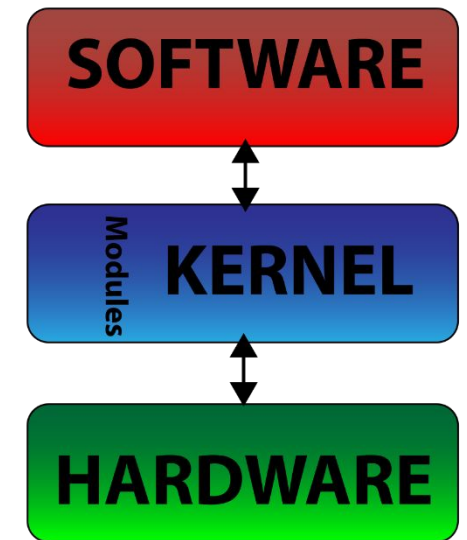




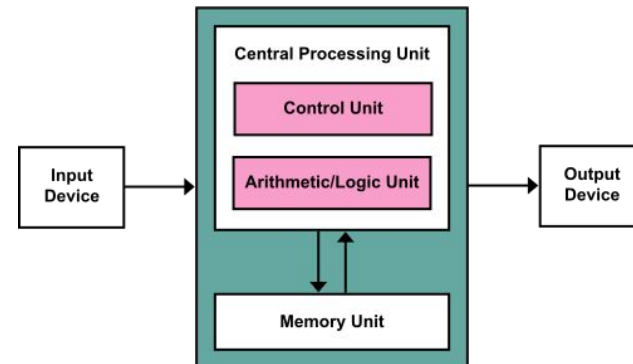
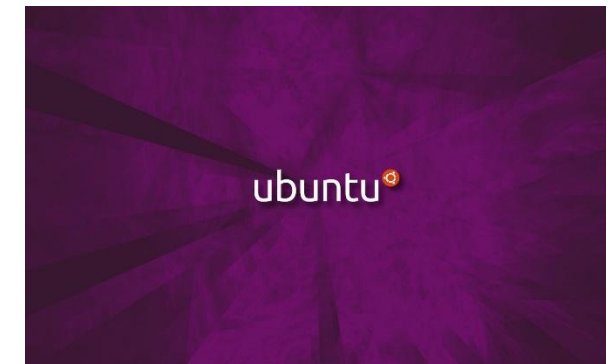
**UNIVERSITÀ DEGLI STUDI
DELLA BASILICATA**

Corso di Sistemi Operativi

Strutture dei sistemi operativi

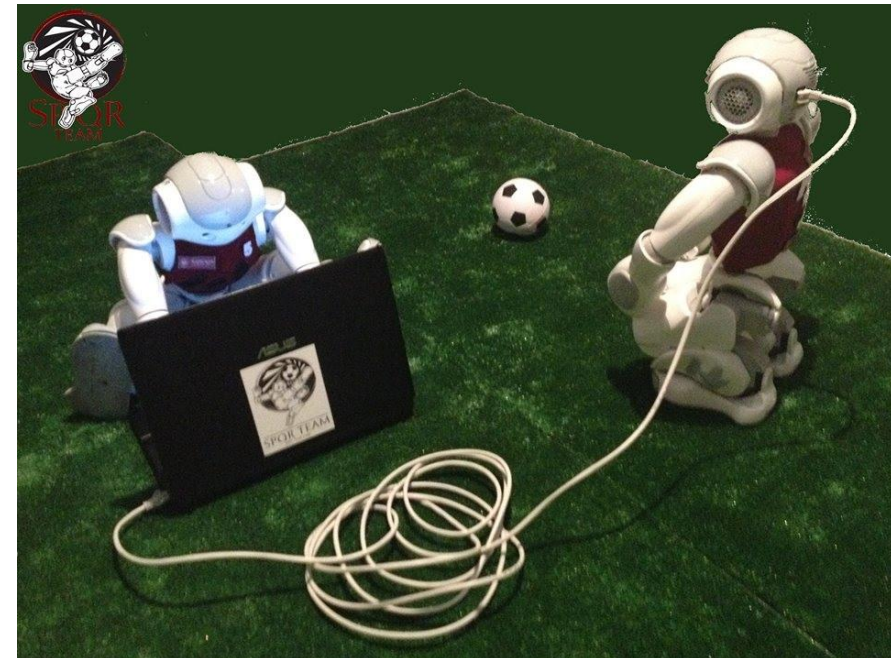
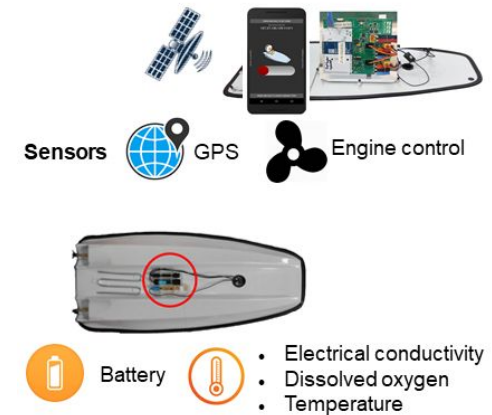


Docente:
**Domenico Daniele
Bloisi**



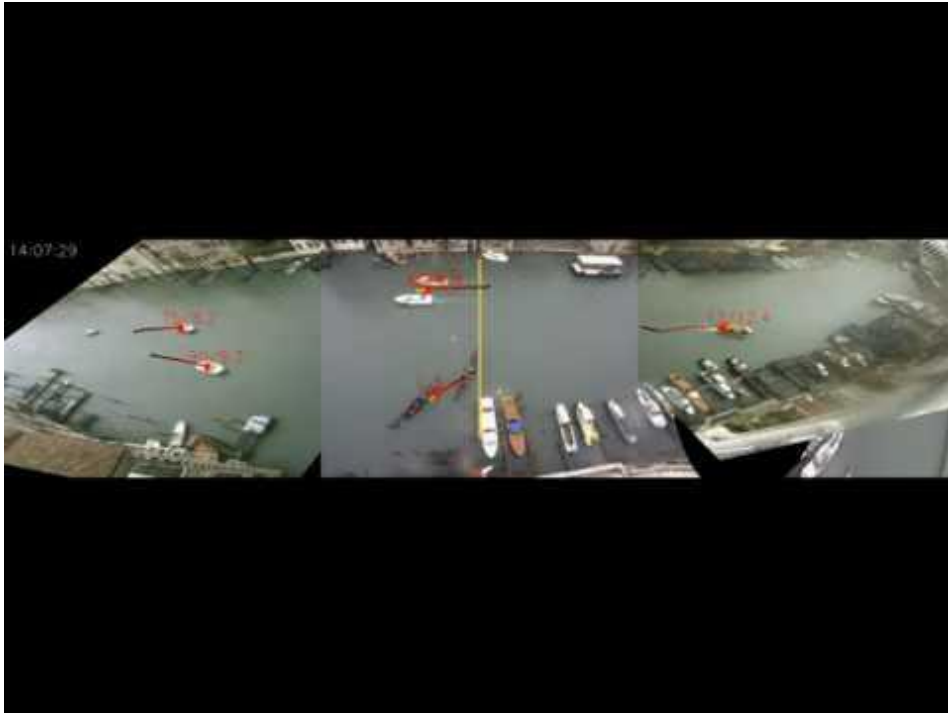
Domenico Daniele Bloisi

- Professore Associato
Dipartimento di Matematica, Informatica
ed Economia
Università degli studi della Basilicata
<http://web.unibas.it/bloisi>
- SPQR Robot Soccer Team
Dipartimento di Informatica, Automatica
e Gestionale Università degli studi di
Roma “La Sapienza”
<http://spqr.diag.uniroma1.it>



Interessi di ricerca

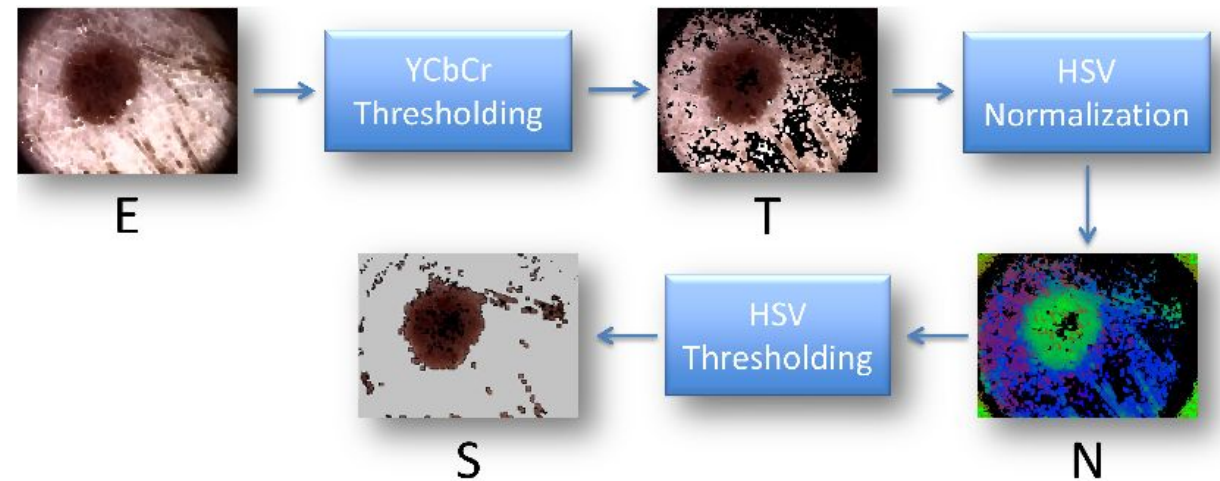
- Intelligent surveillance
- Robot vision
- Medical image analysis



https://youtu.be/9a70Ucgbi_U



<https://youtu.be/2KHNZX7UIWQ>



UNIBAS Wolves <https://sites.google.com/unibas.it/wolves>



- UNIBAS WOLVES is the robot soccer team of the University of Basilicata. Established in 2019, it is focussed on developing software for NAO soccer robots participating in RoboCup competitions.

- UNIBAS WOLVES team is twinned with SPQR Team at Sapienza University of Rome



<https://youtu.be/ji0OmkaWh20>

Informazioni sul corso

- Home page del corso:
<http://web.unibas.it/bloisi/corsi/sistemi-operativi.html>
- Docente: Domenico Daniele Bloisi
- Periodo: I semestre ottobre 2022 – gennaio 2023
 - Lunedì dalle 15:00 alle 17:00 (Aula Leonardo)
 - Martedì dalle 08:30 alle 10:30 (Aula 1)

Ricevimento

- In presenza, durante il periodo delle lezioni:
Lunedì dalle 17:00 alle 18:00 □ Edificio 3D, Il piano, stanza 15
Si invitano gli studenti a controllare regolarmente la bacheca degli avvisi per eventuali variazioni
- Tramite google Meet e al di fuori del periodo delle lezioni:
da concordare con il docente tramite email

Per prenotare un appuntamento inviare
una email a
domenico.bloisi@unibas.it



Programma – Sistemi Operativi

- **Introduzione ai sistemi operativi**
- Gestione dei processi
- Sincronizzazione dei processi
- Gestione della memoria centrale
- Gestione della memoria di massa
- File system
- Sicurezza e protezione

Servizi di un sistema operativo

Un sistema operativo offre un ambiente in cui eseguire i programmi e fornire i seguenti servizi.



