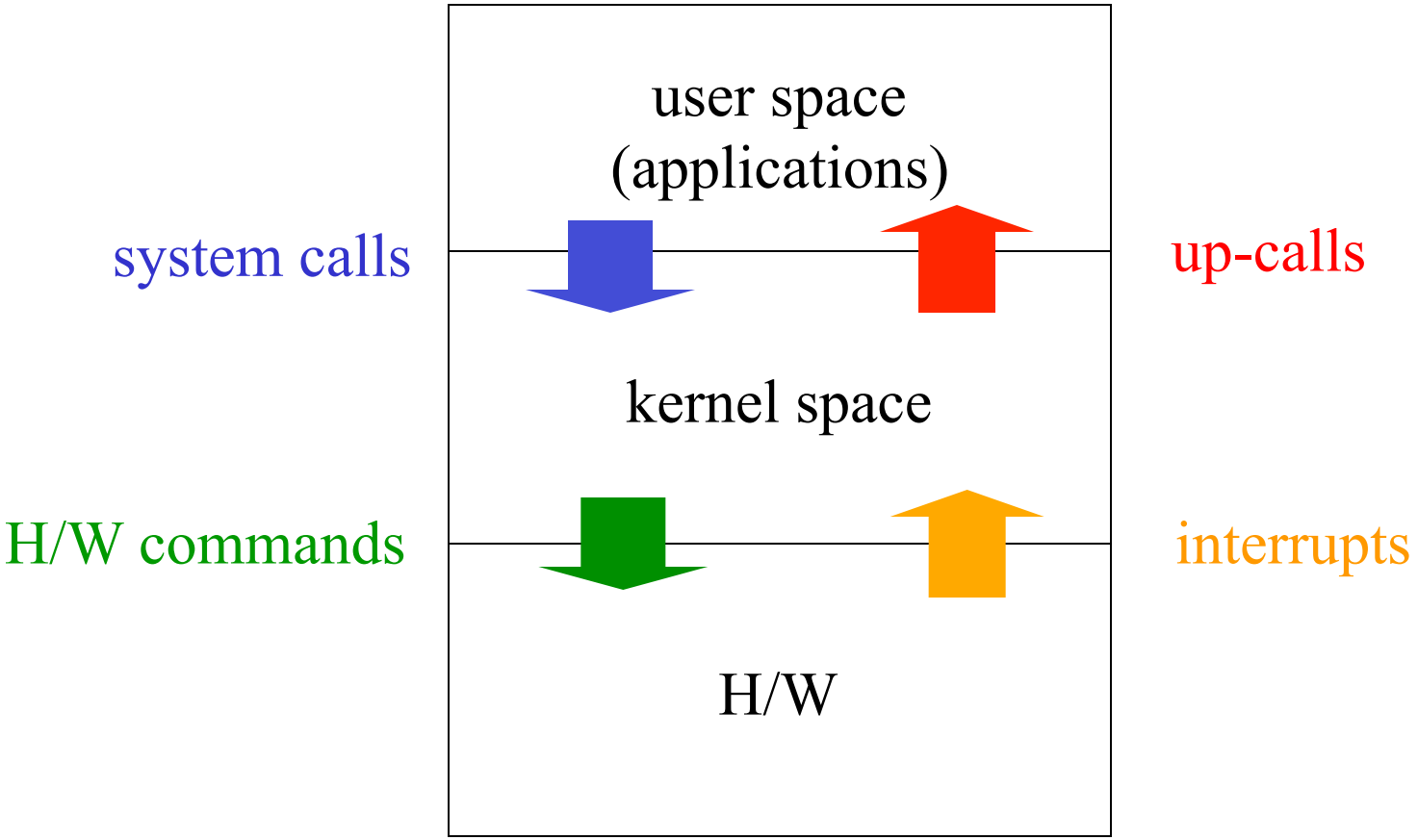
System call: cosa sono

- Le System calls sono funzioni implementate nel SO che possono essere usate da processi in user space
- Da un punto di vista concettuale la chiamata di una syscall è equivalente ad una chiamata di procedura
- La differenza sostanziale è che l'indirizzo della sycall è presente in una tabella del kernel e che durante la chiamata di una sycall il processore entra in Kernel Mode
- Tutti i processori contengono nel loro ISA un'istruzione per effettuare una syscall
  - x86 → int/iret.
  - MIPS → syscall.
  - In altri casi → trap.

System Call: cosa sono

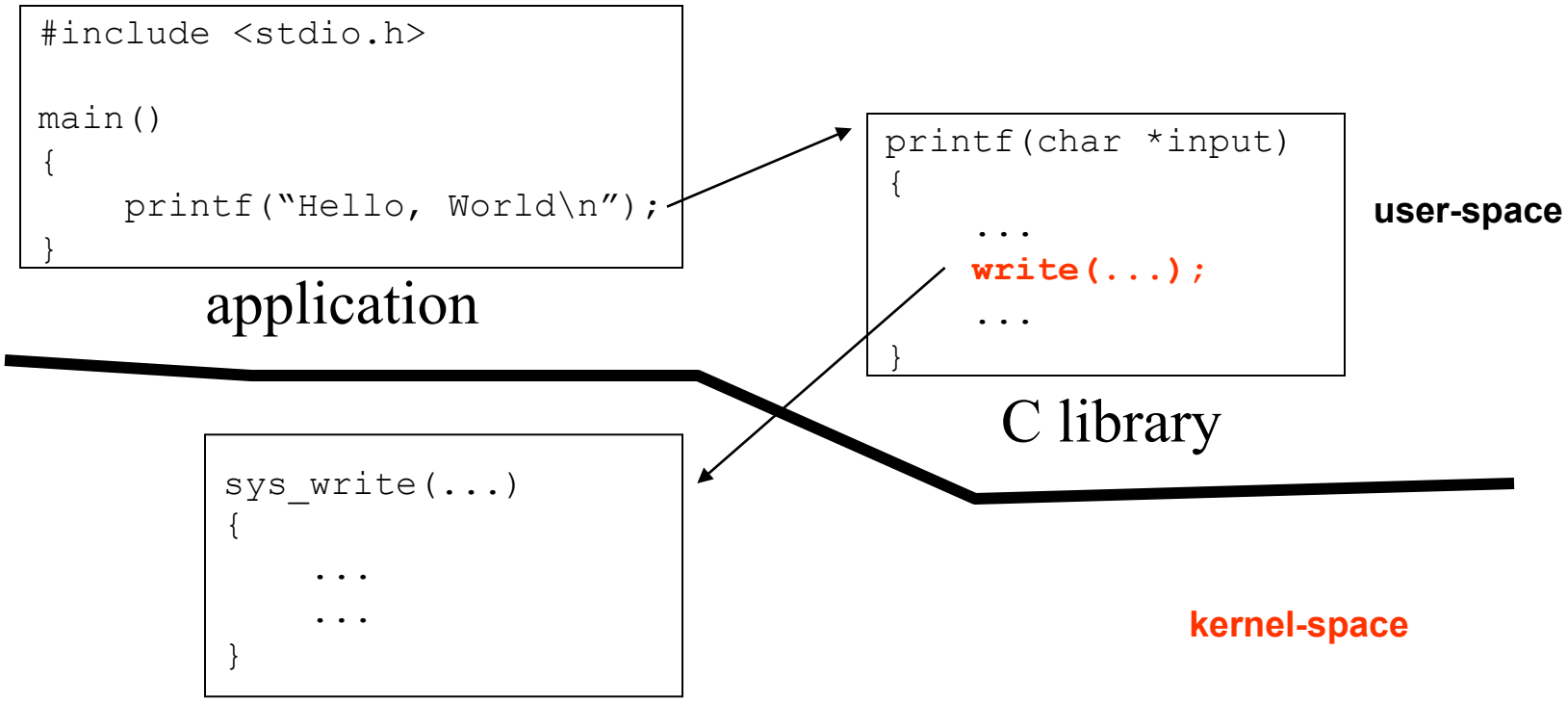
- L'interfaccia con cui i processi operanti in user space possono:
  - Accedere all'hardware e altre risorse gestite dal sistema operativo.
  - Comunicare con il kernel
- Le System call costituiscono uno strato tra hardware e processi user-space, che:
  - Fornisce un'interfaccia per semplificare l'uso delle risorse Hw da parte dei processi user-space
  - Assicura la sicurezza del sistema
    - È il kernel che decide chi e come può accedere alle diverse risorse del sistema
  - Consente la virtualizzazione delle risorse
    - Se le applicazioni fossero libere di accedere al sistema senza la conoscenza globale del kernel, sarebbe impossibile implementare diversi concetti come multitasking e memoria virtuale

