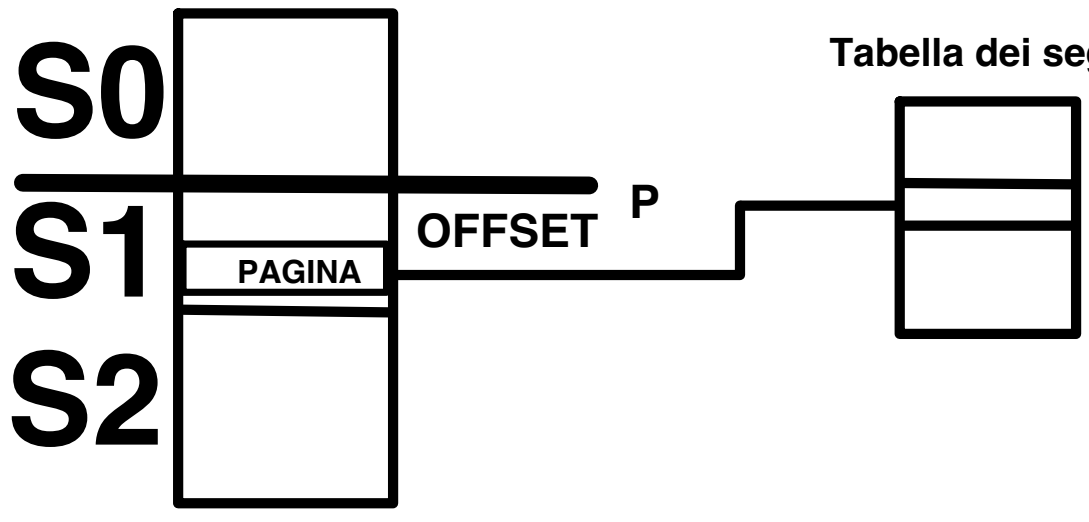
Segmentazione e paginazione in processori x86

domenica 14 maggio 2023 11:11

Segmentazione e paginazione in processori x86

- Ogni applicazione eseguibile per macchine x86 usa di fatto uno schema di accesso alla memoria basato su segmentazione e paginazione

All'interno dei sistemi operativi convenzionali, ogni elemento della tabella dei segmenti ci indica la base del segmento nella memoria logica.
Quando abbiamo la segmentazione in combinazione con la paginazione su processori X86 supponiamo di avere il segmento S0 e il segmento S1, e supponiamo di avere la tabella dei segmenti che è un'informazione che viene mandata in set-up dal nostro sistema operativo quando questo address space deve essere gestito.
La tabella dei segmenti è tale per cui che una sua entry associata al segmento S1, ci dice dove in memoria è presente la tabella delle pagine per questo segmento a cui stiamo accedendo? NO. Succede una cosa differente.
L'informazione che noi abbiamo all'interno di questa entry, è un'informazione che ci dice che dato questo segmento S1, qual è la sua posizione di dove lui inizia all'interno della memoria logica (NON DELLA FISICA).



L'offset di S1 all'interno dell'Address Space cade dove finisce S0. Quindi all'interno di quella tabella abbiamo esattamente l'indicazione del punto P all'interno dell'address space.
Ma se poi conosciamo qual è l'offset all'interno del segmento S1 a cui vogliamo accedere, perché abbiamo espresso un indirizzo logico S1 + OFFSET, per sapere qual è l'esatto punto di questo address space logico, basta sommare a P esattamente questo offset. E otteniamo la posizione esatta a cui stiamo cercando di accedere all'interno dell'address space.
QUESTA POSIZIONE SI CHIAMA INDIRIZZO LINEARE.
Quando parliamo di un indirizzo lineare in un sistema moderno stiamo parlando di un offset a partire dall'inizio dell'Address Space.
Quello che noi facciamo con la segmentazione paginata all'interno dei processori x86 è utilizzare la segmentazione per determinare qual è l'indirizzo lineare (quindi il punto esatto dell'address space a cui stiamo accedendo).
Poi, la paginazione da questo indirizzo lineare va ad indicare esattamente qual è la pagina a cui stiamo accedendo, e quindi dove questa è collocata in memoria.
La paginazione lavora sull'indirizzo lineare.

- In sistemi operativi convenzionali ogni elemento della tabella dei segmenti indica la base del segmento nella memoria logica
- Tale base viene sommata all'offset espresso dall'istruzione per generare l'indirizzo paginato, detto anche **indirizzo lineare**
- Tale indirizzo viene usato per accedere alla tabella delle pagine e calcolare l'indirizzo fisico

Quindi l'indirizzo lineare viene ad essere utilizzato come punto dell'address space a cui vogliamo accedere, e quel punto sarà identificabile come una specifica pagina a cui vogliamo accedere e un offset di pagina.

