THE O.S. INTERFACE

È la parte esterna del SO, attraverso la quale gli utenti richiedono l'esecuzione dei programmi (siano essi programmi "di utilità" dello stesso SO o programmi "applicativi") o accedono ai servizi del kernel.

L'interfaccia può essere di due tipi

.... a caratteri

L'interfaccia a caratteri o **shell** o **command interpreter** è un vero e proprio linguaggio di programmazione che permette all'utente di controllare la esecuzione di comandi, sia in modo interattivo che in modo "batch" (mediante "script" di shell). Tipici SO con interfaccia a caratteri sono il DOS, UNIX, LINUX, ecc.

.... grafica o iconica

L'interfaccia grafica è un sistema d'interazione uomo-macchina basato su simboli (icone) che rappresentano le risorse su cui è possibile operare. Tipici SO con interfaccia iconica sono WINDOWS (in tutte le sue versioni), MAC OS, UNIX vers. OSF Motif, LINUX vers. OSF Motif, ecc.

THE CHARACTER INTERFACE (*)

Una shell si basa sull'esecuzione di tre tipi di **comandi**:

1. **oggetti eseguibili (o comandi esterni)**
   Sono file contenenti programmi in formato eseguibile
   Esempio: sort
2. **comandi built-in (o comandi interni)**
   Sono comandi che realizzano funzioni semplici, eseguite direttamente dalla shell
   Esempio: cd
3. **script (o comandi batch)**
   sono "programmi" in linguaggio di shell

Quando si digita un comando, la shell verifica se si tratta di un comando built-in. In caso positivo, lo esegue. Altrimenti la shell crea (fork) un nuovo processo che esegua il comando.

Oltre all'esecuzione di comandi, la shell consente altre funzioni tipiche:
- **ridirezione** dell'input e dell'output
- **"pipelines"** di comandi
- **filtri**

(*) si fa riferimento ad una tipica interfaccia a caratteri di un SO UNIX-like

THE GRAPHICAL INTERFACE (*)

Un'interfaccia grafica si basa sulla manipolazione di "**risorse**"

Una risorsa è un **oggetto** che fornisce o elabora informazioni

In Windows le risorse sono rappresentate da icone ed hanno un nome

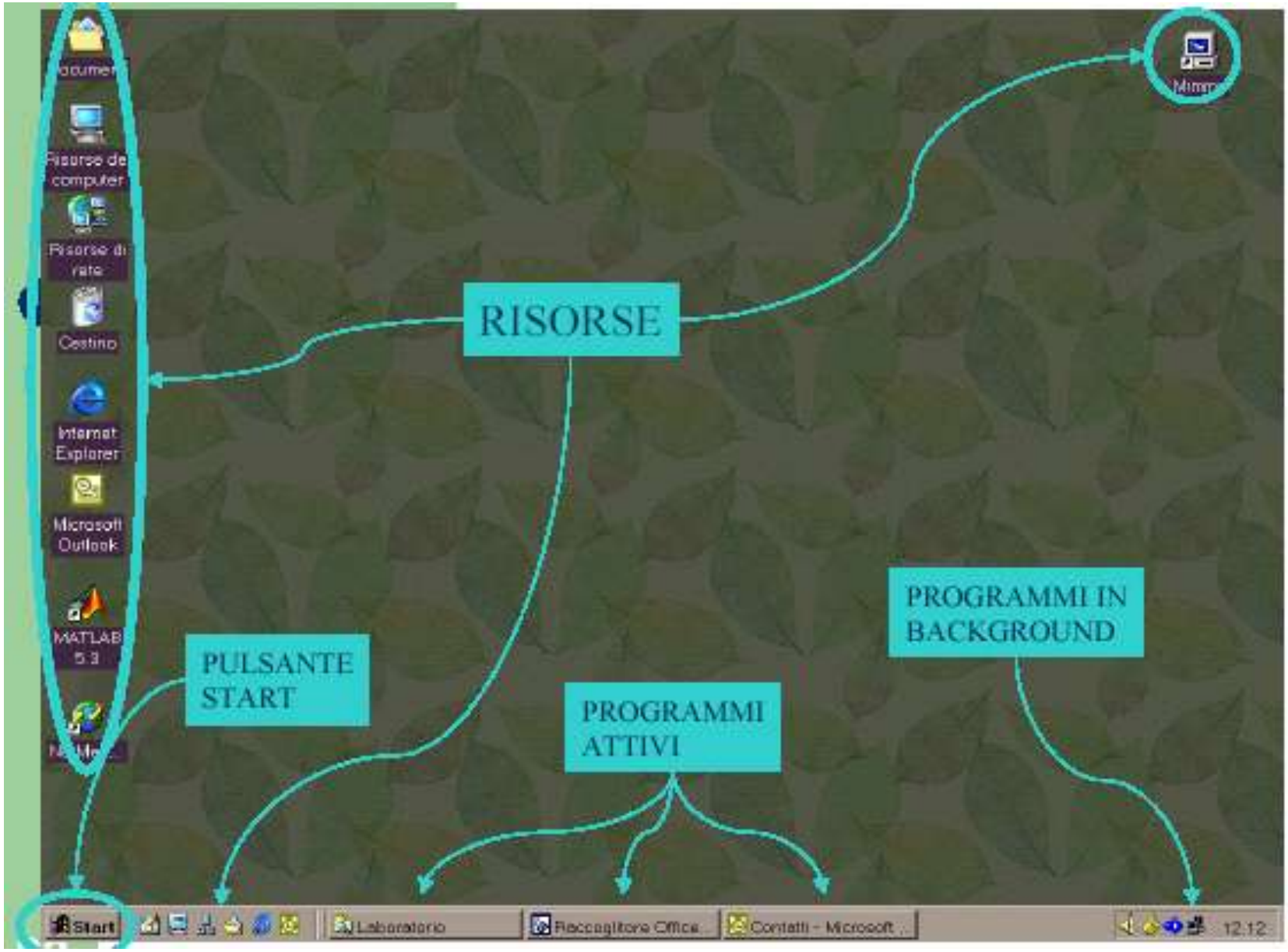
Esempi di risorse
- I dischi e le stampanti
- Tutti i dispositivi esterni
- I file e le cartelle
- I programmi
- I componenti di Windows

Il desktop

È una risorsa speciale di Windows, realizzata come cartella in cui sono contenuti documenti (oggetti) di interesse generale ed organizzata come "scrivania" virtuale, su cui sono disposte le icone degli oggetti.

(*) si fa riferimento ad una tipica interfaccia grafica di un SO WINDOWS-like

THE WINDOWS DESKTOP



THE ACTIVITIES MANAGED BY AN O.S.

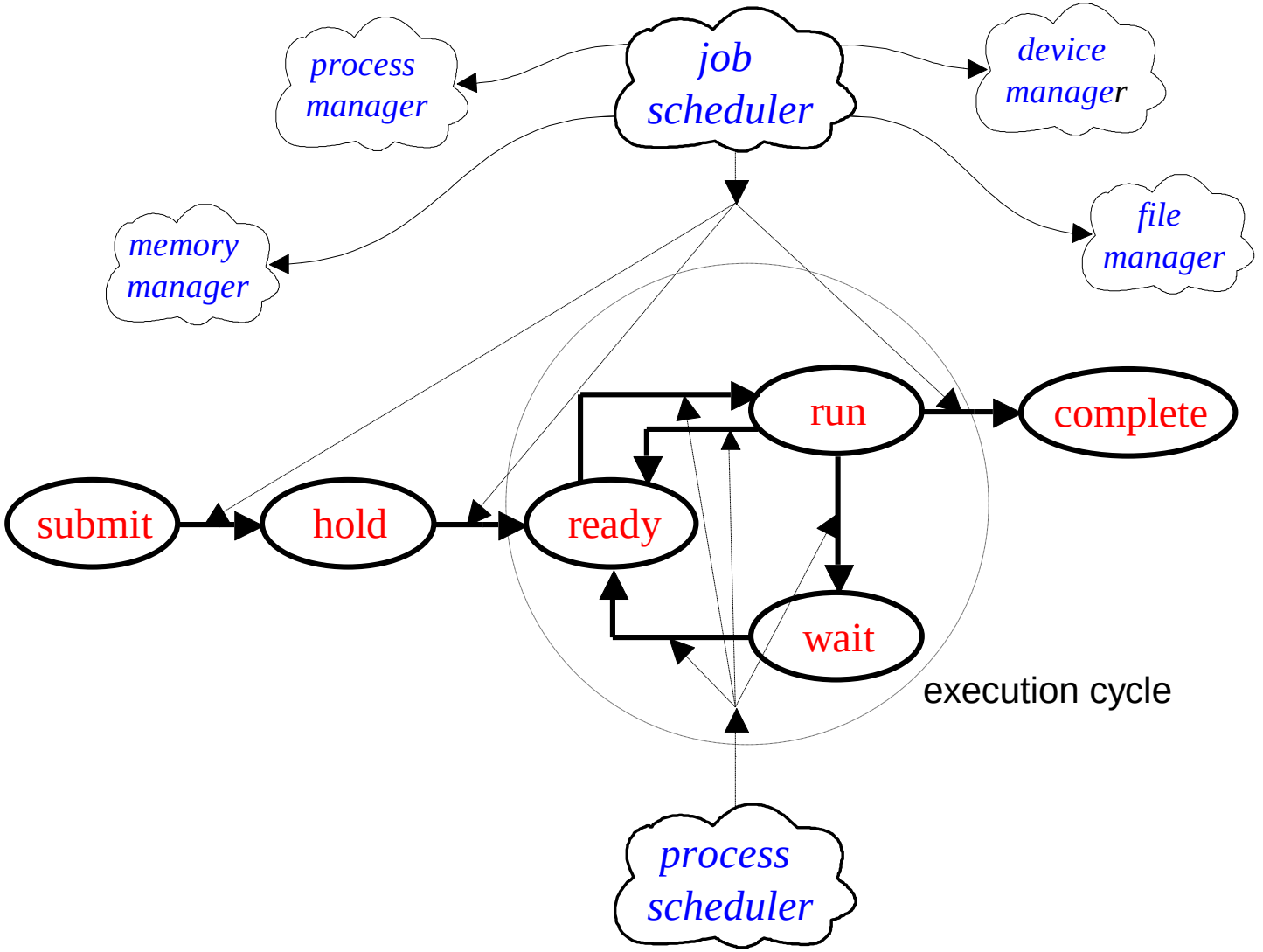
Un ***job*** è costituito da uno o più job-step (programmi) da eseguire in sequenza.

Un ***job-step*** (o programma) è costituito da uno o più task (processi cooperanti) che operano «contemporaneamente» contendendo fra loro, oltre che con gli altri processi costituenti gli altri programmi (processi concorrenti), nell'utilizzo delle risorse (CPU, memoria centrale, memoria di massa, dispositivi, file) presenti nel sistema di elaborazione.

Il sistema operativo stesso è costituito da processi che cooperano per rendere possibile l'esecuzione dei programmi.

Un ***task*** è una attività elementare ed indipendente, con associate le risorse (memoria, file, dispositivi) richieste per la sua esecuzione. Ad ogni task è associato un Task Control Block (TCB) o Process Control Block (PCB).