Servizi di un sistema operativo

Allocazione
delle risorse

Logging

Protezione
e sicurezza

Servizi di un sistema operativo

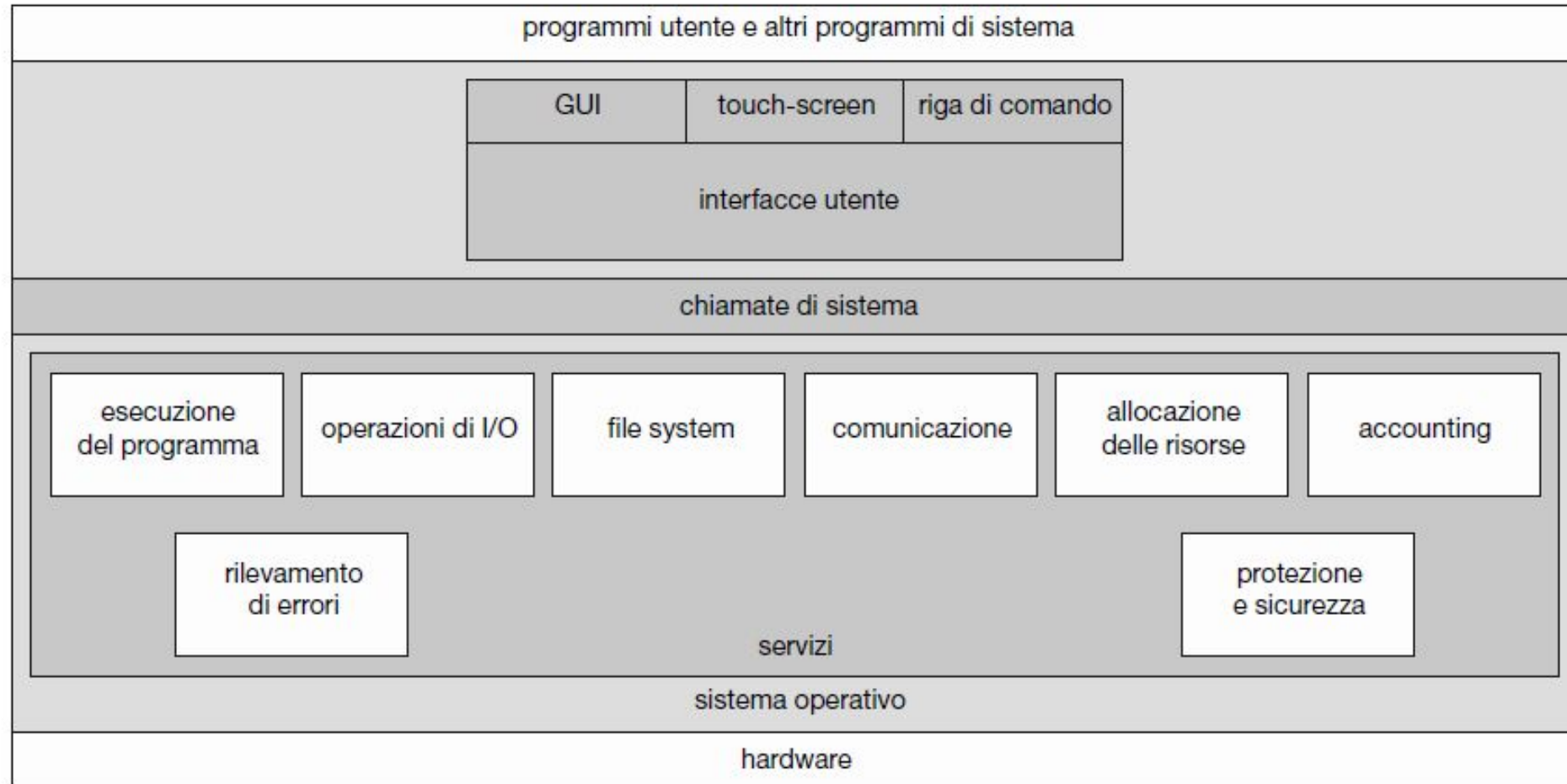


Figura 2.1 Panoramica dei servizi del sistema operativo.

Interfaccia con l'utente del sistema operativo

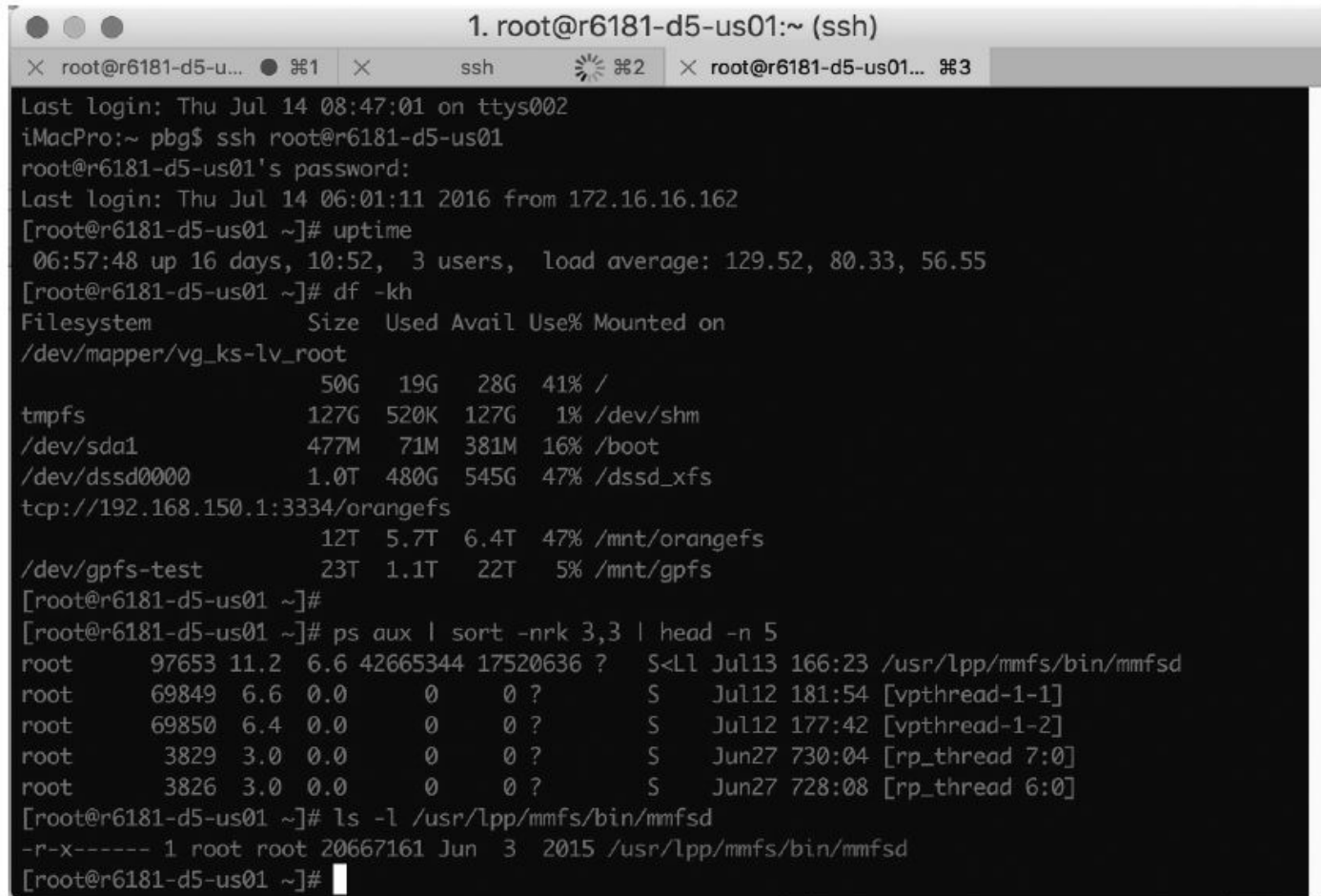
1. interfaccia a riga di comando o

interprete dei comandi

2. Interfaccia **touch-screen**

3. Interfaccia grafica con l'utente o **GUI**

Interprete dei comandi



```
1. root@r6181-d5-us01:~ (ssh)
X root@r6181-d5-u... ● #1 X ssh #2 X root@r6181-d5-us01... #3
Last login: Thu Jul 14 08:47:01 on ttys002
iMacPro:~ pbg$ ssh root@r6181-d5-us01
root@r6181-d5-us01's password:
Last login: Thu Jul 14 06:01:11 2016 from 172.16.16.162
[root@r6181-d5-us01 ~]# uptime
 06:57:48 up 16 days, 10:52,  3 users,  load average: 129.52, 80.33, 56.55
[root@r6181-d5-us01 ~]# df -kh
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/vg_ks-lv_root
                  50G   19G   28G  41% /
tmpfs            127G  520K  127G   1% /dev/shm
/dev/sda1        477M   71M  381M  16% /boot
/dev/dssd0000    1.0T  480G  545G  47% /dssd_xfs
tcp://192.168.150.1:3334/orangefs
                  12T  5.7T  6.4T  47% /mnt/orangefs
/dev/gpfs-test   23T  1.1T  22T   5% /mnt/gpfs
[root@r6181-d5-us01 ~]#
[root@r6181-d5-us01 ~]# ps aux | sort -nrk 3,3 | head -n 5
root    97653 11.2  6.6 42665344 17520636 ?    S<Ll  Jul13 166:23 /usr/lpp/mmfs/bin/mmfstd
root    69849  6.6  0.0    0    0 ?    S    Jul12 181:54 [vpthread-1-1]
root    69850  6.4  0.0    0    0 ?    S    Jul12 177:42 [vpthread-1-2]
root    3829   3.0  0.0    0    0 ?    S    Jun27 730:04 [rp_thread 7:0]
root    3826   3.0  0.0    0    0 ?    S    Jun27 728:08 [rp_thread 6:0]
[root@r6181-d5-us01 ~]# ls -l /usr/lpp/mmfs/bin/mmfstd
-r-x----- 1 root root 20667161 Jun  3 2015 /usr/lpp/mmfs/bin/mmfstd
[root@r6181-d5-us01 ~]#
```

Interprete dei
comandi



shell

Figura 2.2 La shell bash, l'interprete dei comandi utilizzato in macOS.

