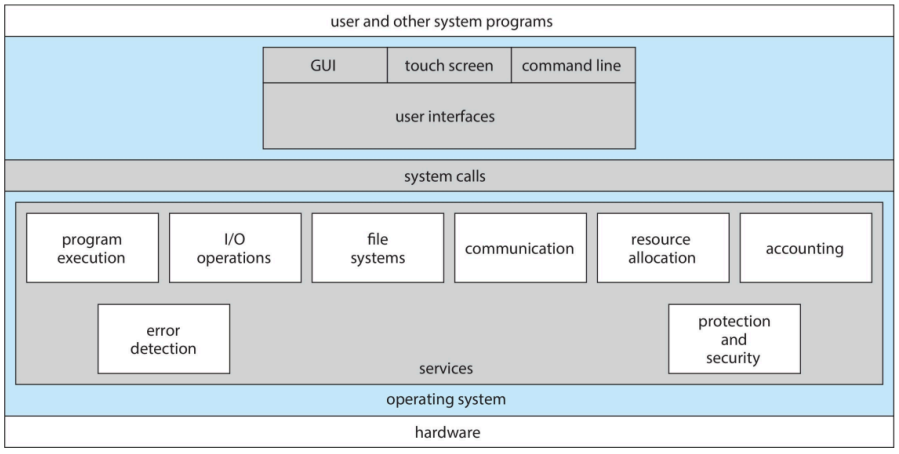
Lezione 5 (24-02-25)

Operating-System Services

Un SO offre un ambiente in cui eseguire i programmi e fornire servizi, questi cambiano a seconda del sistema utilizzato, ma ci sono alcuni servizi comuni:

- **interfaccia con l'utente**: può essere grafica (GUI), a riga di comando (CLI) o touch-screen nei dispositivi mobili.
- **esecuzione di un programma**: Il SO carica ed esegue programmi, permettendone la terminazione normale o anomala.
- **operazioni di I/O**: gestisce le operazioni di I/O evitando che l'utente acceda direttamente ai dispositivi.
- **comunicazioni**: i processi possono scambiare informazioni tramite memoria condivisa o scambio di messaggi, sia localmente che su reti.
- **rilevamento di errori**: il SO deve essere in grado di rilevare e correggere eventuali errori, deve saper intraprendere l'azione corretta per ogni tipo d'errore. Ha diverse scelte possibili, come:
  - l'arresto del sistema;
  - terminare il processo che causa errore;
  - restituire un codice d'errore a un processo che individua e corregge l'errore.

  Altri servizi non direttamente visibili all'utente includono:
- **allocazione delle risorse**: gestione equa delle risorse tra utenti e processi.
- **logging**: tracciamento dell'uso del sistema e delle risorse impiegate.
- **protezione e sicurezza**: controllo dell'accesso alle risorse e protezione da minacce esterne.



Command Line Interpreter

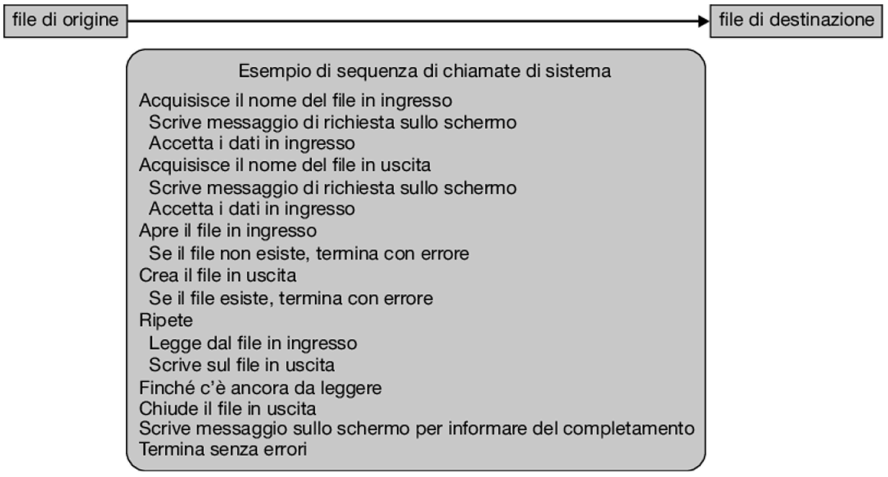
L'interprete dei comandi è un programma speciale, che si avvia all'avvio di un job o non appena un utente effettua il logon. La sua funzione principale è quella di *raccogliere ed eseguire* il successivo comando impartito dall'utente, la maggior parte di questi solitamente riguarda la gestione dei file. I comandi si possono implementare in due modi:

- l'interprete dei comandi contiene il codice per l'esecuzione del comando (es. il comando di cancellazione di un file causa un salto dall'interprete a una sezione del suo stesso codice che imposta i parametri e invoca le chiamate di sistema);

- la maggior parte dei comandi sono implementati tramite programmi di sistema, quindi l'interprete "non capisce" il significato del comando, ma utilizza il nome per identificare un file da caricare in memoria per l'esecuzione. In questo modo si possono aggiungere nuovi comandi al sistema.

System Calls

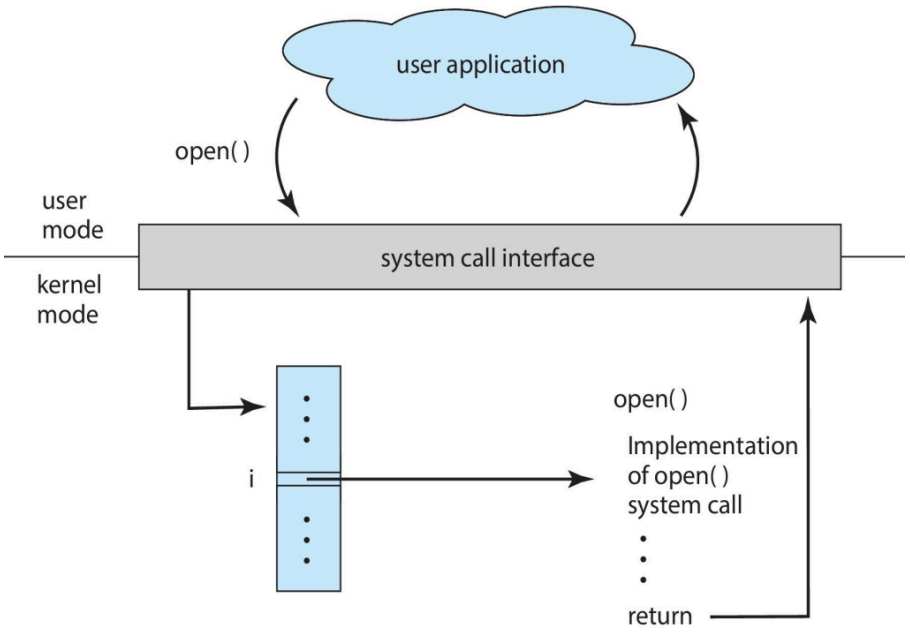
Costituiscono l'interfaccia tra il SO e le applicazioni, sono generalmente disponibili sottoforma di routine scritte in *c* o *c++*, mentre per alcuni compiti di basso livello potrebbe essere necessario utilizzare *assembly*.



Si preferisce utilizzare API(*application programming interface*) per estrarre le system call, rendendo i programmi più portabili e semplici. L'API specifica un insieme di funzioni a disposizione del programmatore e specifica i parametri necessari all'invocazione di queste funzioni, assieme ai valori restituiti.
(es. interfacce più utilizzate: API Windows, API Java, API posix → unix, Linux, macos).
L'accesso all'API si ottiene tramite una libreria di codice fornita dal SO.