THE EXECUTION STEPS OF A PROGRAM



PROGRAM EXECUTION PROCEEDING

Il sistema operativo produce l'avanzamento dell'esecuzione dei programmi.

Dapprima le richieste sottoposte vengono raggruppate, costituendo la coda relativa allo **stato di submit**.

Dopo l'analisi delle risorse necessarie a soddisfare le richieste, queste vengono ordinate in relazione ai criteri di priorità (macroscheduling) del ***Job Scheduler***, costituendo la coda relativa allo **stato di hold**.

A partire dalla prima richiesta di esecuzione presente nella coda di hold, il Job Scheduler, chiamando i gestori delle varie risorse, verifica la disponibilità delle risorse necessarie e, dopo averne avuto conferma, attiva il ***Process Scheduler***. Questo trasforma la richiesta di esecuzione in processo e costruisce un blocco di informazioni (*Process* o *Task Control Block*), che, progressivamente aggiornate, gli permetteranno di gestire il programma nel *ciclo di esecuzione*. Inizialmente il task viene immesso nella coda relativa allo **stato di ready**. La coda in questione è organizzata in relazione ai criteri di priorità (microscheduling) con cui il Process Scheduler ordina i vari task che abbisognano della sola CPU per essere eseguiti.

Non appena la CPU si libera (o il rispetto delle priorità lo richiede), viene operato il cambio del contesto operativo (*context switching*) della CPU e un processo passa nello **stato di run** (ed uno passa in ready).

L'attesa di specifici eventi può costringere il S.O. a far transitare il processo nella coda corrispondente allo **stato di wait**.

Dopo aver ciclato tra gli stati del ciclo di esecuzione, il processo arriva a conclusione e transita nella coda dello **stato di complete**.

O.S. DESIGN AND IMPLEMENTATION

- ✓ Traditionally written in assembly language, operating systems can now be written in higher-level languages.
- ✓ Code written in a high-level language:
  - Φ can be written faster.
  - Φ is more compact.
  - Φ is easier to understand and debug.
- ✓ An operating system is far easier to *port* (move to some other hardware) if it is written in a high-level language.

O.S. INSTALLATION AND USE (SYSTEM GENERATION)

- ✓ Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site.
- ✓ SYSGEN program obtains information concerning the specific configuration of the hardware system.
- ✓ *Booting* – starting a computer by loading the kernel.
- ✓ *Bootstrap program* – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution.

O.S. EVOLUTION

MAINFRAME SYSTEMS

…. Batch Systems

- Reduce setup time by batching similar jobs
- Automatic job sequencing – automatically transfers control from one job to another. First rudimentary operating system.
- Resident monitor
  - initial control in monitor
  - control transfers to job
  - when job completes control transfers back to monitor

O.S. EVOLUTION

MAINFRAME SYSTEMS

…. Multiprogrammed Batch Systems

☞ Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them.

O.S. EVOLUTION

MAINFRAME SYSTEMS

…. time-sharing (interactive)

- Multiple users simultaneously access the system through terminals
- The CPU is multiplexed among several jobs that are kept in memory and on disk (the CPU is allocated to a job only if the job is in memory).
- A job swapped in and out of memory to the disk.
- Interaction between the user and the system is provided; when the operating system finishes the execution of one command, it seeks the next "control statement" from the user's keyboard.
- On-line system must be available for users to access data and code.

O.S. EVOLUTION

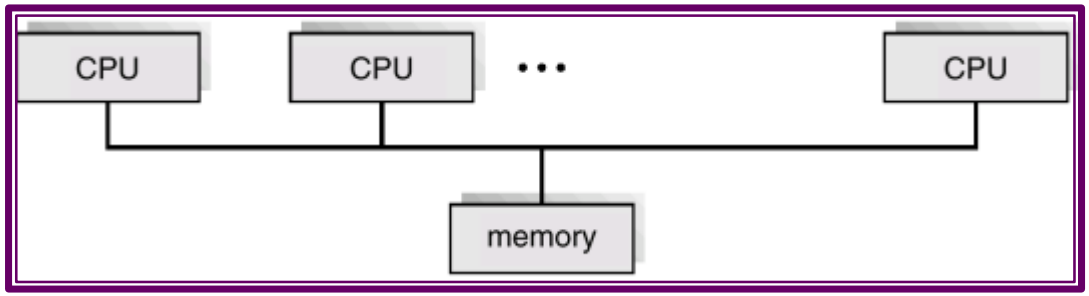
DESKTOP SYSTEMS

- ***Personal computers*** – computer system dedicated to a single user.
- I/O devices – keyboards, mice, display screens, small printers.
- User convenience and responsiveness.
- Can adopt technology developed for larger operating systems. Often individuals have sole use of computer and do not need advanced CPU utilization of protection features.
- May run several different types of operating systems (Windows, MacOS, UNIX, Linux)

O.S. EVOLUTION

PARALLEL SYSTEMS

- Multiprocessor systems with more than on CPU in close communication.
- *Tightly coupled system* – processors share memory and a clock; communication usually takes place through the shared memory.

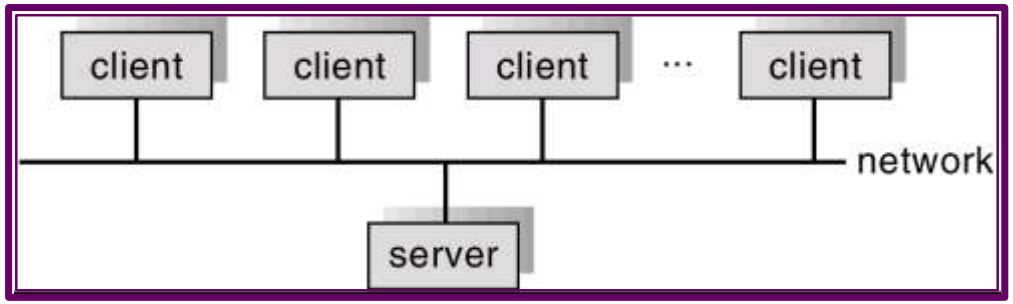


- Advantages of parallel system:
  - Φ Increased *throughput*
  - Φ Economical
  - Φ Increased reliability
    - ✓ graceful degradation
    - ✓ fail-soft systems

O.S. EVOLUTION

DISTRIBUTED SYSTEMS

- Distribute the computation among several physical processors.
- *Loosely coupled system* – each processor has its own local memory; processors communicate with one another through various communications lines, such as high-speed buses or telephone lines.
- Advantages of distributed systems.
  - Φ Resources Sharing
  - Φ Computation speed up – load sharing
  - Φ Reliability
  - Φ Communications
- Requires networking infrastructure.
- Local area networks (LAN) or Wide area networks (WAN)
- May be either client-server or peer-to-peer systems



O.S. EVOLUTION

REAL-TIME SYSTEMS

- Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.
- Well-defined fixed-time constraints.
- Real-Time systems may be either *hard* or *soft* real-time.
- Hard real-time:
  - Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM)
  - Conflicts with time-sharing systems, not supported by general-purpose operating systems.
- Soft real-time
  - Limited utility in industrial control of robotics
  - Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.

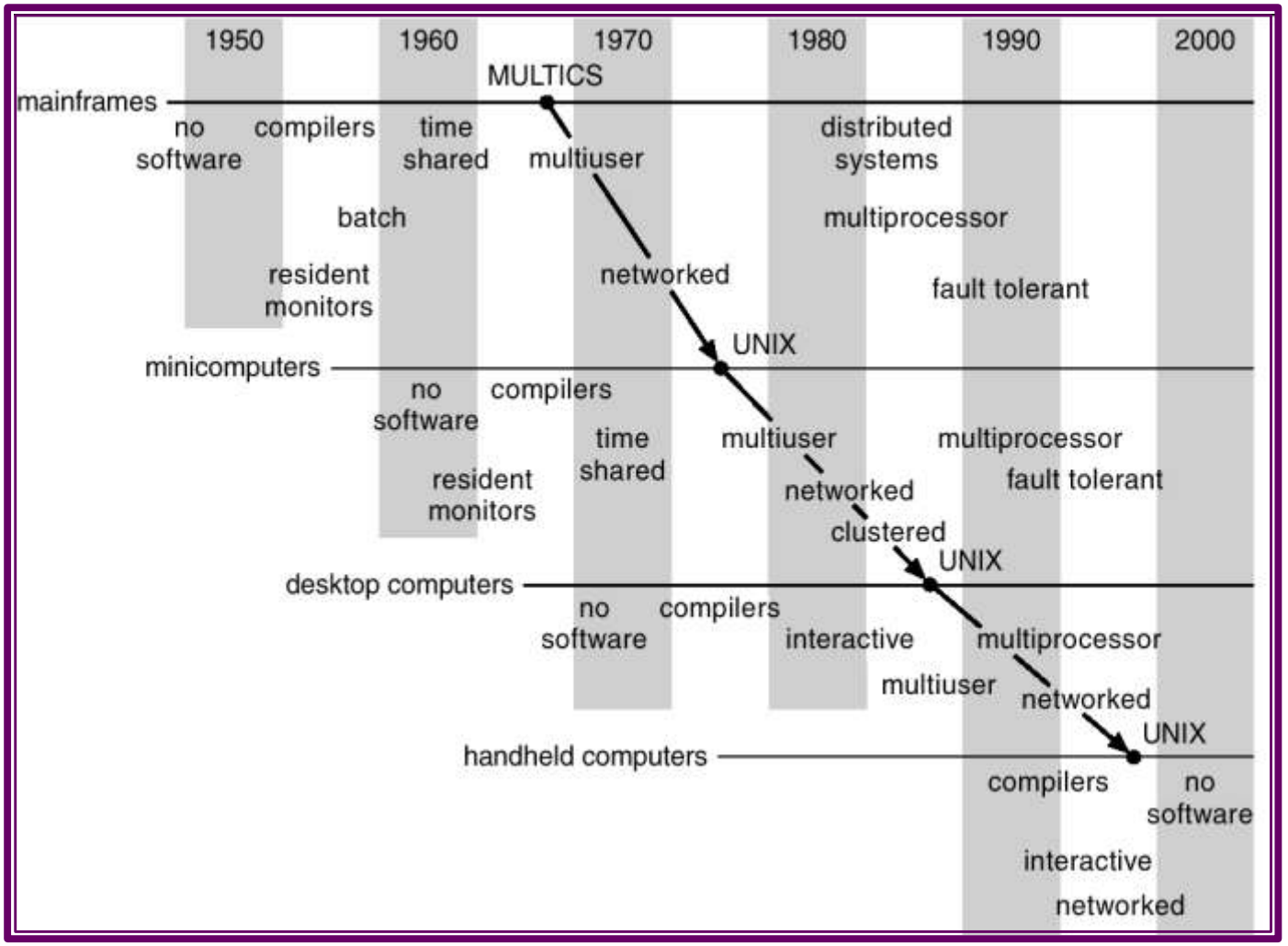
O.S. EVOLUTION

HAND-HELD SYSTEMS

- Personal Digital Assistants (PDAs)
- Cellular telephones
- Issues:
  - Limited memory
  - Slow processors
  - Small display screens.