Se non specifichiamo qual è il segmento a cui vogliamo accedere, quindi supponiamo che all'interno del nostro codice x86 scriviamo:

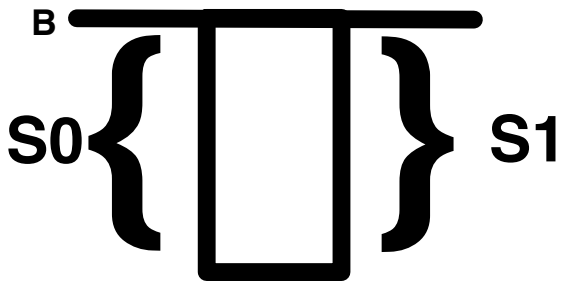
MOV (%rax), rbx

Prendiamo rax (utilizzato come pointer alla memoria) e quello che c'è scritto in questo offset va caricato all'interno di rbx.

Ma l'offset di (%RAX), in quale segmento cade?

Se non specifichiamo questo abbiamo un DEFAULT.
Il default che è classicamente utilizzato in processori x86 è il default relativo al DS (Data segment), dove ovviamente ci sono i dati, ma poi quando esprimiamo un indirizzo per un salto a subroutine stiamo esprimendo un indirizzo che userà il codice, quindi usiamo il default CS (Code Segment).
Quindi se noi non andiamo a scrivere questa specifica del CS all'interno della nostra istruzione, la CPU va ad utilizzare il CS come default.

Quando noi lavoriamo su processori X86 per molti di questi segmenti succede che, dato un address space, ho una zona dove mantengo S0, ma è la stessa zona dove mantengo anche S1, E COSÌ VIA.



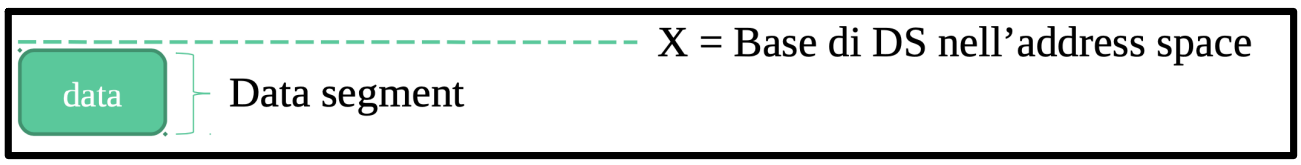
Address Space

In particolare abbiamo la stessa base di posizionamento di questi due segmenti, all'interno dell'address space. Quindi questi due segmenti cascano nello stesso punto logico all'interno dell'address space.
Lo stesso punto logico specificato come indirizzo lineare all'interno di questo AB.

- Se non si specifica il segmento corrisponente ad un accesso in memoria nel codice x86, viene acceduto di fatto un segmento di deafault, ad esempio:
  - ✓ CS (CODE SEGMENT) per il codice
  - ✓ DS (DATA SEGMENT) per i dati

