01 - PIATTAFORME DI ESECUZIONE

Fatta eccezione per i sistemi più elementari, l'esecuzione di un'applicazione avviene nel contesto di un sistema operativo, il quale offre un insieme di servizi, funzionalità e convenzioni che ne permettono il funzionamento, che prendono il nome di API (Application Programming Interface).
Le ABI (Application Binary Interface) definiscono invece quale formato debba avere un prodotto software per essere compatibile con un determinato sistema operativo.

Per sviluppare un programma che si interfacci direttamente con il sistema operativo, occorre accedere e conoscere le API da chiamare (file header), le strutture dati coinvolte e le convenzioni definite dal SO (strutture che descrivono lo stato del sistema ed a cui si può accedere tramite API in modo indiretto attraverso riferimenti opachi come HANDLE di winzoz o FileDescriptor in Linux).

Tutte le funzioni invocate possono anche fallire, per cui è bene anche gestire gli errori, controllando il valore di ritorno di ogni API per capire SE c'è stato un errore e QUALE errore sia.

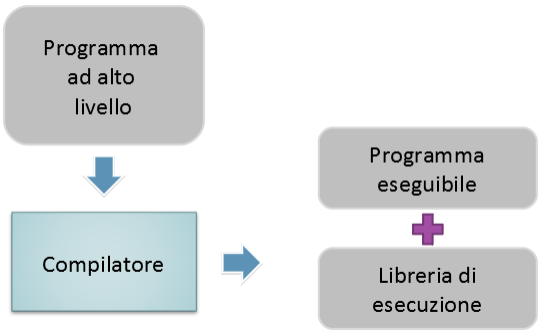
Siccome la rappresentazione dei caratteri a 8 bit è insufficiente per la maggior parte degli alfabeti, occorre anche avere un meccanismo di codifica e gestione del testo (possibili rappresentazioni: UTF-32, UTF-16, UTF-8) che diventa particolarmente complicato nel caso di rappresentazioni a lunghezza variabile, in quanto si possono avere ordinamenti differenti (es: big endian o little endian): spesso quindi si adotta una codifica a 16 bit ignorando i caratteri rari.
In Windows ci sono due tipi base: il char (8 bit) e il wchar_t (16 bit).
In Linux di default GCC codifica eventuali caratteri non-ASCII con UTF-8, che però può creare problemi (per esempio *strlen(str)* non indica più il numero di caratteri effettivamente presenti ma solo il numero di byte non nulli).

Gli standard POSIX (acronimo per Portable Operating System Interface) sono una famigli di standard per mantenere la compatibilità tra i diversi sistemi operativi. Sono delle API a basso livello e un set di comandi *shell* volti a rendere portabili gli applicativi.

02 - IL MODELLO DI ESECUZIONE

Ogni linguaggio di programmazione propone uno specifico modello di esecuzione, che non corrisponde quasi mai a quello di un dispositivo reale. E' compito del compilatore introdurre uno strato di adattamento che implementi il modello nei termini offerti dal dispositivo soggiacente.
Il programma originale quindi viene trasformato in uno nuovo dotato di un modello di esecuzione più semplice tramite la traduzione fatta dal compilatore.

Le librerie di esecuzione offrono agli applicativi meccanismi di base per il loro funzionamento e sono costituite da due tipi di funzione: alcune sono inserite in fase di compilazione per supportare l'esecuzione (come il controllo dello stack) e sono quindi invisibili al programmatore; altre offrono funzionalità standard che gestiscono opportune strutture dati ausiliarie e/o richiedendo al SO quelle non altrimenti realizzabili (malloc).



Un programma C/C++ assume di poter accedere a qualsiasi indirizzo di memoria all'interno del quale può leggere o scrivere dati o dal quale può eseguire codice macchina. Inoltre, l'insieme di istruzioni che definiscono il programma vengono eseguite, una per volta, nell'ordine indicato dal programmatore.
Il punto di partenza del flusso di esecuzione è predefinito (*main()* in C e i costruttori delle variabili globali in C++) e lo stack permette di gestire chiamate annidate tra le funzioni.

Ad ogni modo, le sole librerie di esecuzione non possono implementare tutte le astrazioni del modello di esecuzione, si ha bisogno che i programmi siano separati in modo tale che un malfunzionamento di uno non vada a toccare gli altri. Per fare ciò venne introdotto il concetto di processo, che porta con sè il contesto di un programma e vengono gestiti dal SO (non solo a livello di creazione ma anche di gestione di flusso di esecuzione).
Quando un processo viene selezionato per l'esecuzione, il SO configura il processore con le relative risorse e ripristina lo stato ad esse associato: si richiede quindi un supporto hardware per distinguere due modalità di esecuzione:

- Modalità **utente**: può eseguire un sottoinsieme delle istruzioni offerte dalla CPU ed esegue il codice associato ai diversi processi, sottostando a vincoli di accesso sia per memoria che per periferiche

- Modalità **supervisore**: crea e gestisce le strutture dati che modellano i processi, il loro spazio di indirizzamento e le altre risorse che posseggono. Accesso illimitato a tutte le funzionalità e parti (incluse periferiche, filesystem e rete)

L'innalzamento di privilegio comporta la sostituzione dello stato della CPU e delle struttture di supporto all'esecuzione tramite istruzioni sicure, per cui il costo per attraversa la barriera tra le due modalità è elevato.

L'esecuzione di un programma comporta quindi la creazione di un nuovo processo, articolata in diverse fasi:

1. Creazione spazio di indirizzamento: insieme di locazioni di memoria accessibili tramite indirizzo virtuale. Attraverso la MMU del processore, gli indirizzi virtuali vengono tradotti in fisici, per cui si possono creare innumerevoli spazi tra loro separati. Attenzione ai *page_fault* (tentativi di accesso a indirizzi non mappati)

2. Caricamento dell'eseguibile in memoria: dopo aver inizializzato lo spazio di indirizzamento, il loader inserisce il contenuto del programma da eseguire (incluse le dipendenze). Si hanno quindi due sezioni, una per il codice e una per i dati e altre sezioni per stack e heap

3. Caricamento delle librerie: una sezione del file eseguibile è dedicata ad elencare ulteriori eseguibili necessari, che vengono mappati ricorsivamente nello spazio di indirizzamento, mentre tutti i riferimenti a variabili/funzioni delle dipendenze vengono aggiornati con gli indirizzi effettivi.

4. Avvio dell'esecuzione: quando il processo di caricamento termina, è possibile avviare l'esecuzione, tramite un'apposita funzione della libreria di esecuzione. Il comportamento di tale funzione dipende dal linguaggio di programmazione e dal SO, ma in sostanza inizializza stack, registri e strutture per gestire eccezioni; invoca costruttori e distruttori di oggetti globali e la funzione principale e infine rilascia l'intero spazio di indirizzamento (con le risorse in esso allocate) attraverso la system call *exit()*.