Struttura interna di un address-space

lunedì 15 maggio 2023 18:55

Struttura interna di un address-space

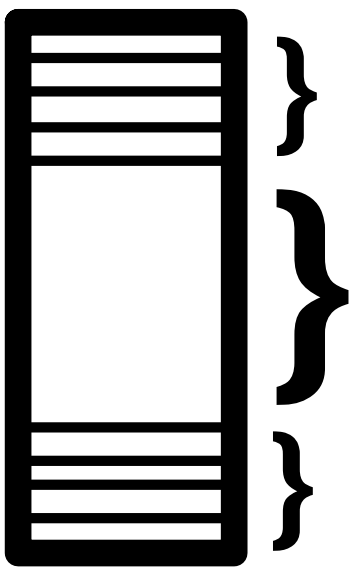
Fin ora abbiamo analizzato questo.
Dato un address space abbiamo studiato come questo è mappato all'interno della RAM.
Ci interesserebbe, nel momento in cui noi vogliamo scrivere il software all'interno di questo address space, come (questa struttura interna dell'address space) è fatta.
Quindi NON la struttura di collocamento dell'address space in memoria fisica, quanto la struttura interna sia fatta e come possiamo lavorarci su.

- Come detto, utilizzando X bit per l'indirizzamento logico la taglia globale di un address space è 2^X differenti locazioni (e.g. byte)

Quindi se noi abbiamo X bit per discriminare, nell'indirizzamento lineare, un punto qualsiasi di tutto l'address space, noi sappiamo che di questi punti con X bit né possiamo discriminare 2^X.

E poi tutti i punti dell'address space possono essere byte differenti a cui accediamo una volta che abbiamo specificato una posizione.

Ogni offset possibile all'interno di queste 2^X possibilità possono essere lecite.
Di queste possibilità però, non necessariamente ad ogni istante le possiamo utilizzare tutte: dato un address space noi possiamo avere alcune pagine che sono realmente utilizzabili ORA, per esempio una zona sopra e sotto, e una zona intermedia che non è utilizzabile ORA.



Se ora esprimo la volontà di accedere esattamente nei punti utilizzabili, non ci sono problemi.
Ovviamente l'accesso deve essere compatibile col tipo di protezione che abbiamo su queste pagine, se sono pagine READ ONLY (e questo me lo dice la Page Table, perché oltre a dirmi dove è collocata in memoria fisica la pagina, ha anche dei bit di protezione) tutto apposto, altrimenti se accedo nella zona NON UTILIZZABILE, e quindi esprimo un offset che mi porta esattamente lì, posso avere dei problemi.

Questi problemi sono legati al concetto di "*Mappatura delle pagine dell'address space*".

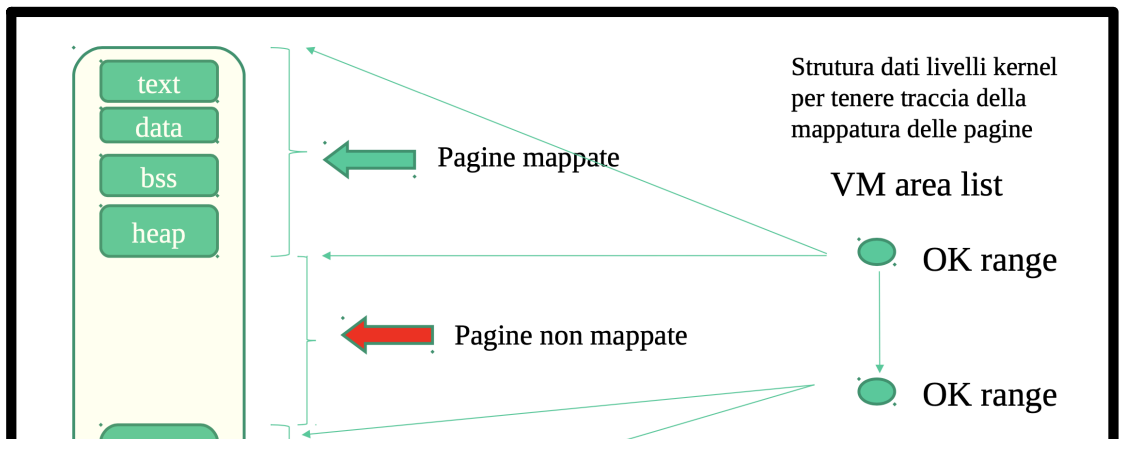
Se io accedo all'interno dell'address space nella zona non utilizzabile, sto accedendo ad una zona in cui le pagine che "dovrebbero" risiedere in quella zona dell'address space non sono correntemente mappate.

