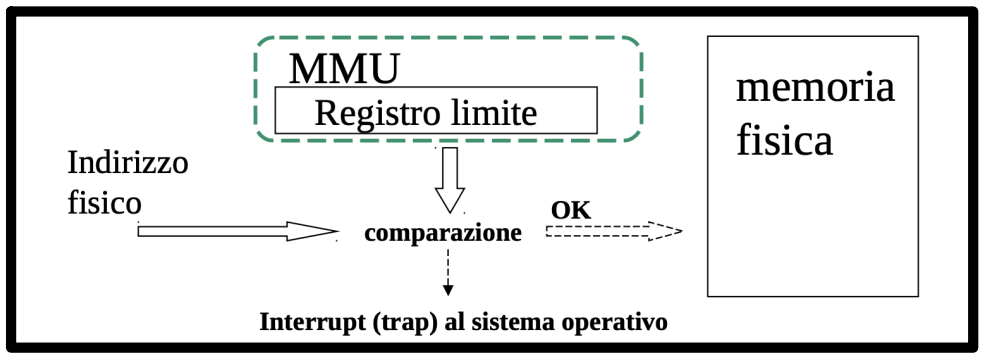
Protezione degli accessi in memoria

mercoledì 10 maggio 2023 17:32

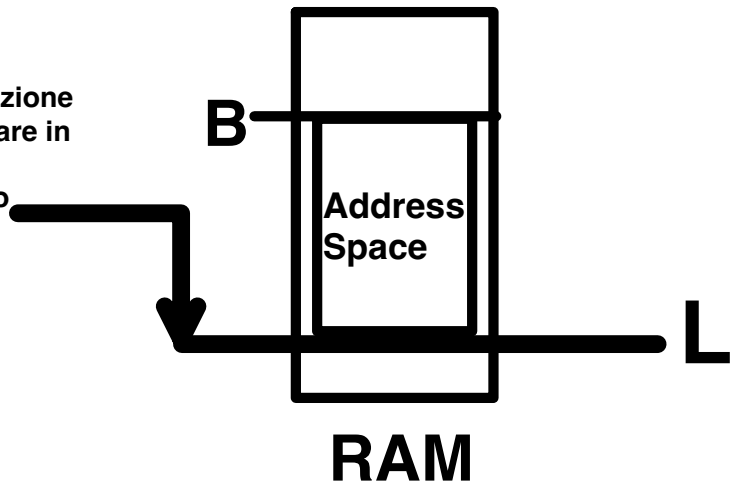
Protezione degli accessi in memoria

Per risolvere il problema delle interferenze, la MMU è andata ancora avanti, e all'interno di esso è stato aggiunto un altro registro che si chiama "REGISTRO LIMITE".

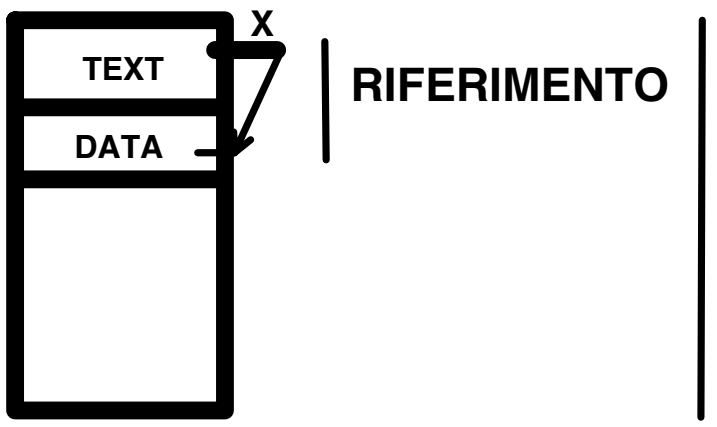


Il registro limite è il registro che ci dice qual è il limite massimo, all'interno della memoria fisica, dove un'applicazione è collocata. E quindi questo registro LIMITE ci va ad indicare in maniera ultimativa, la posizione di questa informazione.
Dentro alla MMU abbiamo anche l'identificazione di questo punto:

Questo punto viene chiamato LIMITE.
Quindi abbiamo la base, ossia il punto in cui inizia la rappresentazione dell'address space all'interno della RAM, e anche il limite. Chiaramente all'interno di una MMU, dopo aver calcolato l'indirizzo fisico come spiegato prima, questo indirizzo fisico calcolato viene comparato con il registro limite; Questo viene fatto con lo scopo di capire se andiamo oltre il limite dello spazio di indirizzamento dell'address space.
Se andiamo oltre il limite, chiaramente il sottosistema di controllo genera una TRAP, e quindi il flusso passa al sistema operativo.
Prende il controllo il codice del sistema operativo perché in qualche modo abbiamo la necessità chiaramente di evitare che ci sia un accesso effettivo alla memoria su un area che non è riservata per l'applicazione che correntemente è in esercizio.



Ovviamente tutti questi controlli non possiamo imporli soltanto quando compiliamo le applicazioni.
Quando noi compiliamo un'applicazione possiamo generare un codice tale per cui quando abbiamo un'istruzione su X all'interno della zona testo, questa istruzione riferisce qualcosa all'interno della zona dati, e andiamo a controllare che questo riferimento sia mantenuto all'interno dell'address space della nostra applicazione.
NON POSSIAMO FARLO, perché alcune volte i riferimenti vengono eseguiti utilizzando i pointers.



Ma con questa soluzione che opera a run-time, stiamo tranquilli che anche con i pointers (anche se c'è un bUG all'interno dell'applicazione e con un pointer stiamo cercando di andare fuori dall'address space) abbiamo la possibilità di far intervenire il software del sistema.

Address Space

e.g. `mov (%rax), %rbx`

Se noi consideriamo questa istruzione, noi abbiamo caricato il VALORE di un pointer all'interno di RAX, e con le () lo stiamo realmente utilizzando per accedere alla memoria, e questa cosa qui viene determinata in base a ciò che succede a run-time, non a compile-time.
Perché magari all'interno di questo registro ci finiscono informazioni che sono funzione di valori di input che sono letti dalla mia applicazione. Il che implica dire che a tempo di compilazione non abbiamo la garanzia di generare qualcosa per cui un'istruzione riferisca necessariamente sempre all'interno del contenitore (indipendentemente dalla sua taglia), piuttosto che fuori.

- ogni processo deve essere protetto contro interferenze di altri processi, siano esse accidentali o intenzionali
- i riferimenti di memoria generati da un processo devono essere controllati per accertarsi che

l'indirizzi di memoria generati da un processo devono essere controllati per accertarsi che cadano nella regione di memoria realmente riservata a quel processo

- il controllo va effettuato run-time poichè la maggioranza dei linguaggi di programmazione supporta il calcolo degli indirizzi tempo di esecuzione (vedi indici di array o puntatori a strutture dati)
- il controllo avviene via hardware per motivi di efficienza

Avere hardware che ci supporta questi controlli per quanto riguarda il binding a tempo d'esecuzione, ci risolve anche il problema dell'efficienza: tutti questi controlli eseguiti da una MMU che è un oggetto hardware ci permette un'efficienza molto molto ampia.