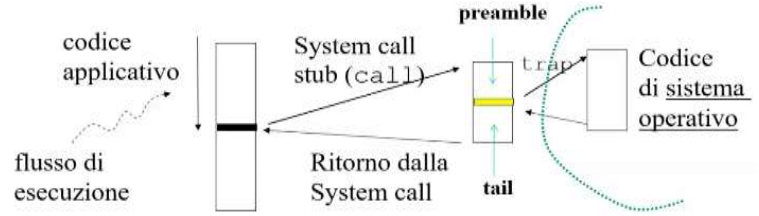
Introduzione2 System call

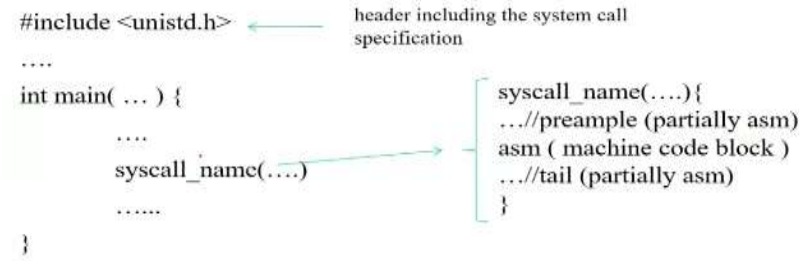
venerdì 3 ottobre 2025 17:38

Vediamo ora la relazione tra il software delle nostre applicazioni con quello del sistema operativo : si utilizza un meccanismo di **system call** (chiamate di sistema) . Queste system call forniscono un'interfaccia per l'accesso al software del sistema operativo.

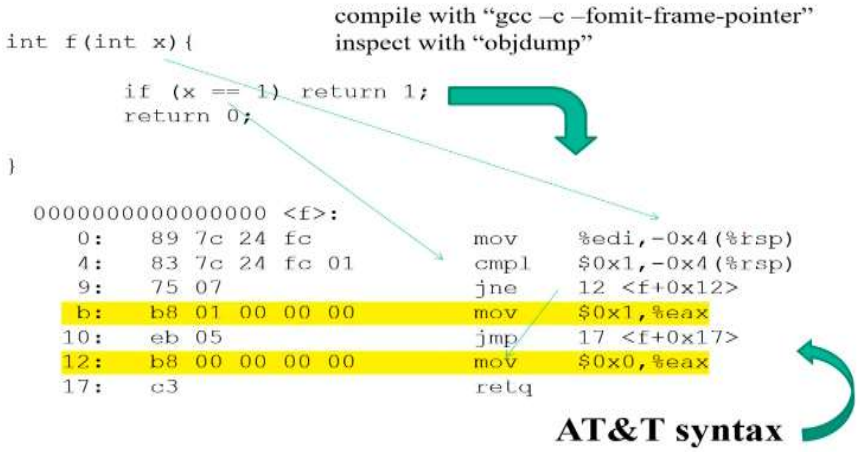
Inizialmente erano implementate solo in assembly , ma alcuni linguaggi di alto livello (tipo C e C++) permettono l'invocazione diretta di una system call. Quanto detto finora vale per solo assembler. Per quanto riguarda il C , il discorso è diverso :