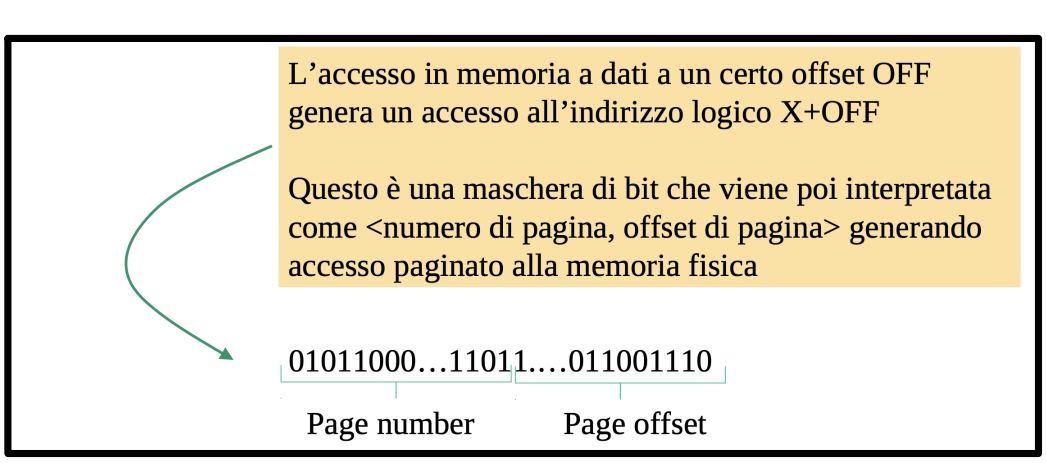
Uno schema

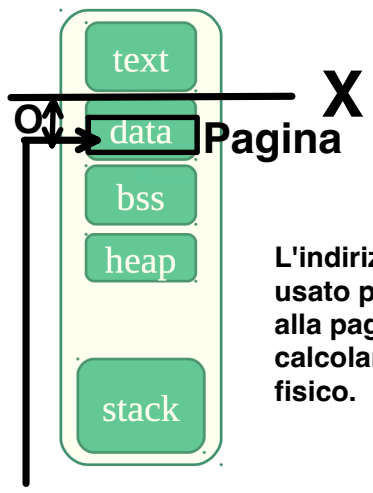
Quando noi abbiamo un accesso su X86, e magari questo accesso è relativo al data segment (DS) nell'address space, quindi vogliamo accedere a dei dati all'interno dell'address space che hanno un certo posizionamento, X rappresenta la base di questo segmento nell'address space.



X è la base dell'indirizzamento lineare di ciò a cui stiamo accedendo, all'interno di questo data segment esprimiamo un <u>offset</u> interno per accedere a dei dati.

Bene, tutta questa informazione <u>X</u> + <u>offset</u> viene a diventare un'unica maschera di bit che esprime un indirizzo lineare all'interno di questo address space, suddiviso in una zona che specifica il numero di pagina nell'indirizzamento lineare, e una zona che specifica l'OFFSET su quella pagina.

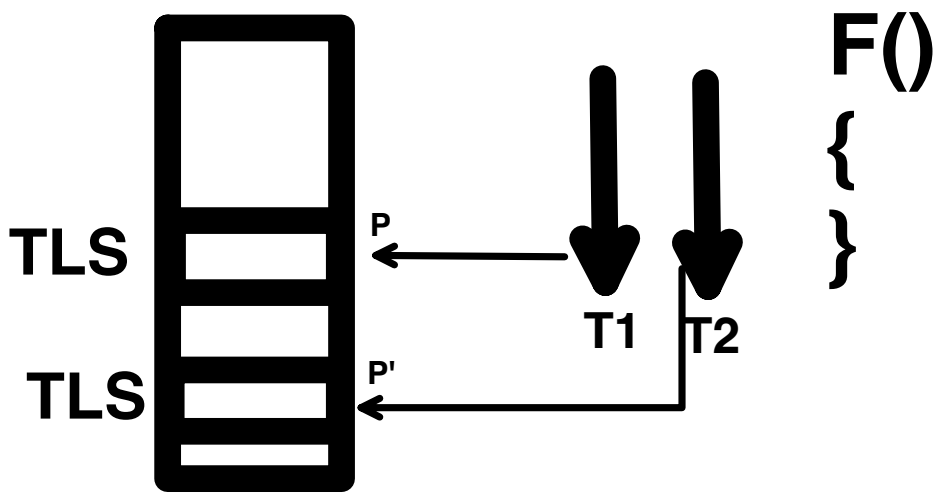




L'indirizzo lineare viene usato per accedere poi alla page table e calcolare l'indirizzo fisico.

Voglio accedere qui

Ragioniamo sulla SEGMENTAZIONE e il thread local storage: quando noi abbiamo un'applicazione compilata con un utilizzo del TLS, abbiamo che questa applicazione avrà delle istruzioni macchina che saranno in grado di arrivare su un TLS o su un altro TLS, esattamente allo stesso modo. La stessa funzione F() può essere utilizzata da un thread o da un altro thread, quindi lo stesso blocco di codice macchina può essere utilizzato da un thread o un altro thread, sul thread 1 andremo in una zona di variabili per thread, sul thread 2 andremo in un'altra zona.



Come si fa a fare questo? Quando noi accederemo al Thread Local Storage, in particolare per quanto riguarda le istruzioni macchina che andremo ad utilizzare, queste istruzioni macchina vanno esattamente ad esprimere qual è il segmento a cui vogliamo accedere. Quindi utilizzeremo un segmento per andare a specificare che ci stiamo

muovendo a partire dal punto P' dell'address space per andare in quella zona, o a partire dal punto P.
Ma il segmento che utilizzeremo è lo stesso, soltanto che quando T2 va in CPU o T1 va in CPU, la nostra CPU viene tarata con una SEGMENT TABLE per cui quel segmento una volta va da una parte e un'altra volta va dall'altra.
Quando cambiamo thread in CPU possiamo cambiare anche le tabelle di gestione della memoria, questo implica dire che stiamo riportando quel software stesso in una zona o in un'altra zona.
In particolare per l'implementazione del TLS.
Il tls utilizza questi schemi segmentati per identificare specifiche zone all'interno dell'address space dove noi dobbiamo andare ad accedere.