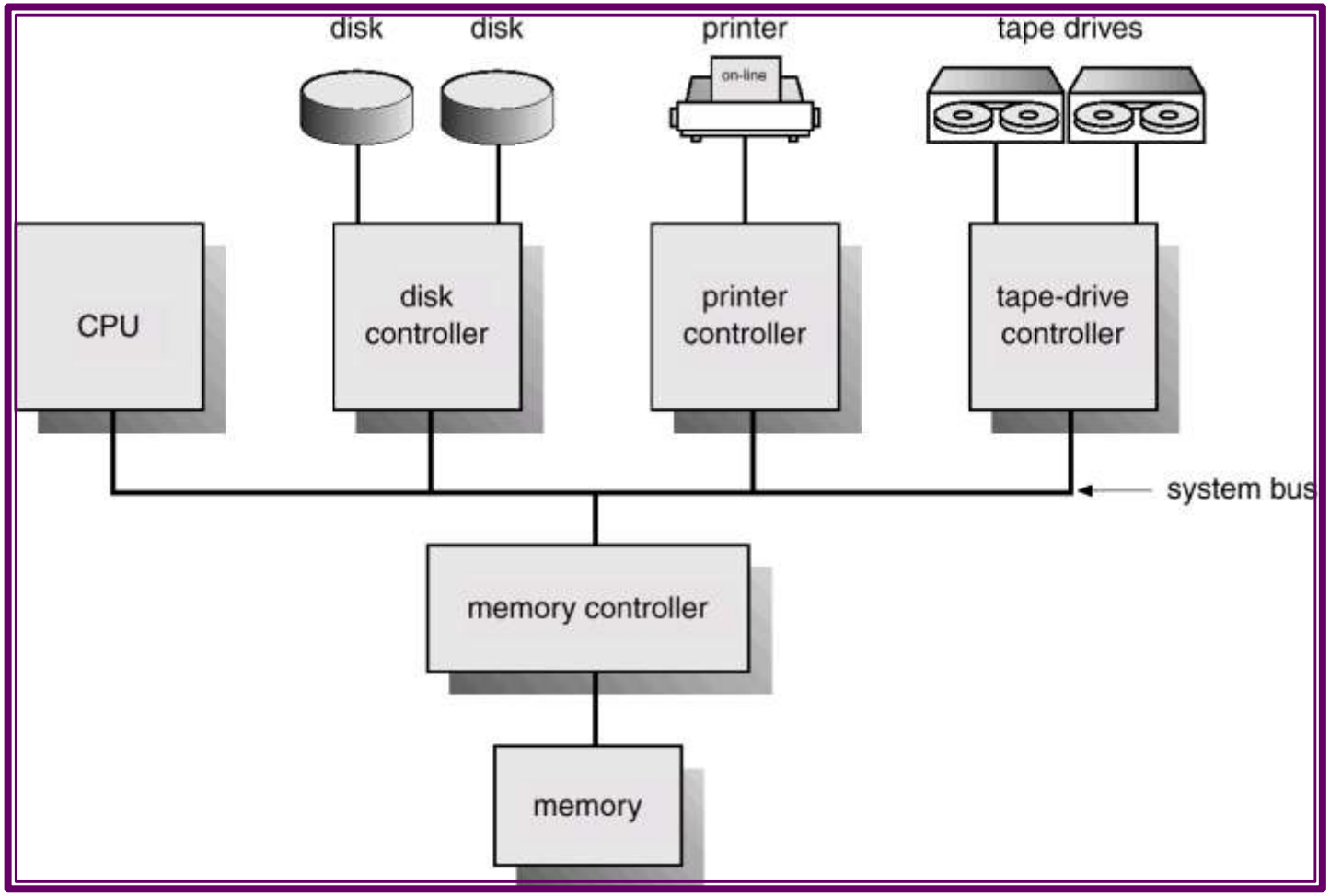
O.S. EVOLUTION

Migration of O.S. concepts and features



PROCESSOR AND O.S. MAIN FEATURES

A generic computer system architecture



PROCESSOR AND O.S. MAIN FEATURES

A generic computer system operation

The initial program (***bootstrap program***) stored in the ROM initializes the system (registers, controllers, memory).

Then localizes, locates and loads into memory the ***kernel***, starting the first process (***initiator***) and waiting for some ***event*** to occurr.

An event is usually signaled by an ***interrupt*** from either the hardware or the software.

The O.S. is interrupt driven.

Hardware may trigger an interrupt by *sending a signal to the CPU*.
Software may trigger an interrupt by *executing a supervisor call* (SVC).

PROCESSOR AND O.S. MAIN FEATURES

Tipi di interruzioni

Interruzione interna (o *trap*): è quella generata dal software ed è causata da un errore (divisione per zero, indirizzo errato di memoria) o dalla richiesta di un servizio del S.O. Ha carattere sincrono, in quanto il processo che lancia la trap va nello stato di wait provocando l'avvio di un segmento di codice del nucleo del sistema operativo (*Supervisor Call o SVC*).

Interruzione esterna (o *interrupt*): ha carattere asincrono, in quanto al verificarsi di un evento, la sua occorrenza è segnalata alla CPU tramite un segnale in ragione del quale viene provocato l'avvio di un segmento di codice del nucleo del sistema operativo.

Ogni processore ha una propria architettura di SVC ed interrupt, anche se le diverse architetture hanno molte caratteristiche in comune.

PROCESSOR AND O.S. MAIN FEATURES

SVC operation

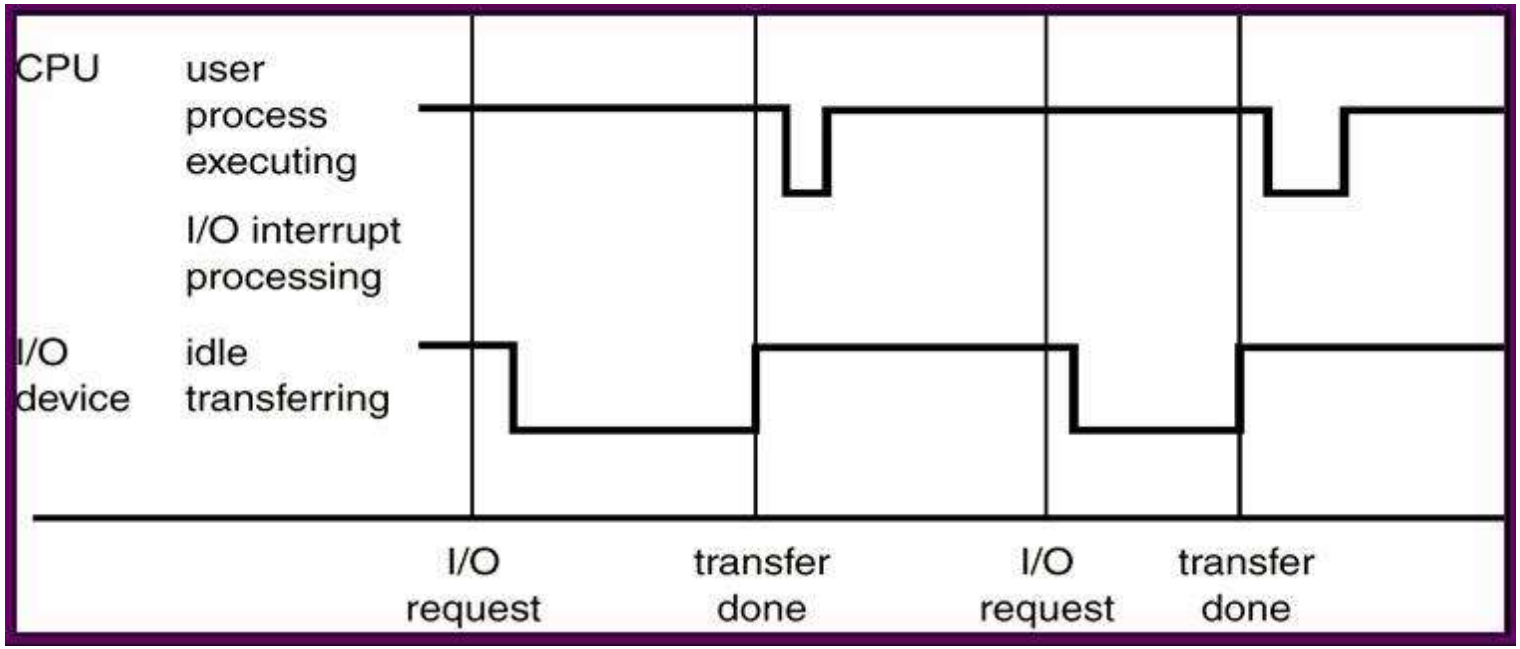
- System calls provide the interface between a running program and the operating system.
  - Generally available as assembly-language instructions.
  - Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, C++)

- Three general methods are used to pass parameters between a running program and the operating system.
  - Pass parameters in *registers*.
  - Store the parameters in a table in memory, and the table address is passed as a parameter in a register.
  - *Push* (store) the parameters onto the *stack* by the program, and *pop* off the stack by operating system.

PROCESSOR AND O.S. MAIN FEATURES

Interrupt operation

- Interrupt transfers control to the interrupt service routine generally, through the *interrupt vector*, which contains the addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction.
- Incoming interrupts are disabled (*interrupt masking*) while another interrupt is being processed to prevent a *lost interrupt*.



PROCESSOR AND O.S. MAIN FEATURES

Gestione di un interrupt

Ogni processore ha un proprio meccanismo di gestione degli interrupt.

Quando avviene un'interruzione il sistema operativo preserva lo stato della CPU.
Il meccanismo più semplice prevede che quando la CPU viene interrotta, si completi l'istruzione corrente, si salvi il contesto computazionale corrente (registri e program counter) in una locazione di memoria (fissa o indicizzata in base al dispositivo da cui è pervenuto l'interrupt) e si passi ad eseguire l'***interrupt handler***, che si trova ad un indirizzo prefissato di memoria e che, in base al codice dell'interrupt, trasferisce l'esecuzione all'indirizzo specificato nell'***interrupt vector***, che occupa le prime posizioni di memoria e che è costituito da tanti indirizzi quanti sono gli interrupt previsti dal processore. Ogni indirizzo rimanda al segmento di codice che serve quel particolare interrupt.

Per rendere più rapide le operazioni, **negli attuali meccanismi di gestione degli interrupt viene usato direttamente il vettore delle interruzioni, senza interrupt handler**.

Al termine dell'esecuzione del segmento di codice indirizzato dall'interrupt vector, si ripristina il contesto computazionale e la CPU riprende l'esecuzione interrotta.

Nelle architetture più recenti il contesto computazionale viene salvato sullo stack del sistema.

Una SVC viene servita facendo anche essa riferimento al vettore delle interruzioni.

PROCESSOR AND O.S. MAIN FEATURES

I/O operation

- I/O devices and the CPU can execute concurrently.
- Each device controller is in charge of a particular device type.
- Each device controller has a local buffer.
- CPU moves data from/to main memory to/from local buffers of suitable size.
- I/O is from the device to local buffer of controller.
- Device controller informs CPU that it has finished its operation by causing an *interrupt*.

PROCESSOR AND O.S. MAIN FEATURES

Servicing an I/O operation

To start an I/O operation the CPU loads the appropriate registers into the device controller.

The device controller examines the content of the registers to determin the action to take and starts the requested transfer of data. The I/O operation may be accomplished in two different ways: Synchronous or asynchronous I/O.