Interfaccia touch-screen

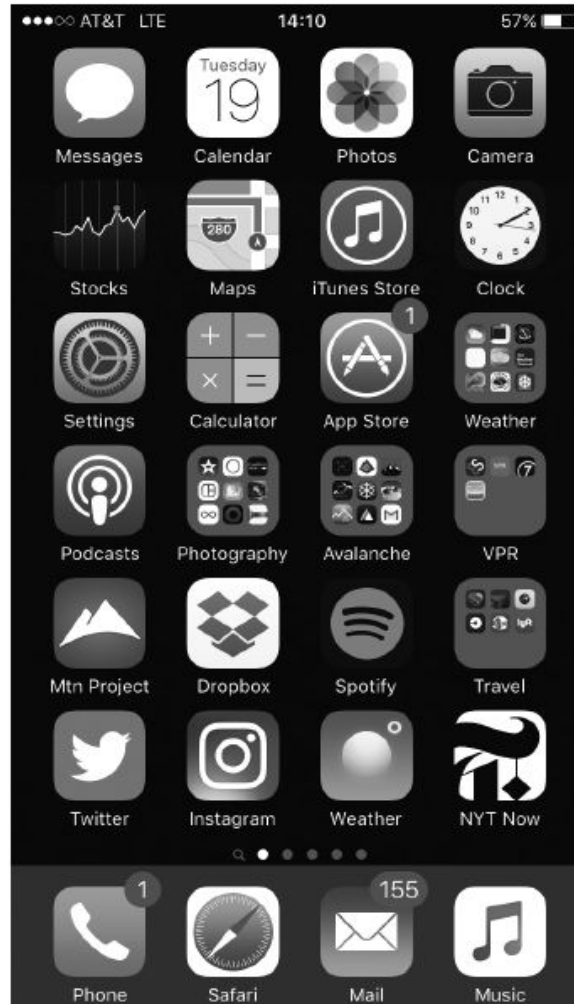


Figura 2.3 Il touch-screen di un iPhone.

GUI

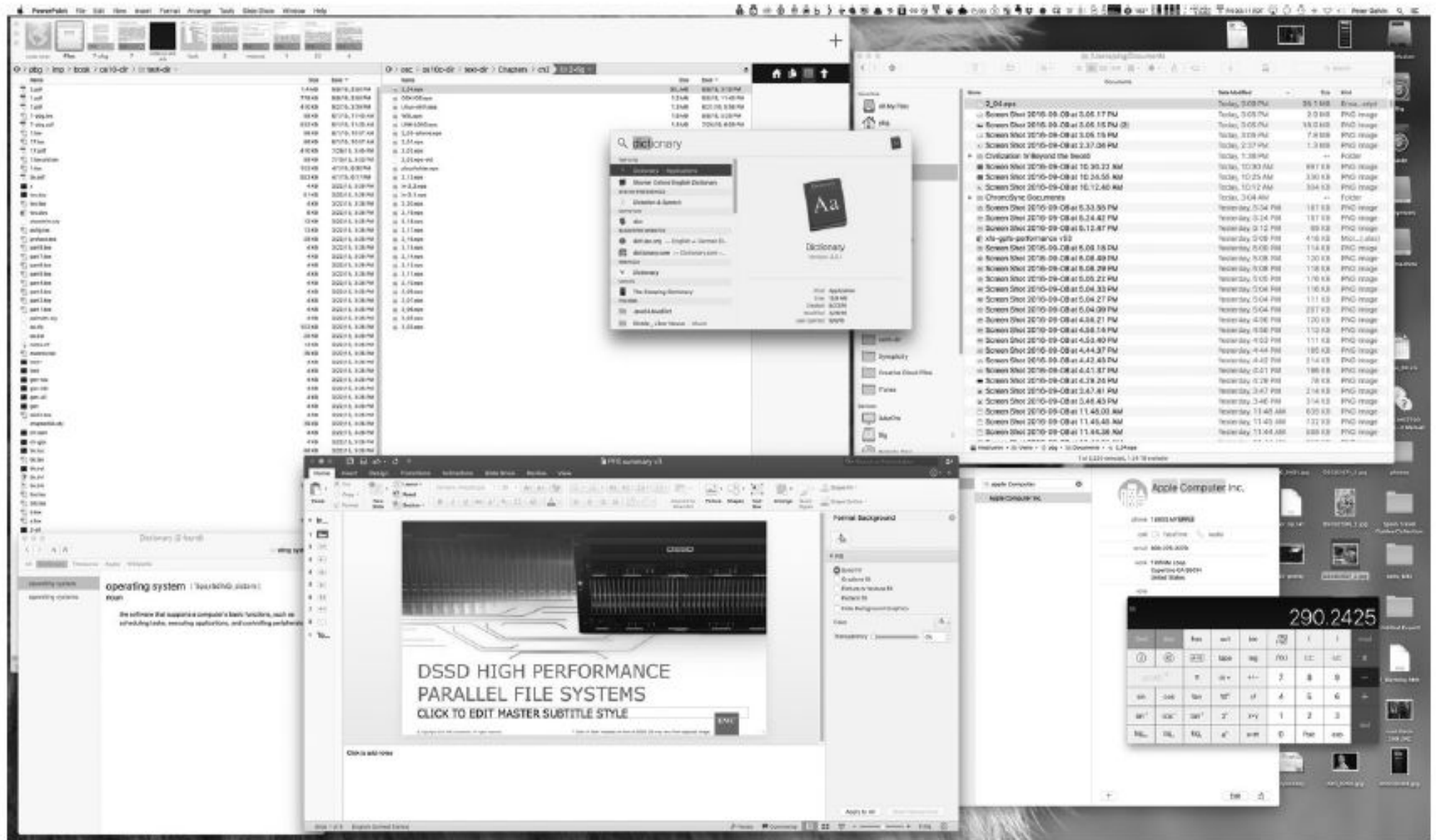


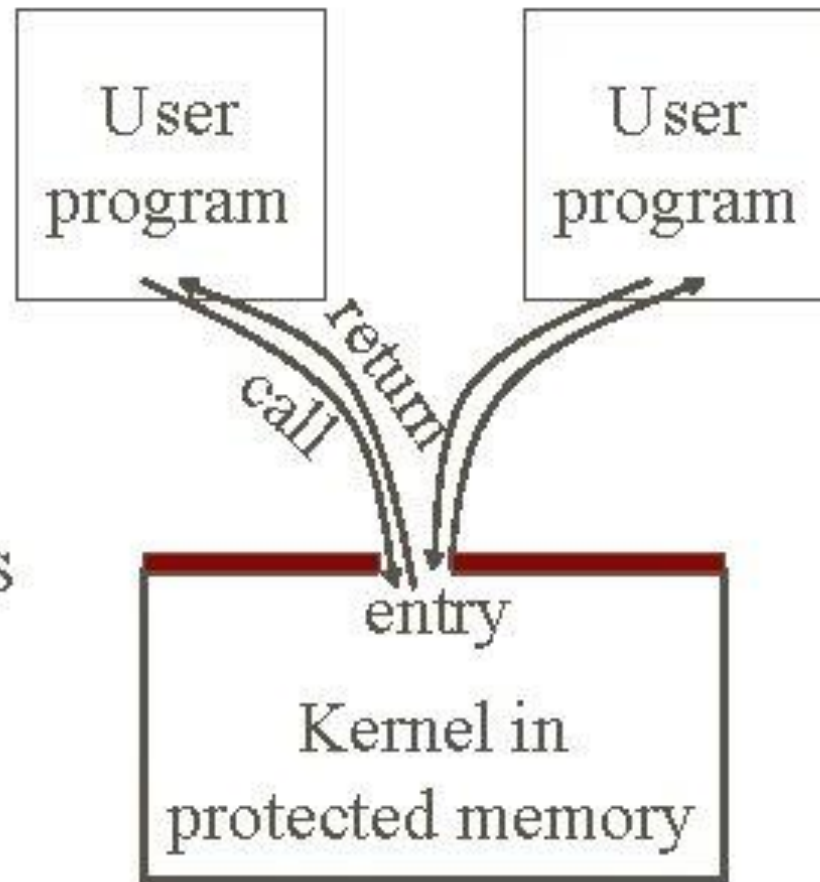
Figura 2.4 Interfaccia grafica di macOS.

System calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs are **Win32 API** for Windows, **POSIX API** for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and **Java API** for the Java virtual machine (JVM)

System call mechanism

- User code can be arbitrary
- User code cannot modify kernel memory
- Makes a system call with parameters
- The call mechanism switches code to kernel mode
- Execute system call
- Return with results



Esempio system call

Le **chiamate di sistema** (*system call*) costituiscono un'interfaccia per i servizi resi disponibili dal sistema operativo.

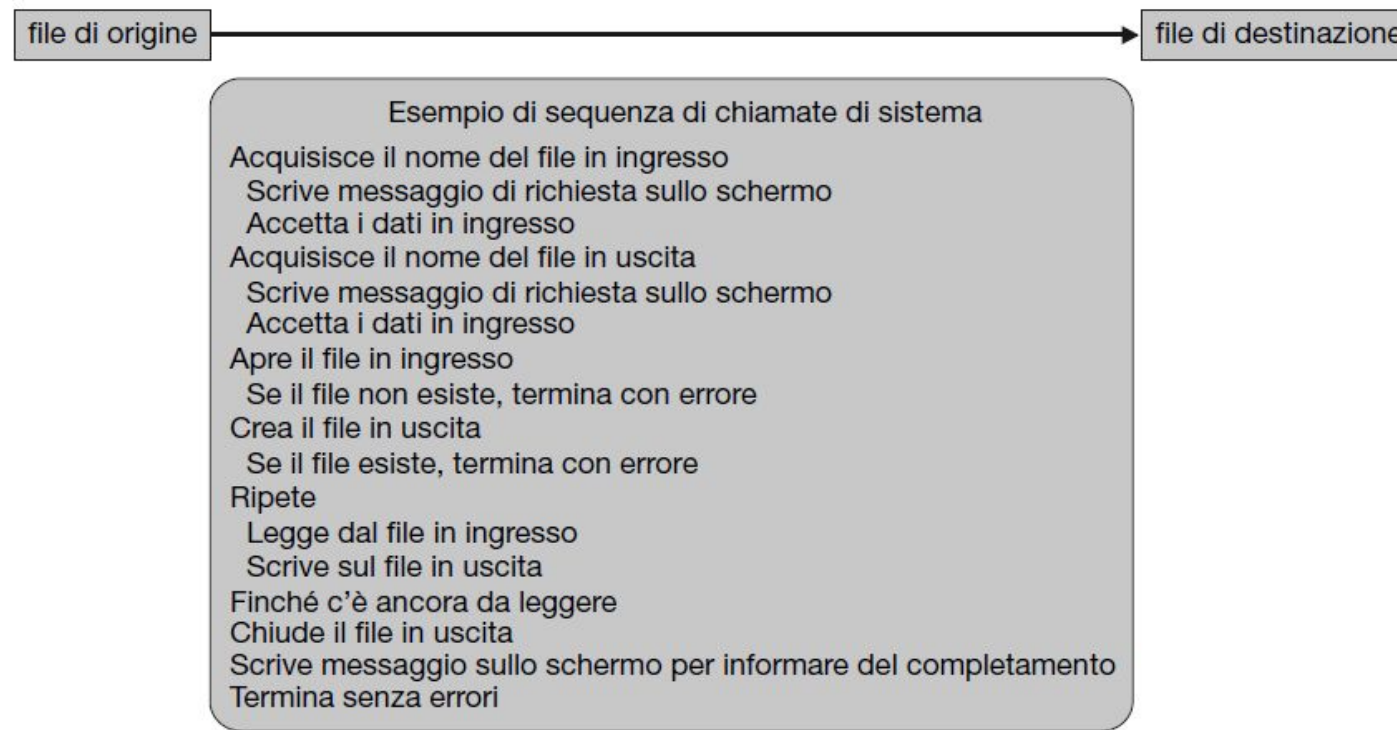


Figura 2.5 Esempio d'uso delle chiamate di sistema.