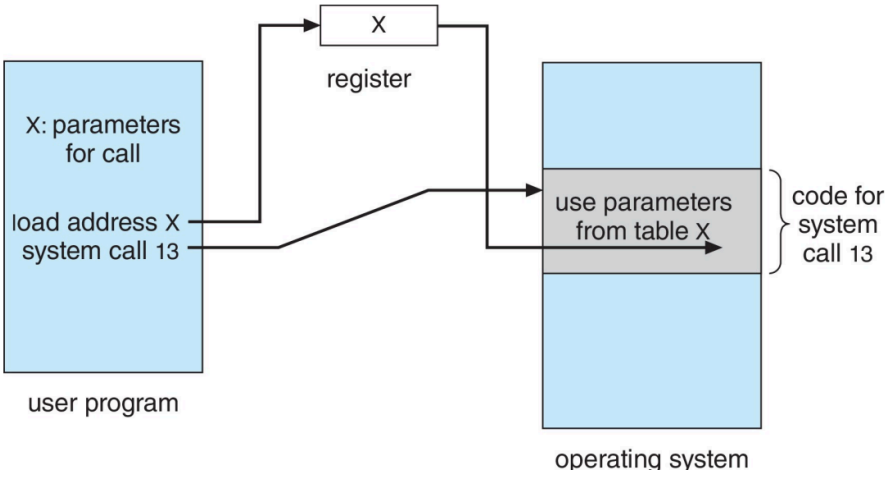
Altro fattore importante nella gestione delle chiamate di sistema è l'RTE (run-time environment), suite completa di programmi necessari per eseguire applicazioni scritte in un determinato linguaggio; include:

- compilatori o interpreti;
- librerie;
- loader;
- ecc..
  Fornisce un'interfaccia alle chiamate di sistema (system call interface) che collega il linguaggio alle system call disponibili. L'interfaccia intercetta le chiamate a funzione nella API e invoca le relative system call.
  Ogni chiamata di sistema è codificata da un numero.
  L'interfaccia alle chiamate di sistema mantiene una tabella delle chiamate e invoca di volta in volta la chiamata richiesta, che risiede nel kernel del sistema, restituendo al chiamante lo stato della chiamata.



Spesso le system call richiedono informazioni aggiuntive all'identità della chiamata, il tipo e la quantità delle informazioni varia a seconda del SO e alla specifica chiamata.
Per passare parametri al SO si usano tre metodi:

- **passaggio dei parametri nei registri**, in alcuni casi può succedere di avere più parametri che registri;
- **parametri memorizzati in un *blocco*** o tabella di memoria e si passa l'indirizzo del blocco, come parametro, in un registro;
- **si utilizza uno stack**, ovvero, un programma colloca (push) i parametri nello stack, da cui sono poi prelevati (pop) dal SO.
  I metodi del blocco e dello stack non limitano il numero o la lunghezza dei parametri da passare.

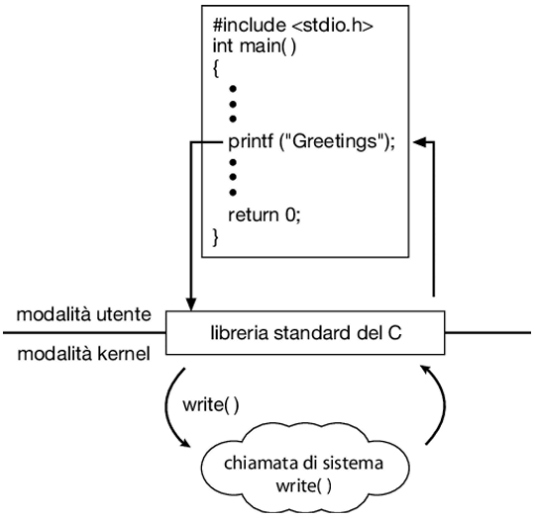


Types of System Calls

Le chiamate di sistema sono classificabili in:

- **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.



- **gestione dei file**:
  - creazione e cancellazione dei file;
  - apertura e chiusura;
  - lettura, scrittura e posizionamento;
  - esame e impostazione degli attributi di un file.
- **gestione dei dispositivi**:
  - richiesta e rilascio di un dispositivo;
  - lettura, scrittura e riposizionamento;
  - esame e impostazione degli attributi di un dispositivo;
  - inserimento logico ed esclusione logica di un dispositivo.
- **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 dei 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.