Once the transfer is completed, the controller sends an interrupt to the CPU to signal it has finished the requested operation.

PROCESSOR AND O.S. MAIN FEATURES

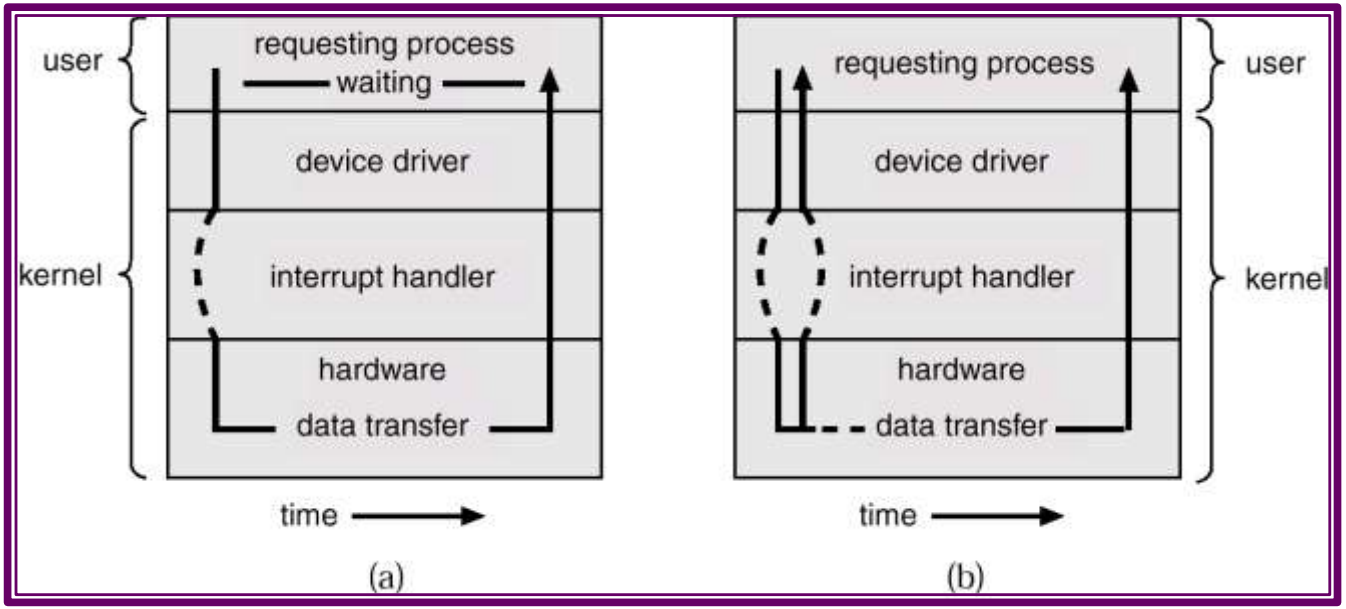
Synchronous and asynchronous I/O

(a) Synchronous method

- After I/O starts, control returns to user program only upon I/O completion.
  - A special ***wait instruction*** idles the CPU until the next interrupt or a ***tight wait loop*** continues until an interrupt occurs, transferring control to another part of the O.S. (contention for memory access).
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing.

(b) Asynchronous method

- After I/O starts, control returns to user program without waiting for I/O completion.
  - A system call is requested to allow user to wait for I/O completion.
  - If no user or O.S. process needs for CPU, a wait instruction or wait loop is requested



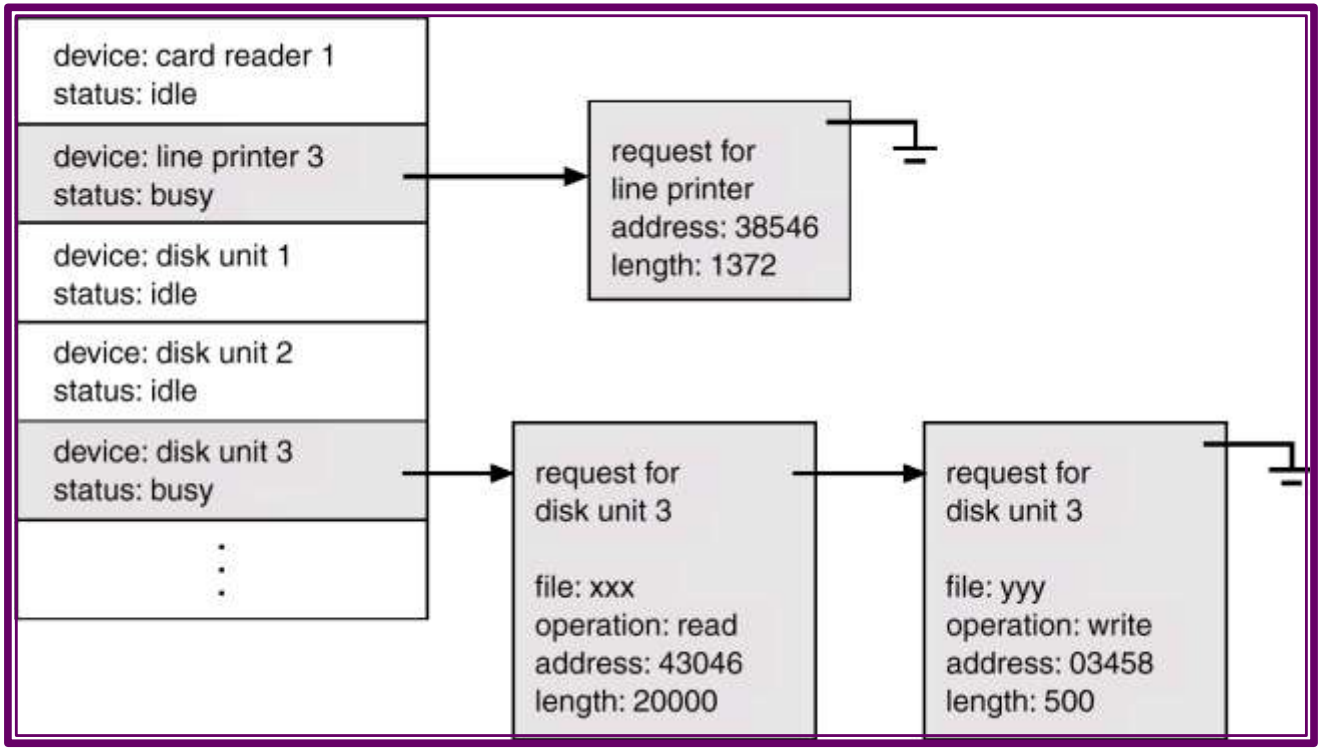
PROCESSOR AND O.S. MAIN FEATURES

Device Status Table

To override the problem of the wait instruction or of the tight loop, and to be able to keep track of many I/O requests at the same time, the O.S. uses a table, the Device Status table, containing an entry for each I/O device.

The O.S. indexes into I/O device table to determine device status and to modify table entry to include interrupt.

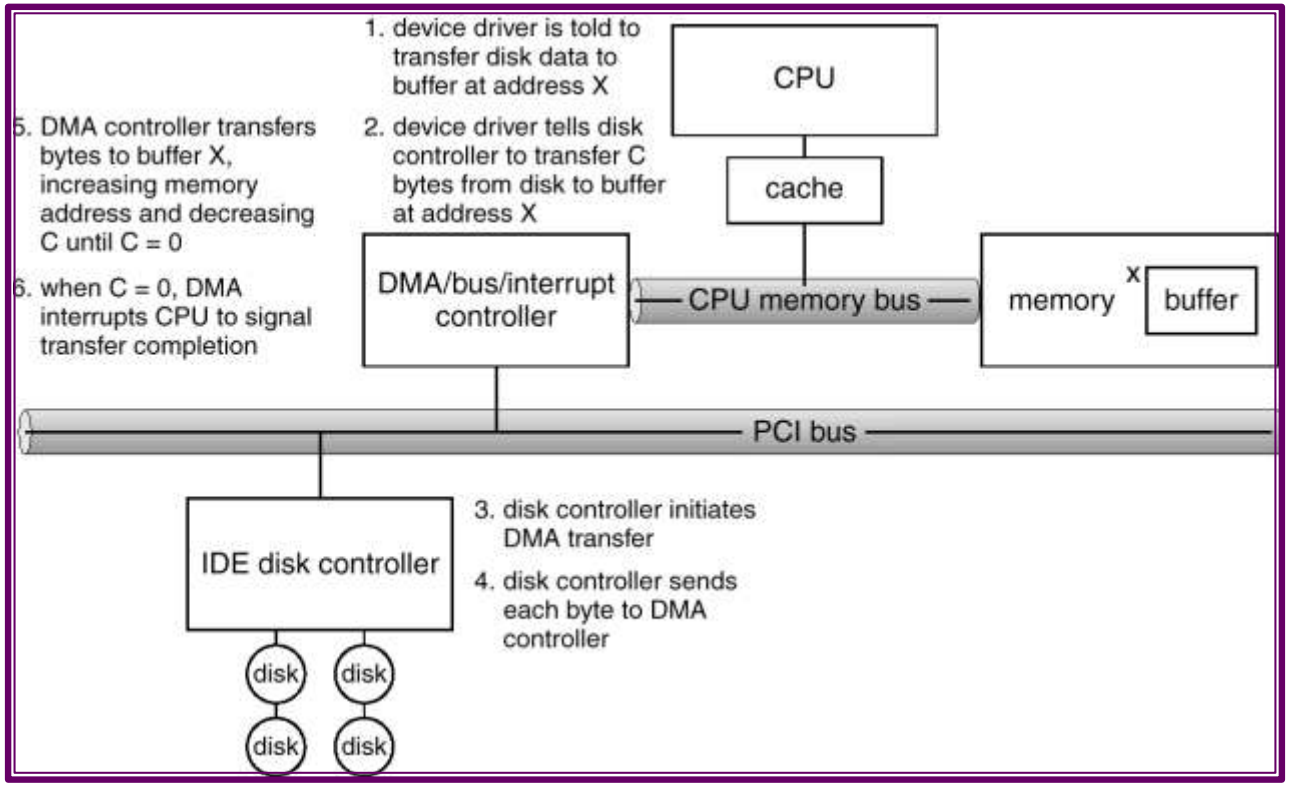
Device-status table contains entry for each I/O device indicating its type, address, and state.



PROCESSOR AND O.S. MAIN FEATURES

Direct Memory Access (DMA)

- Used for high-speed I/O devices, to avoid programmed I/O for large data movement.
- Able to transmit information at close to memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Only one interrupt is generated per block, rather than the one interrupt per byte.