System call - implementazione

- Typically, a **number associated with each system call**
 - **System-call interface** maintains a **table indexed according to these numbers**
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
 - Just needs to obey API and understand what OS will do as a result call
 - Most details of OS interface **hidden from programmer by API**
 - 4 Managed by **run-time support library** (set of functions built into libraries included with compiler)

API - Interfaccia per la programmazione di applicazioni

API: Specifica un insieme di funzioni a disposizione del programmatore e dettaglia i parametri necessari all'invocazione di queste funzioni, insieme ai valori restituiti.

ESEMPIO DI API STANDARD

Come esempio di API standard consideriamo la funzione `read()` disponibile in Unix e Linux. L'API per questa funzione si può ottenere digitando

```
man read
```

da riga di comando. Una descrizione di questa API è la seguente:

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count)
```

valore restituito nome della funzione parametri

Un programma che utilizza la `read()` deve includere il file `unistd.h` che, tra le altre cose, definisce i tipi di dato `ssize_t` e `size_t`. I parametri passati alla `read()` sono i seguenti:

- `int fd` — il descrittore del file da leggere
- `void *buf` — un buffer nel quale vengono messi i dati letti
- `size_t count` — il massimo numero di byte da leggere e inserire nel buffer

Quando una `read()` è completata con successo viene restituito il numero di byte letti. La `read()` restituisce 0 in caso di fine del file e -1 quando si è verificato un errore.

API, system call e SO

Le relazioni fra una API, l'interfaccia alle chiamate di sistema e il sistema operativo

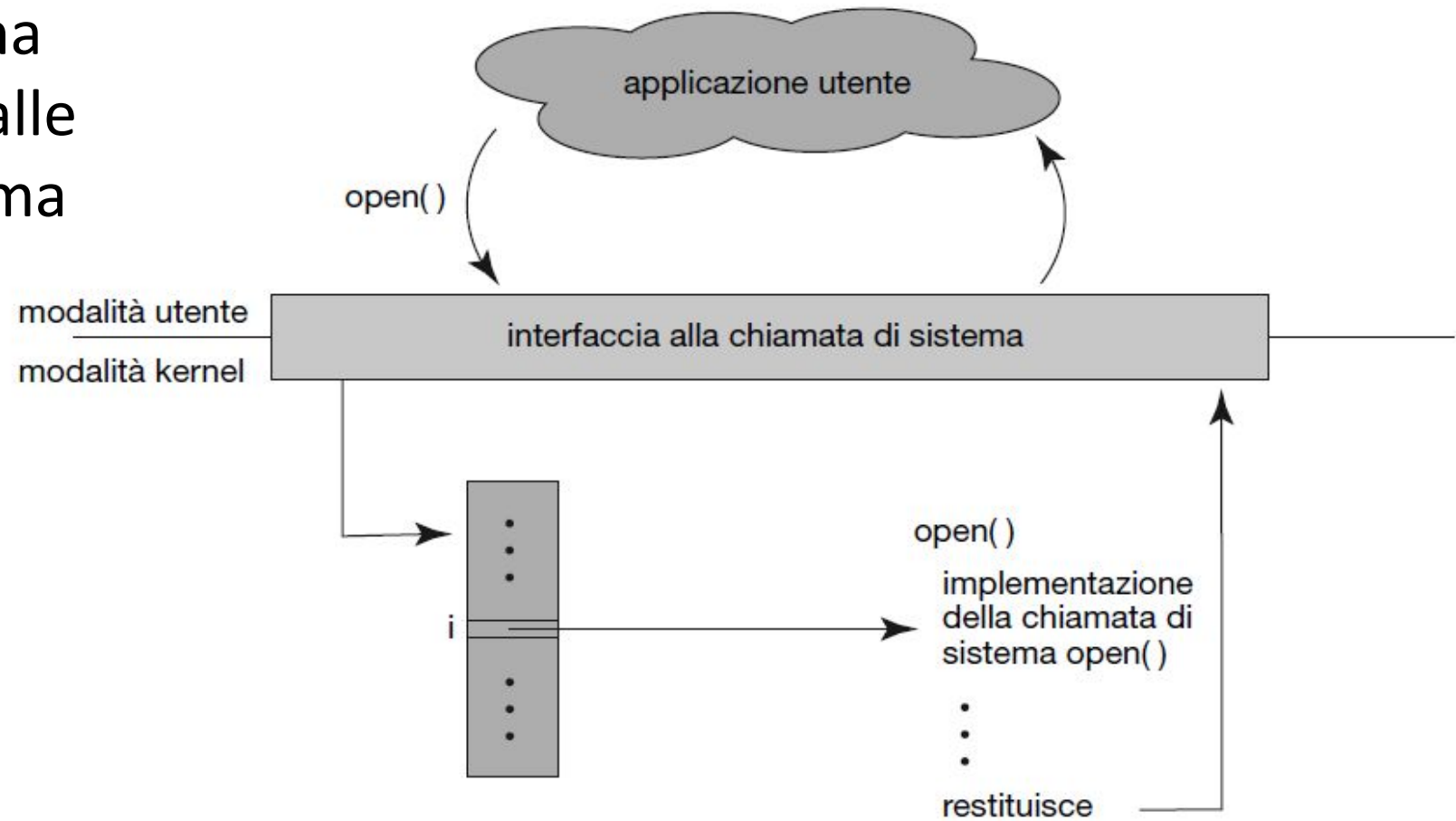


Figura 2.6 Gestione della chiamata di sistema `open()` invocata da un'applicazione utente.

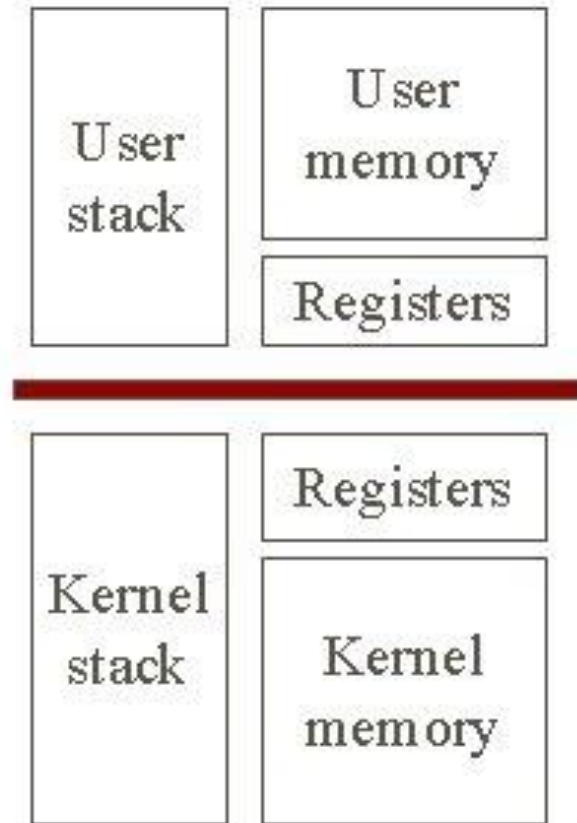
System call – passaggio dei parametri

Per passare **parametri** al sistema operativo si usano tre metodi generali:

1. in *registri*
2. in un *blocco o tabella di memoria*
3. nello *stack* da cui sono prelevati (*pop*) dal sistema operativo

Passaggio dei parametri in registri

- È il metodo più semplice e veloce
- In alcuni casi potrebbero esserci più parametri che registri



Passaggio dei parametri in forma di tabella