Kernel space vs. User space



Applicazioni, librerie e kernel

- La chiamata di una system call da parte di un'applicazione avviene attraverso la chiamate di opportune funzioni di libreria, che a loro volta si rifanno all'interfaccia della system call per svolgere il loro compito



Librerie e system call

- Rispetto alle system call le funzioni di libreria possono svolgere una delle seguenti funzioni:
  - wrapper alla system call
    - Fornisce elaborazioni supplementari a quelle fornite dalla system call
      - `Printf()`: effettua le formattazione dei dati prima di chiamare la system call `write()` che scrive i dati su standard output
  - Relazione 1-1 con la system call
    - Chiama esclusivamente la system call di competenza
      - `open()`
  - Non chiamano nessuna system call
      - `strcpy()`

System Call

- Tutti i programmi applicativi fanno uso di tale interfaccia; anche le librerie di sistema si appoggiano sulle system call.
- In Linux (kernel 2.6) sono circa 280 (**/usr/include/asm/unistd.h**)
- Spesso ad una system call corrisponde (a più alto livello) una funzione di libreria standard C (e.g.: **open** → **fopen**)
- L'invocazione della system call segue la sintassi di chiamata di una normale funzione C
- Ogni system call ha un prototipo, definito negli include file di sistema (nella directory **/usr/include** e sue subdirectory): ad esempio **pid_t fork(void);**
- Alcune system call possono essere invocate con successo soltanto se il processo chiamante gira con i privilegi dell'utenza root
- **Nota**: utilizzare **man *nome_system_call*** per visualizzare il prototipo della system call e gli include file necessari

La libreria standard C

- La libreria standard del C (**libc**) contiene funzioni di utilità che forniscono servizi *general purpose* al programmatore
- Queste funzioni **NON** sono chiamate dirette a servizi del kernel, sebbene alcune di esse possono fare uso delle system call per realizzare i propri servizi
- Le funzioni di libreria standard C risiedono generalmente in una libreria dinamica
- Esempi:
  - la funzione **fopen** invoca la system call **open** per accedere ad un file
- Invece
  - la funzione **strcpy** (string copy) e la funzione **atoi** (convert ASCII to integer) non coinvolgono alcuna system cal

Application Programming Interface (API)

- Le applicazioni sono scritte facendo riferimento ad un interfaccia di alto livello (API) e non direttamente alle system call
- Un API definisce un insieme di interfacce di facile uso che possono essere usate per la stesura di programmi applicativi
  - Queste interfacce possono essere implementate ricorrendo a system call,
  - Usando più system call,
  - O non usando nessuna system call
- La stessa API può esistere su sistemi diversi, fornendo così un'interfaccia unica alle applicazioni e consentendo di fatto la portabilità del codice sorgente.

POSIX & LibC

- POSIX (Portable OS interface)
  - API più diffusa nel mondo UNIX
  - Comprende un serie di standard definiti da IEEE
  - "*Linux è POSIX compliant*"
    - Windows NT offre librerie POSIX-compatibili
- L'interfaccia più diffusa alle system call in Linux è fornita dalla libreria C
- La libreria C implementa
  - La principale API sui sistemi Unix
  - L'interfaccia per le chiamate di sistema
    - Standard c
- Libc fornisce anche la maggior parte di POSIX API

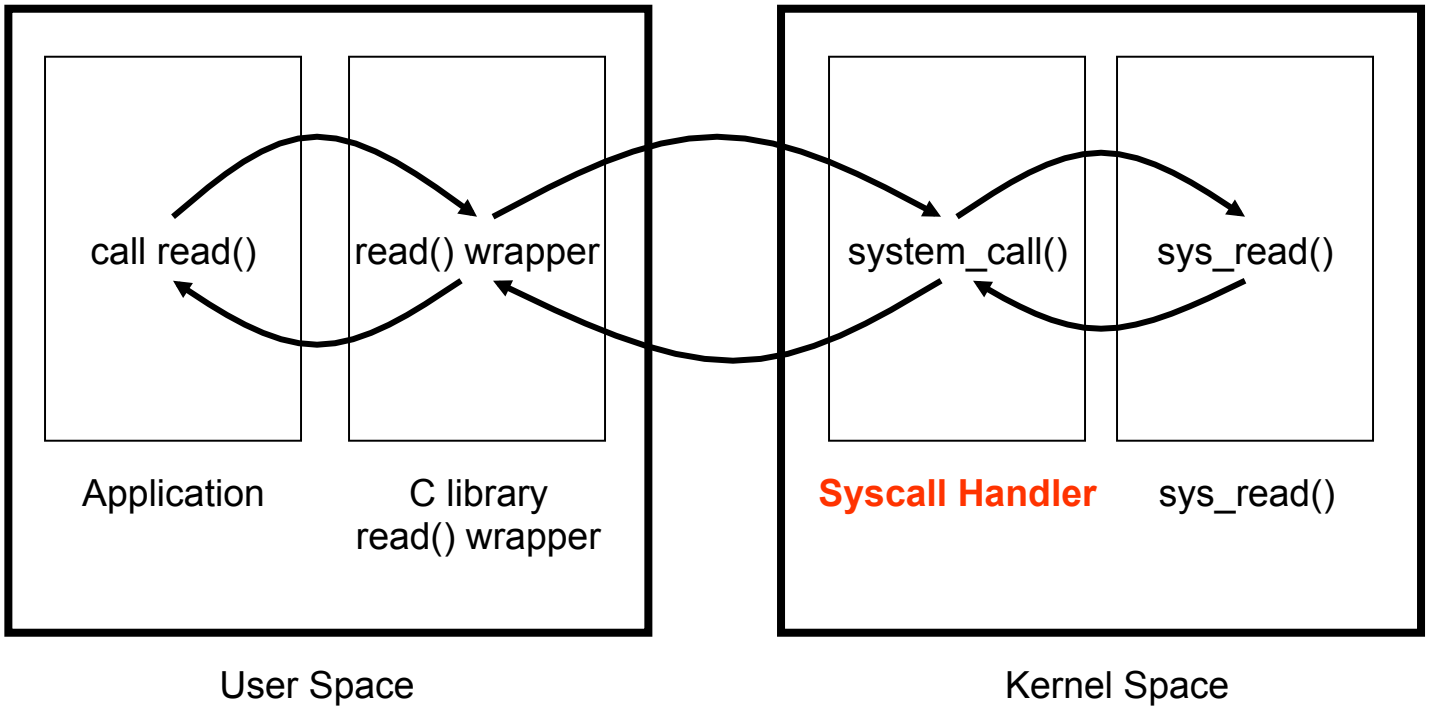
Definire una System Call

- Uno o più argomenti di input
- Un valore di ritorno che indichi la corretta esecuzione o meno
  - Di solito un valore negativo denota un errore
  - Un valore di ritorno pari a zero è di solito indicazione di successo
- Un comportamento ben definito

System Call

- Ad ogni system call è assegnato un numero che la contraddistingue, il kernel riconosce le syscall per numero e non per nome
- Una volta assegnato il numero ad una system call, non potrà più essere mutato, altrimenti le applicazioni precedentemente compilate non troveranno la corretta corrispondenza
- Se una syscall viene rimossa il suo identificativo non potrà essere riciclato
- Il kernel linux mantiene una lista di tutte le syscall registrate in una tabella memorizzata in `sys_call_table` in `entry.S`