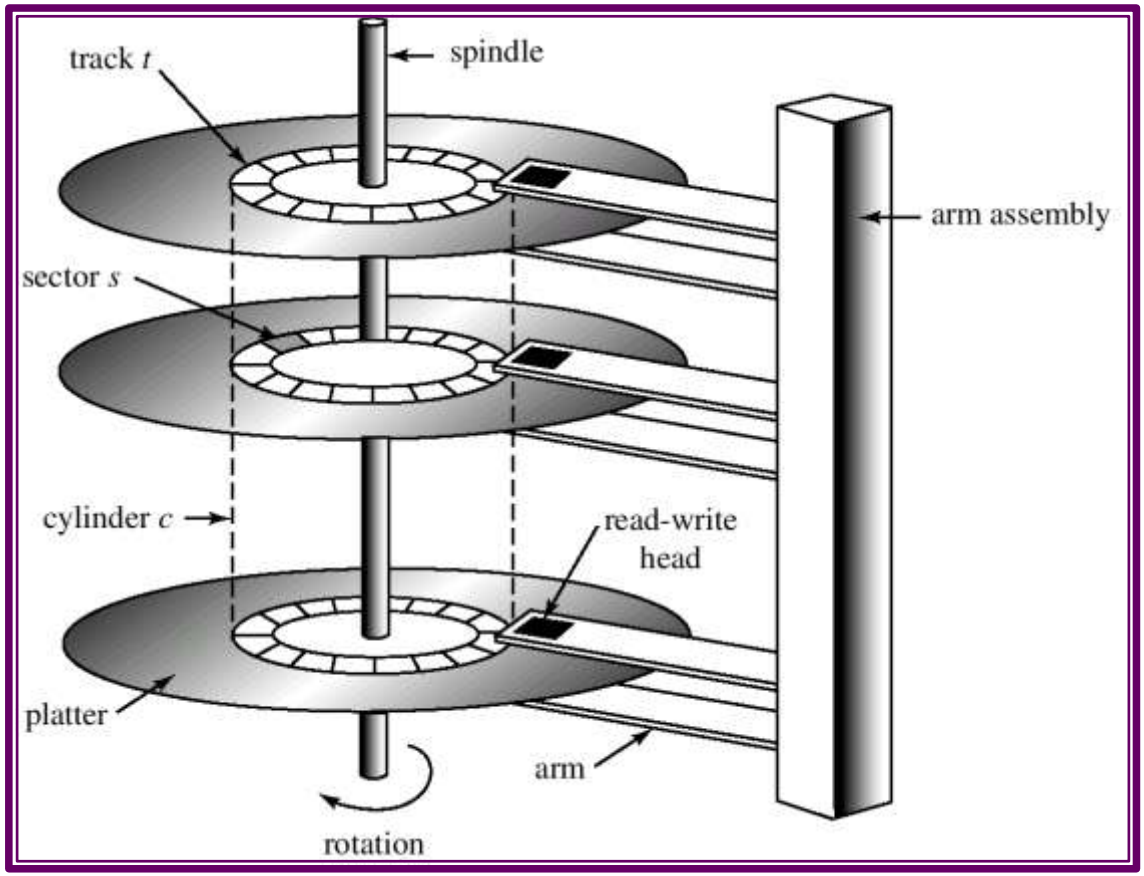
MEMORY TYPES

- ***Main memory*** – only large storage media that the CPU can access directly.
- ***Secondary storage*** – extension of main memory that provides large nonvolatile storage capacity.

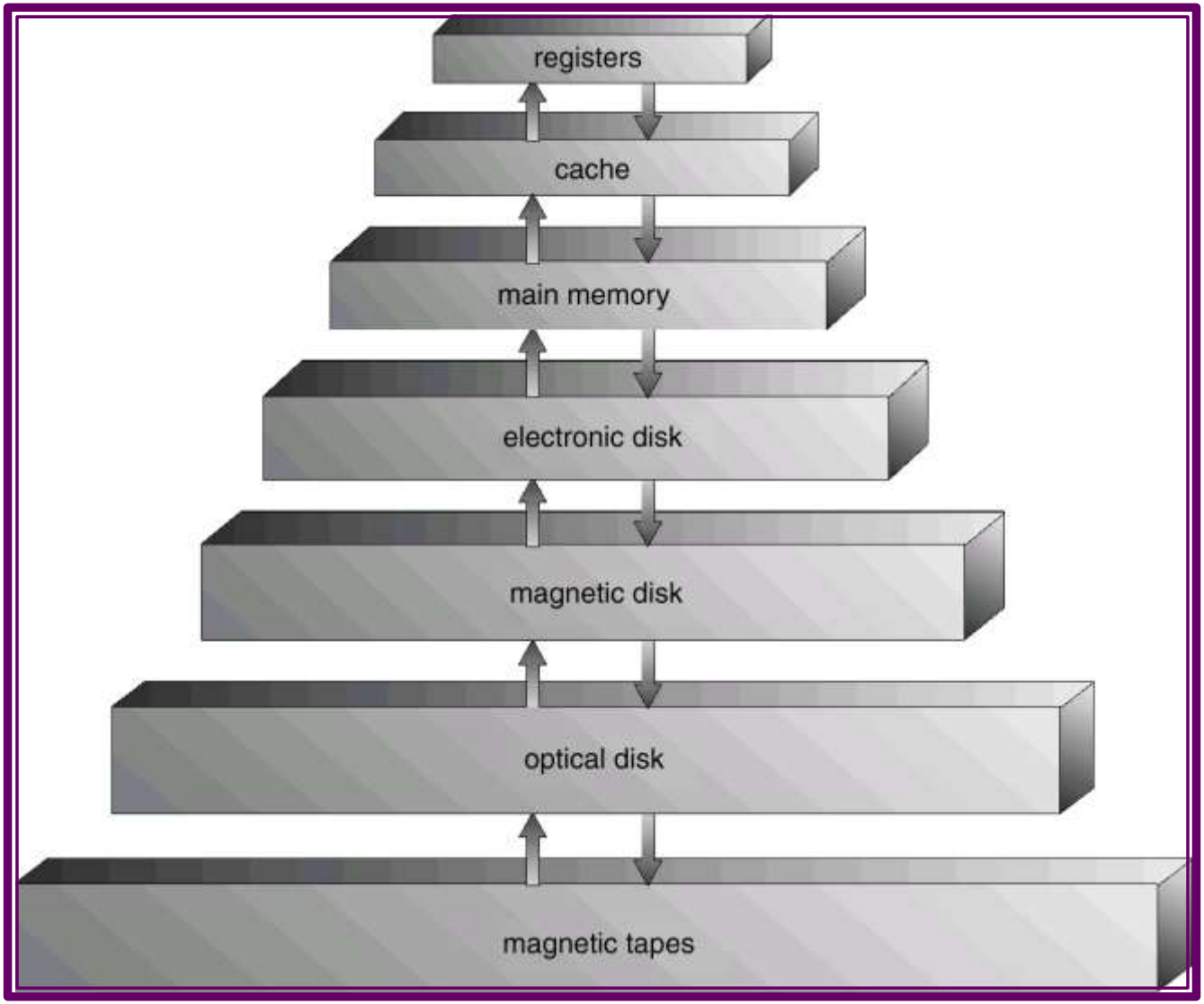
  Most common secondary storage: Magnetic disks – rigid metal or glass platters covered with magnetic recording material

  Φ Disk surface is logically divided into *tracks*, which are subdivided into *sectors*.

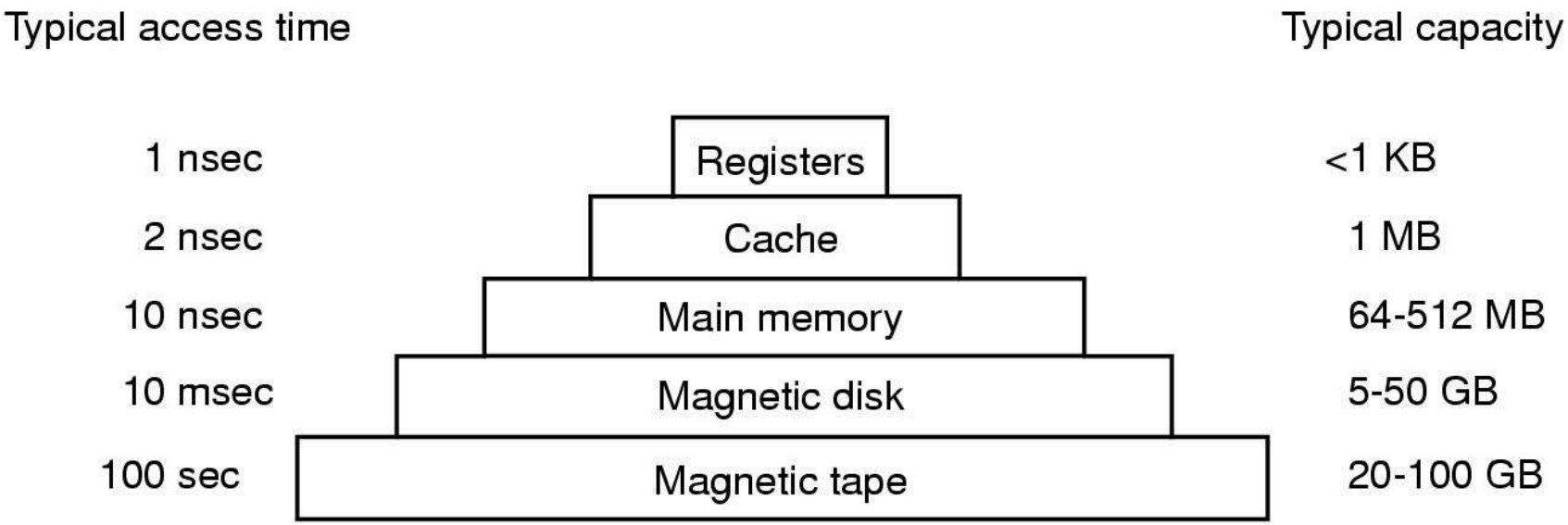
  Φ The *disk controller* determines the logical interaction between the device and the computer.



STORAGE DEVICE HIERARCHY

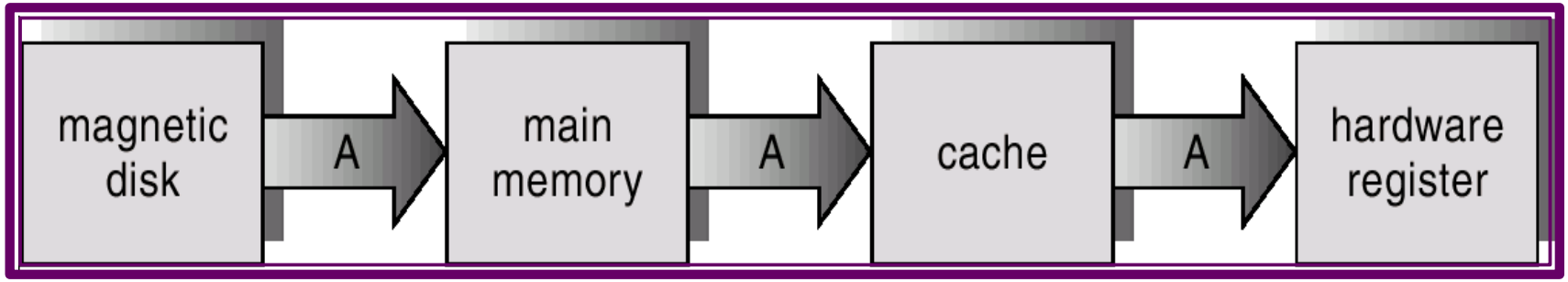


STORAGE DEVICE HIERARCHY



CACHE MEMORY

- Use of high-speed memory to hold recently-accessed data.
- Requires a *cache management* policy.
- Caching introduces another level in storage hierarchy. This requires data that is simultaneously stored in more than one level to be *consistent*.