Parameters stored in a **block**, or table, in memory, and **address** of block passed as a parameter in a **register**

- This approach taken by Linux and Solaris

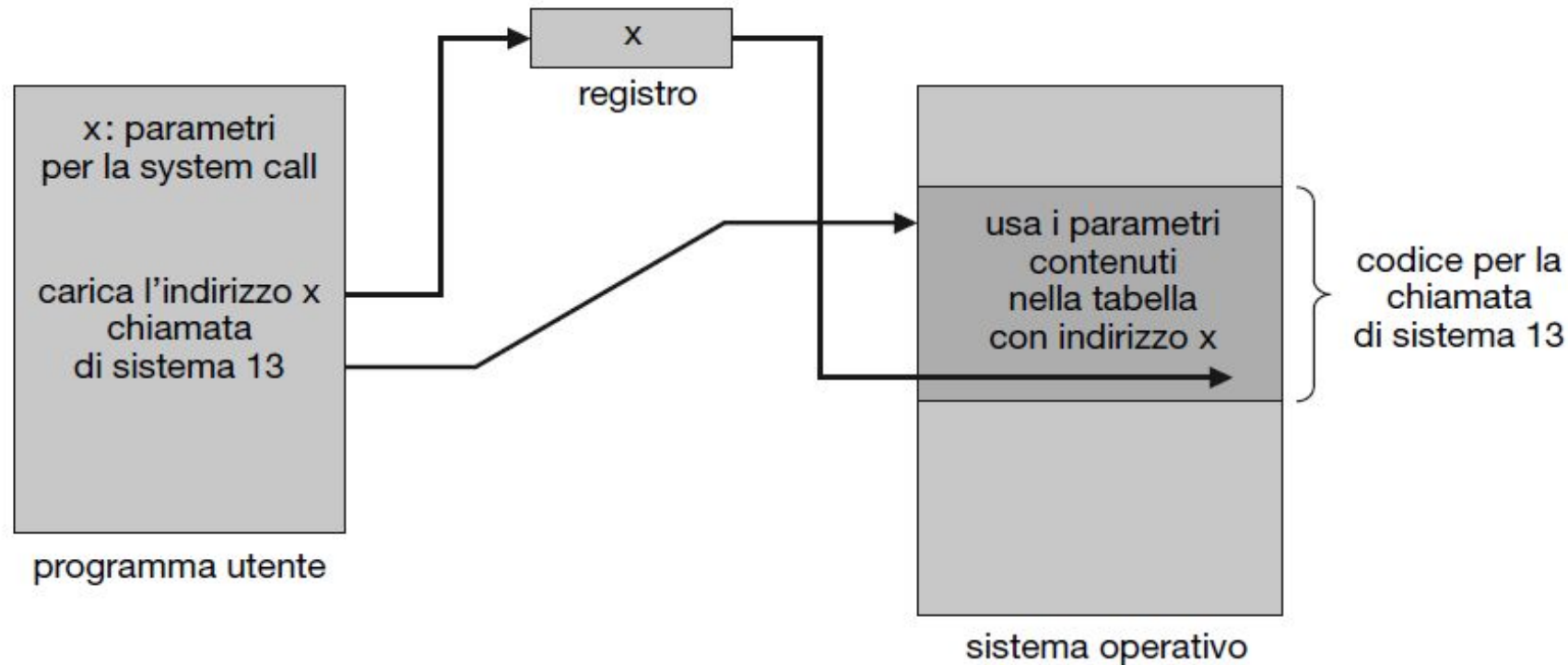
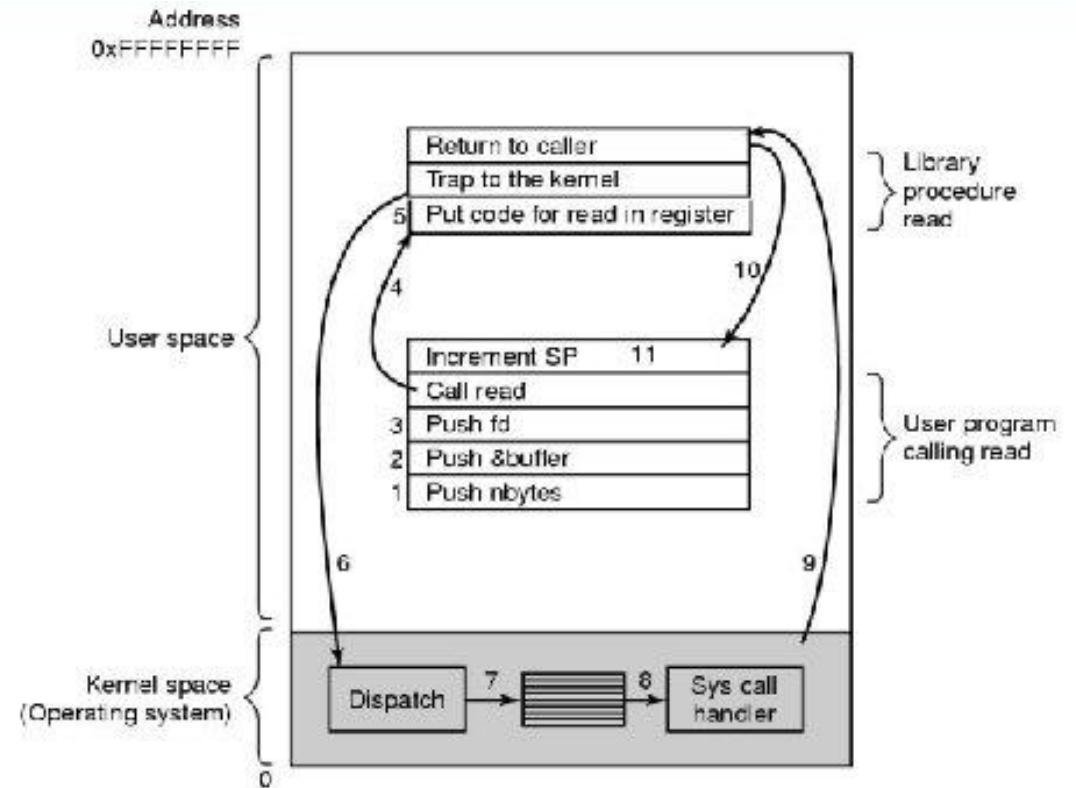


Figura 2.7 Passaggio di parametri in forma di tabella.

Passaggio dei parametri nello stack

- Parameters placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system
- Block and stack methods **do not limit** the number or length of parameters being passed



The 11 steps in making the system call read(fd, buffer, nbytes)

Categorie di chiamate di sistema

Controllo dei
processi

Gestione dei file

Gestione dei
dispositivi

Gestione delle
informazioni,
comunicazioni e
protezione

Categorie di chiamate di sistema

Controllo dei processi

- creazione e arresto di un processo
- caricamento, esecuzione
- terminazione normale e anormale
- esame e impostazione degli attributi di un processo
- attesa per il tempo indicato
- attesa e segnalazione di un evento
- assegnazione e rilascio di memoria

Categorie di chiamate di sistema

Gestione dei file

- creazione e cancellazione di file
- apertura, chiusura
- lettura, scrittura, posizionamento
- esame e impostazione degli attributi di un file

Categorie di chiamate di sistema

Gestione dei dispositivi

- richiesta e rilascio di un dispositivo
- lettura, scrittura, posizionamento
- esame e impostazione degli attributi di un dispositivo
- inserimento logico ed esclusione logica di un dispositivo

Categorie di chiamate di sistema

Gestione delle
informazioni,
comunicazioni e
protezione

- Gestione delle informazioni
 - esame e impostazione dell'ora e della data
 - esame e impostazione dei dati del sistema
 - esame e impostazione degli attributi dei processi, file e dispositivi
- Comunicazione
 - creazione e chiusura di una connessione
 - invio e ricezione di messaggi
 - informazioni sullo stato di un trasferimento
 - inserimento ed esclusione di dispositivi remoti
- Protezione
 - visualizzazione dei permessi di un file
 - impostazione dei permessi di un file

Esempi di chiamate di sistema

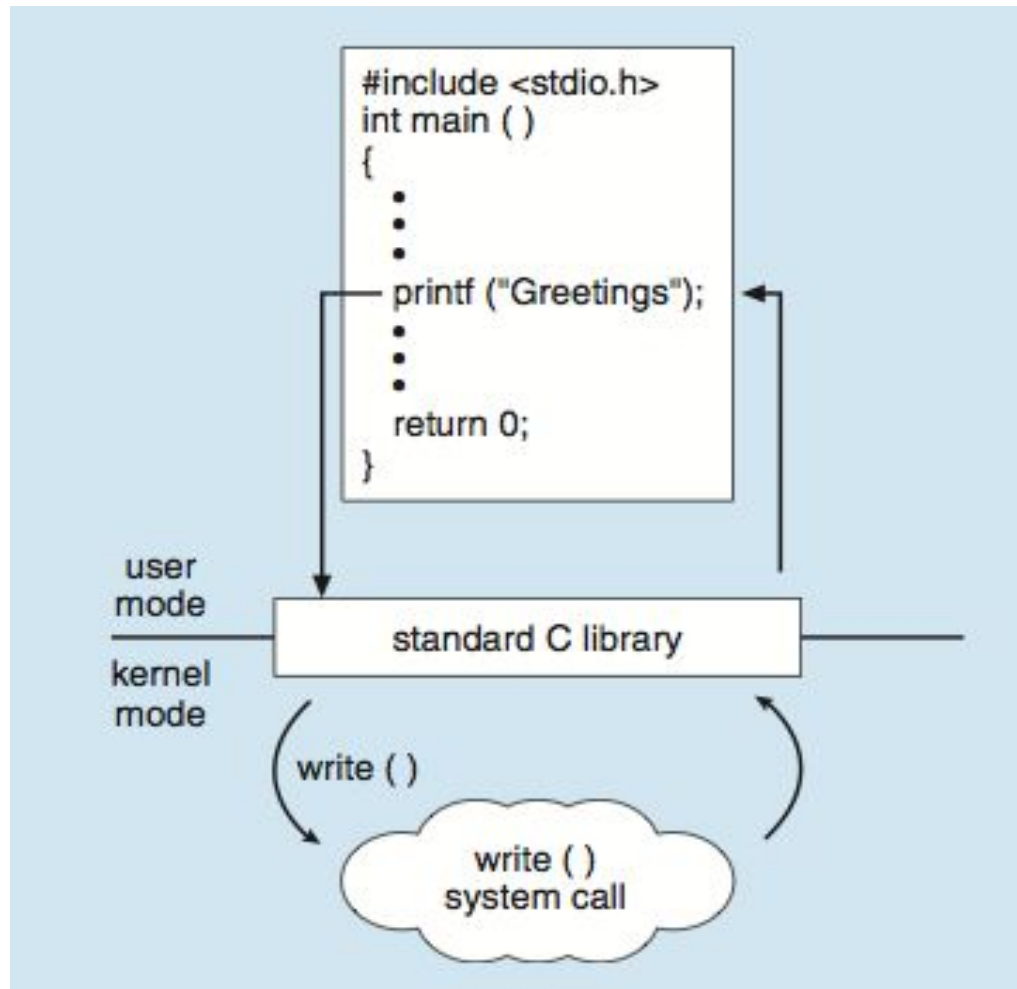


ESEMPIO DI CHIAMATE DI SISTEMA DI WINDOWS E UNIX

	Windows	UNIX
Controllo dei processi	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