System Call Handler



Parameter Passing

- Molte syscall richiedono opportuni parametri per poter essere eseguite
- I parametri sono scritti nei registri di CPU prima della chiamata di syscall
  - Nelle architetture IA-32 I registri usati sono `ebx`, `ecx`, `edx`, `esi`, e `edi`
  - Se sono richiesti più argomenti, gli stessi sono passati attraverso stack o puntatori
- Il valore di ritorno è restituito all'applicazione attraverso il registro eax

System Call via Libc

Process Management

pid = fork()	Create a child process identical to the parent
pid = waitpid(pid, &statloc, opts)	Wait for a child to terminate
s = wait(&status)	Old version of waitpid
s = execve(name, argv, envp)	Replace a process core image
exit(status)	Terminate process execution and return status
size = brk(addr)	Set the size of the data segment
pid = getpid()	Return the caller's process id
pid = getpgrp()	Return the id of the caller's process group
pid = setsid()	Create a new session and return its process group id
l = ptrace(req, pid, addr, data)	Used for debugging

Esempio

```
#define TRUE 1

while (TRUE) {                                  /* repeat forever */
      type_prompt( );                           /* display prompt on the screen */
      read_command(command, parameters);        /* read input from terminal */

      if (fork( ) != 0) {                       /* fork off child process */
           /* Parent code. */
           waitpid(–1, &status, 0);             /* wait for child to exit */
      } else {
           /* Child code. */
           execve(command, parameters, 0);      /* execute command */
      }
}
```

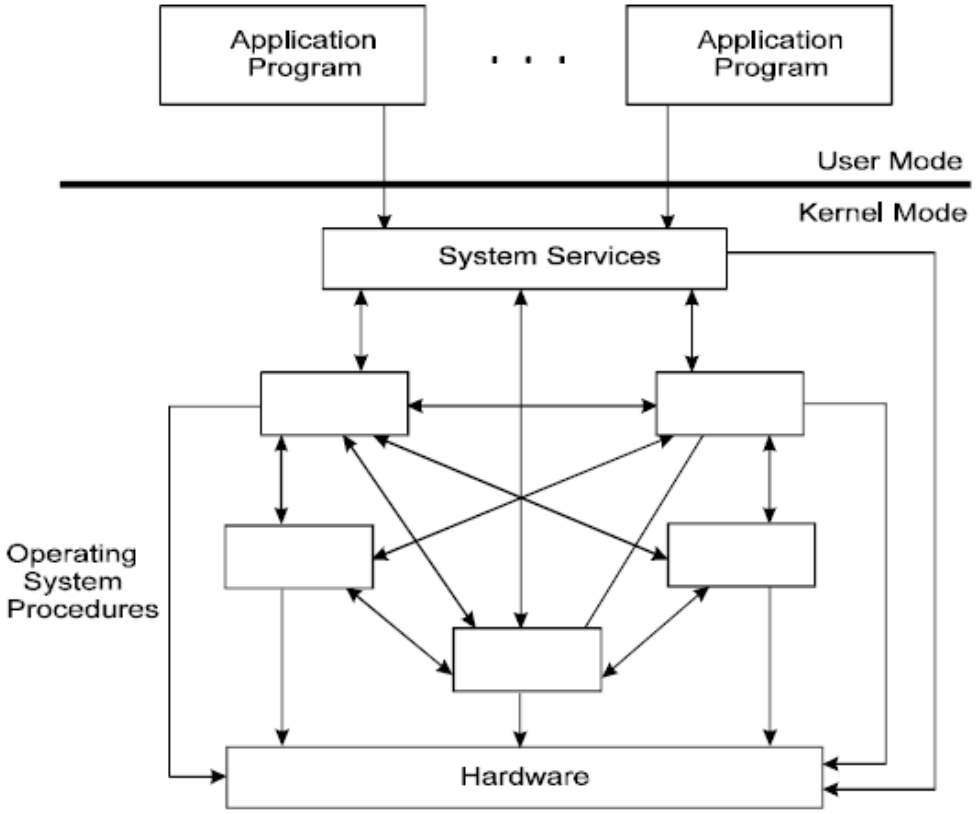
Struttura di un SO

- Nel corso degli anni sono emersi diversi approcci sulla struttura da dare al programma sistema operativo
- In particolare sono emerse 4 tipologie diverse di struttura:
  - Monolitici
  - Layered (stratificati)
  - Microkernel
  - VMM

Monolitici

- I sistemi monolitici sono costituiti da un insieme di moduli, sena alcuna relazione gerarchica
- Ogni modulo può chiamare ogni altro modulo
- Tutti i moduli possono accedere indiscriminatamente a tutti i dati presenti nell'area kernel

Monolitico



Stratificati

- In questo caso esiste una relazione gerarchica tra i moduli che compongono il SO, ad ogni modulo è assegnato un numero che indica il livello di appartenenza
- Un modulo di livello N potrà accedere solo a funzionalità e dati presenti al livello sottostante
- In questo caso il sistema risulta di più facile manutenzione anche se meno efficiente del sistema monolitico