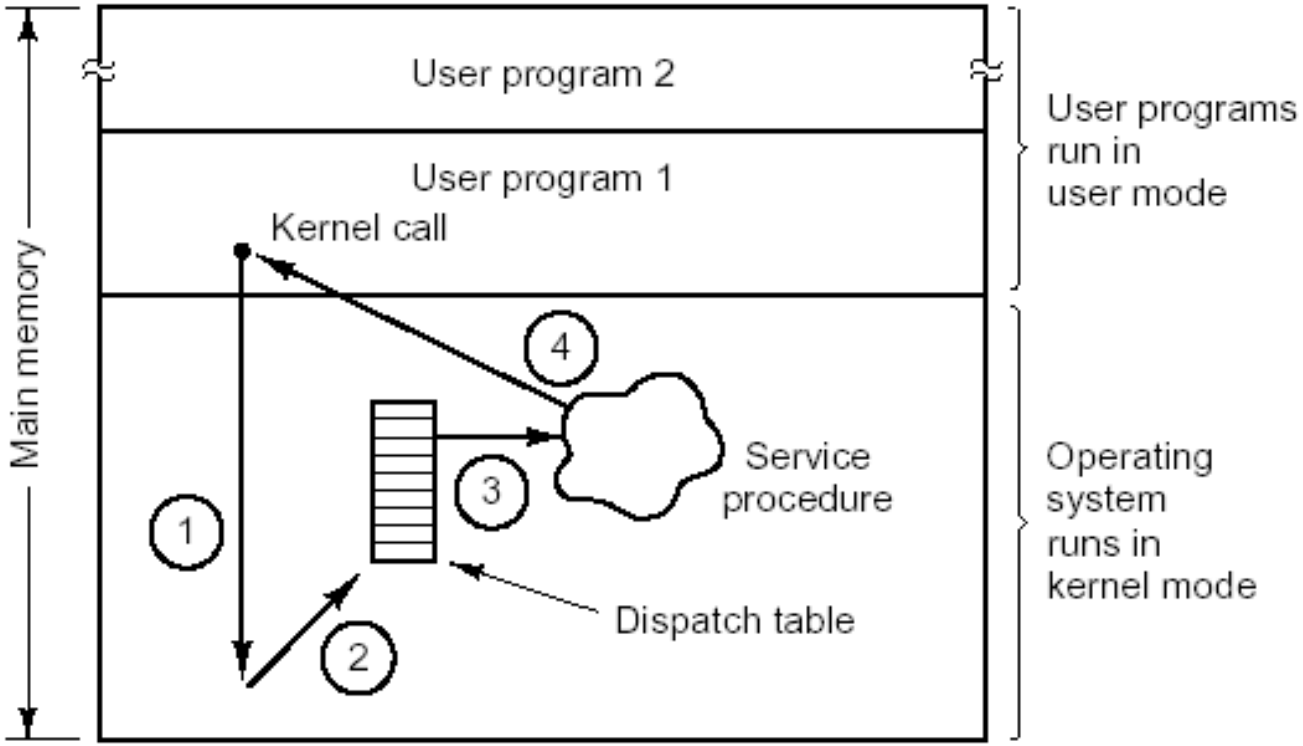


O.S. ARCHITECTURE

L'architettura del S.O. può essere

- ✓ **monolitica**, quando esso è composto da un unico modulo che serve le richieste dei programmi-utente una alla volta;
- ✓ **a macchina virtuale**, se esso offre a ciascun programma-utente visibilità di un particolare hardware
- ✓ **client-server,** se esso prevede un nucleo minimo di funzioni comuni (*microkernel*) a tutte le stazioni di un sistema distribuito, a cui alcune stazioni (*server*) aggiungono funzioni specifiche per offrire servizi ad altre stazioni (*client*).
- ✓ **a livelli**, se esso è articolato in diversi moduli, ciascuno dei quali svolge specifiche funzioni, ed ogni modulo può servire le richieste di più programmi-utente

MONOLITIC ARCHITECTURE OF AN O.S.

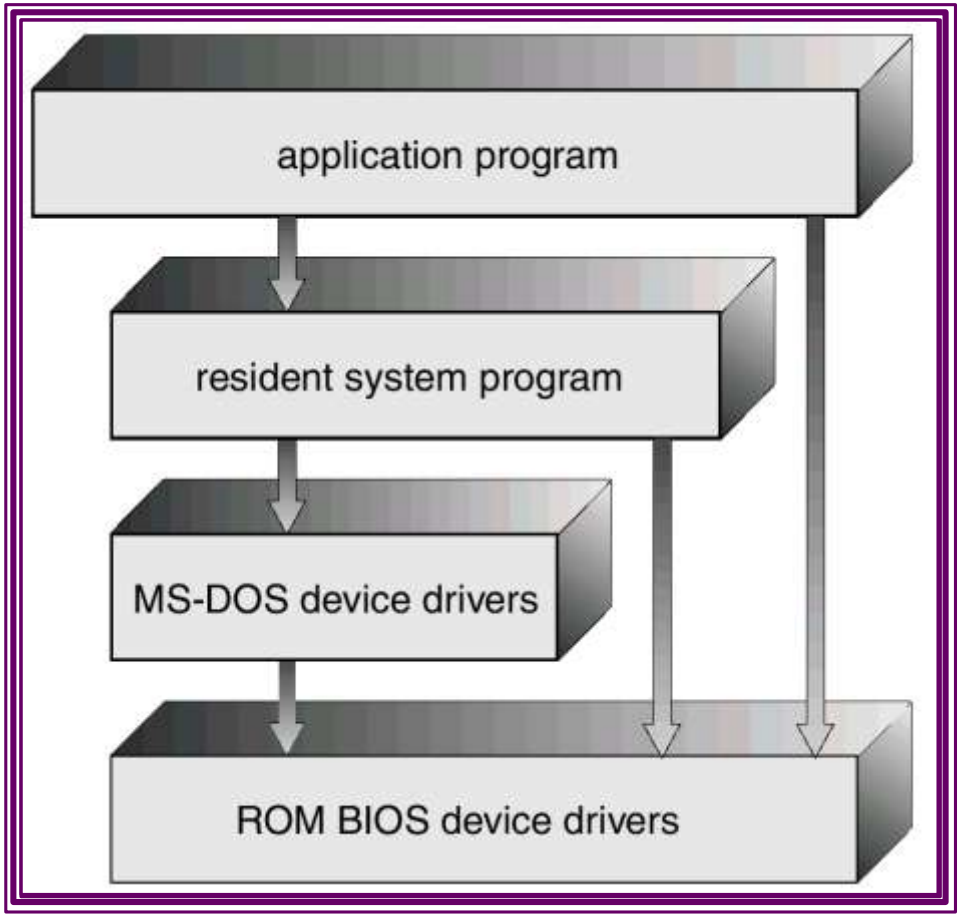


How a system call can be made:

(1) User program traps to the kernel.
(2) Operating system determines service number required.
(3) Operating system calls service procedure.
(4) Control is returned to user program.

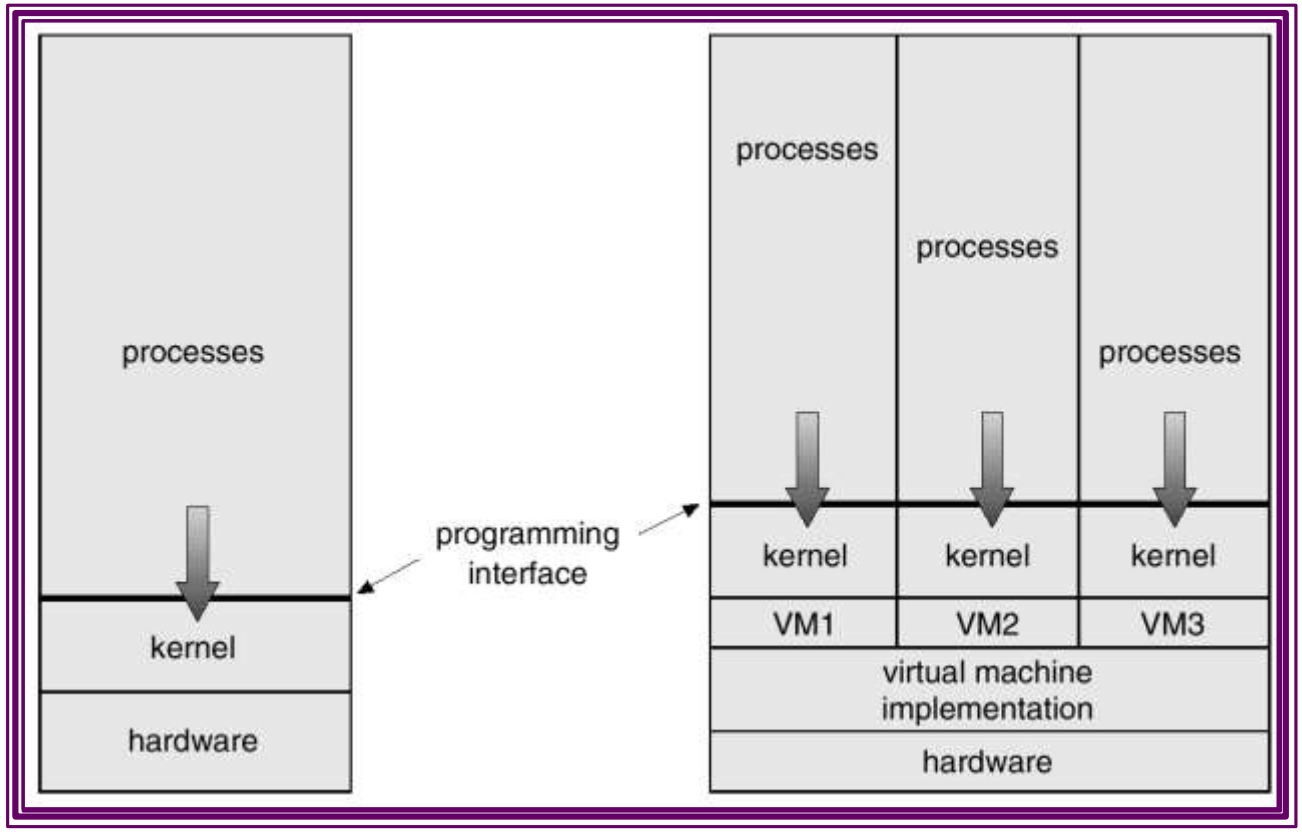
MONOLITIC ARCHITECTURE OF AN O.S.

- ✓ MS-DOS – written to provide the most functionality in the least space
  - Φ not divided into modules
  - Φ although MS-DOS has some structure, its interfaces and levels of functionality are not well separated.



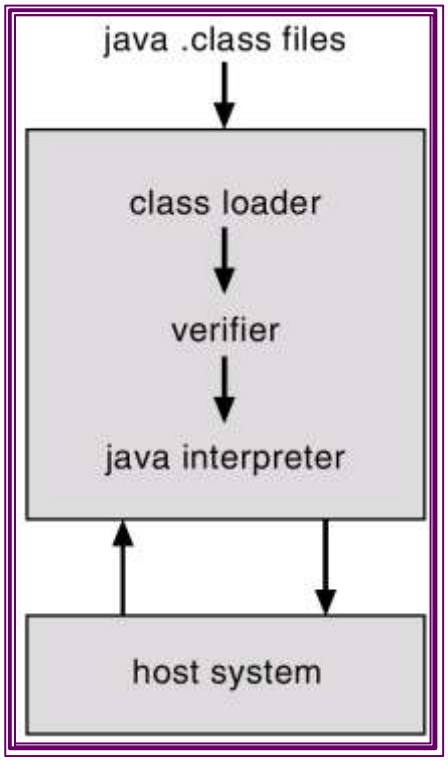
VIRTUAL MACHINE ARCHITECTURE OF AN O.S.

