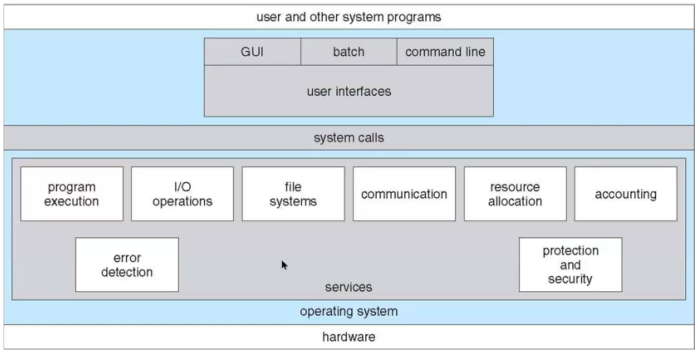
1.0 SERVIZI PER L'UTENTE GENERICO

La visione della macchina che abbiamo, e che l'utente generico osserva, è la seguente:



Alla base di tutto c'è l'hardware (CPU, parti meccaniche, etc…). Al di sopra di queste parti meccaniche è presente il sistema operativo, il quale mette a disposizione le risorse che hanno l'obiettivo di far eseguire un dato programma all'utente (comprende tutta la parte che nell'immagine viene vista in azzurro). In particolare, i servizi che il sistema operativo mette a disposizione sono:

- **eseguire programmi**: il SO deve caricare in memoria ed eseguire i programmi e garantire la terminazione dell'esecuzione in modo normale o anormale (eventualmente indicandone l'errore).
- Come noi sappiamo, i programmi che scriviamo risiedono nell'Hard Disk, però, ogniqualvolta un esecutivo deve essere eseguito, bisogna caricarlo in memoria principale perché l'unico modo che ci permette di eseguire un programma, è che esso risieda in memoria principale, in quanto la CPU comunica solo con essa. Una volta che il programma è in memoria principale, la CPU esegue riga dopo riga il programma.
- **effettuare operazioni di input/output**: quando stiamo eseguendo un programma in memoria principale, il SO deve garantire che in un momento qualsiasi del programma si necessiti di un file (che naturalmente risiede sul disco), anche quest'ultimo deve essere portato all'interno del pezzo di memoria principale che è stato assegnato precedentemente al programma.
- **gestire il file system**: significa che il SO, non solo deve mettere a disposizione la possibilità di effettuare input/output su un file, ma anche creare file, creare una directory ed all'interno inserire un file, sapere quali sono i permessi di accesso di un file. Il SO deve garantire questi servizi.
- **comunicazione tra processi**: il SO deve garantire che più processi possono scambiarsi informazioni, sullo stesso computer o tra computer su una rete.
- **rilevazione dell'errore**: il SO deve essere sempre informato sugli errori di tipo hardware o software.
- **allocazione risorse**: il SO deve garantire il corretto passaggio dell'esecutivo dal disco alla memoria principale.
- **contabilizzazione**: contare quanti utenti ci sono, e quanta memoria stanno utilizzando, utile per la gestione totale del sistema.
- **protezione e sicurezza**;

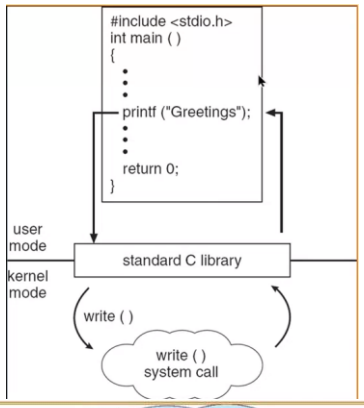
Tutti questi servizi che sono stati elencati, vengono realizzati mediante le **system calls**, che sono l'unica modalità che ci consente di andare in kernel mode ed eseguire fisicamente ciò di cui abbiamo bisogno**.**

Al disopra del sistema operativo ci sono le interfacce utente che ci consentono di comunicare col SO e che fondamentalmente possono essere di 2 tipi: **interfaccia CLI** (Command Line Interface), in cui si comunica al sistema operativo attraverso comandi. Un'operazione del genere è garantita dalla shell. Ci sono poi le **interfaccia grafiche** (GUI) che mettono a disposizione ad esempio, un desktop amichevole, i comandi sono forniti tramite mouse, tastiera e monitor.

1.1 SERVIZI PER IL PROGRAMMATORE: SYSTEM CALL

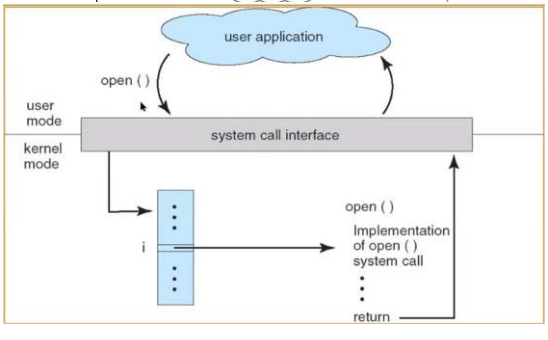
Quando l'utente chiede al SO di voler effettuare un servizio, questo viene realizzato attraverso le system call. Tipicamente le system call sono delle interfacce che il SO mette a disposizione del programmatore in maniera tale che una qualsiasi richiesta venga fatta al sistema operativo, viene gestita mediante system call. Usando le system call, si usano delle funzioni di libreria (API) che sono più facili da gestire nella pratica e che nascondono i dettagli implementativi che la system call utilizza effettivamente.

Ad esempio, la printf è una funzione di libreria →



Immaginiamo di avere questa porzione di codice, e ad un certo punto viene invocata la printf. Quando viene invocata la printf, il SO va a prendere l'effettivo codice della printf scritto da qualche parte, lo porta in memoria principale e inizia ad interpretarlo. Ad un certo punto, nel codice della printf appare la funzione *write()*, che è una system call, e di conseguenza di passa in kernel mode e ciò permetterà di stampare a schermo l'informazione che l'utente richiede. La write() ha esattamente l'aspetto di una normale funzione C, solo che essendo una system call *non esiste una porzione di codice che implementa tale funzione*, infatti le system call sono qualcosa di fisico che il kernel utilizza per poter realizzare quello che è stato chiesto.

Poiché non esiste una porzione di codice che implementa le system call, chiaramente esiste un altro tipo di gestione che viene adesso descritto →



Il SO associa ad ogni system call un numeretto. Ogni qualvolta si chiede al SO di mandare una system call, nell'esempio la funzione open() e per esempio sa che la funzione ha numero 10, nel kernel del SO c'è una sorta di tabella al cui interno ci sono i numeri contenenti ciascuna system call, per cui va nell'entry 10 ed è presente un link ad una sequenza di istruzioni che devono eseguite per l'implementazione della system call open(). Di conseguenza noi non avremo mai a disposizione l'implementazione della system call open, ma sappiamo naturalmente ciò che fa.