SO stratificato

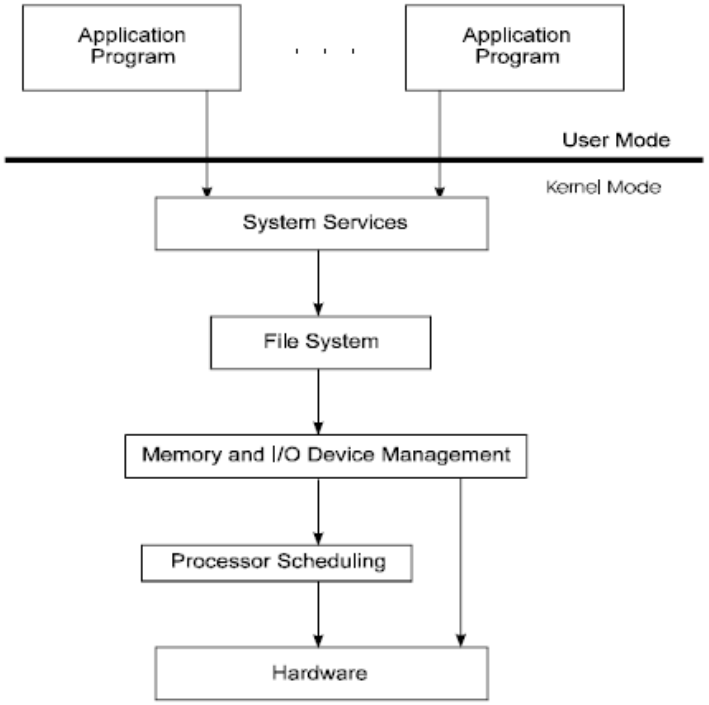
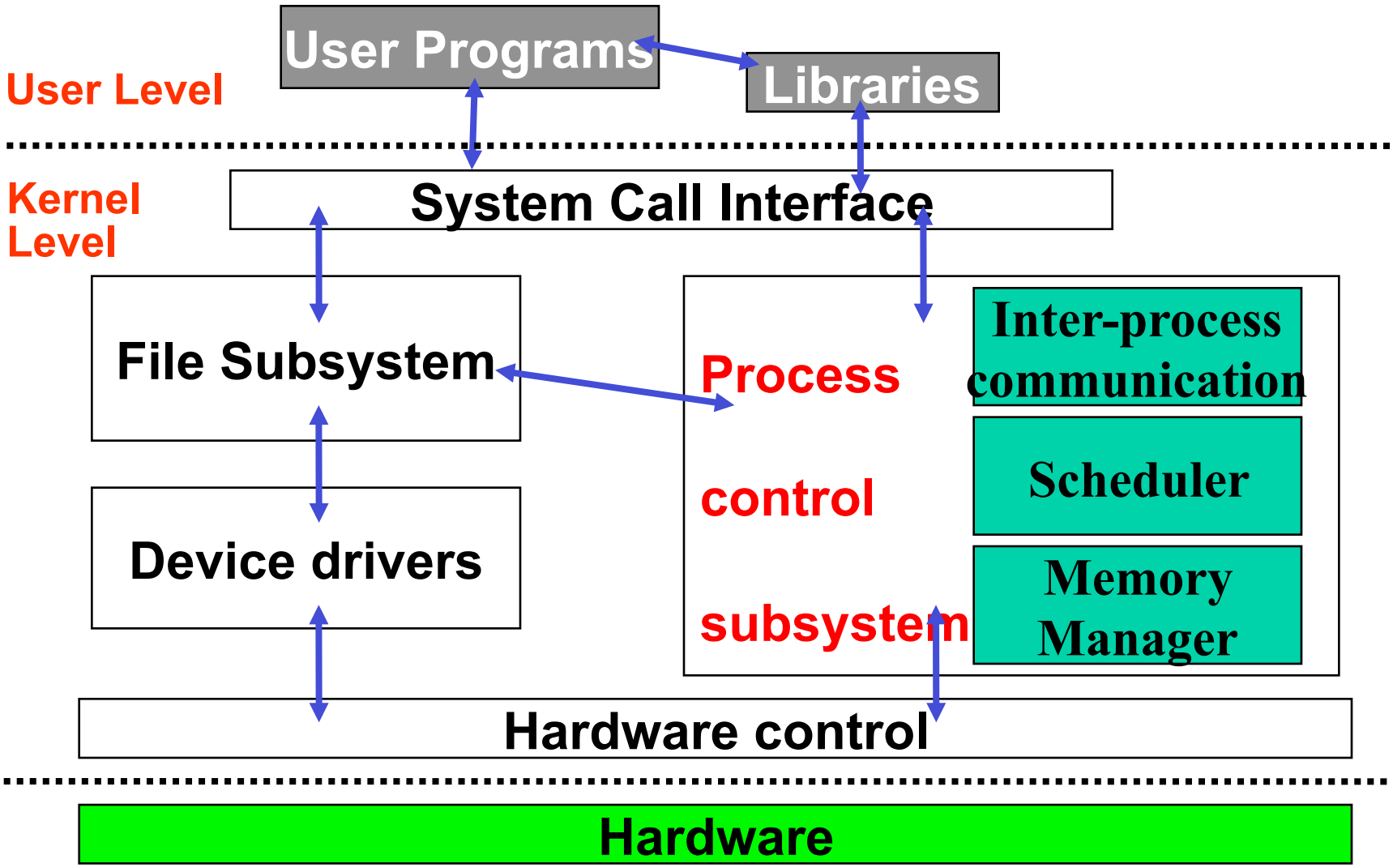


Figure 2.2: Layered Operating System

Architettura Unix



Architettura di un SO: W2000

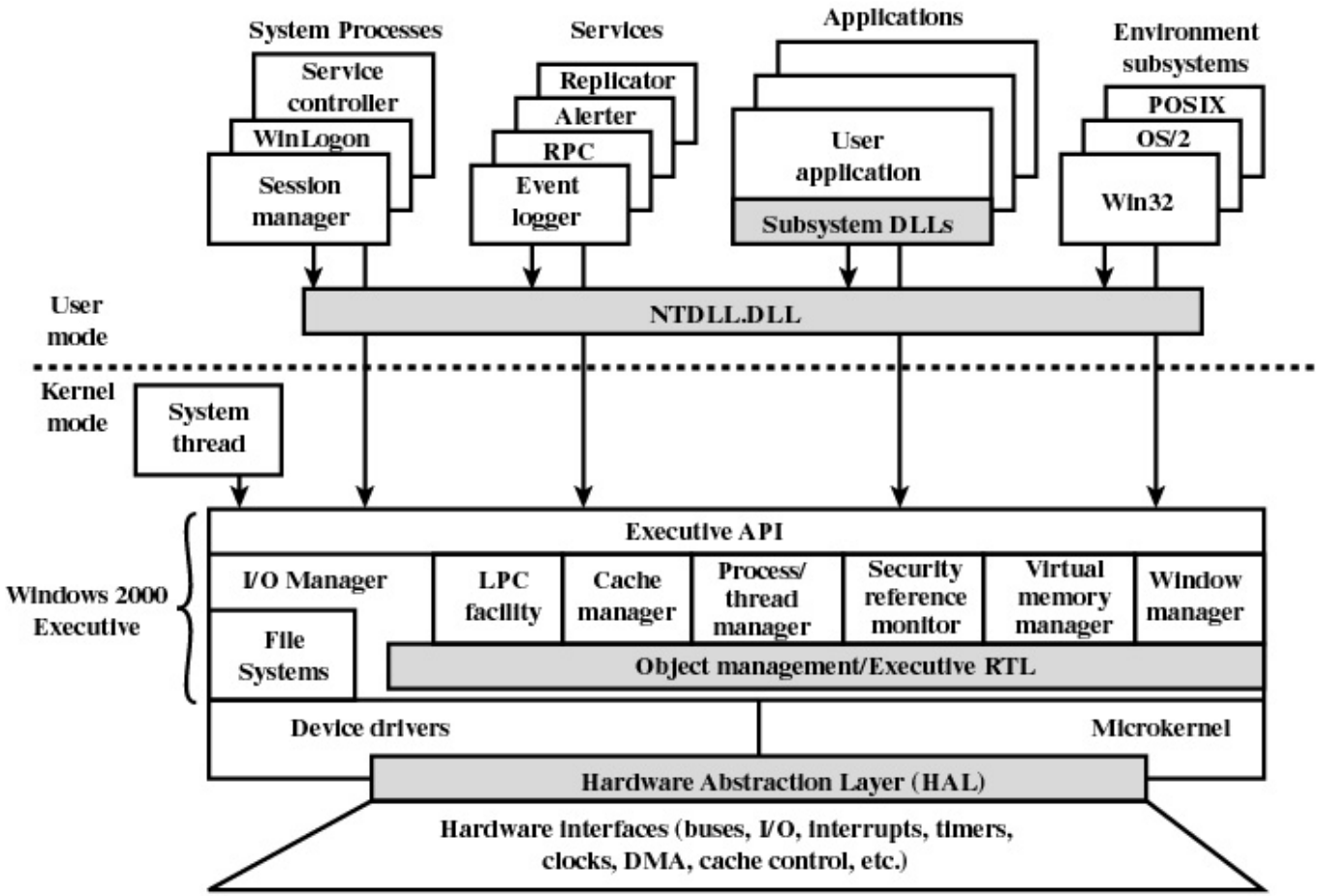
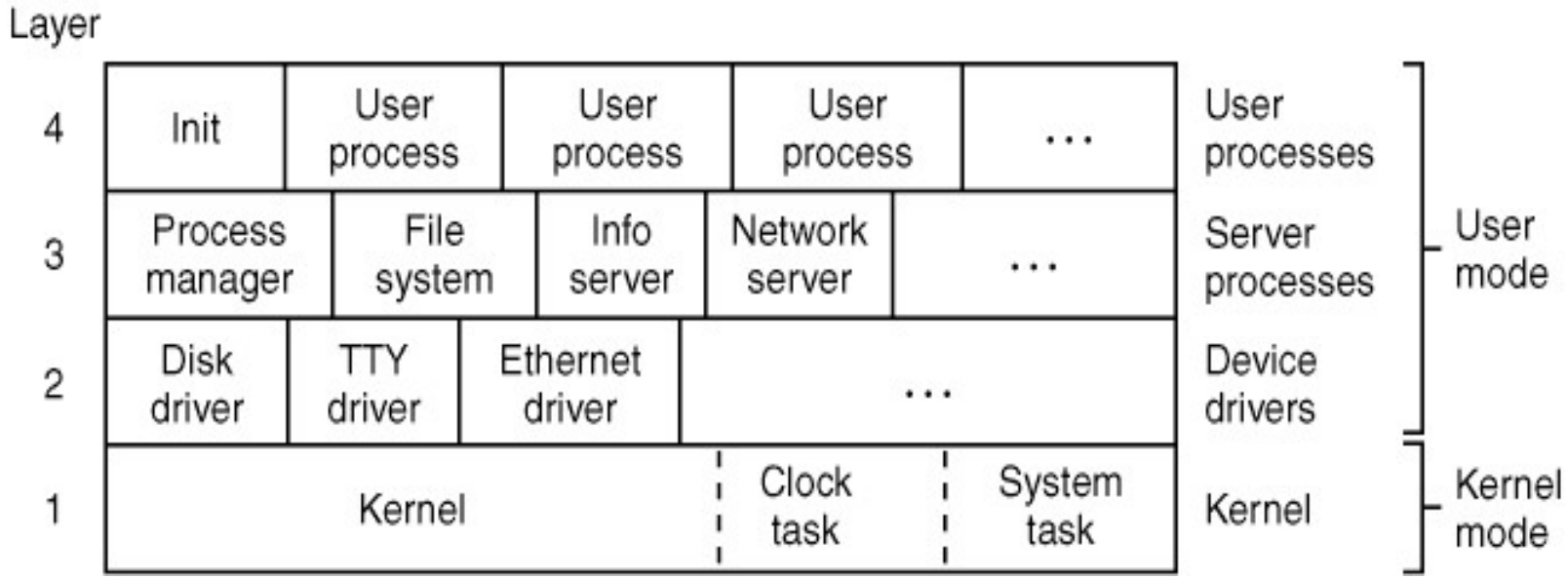