- ✓ A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware.
- ✓ A virtual machine provides an interface *identical* to the underlying bare hardware.
- ✓ The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.
- ✓ The resources of the physical computer are shared to create the virtual machines.



Java Virtual Machine (JVM)

- ✓ Compiled Java programs are platform-neutral bytecodes executed by a Java Virtual Machine (JVM).
- ✓ JVM consists of
  - Φ class loader
  - Φ class verifier
  - Φ runtime interpreter
- ✓ Just-In-Time (JIT) compilers increase performance

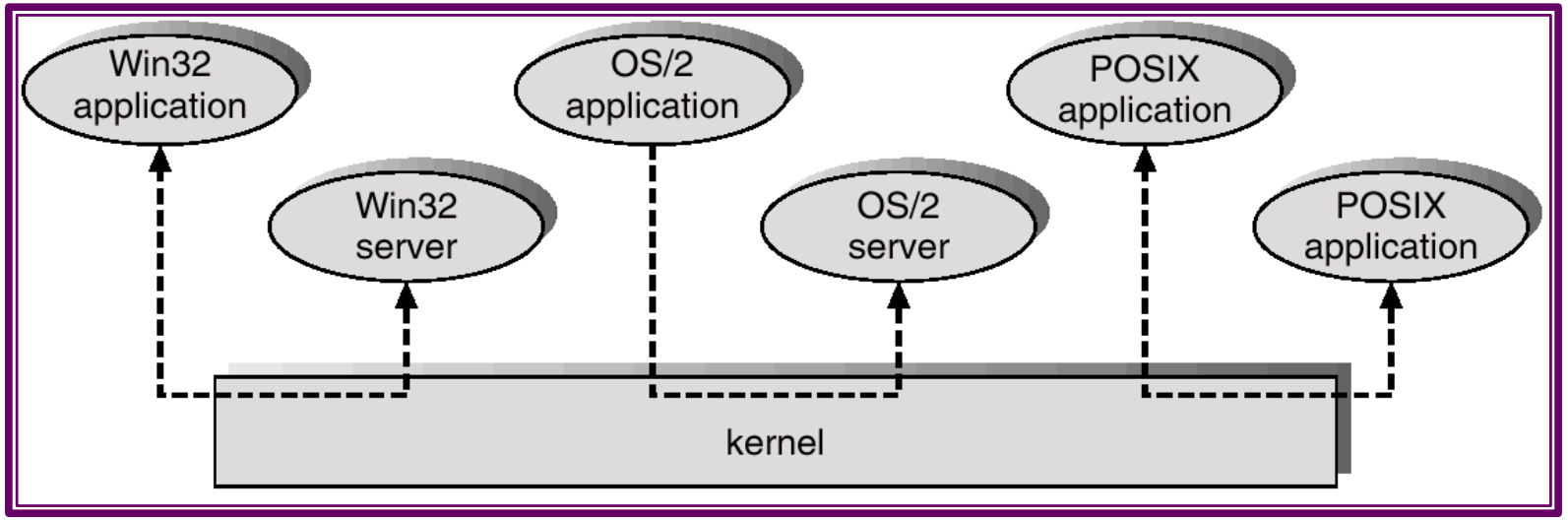


CLIENT-SERVER ARCHITECTURE OF AN O.S.

The microkernel

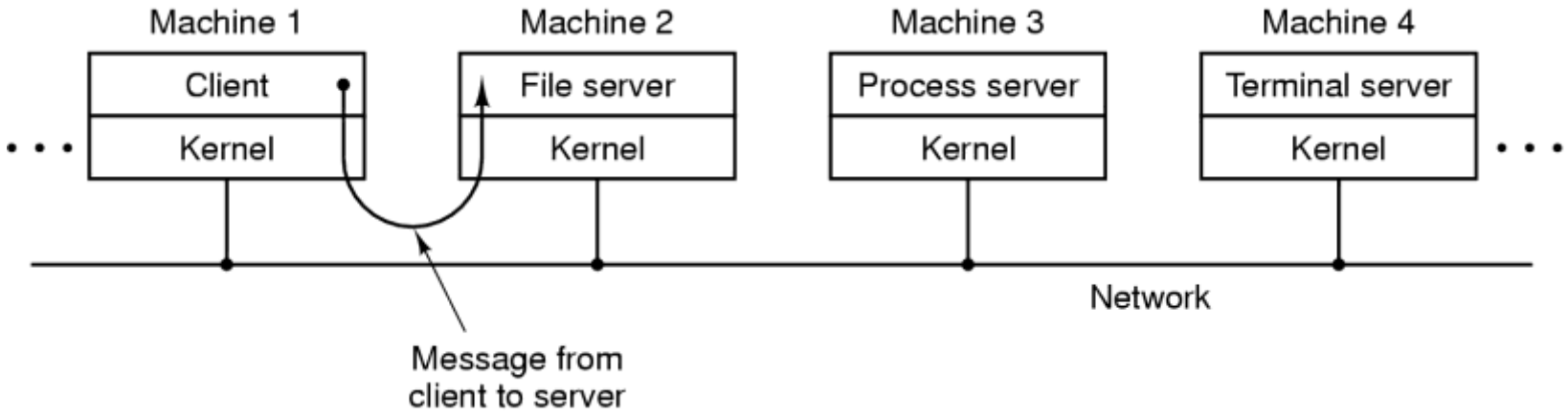
- ✓ Moves as much from the kernel into "*user*" space.
- ✓ Communication takes place between user modules using message passing.
- ✓ Benefits:
  - easier to extend a microkernel
  - easier to port the operating system to new architectures
  - more reliable (less code is running in kernel mode)
  - more secure

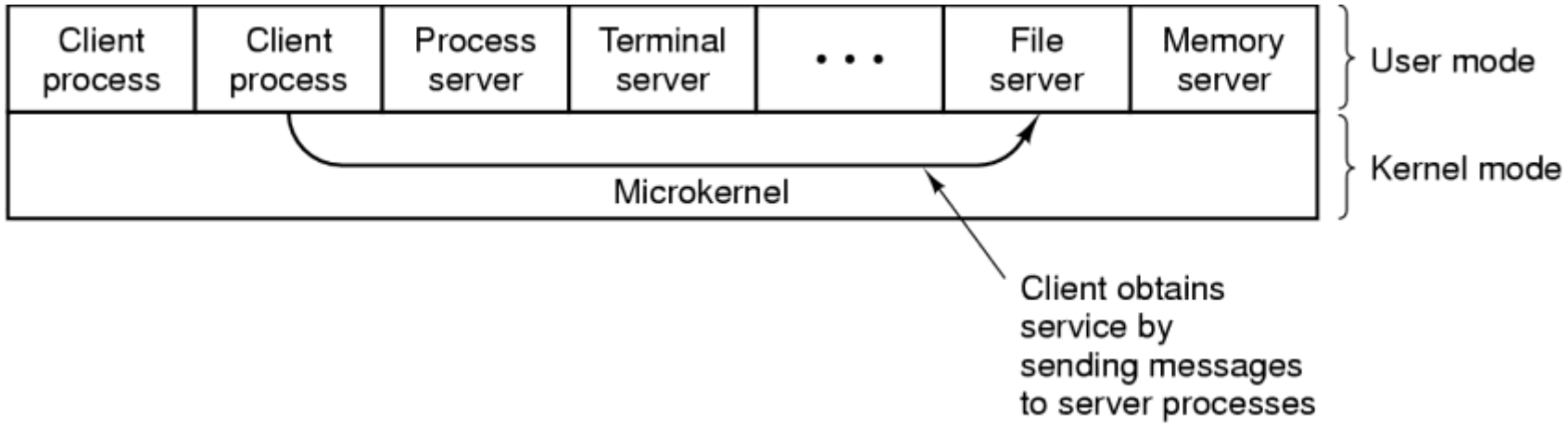
Windows NT client-server architecture



CLIENT-SERVER ARCHITECTURE OF AN O.S.

General model





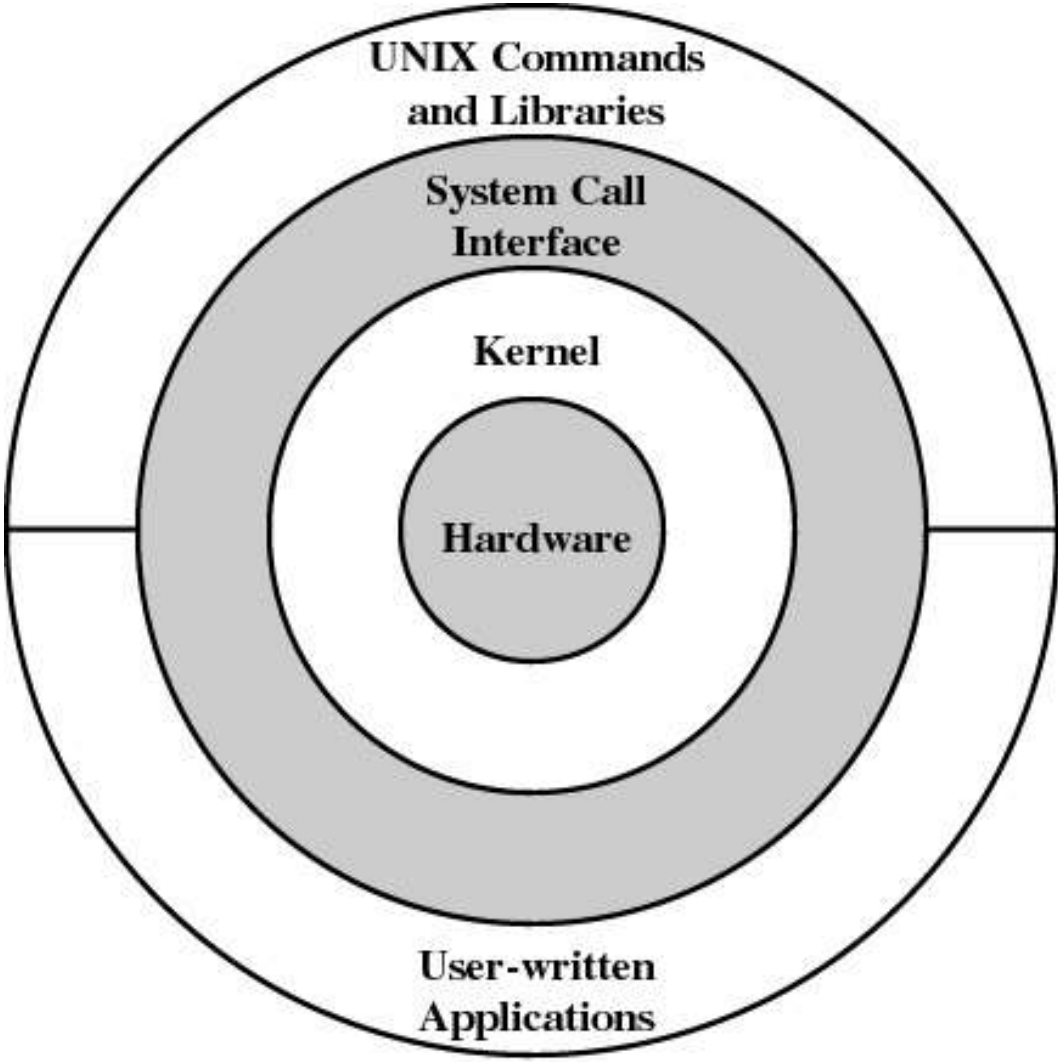
LAYERED ARCHITECTURE OF AN O.S.