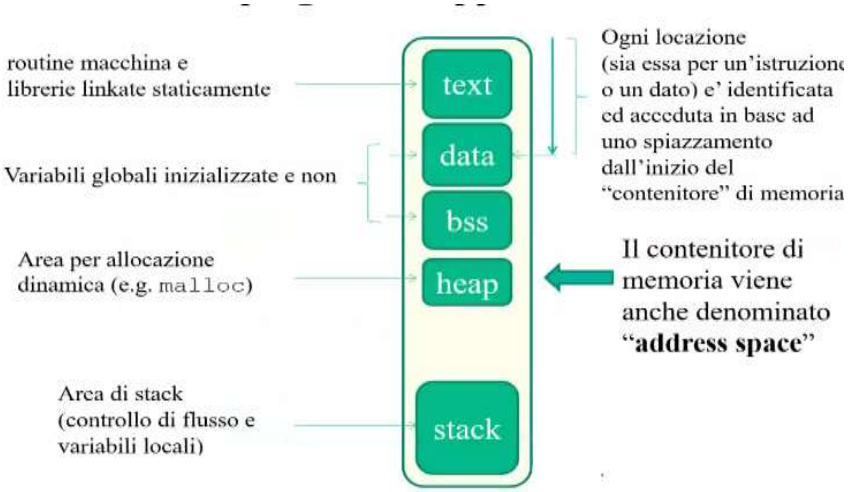
Il supporto per le system call sono le istruzioni di trap, quindi si ha invocazione indiretta ad istruzione macchina : ovvero salto a subroutine e poi ad istruzione macchina. Con il salto alla subroutine si intende l'invocazione di una libreria . Una domanda sorge: cosa facciamo nel preambolo e nella coda? Nel preambolo scrivo le informazioni nei registri (servizio da invocare e parametri), mentre nella coda si fa qualcosa con il valore di ritorno della chiamata della funzione del sistema operativo. Quindi in realtà noi usiamo la libreria , quindi si punta ad una architettura machine dependent, la quale a sua volta usa programmazione assembler per poter utilizzare trap e istruzioni per manipolare registri. Vediamo un esempio :



In dettaglio:



Vediamo ora la struttura complessiva di un programma , o meglio **lo spazio di indirizzamento** :



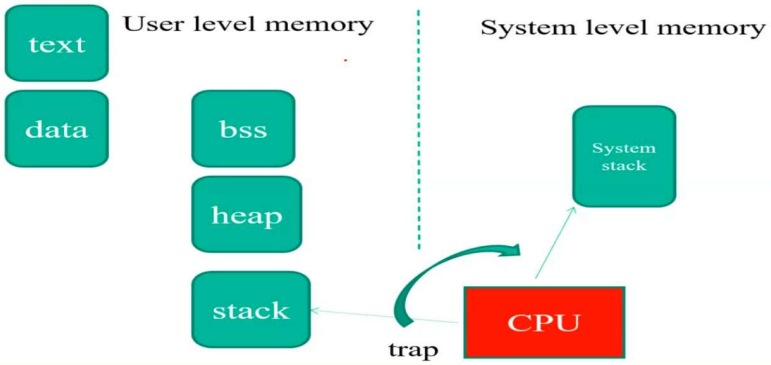
Andiamo a vedere i formati della compilazione :

1. Unix
    a. Si usa gcc
    b. Il formato della compilazione è ELF(Executable and linkable format)
2. Windows
    a. Il formato della compilazione è PE(portable executable), il quale è basato su COFF(Common object file format), tramite il quale si descrive il contenuto effettivo di un file exe
3. In generale si ha che : è compito del tool di compilazione di generare ELF/EXE a partire dai sorgenti del programma e dalle direttive di compilazione, specificando eventualmente quali moduli vanno inclusi e quali sezioni vanno incluse

Vediamo ora lo stub delle **system call:**

```
int syscall_name(int , void *,  struct struct_name * …)
```

Sono caratterizzate da nome e parametri , tra i quali vi possono essere dei puntatori . I parametri tengono a cambiare lo sato del processore (cambio e/o scrivo/leggo nello spazio di indirizzamento). Per quanto riguarda il valore di ritorno della system call (ritornato al chiamante dalla coda delle istruzioni macchina) viene gestito dalla coda della routine macchina, quindi dipende dai valori dei registri della CPU. Vediamo ora come viene gestita la memoria :



Quindi in generale oltre all'address space (user level memory) , per ogni applicazione viene utilizzata "altra memoria" destinata all'uso di questa specifica applicazione (anch'essa è stack area); quindi in generale per ogni programma in esecuzione si usano almeno due stack area. Per arrivare a questa altra area di memoria serve che lo STACK POINTER punti a questa area , ma come?? In seguito ad invocazione di trap che invocando il sistema operativo per un modulo.