Figure 2.13 Windows 2000 Architecture

Architettura di un SO: minix



Microkernel

- Un microkernel è un kernel di sistema operativo che contiene solo ed esclusivamente i servizi indispensabili per il funzionamento dell'intero sistema
  - Gestore dei processi
  - Meccanismo di comunicazione tra processi
  - Gestore dei messaggi
- I restanti servizi (Gestione memoria, file system, Gestore finestre, ecc.) sono considerati alla stregua di processi utente

Vantaggi e Svantaggi

- Kernel molto contenuto → manutenzione più facile, meno errori
- Codice eseguito in kernel mode molto contenuto
- Sistema operativo nel complesso facilmente più estendibile e personalizzabile
- Svantaggi
  - Prestazioni

Microkernel

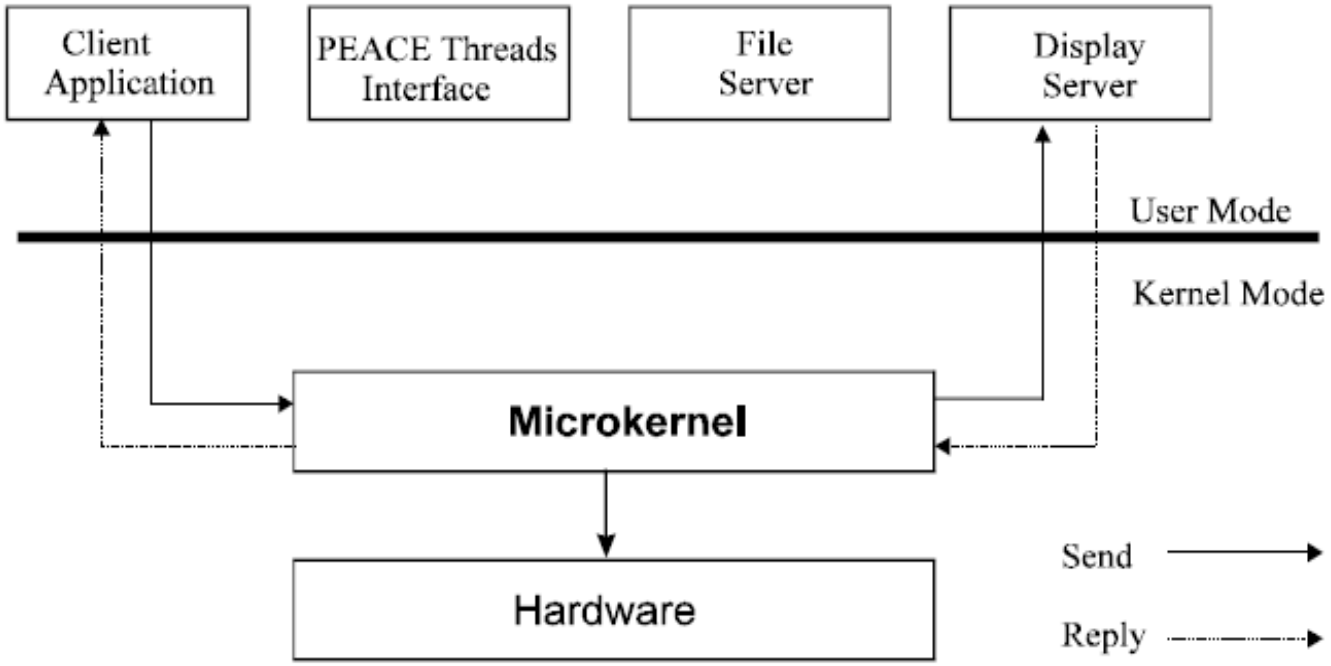
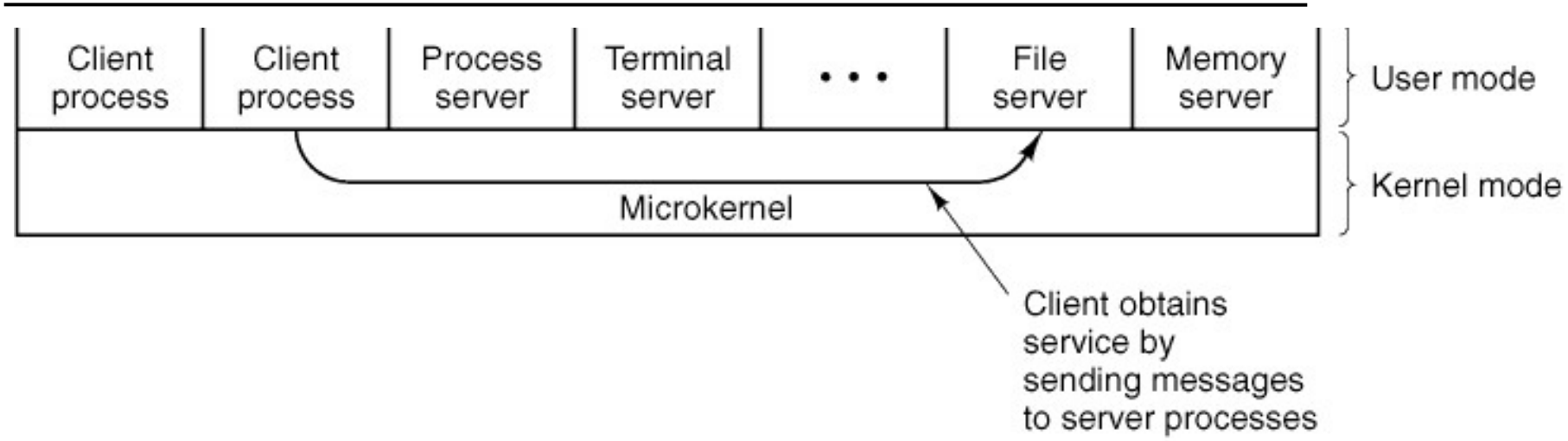


Figure 2.4: Microkernel Operating System

Client -server

- Nei sistemi a microkernel le applicazioni che implementano i diversi servizi possono essere pensate come dei server che forniscono funzionalità agli altri server o alle applicazioni utente che in questo caso operano come client
- Per questo motivo i sistemi microkernel sono chiamati anche client-server OS