System Services

L'insieme dei servizi di sistema offre un ambiente più conveniente per lo sviluppo e l'esecuzione dei programmi e sono divisi in diverse categorie:

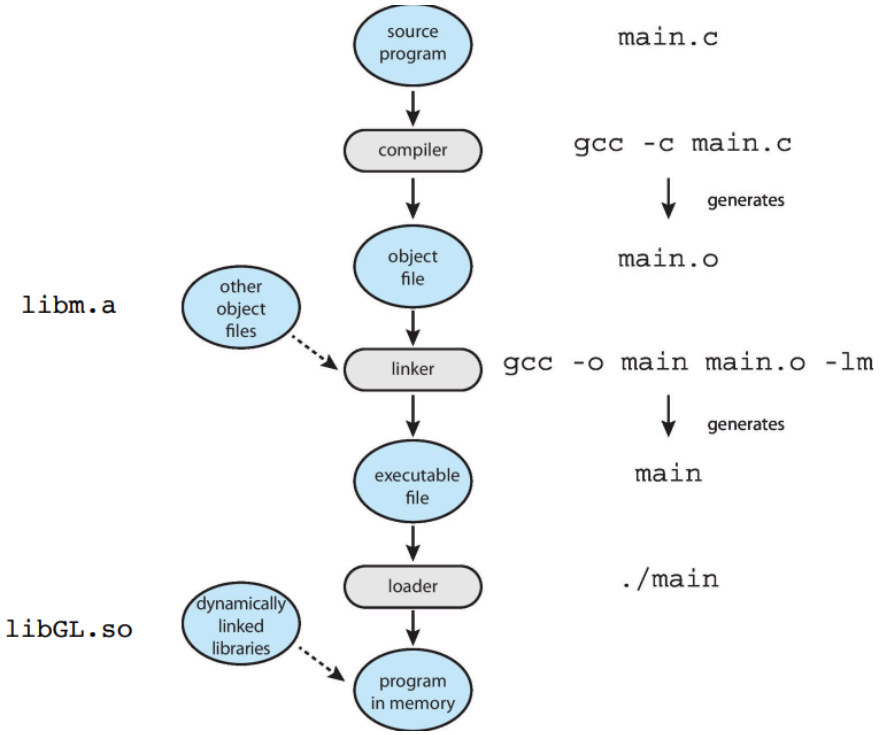
- **gestione dei file**: programmi che creano, cancellano, copiano, rinominano, stampano, elencano e in genere compiono operazioni sui file e le directory.
- **informazioni di stato**: programmi che richiedono al sistema di indicare data, ora, quantità di memoria disponibili o spazio sui dischi, numero degli utenti, ecc... Oppure forniscono informazioni

più dettagliate su prestazioni, accessi al sistema e debug. In alcuni sistemi possiamo trovare un registro che ha lo scopo di archiviare informazioni sulla configurazione del sistema.
- **modifica dei file**: editor per creare e modificare il contenuto di file memorizzati su dischi o altri dispositivi.
- **ambienti di supporto alla programmazione**: compilatori, assemblatori, debugger e interpreti dei linguaggi di programmazione.
- **caricamento ed esecuzione dei programmi**: una volta assemblato o compilato, un programma deve essere caricato in memoria per essere eseguito. Sono messi a disposizione caricatori assoluti, caricatori rilocabili, editor dei collegamenti (linkage editor) e caricatori di sezioni sovrapponibili di programmi (overlay loader). Sono disponibili anche sistemi d'ausilio all'individuazione e correzione degli errori (debugger).
- **comunicazioni**: programmi che creano collegamenti virtuali tra processi, utenti e calcolatori diversi.
- **servizi in background**: i sistemi general-purpose hanno metodi per lanciare programmi di sistema al momento dell'avvio. Alcuni terminano, mentre altri restano in esecuzione fino a quando il sistema viene arrestato, questi processi in esecuzione sono noti come servizi, sottosistemi oppure demoni. Oltre ai programmi di sistema vengono forniti anche programmi che risolvono problemi comuni o che eseguono operazioni comuni.