Gestione dei file	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Gestione dei dispositivi	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Gestione delle informazioni	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Comunicazione	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()
Protezione	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Standard C Library Example

- C program invoking `printf()` library call, which calls `write()` system call



Sistema monoprogrammato

Arduino è una semplice piattaforma hardware composta da un microcontrollore e da sensori di ingresso che rispondono a diversi eventi

Arduino è un esempio di un sistema **monoprogrammato**



Sistema monoprogrammato

La creazione di un programma per Arduino prevede:

1. la scrittura del programma su un PC
2. Il caricamento del programma compilato (noto come *sketch*) dal PC alla memoria flash di Arduino tramite una connessione USB

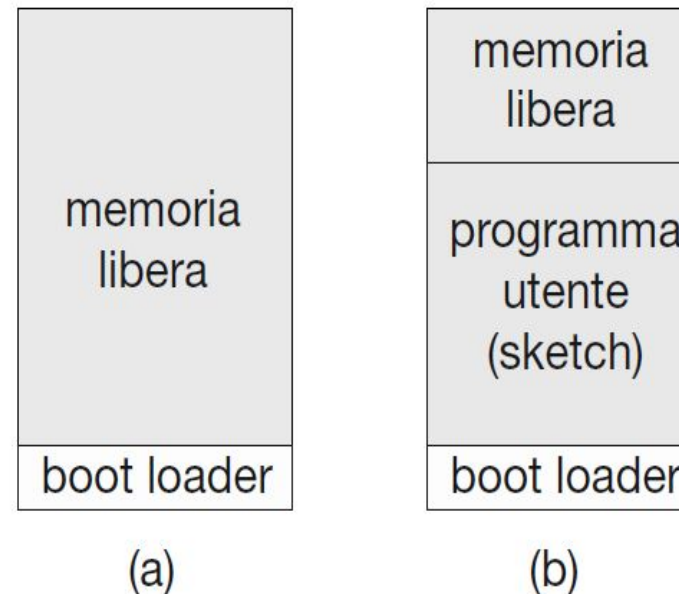


Figura 2.9 Esecuzione in Arduino. (a) All'avviamento del sistema. (b) Durante l'esecuzione di un programma.

Sistema multitasking

FreeBSD (derivato da UNIX Berkeley)
è un esempio di
sistema
multitasking

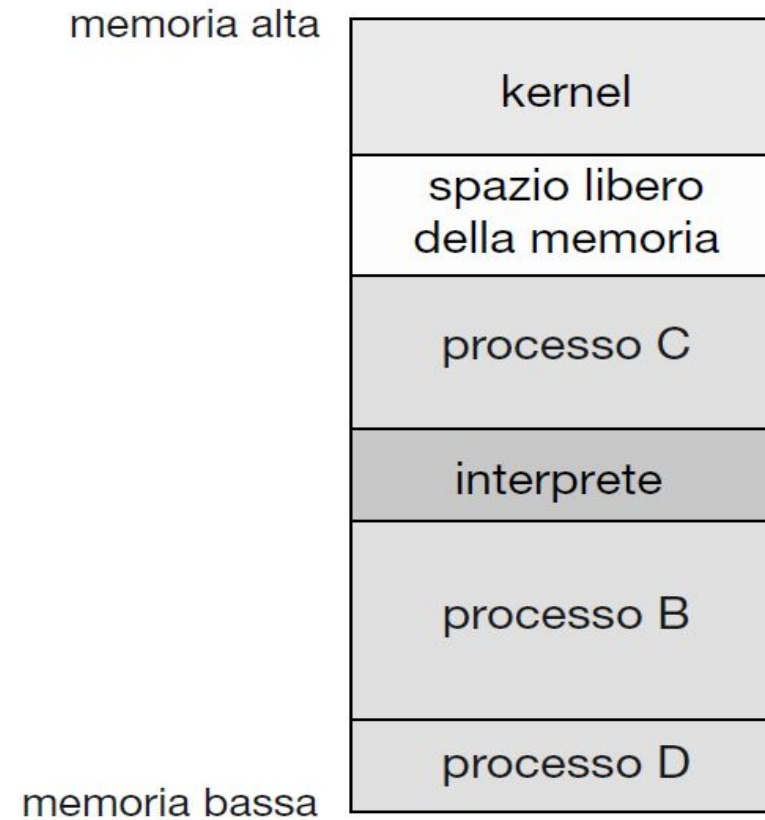


Figura 2.10 Esecuzione di più programmi nel sistema operativo FreeBSD.

Servizi di sistema/utilità di sistema



Linker e loader

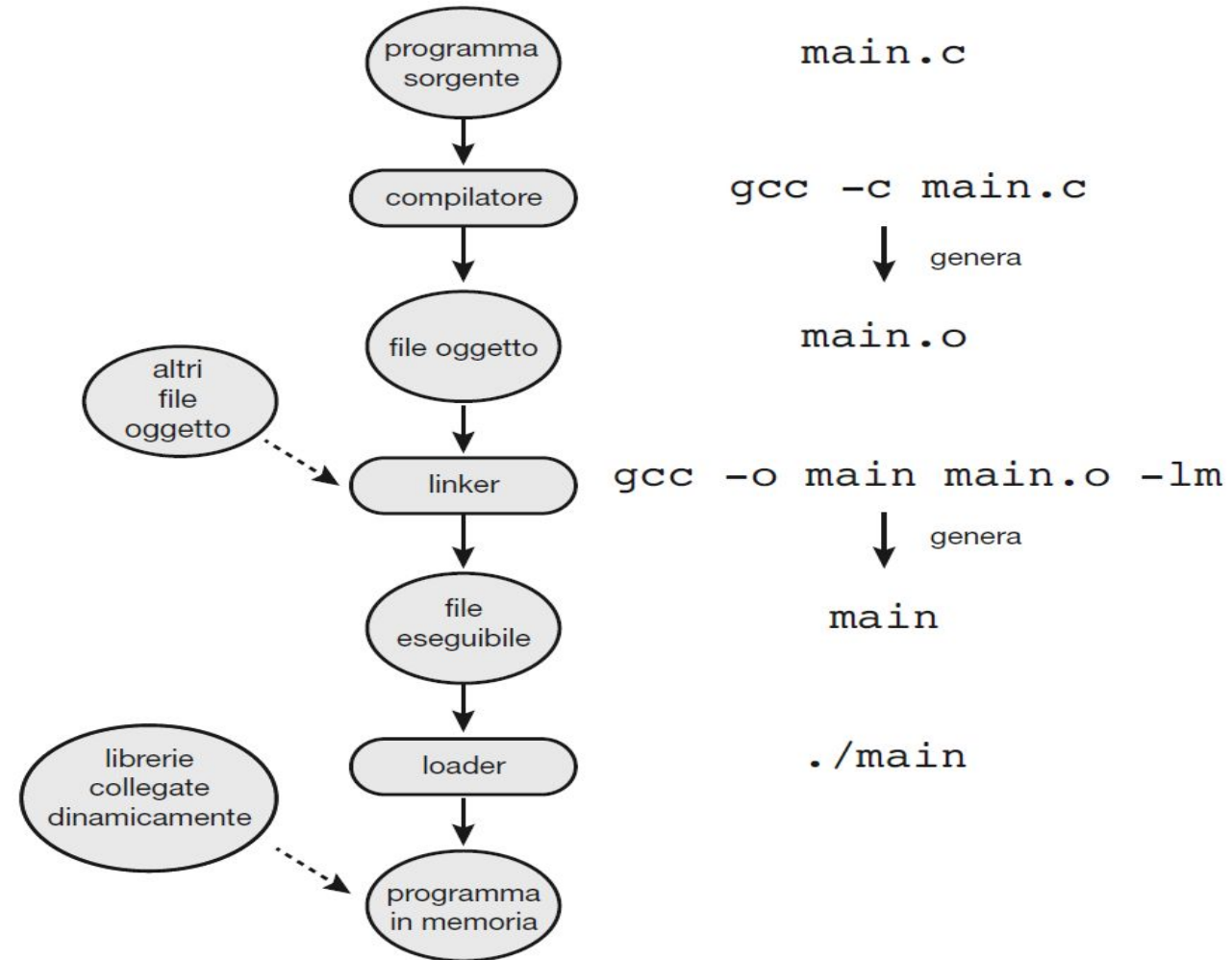


Figura 2.11 Il ruolo di linker e loader.

Perchè le applicazioni dipendono dal sistema operativo

- Fondamentalmente le applicazioni compilate su un sistema operativo *non sono eseguibili* su altri sistemi operativi.
- Ogni sistema operativo fornisce *un insieme univoco di chiamate di sistema*.

Esecuzione su più sistemi operativi

Tre modi per consentire a un'applicazione di essere resa disponibile per l'esecuzione su più sistemi operativi:

1. Può essere scritta in un linguaggio interpretato che ha un interprete disponibile per più sistemi operativi (per esempio Python)
2. Può essere scritta in un linguaggio che utilizza una macchina virtuale contenente l'applicazione in esecuzione (per esempio Java)
3. Lo sviluppatore di un'applicazione può utilizzare un linguaggio o un'API standard in cui il compilatore genera binari nel linguaggio specifico del sistema operativo e della macchina (per esempio C/C++)

Struttura del sistema operativo

Struttura monolitica

Un sistema monolitico viene anche chiamato **sistema strettamente accoppiato** (*tightly coupled*)