Client-Server Model (1)



Virtualizzazione

- Il processo attraverso cui le risorse e l'interfaccia per l'accesso ad un sistema (CPU, memoria, disco) sono mappate, da un opportuno agente, sulle risorse e l'interfaccia del sistema reale che non viene "visto" dall'utilizzatore finale

Virtualizzazione

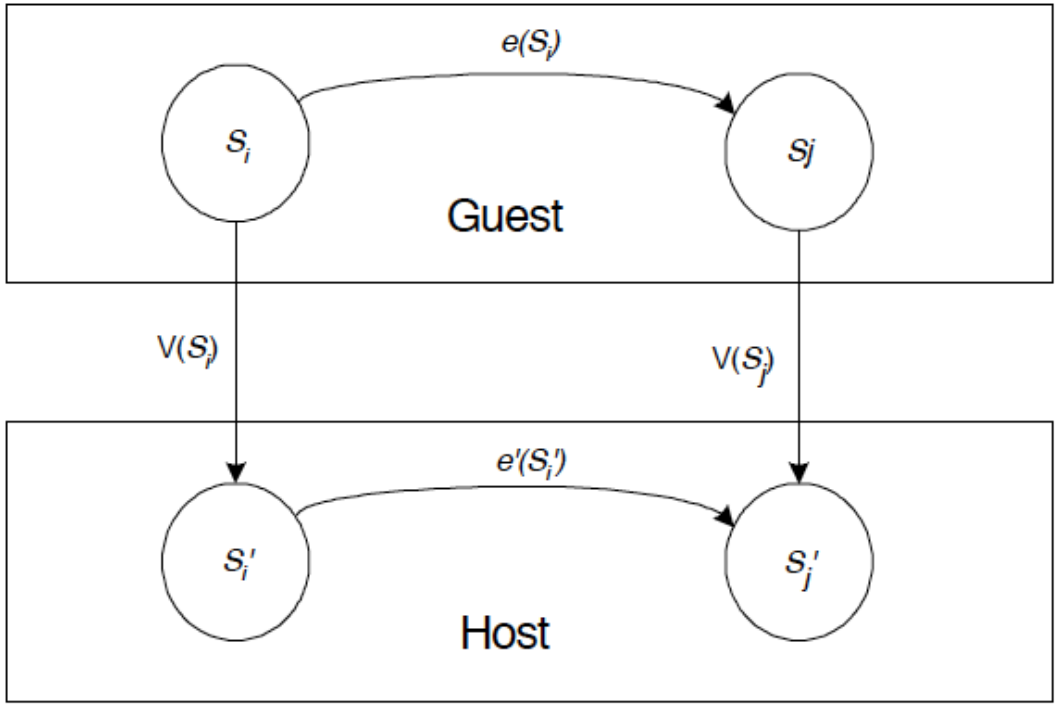


Figure 2. Formally, virtualization is the construction of an isomorphism between a guest system and a host; $e' \circ V(S_i) = V \circ e\ (S_i)$.

Macchine virtuali (VMM)

- Un VMM è uno strato di software che si pone immediatamente al di sopra dell'hardware di un sistema il cui compito è quello di simulare una o più copie dell'unica macchina fisica
- In questo modo è possibile disporre di sistemi architetturalmente diversi pur possedendone fisicamente uno solo
- Le "applicazioni" del VMM sono dette macchine virtuali

Implicazioni e vantaggi

- È possibile eseguire contemporaneamente più copie di un OS sulla stessa macchina fisica
- È possibile eseguire contemporaneamente OS diversi sulla stessa macchina fisica
- Vantaggi
  - Miglior sfruttamento dell'HW
  - Sviluppo e uso del sw (portabilità, compatibilità)
  - Sicurezza (Isolamento, fault tolerance)

Approcci di riferimento

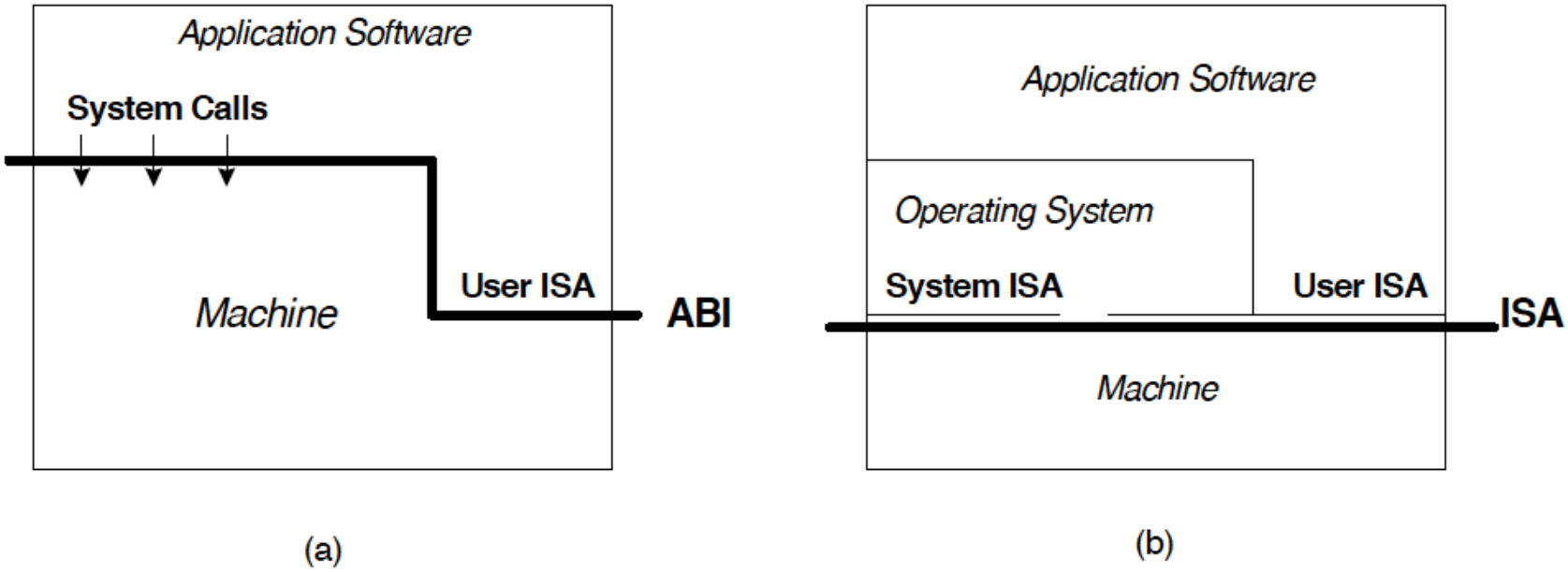
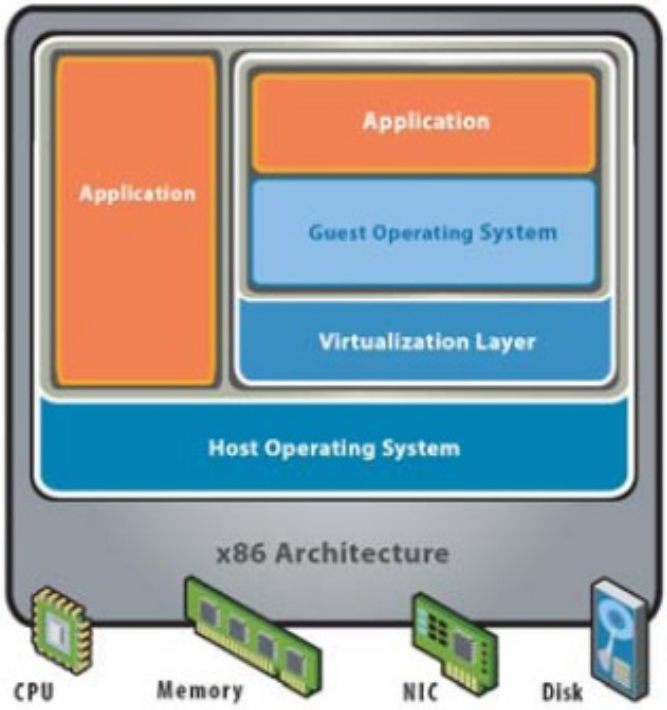