- ✓ The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- ✓ With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.

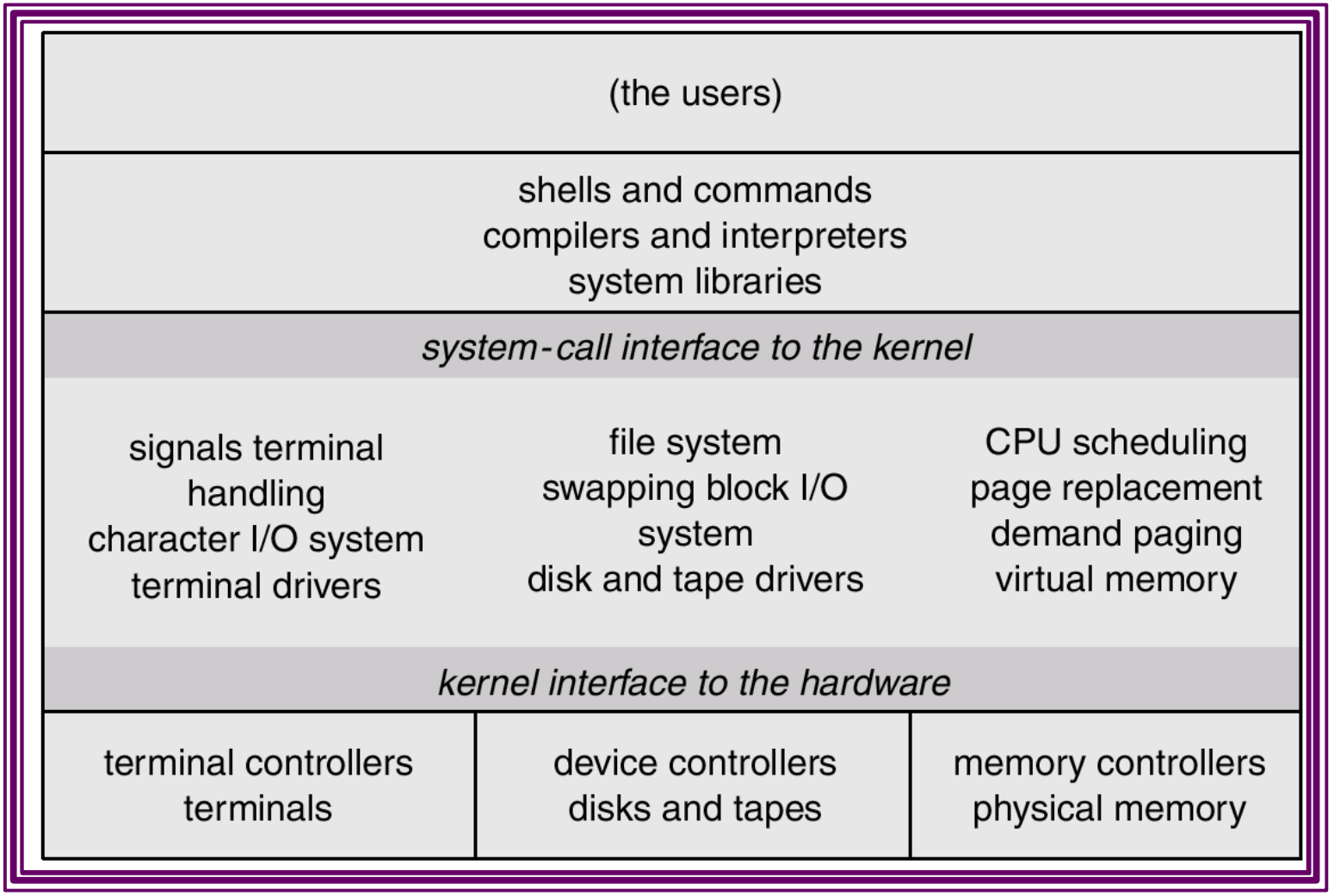
The original UNIX operating system had limited structuring. It consists of two separable parts.

- Φ Systems programs
- Φ The ***kernel***
  - consists of everything below the system-call interface and above the physical hardware
  - provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.

LAYERED ARCHITECTURE OF UNIX

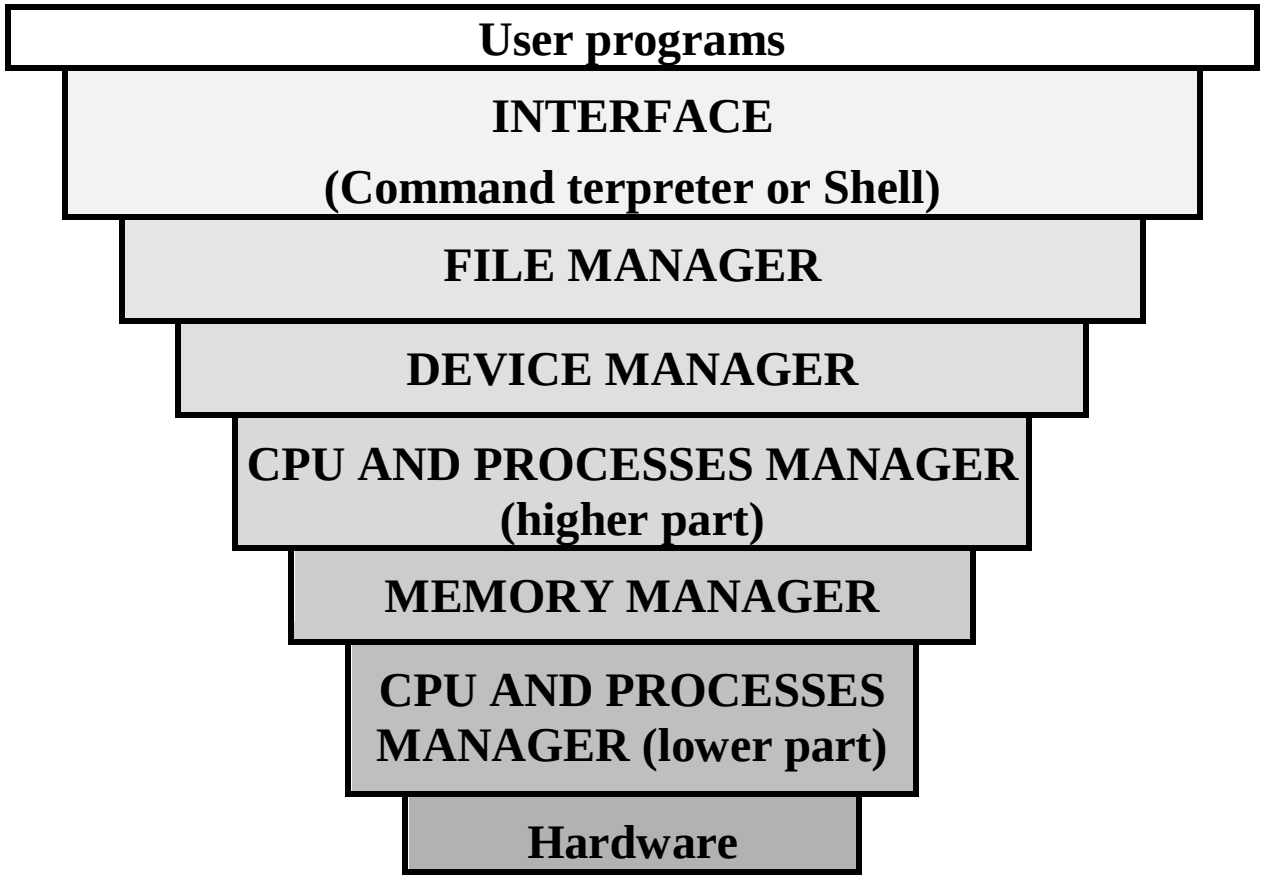


LAYERED ARCHITECTURE OF UNIX



GENERAL MODEL OF AN O.S. LAYERED ARCHITECTURE

The general model



GENERAL MODEL OF AN O.S. LAYERED ARCHITECTURE

General model

L'Interfaccia comprende il Job Scheduler.

I livelli tra l'Interfaccia e l'Hardware costituiscono il **Nucleo** o **kernel**.

I moduli di un livello possono utilizzare moduli dei livelli sottostanti mediante chiamate al Supervisore o ***primitive*** sincrone (***trap*** o ***interrupt interni)***.

Ogni livello garantisce la gestione (*management*) di una risorsa e comprende tutti i moduli che contribuiscono a svolgere tale compito (*resource manager*).

I *resource manager* sono:

- Φ il gestore dei ***file***
- Φ il gestore dei ***dispositivi***
- Φ il gestore della ***CPU*** e dei ***processi***
- Φ il gestore della ***memoria centrale***

Il Manager della CPU e dei processi (o Process Scheduler) è suddiviso in due parti:

- ☞ la parte alta comprende i moduli per la ***creazione/distruzione di processi*** e per la ***comunicazione tra processi***;
- ☞ la parte bassa comprende i moduli per la ***sincronizzazione di processi***.

CPU AND PROCESS MANAGEMENT

The operating system is responsible for the following activities connected with CPU management:

- ✓ Keep track of which process is currently using CPU.
- ✓ Decide to which process (among those in the ready status) to assign CPU.
- ✓ Allocate and deallocate CPU according to the process priorities.

The operating system is responsible for the following activities in connection with process management.

- Φ Process creation and deletion.
- Φ Process suspension and resumption.
- Φ Provision of mechanisms for:
  - ☞ process synchronization
  - ☞ process communication

MAIN MEMORY MANAGEMENT

- ✓ Main memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.
- ✓ RAM main memory is a volatile storage device. It looses its contents in the case of system failure.
- ✓ The operating system is responsible for the following activities in connections with memory management:
  - Φ Keep track of which parts of memory are currently being used and by whom.
  - Φ Decide which processes to load when memory space becomes available.
  - Φ Allocate and deallocate memory space as needed.

FILE MANAGEMENT

- ✓ A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data.
- ✓ The operating system is responsible for the following activities in connections with file management:
  - Φ File creation and deletion.
  - Φ Directory creation and deletion.
  - Φ Support of primitives for manipulating files and directories.
  - Φ Mapping files onto secondary storage.
  - Φ File backup on stable (nonvolatile) storage media.

DEVICE MANAGEMENT

- ✓ The I/O system consists of:
  - Φ A buffer-caching system
  - Φ A general device-driver interface
  - Φ Drivers for specific hardware devices (secondary storage, printers, tablets, ecc.)