In alternativa, è possibile progettare un **sistema debolmente accoppiato** (*loosely coupled*)

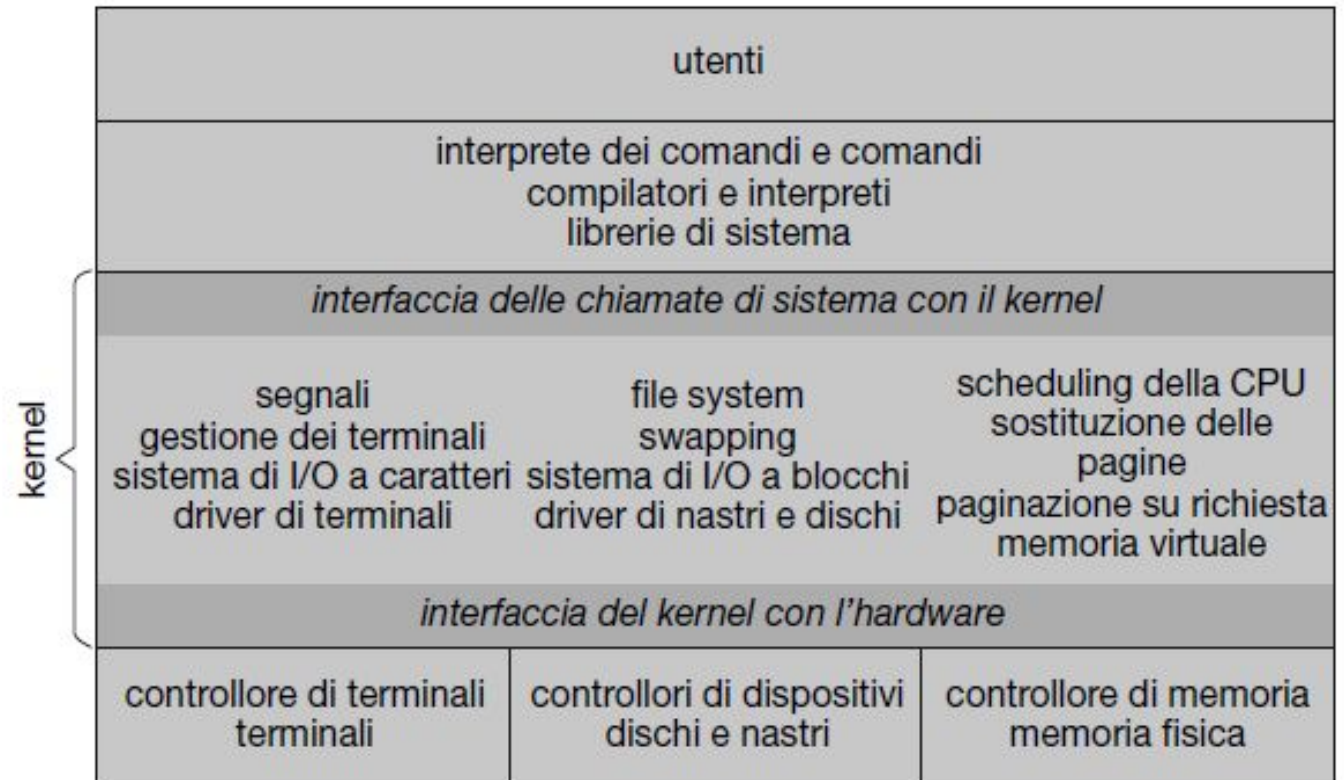


Figura 2.12 Struttura del sistema UNIX.

Struttura del sistema Linux

Il sistema operativo **Linux** è basato su UNIX ed è strutturato in modo simile

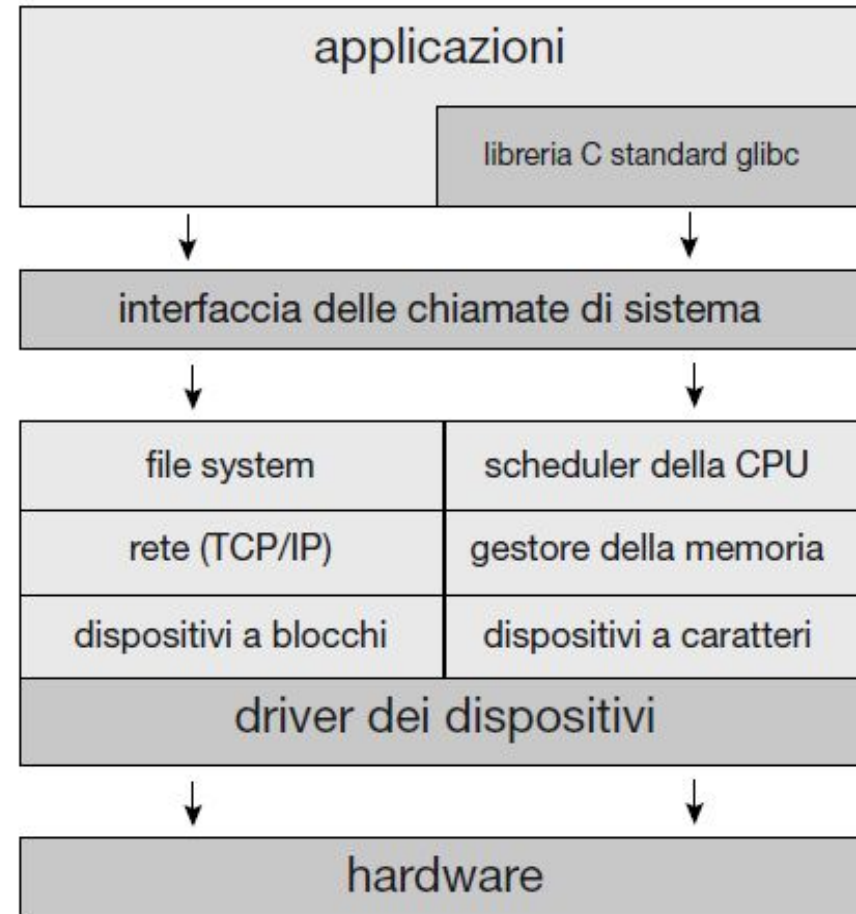


Figura 2.13 Struttura del sistema Linux.

Approccio stratificato

Per realizzare un **sistema debolmente accoppiato** (*loosely coupled*) è possibile suddividere le funzioni del kernel in moduli

Vi sono molti modi per rendere modulare un sistema operativo. Uno di essi è l'**approccio stratificato**.

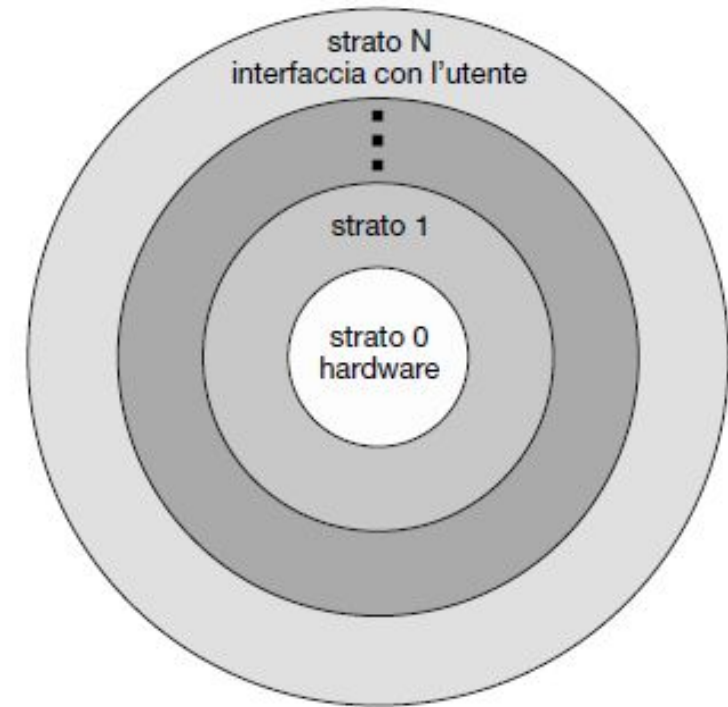


Figura 2.14 Struttura a strati di un sistema operativo.

Microkernel

Verso la metà degli anni '80 fu realizzato un sistema operativo, **Mach**, con il kernel strutturato in moduli secondo il cosiddetto **orientamento a microkernel**.

Scopo principale del microkernel → fornire funzioni di comunicazione tra i programmi client e i vari servizi, anch'essi in esecuzione nello spazio utente.

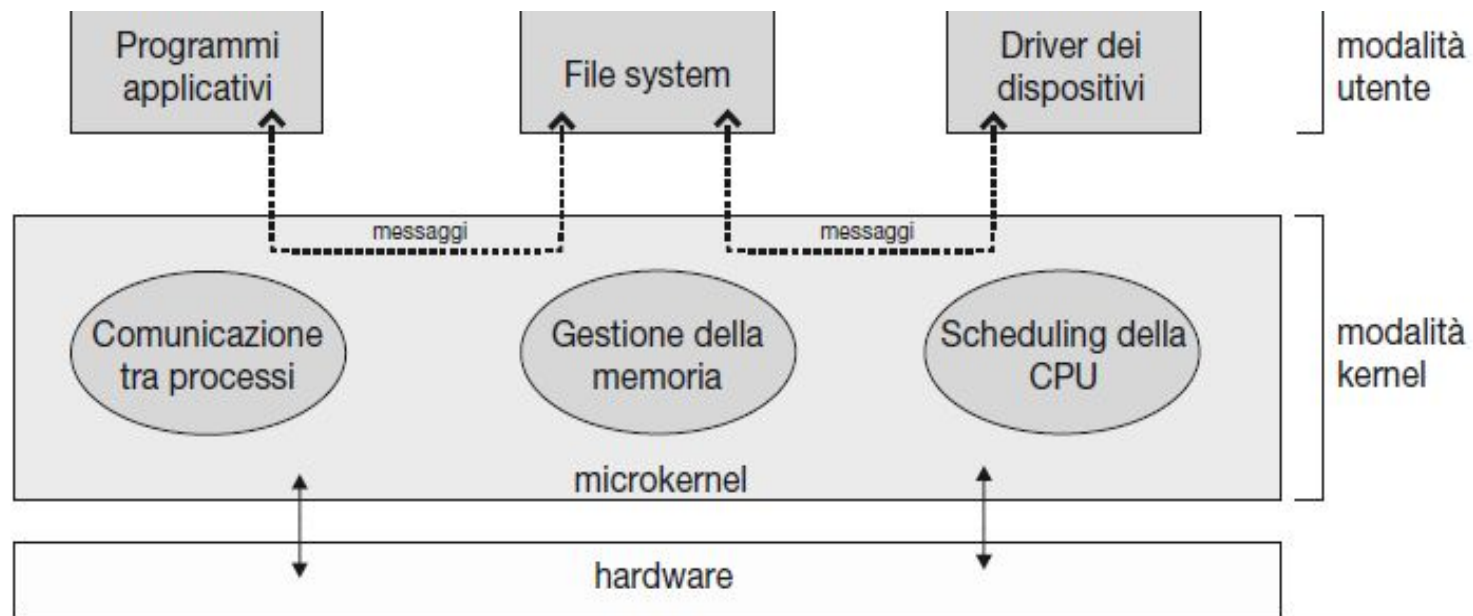


Figura 2.15 Architettura tipica di un microkernel.

macOS e iOS

Il sistema operativo **macOS** di Apple è progettato per funzionare principalmente su *computer desktop e laptop*, mentre **iOS** è un sistema operativo mobile progettato per *iPhone e iPad*.

- Strato dell'interfaccia utente (***user experience***)
- Strato degli **ambienti applicativi**
- **Ambienti di base (core)**
- **Ambiente kernel** (noto anche come **Darwin**)

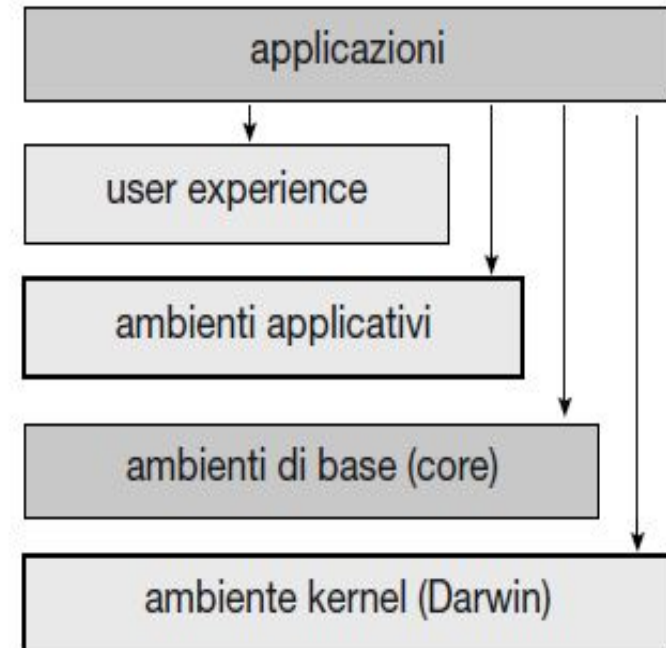


Figura 2.16 Architettura dei sistemi operativi macOS e iOS di Apple.

Darwin

Darwin è un sistema a strati costituito principalmente dal microkernel Mach e dal kernel BSD UNIX.

Apple ha rilasciato il sistema operativo **Darwin** come **open-source**

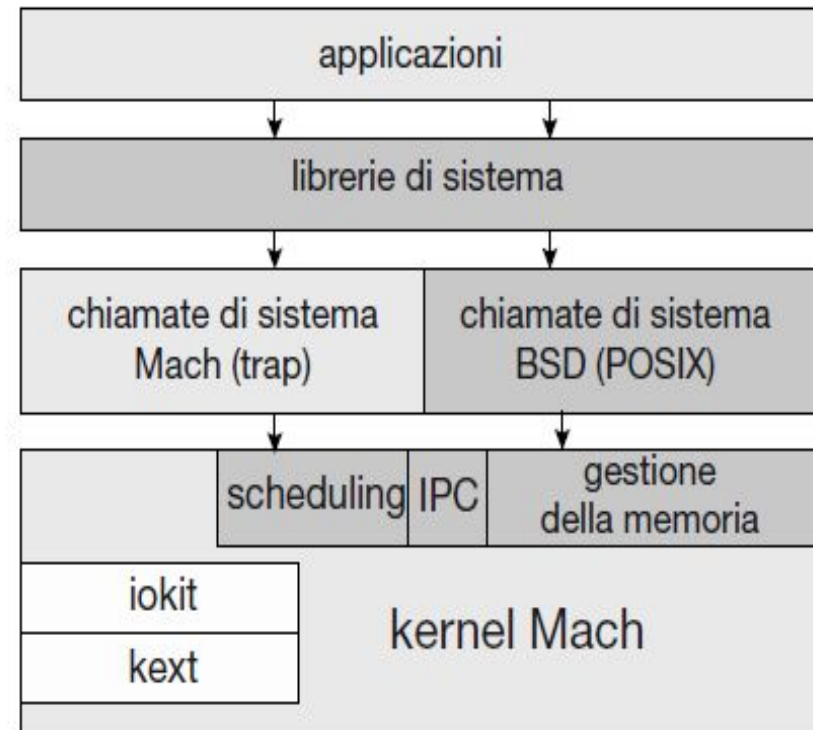


Figura 2.17 La struttura di Darwin.

Android

Mentre iOS è progettato per funzionare su dispositivi mobili di Apple ed è un software proprietario, **(Google) Android** gira su una varietà di piattaforme mobili ed è **open-source**.

Poiché Android può essere eseguito su un numero quasi illimitato di dispositivi, Google ha scelto di astrarre l'hardware attraverso uno strato di astrazione hardware detto **HAL** (hardware abstraction layer).

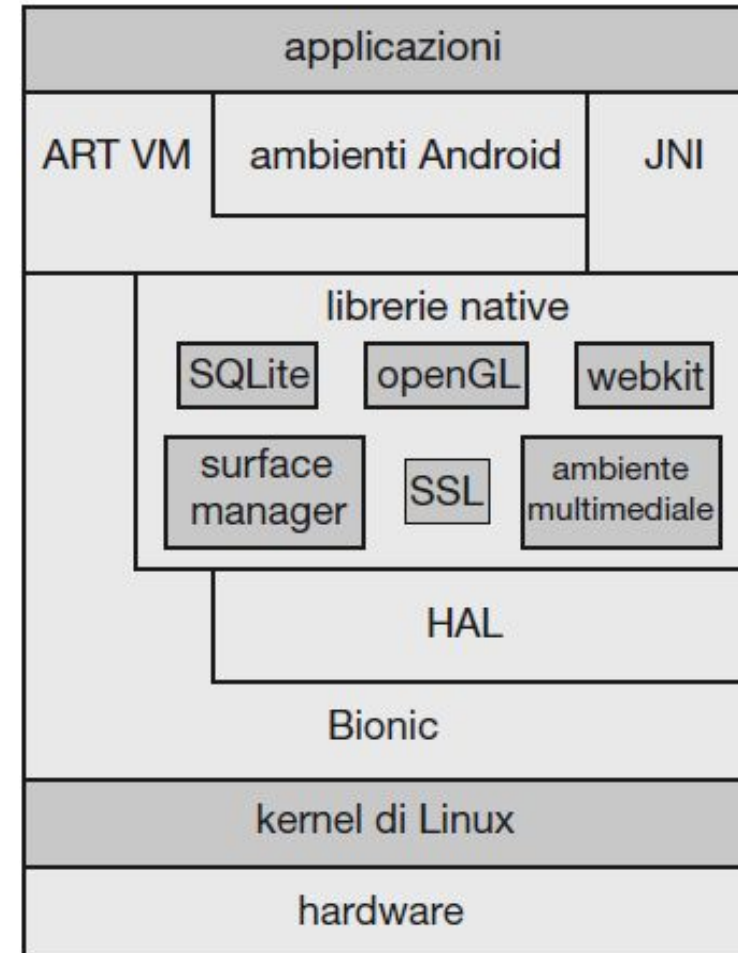