Figure 5. Machine Interfaces
Application Binary Interface (ABI)
b) Instruction Set Architecture (ISA) interface

a)

Hosted Architecture



Hosted Architecture

Hypervisor

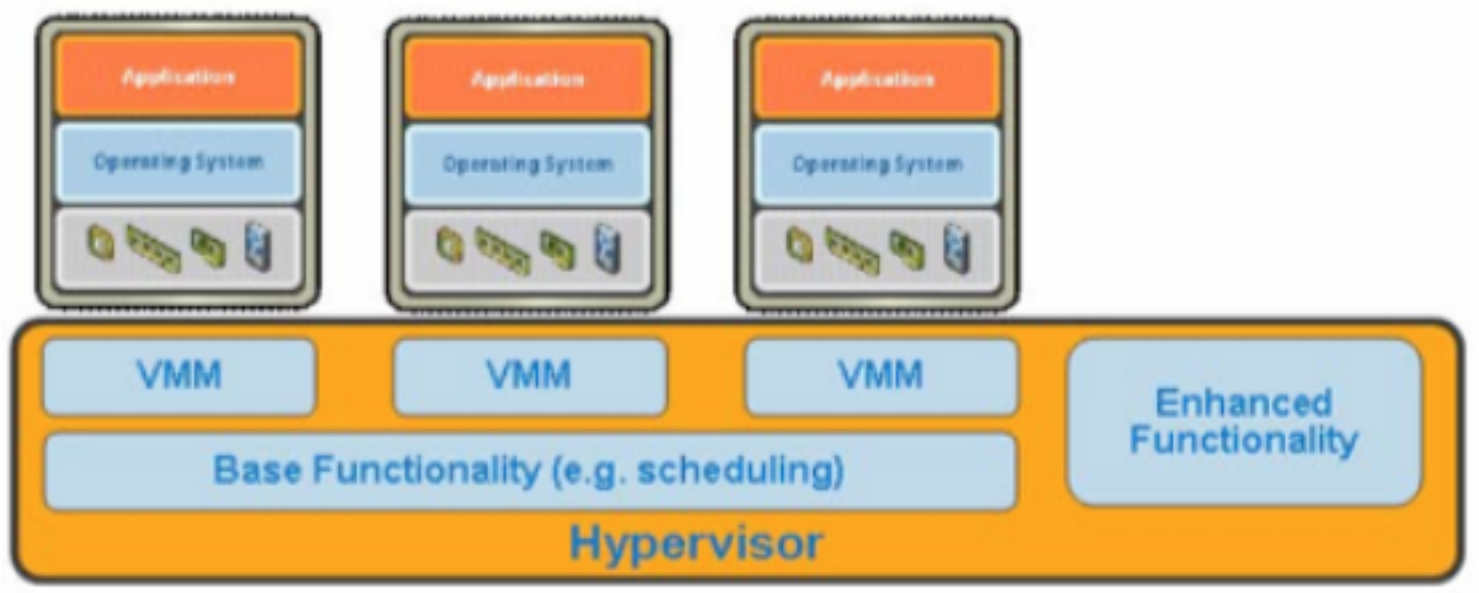


Figure 3 – The hypervisor manages virtual machine monitors that host virtual machines