Linkers and Loaders

Generalmente un programma si trova su disco in forma di file binario eseguibile, per essere eseguito su una CPU deve essere caricato in memoria e inserito nel contesto di un processo.
I file sorgenti vengono compilati in file oggetto per essere caricati in memoria fisica e sono noti come *file oggetto rilocabili*. Il linker combina i file oggetto rilocabili in un singolo file binario eseguibile, durante questa fase (***linking***) possono essere inclusi anche altri file oggetto o alcune librerie.
Per caricare il file eseguibile in memoria viene utilizzato un ***loader***.
Un'altra attività associata a questo processo è la ***rilocazione*** (relocation), che assegna gli indirizzi definitivi alle componenti del programma e riaggiusta il codice e i dati nel programma secondo questi indirizzi in modo che, per esempio, il codice possa richiamare le funzioni della libreria e accedere alle sue variabili durante l'esecuzione.
Questo processo presuppone che tutte le librerie siano collegate nel file eseguibile e caricate in memoria, la maggior parte dei sistemi però consente a un programma di collegare dinamicamente le librerie quando viene caricato, cosi è possibile evitare di collegare a caricare librerie che potrebbero non

essere utilizzate.



Static Libraries vs Dynamic Libraries

Librerie Statiche
Vengono incorporate direttamente nell'eseguibile del programma al momento della compilazione.

- maggiore sicurezza: poiché il codice della libreria viene copiato all'interno dell'eseguibile, non c'è il rischio che venga modificato esternamente.
- maggiore velocità di esecuzione: il codice della libreria è già incluso nell'eseguibile, quindi il programma non deve cercarlo o caricarlo in memoria a run-time.
- richiede ricompilazione e rilinking: se il codice della libreria viene aggiornato, ogni programma deve essere ricompilato e rilinkato con la nuova versione.
- file di grandi dimensioni.
  Librerie Dinamiche
  Vengono collegate al programma solo durante l'esecuzione (runtime).
- condivisione tra più programmi: più programmi utilizzalo la stessa libreria, riducendo cosi anche lo spazio occupato su disco.
- rischio di corruzione o incompatibilità: se la libreria dinamica viene modificata c'è il rischio che alcuni programmi non funzionino più correttamente.
- file più piccoli.
- aggiornamenti senza ricompilazione.

Why Applications are OS Specific

Le applicazioni compilate su un SO non sono eseguibili su altri SO, in quanto ogni sistema fornisce un insieme univoco di chiamate di sistema, che fanno parte dell'insieme di servizi forniti dai SO alle applicazioni.
Un'applicazione può essere resa disponibile per l'esecuzione su più sistemi operativi in tre modi:

- può essere scritta in un **linguaggio interpretato** (es. Python, Ruby), ovvero un linguaggio che ha un interprete disponibile per più SO. L'interprete legge ogni riga del programma sorgente, esegue

istruzioni equivalenti del set di istruzioni nativo e invoca le chiamate proprie del SO. Le prestazioni sono inferiori rispetto a quelle delle applicazioni native in quanto l'interprete offre solo un sottoinsieme delle funzionalità di ciascun SO, limitando potenzialmente l'insieme delle funzionalità delle applicazioni associate.

- può essere scritta in un **linguaggio che utilizza una macchina virtuale** contenente l'applicazione in esecuzione. La macchina virtuale fa parte dell'rte completo del linguaggio. (es. Java → JVM, Java Virtual Machine). Presenta svantaggi simili a quelli dell'interprete.
- può utilizzare un **linguaggio o un'API standard** in cui il compilatore genera binari nel linguaggio specifico del SO e della macchina. L'applicazione deve essere portata su ciascun sistema operativo su cui verrà eseguita. Il porting può richiedere molto tempo e deve essere fatto per ogni nuova versione dell'applicazione, con test e debug. (es. C)
  Mentre le API specificano determinate funzioni a livello di applicazione, a livello di architettura troviamo l'ABI (*application binary interface*) per definire in che modo i diversi componenti di un codice binario possano interfacciarsi su un determinato SO e una determinata architettura.
  Un'ABI specifica:
- dettagli di basso livello;
- dimensione degli indirizzi;
- metodi di passaggio dei parametri alle chiamate di sistema;
- organizzazione dello stack di runtime;
- formato binario delle librerie di sistema;
- dimensione dei tipi di dati;
- ecc...