Figura 2.18 Architettura di Google Android.

Generare e avviare un OS

Scrivere il codice sorgente del sistema operativo (o ottenere il codice sorgente già scritto)

Configurare il sistema operativo per il sistema su cui verrà eseguito

Compilare il sistema operativo

Installare il sistema operativo

Avviare il computer e il nuovo sistema operativo

Avvio del sistema operativo

Il processo di avvio di un computer, caricando il kernel del sistema operativo, è noto come *boot*.



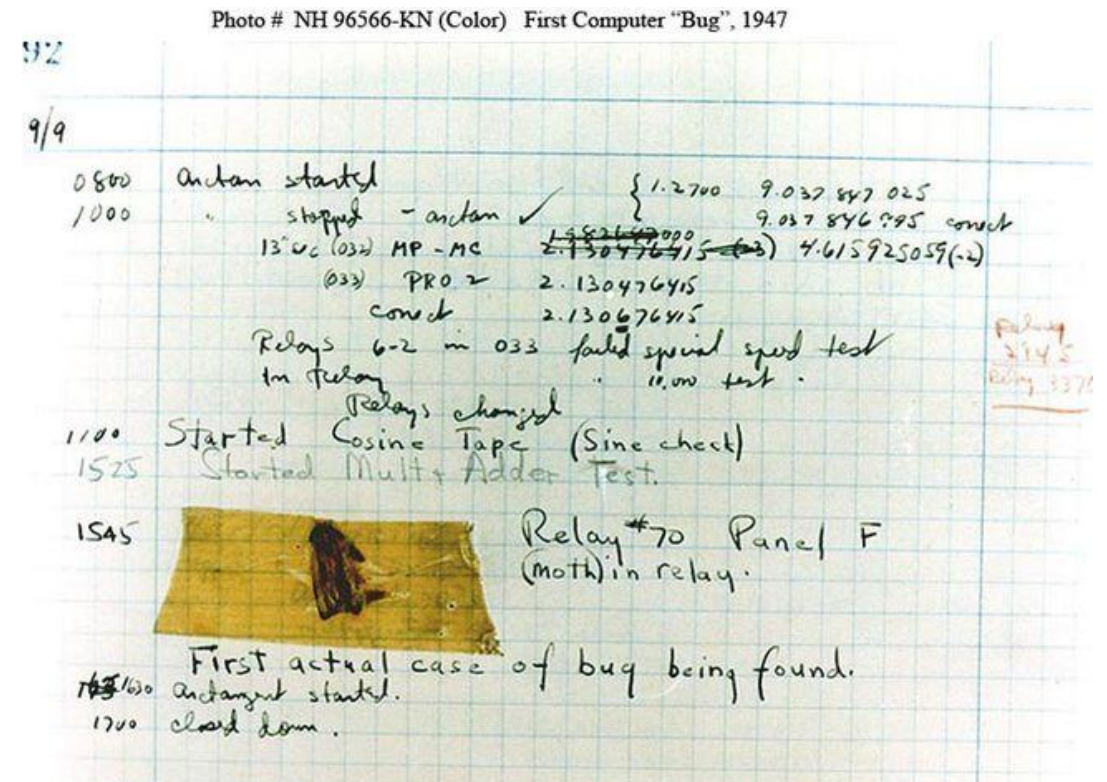
System boot

- When power initialized on system, execution starts at a fixed memory location
 - Firmware ROM used to hold initial boot code
- Operating system must be made available to hardware so hardware can start it
 - Small piece of code – **bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
 - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**

Debugging

Debugging → l'attività di individuare e risolvere errori hardware e software nel sistema, i cosiddetti **bachi (bug)**, ma anche

- **regolazione delle prestazioni (performance tuning)**
- **colli di bottiglia (bottleneck)** del sistema



<https://www.nationalgeographic.org/thisday/sep9/worlds-first-computer-bug/>

Debugging

Se il *debugging di processi* a livello utente è una sfida, **a livello del kernel** del sistema operativo è un'attività ancora più difficile a causa della dimensione e della complessità del kernel, del suo controllo dell'hardware e della mancanza di strumenti per eseguire il debugging a livello utente.



Un guasto nel kernel viene chiamato **crash**



LA LEGGE DI KERNIGHAN

“Il debugging è due volte più complesso rispetto alla stesura del codice. Di conseguenza, chi scrive il codice nella maniera più intelligente possibile non è, per definizione, abbastanza intelligente per eseguirne il debugging.”

Prestazioni

Tracing o tracciamento

gli strumenti di tracing raccolgono i dati relativi a uno specifico evento

Contatori

contano per esempio il numero di chiamate di sistema effettuate o il numero di operazioni eseguite su un dispositivo o su un disco di rete

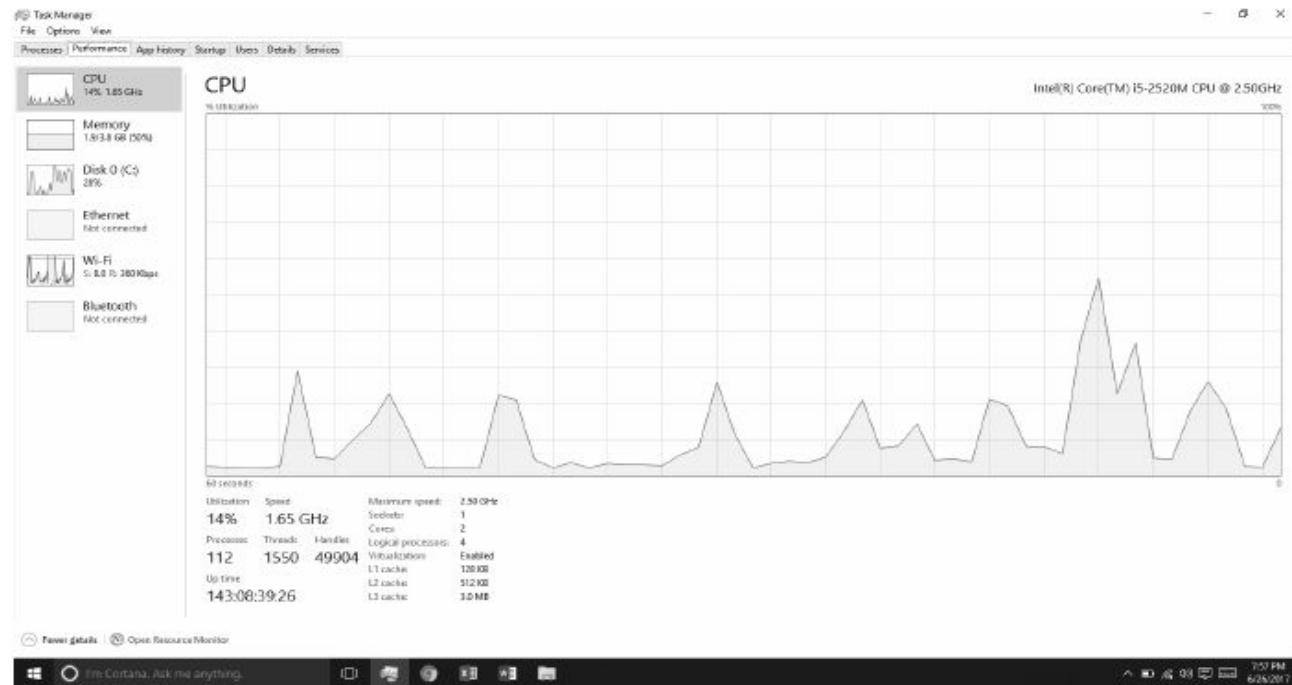


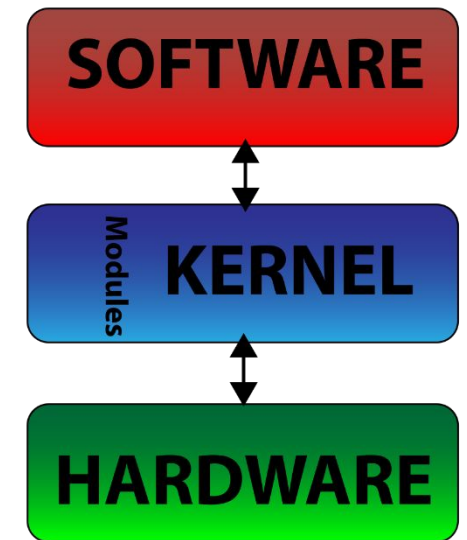
Figura 2.19 Il task manager di Windows 10.



**UNIVERSITÀ DEGLI STUDI
DELLA BASILICATA**

Corso di Sistemi Operativi

Strutture dei sistemi operativi



Docente:
**Domenico Daniele
Bloisi**