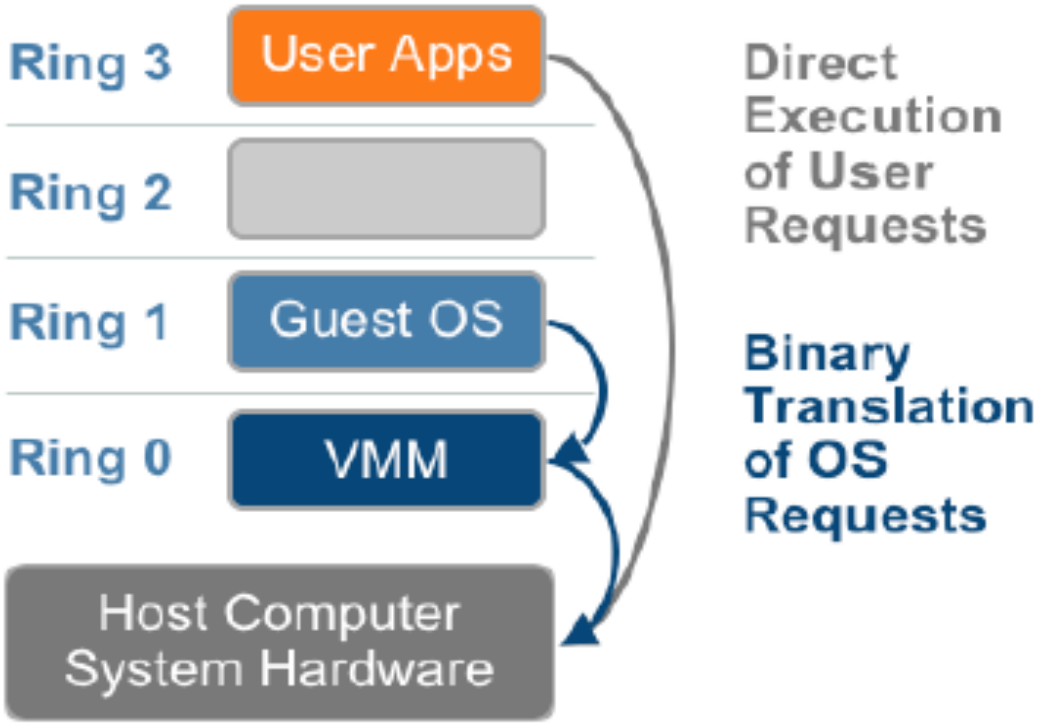


Figure 5 – The binary translation approach to x86 virtualization

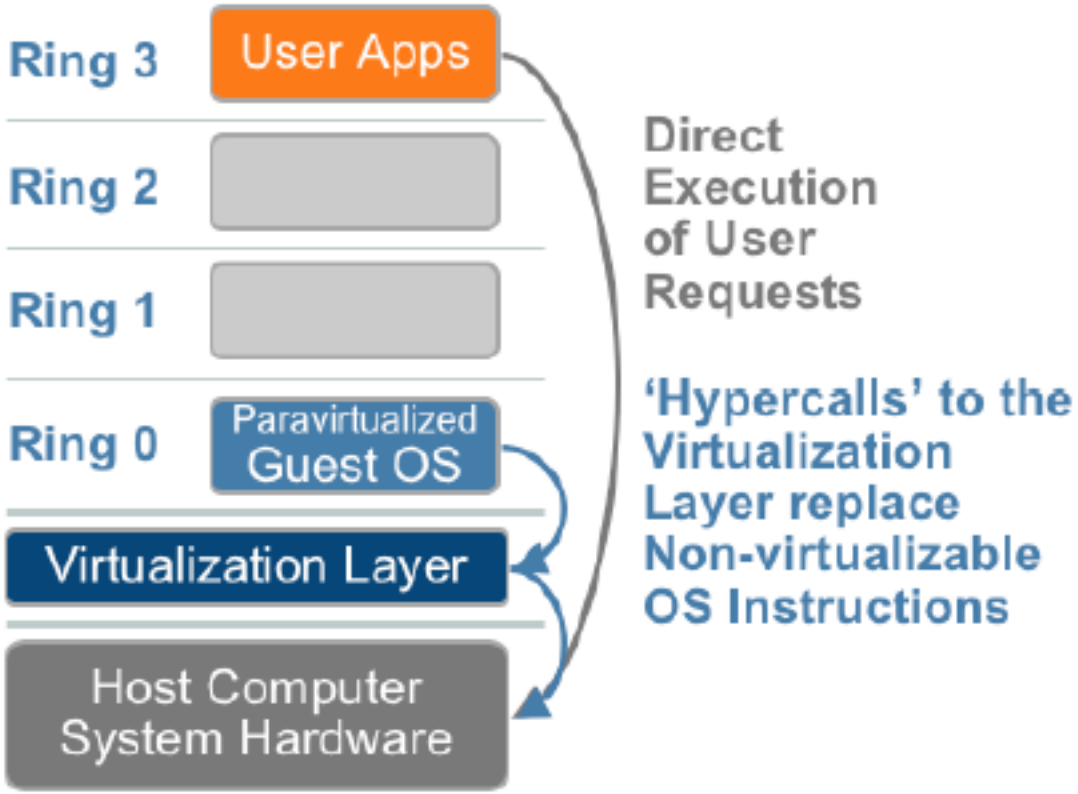
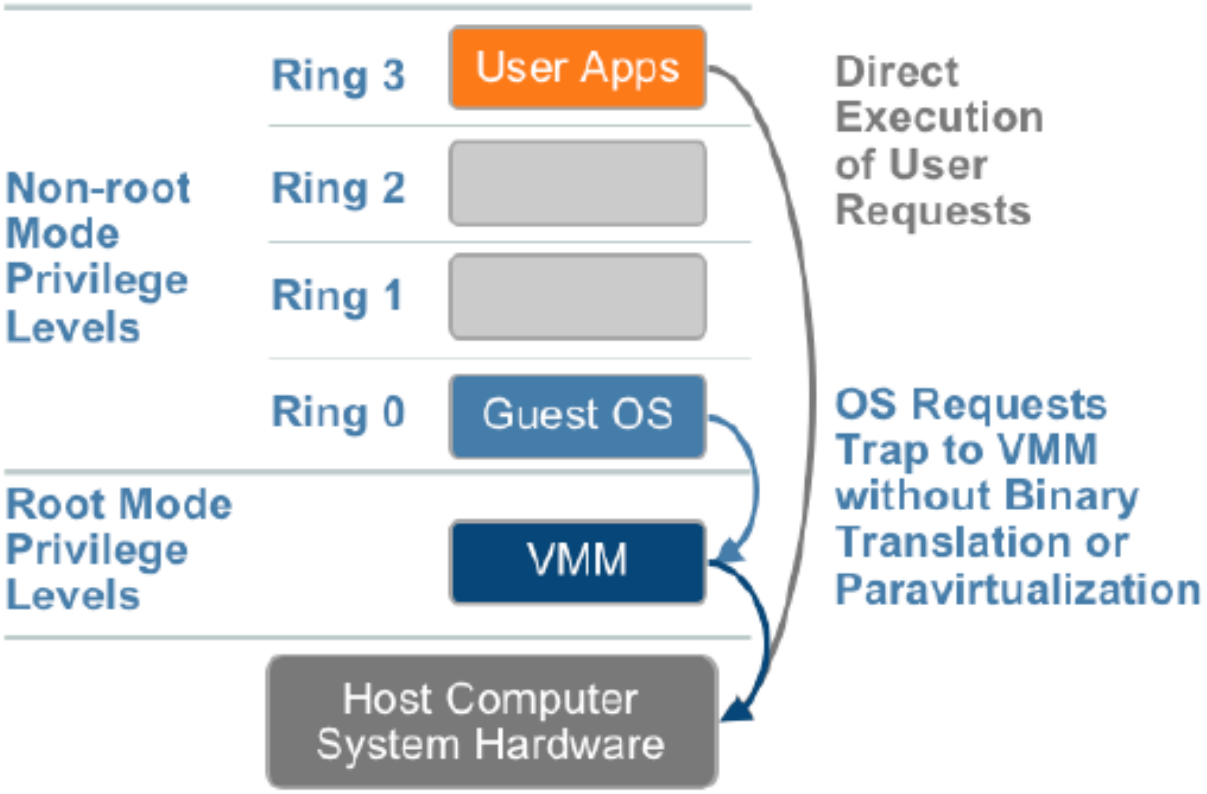


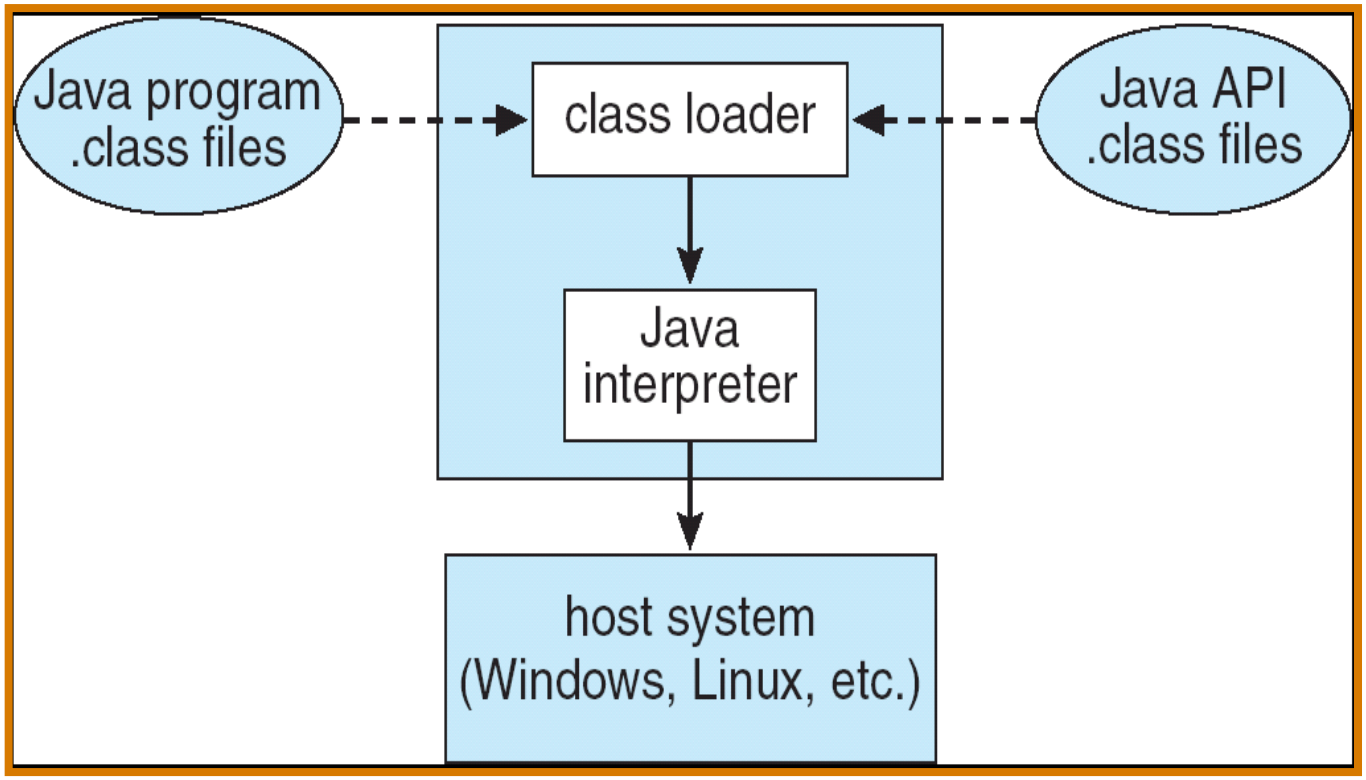
Figure 6 – The Paravirtualization approach to x86 Virtualization

Virtualization Technology (VT)



- **Figure 7 – The hardware assist approach to x86 virtualization**

JVM



Historical Computers

Alcune delle fotografie presenti in questa presentazione sono del

Computer History Museum

http://www.computerhistory.org