Stub Sys.Call

martedì 21 marzo 2023 17:25

Tornando allo “stub” delle system call

Torniamo allo Stub delle System call, ovvero queste funzioni di libreria che possono riportare il controllo al software del sistema operativo per eseguire specifiche attività e andiamo a vedere qualche dettaglio di come sono strutturate.

Ciascuna di esse ha un “nome” e possiamo passare dei “parametri”.

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

Alcuni di questi parametri possono essere dei pointers.

Il secondo parametro è un puntatore generico, il terzo è un puntatore ad una struttura.

I parametri che vengono essere ammessi in input a questa funzione vengono scritti dalla stessa all’interno di “registri di processore”.

Nel caso di pointers, scrivere questi pointers dentro i registri di processore, ossia “*COMUNICARE*” GLI INDIRIZZI DI MEMORIA all’interno dei registri di processore, permette, nel momento in cui il software del sistema operativo prende il controllo, di sapere in quali posizioni di memoria operare.

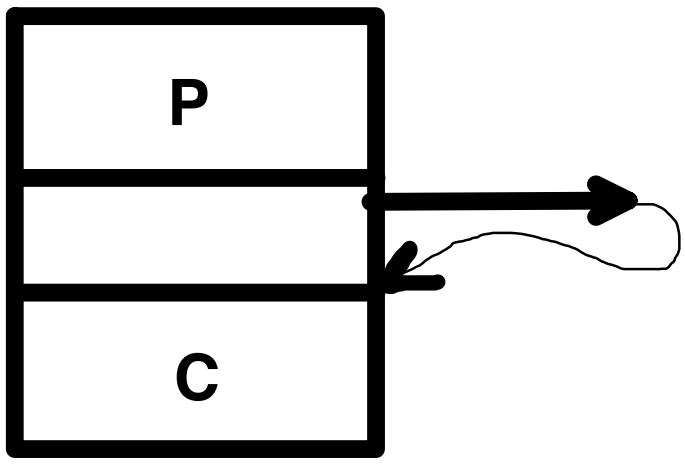
Il software del S.O, conoscendo dei POINTERS, può andare a scrivere nello spazio di indirizzamento dell’applicazione (Address Base) durante l’esecuzione della System call.

Tipicamente quando noi passiamo dei pointers come dei parametri a delle System Call, il software del S.O va a lavorare all’interno dell’address space.

Il valore di ritorno di una System call viene ritornato al

chiamante dalla coda delle istruzioni macchina che rappresentano l'eseguibile di questo Stub.

La sys.Call è un blocco di codice dove dentro abbiamo il passaggio di controllo (tramite un'istruzione macchina particolare) al software del S.O, ma abbiamo un PREAMBOLO e una CODA di istruzioni macchina, e quando il software del sistema operativo ritorna il controllo, ci rimane la coda delle istruzioni macchina da eseguire.



Ritorneremo al chiamante utilizzando un'istruzione macchina all'interno di questa coda

Per esempio una retq nel caso dei processori X86.

Il valore di ritorno che è gestito da questa System call, è gestito dalla coda.

Nel momento in cui il S.O prende il controllo, questa coda può andare a verificare in CPU, quali siano i valori all'interno dei registri e, sulla base di essi, definire quale valore ritornare all'applicazione.

Oppure prelevare questi valori che sono all'interno dei registri e in base a ciò che c'è scritto, generare un nuovo valore da ritornare al chiamante.

- ✓ il valore di ritorno è definito dalla "coda" della routine macchina
- ✓ questo generalmente dipende dal valore scritto dal sistema operativo in qualche registro di processore

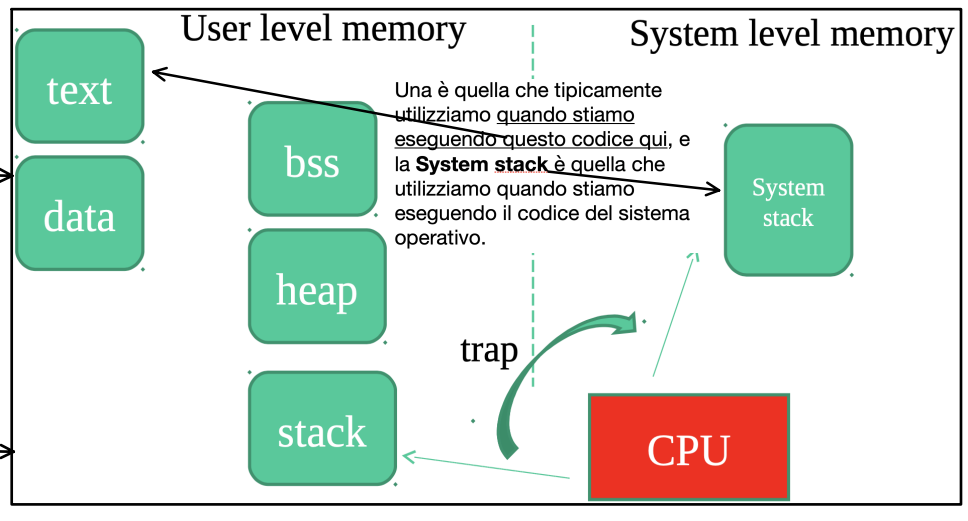
Qual'è la traccia che il S.O ha lasciato all'interno dei registri di CPU quando questo Stub riprende il controllo.

Torniamo un valore all'interno dei registri a questo Stub e lavoriamo nella zona della coda per elaborare quale valore ritornare al chiamante.

Una cosa da considerare quando abbiamo una System Call è un aspetto riguardante la GESTIONE DELLA MEMORIA.

Quando noi abbiamo un'applicazione attiva con il suo relativo Address Space, in realtà all'applicazione SPECIFICA viene associata anche un'altra zona di memoria *destinata all'uso di questa specifica applicazione*: la zona si chiama **System Stack**.

Questa zona della memoria è ancora una stack area, e lavoreremo perlomeno con due stack areas.



Quando avviene una ***TRAP***, e quindi il flusso d'esecuzione viene portato ad un modulo del S.O, viene cambiato lo stato del processore per far sì che lo ***stack pointer*** ci porti nella System Stack.

Abbiamo sicuramente uno stack USER e uno stack SISTEMA all'interno della gestione delle applicazioni convenzionali. Questo è dovuto al fatto che il software del S.O **non vuole utilizzare la Stack User** per andare ad inserire dentro le informazioni che riguardano ciò che il software stesso dovrà fare o i valori che il software stesso deve poter gestire.

Il cambio di stack avviene quando uno stub di system call esegue l'istruzione di trap che passa il controllo al sistema operativo