Un'address space può essere molto grande, e la grandezza dipende da quegli X bit che noi utilizziamo per esprimere gli offset.

Ma possiamo avere applicazioni che possono utilizzare questo address space in maniera massiva oppure NO.
Quindi possiamo avere delle applicazioni che hanno bisogno di usare pochissime pagine rispetto a tutte le pagine utilizzabili.
Per queste applicazioni ci possono essere alcune pagine che correntemente sono non mappate all'interno dell'address space: non sono ritenute valide dal sistema operativo.

- Applicazioni diverse e scenari di esecuzione diversi portano talvolta a voler usare (mappare) solo un sottoinsieme delle pagine logiche contenibili nell'address space

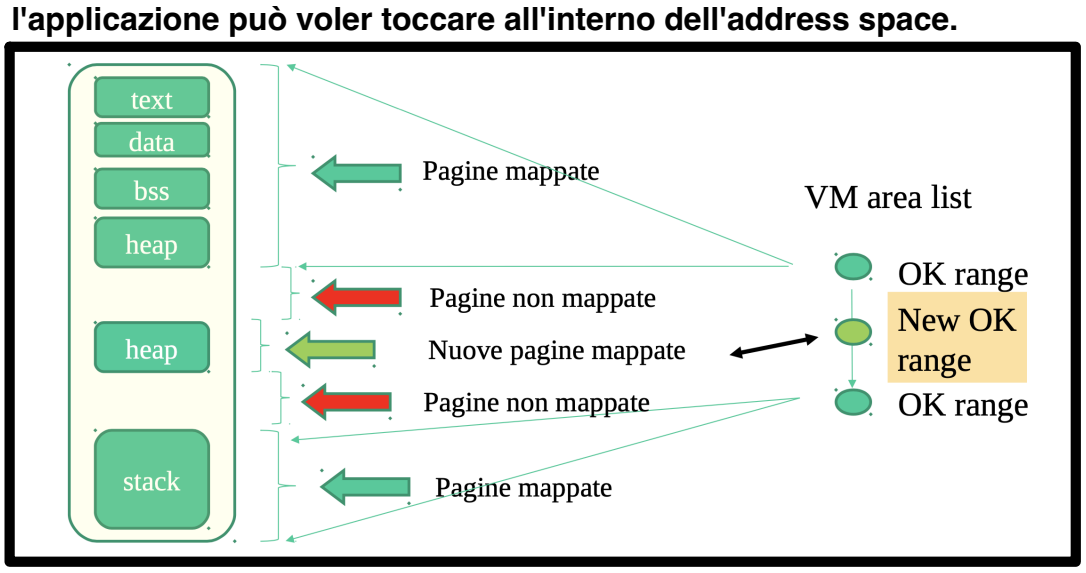
Supponiamo di dover considerare l'address space di un'applicazione che viene attivata su un sistema operativo, all'interno di questo AB abbiamo alcune pagine per il testo, alcune pagine per i dati, alcune pagine per il bss e alcune per l'allocazione dinamica, che sono correntemente utilizzate. Quindi l'applicazione ha chiesto di voler utilizzare queste pagine all'interno di questo address space.
Poi sotto abbiamo un'altra zona per la stack area.
IN MEMORIA FISICA TUTTE QUESTE PAGINE POSSONO ESSERE SPARSE IN CHISSÀ QUALI FRAME.



In mezzo abbiamo una zona di pagine che sono "NON MAPPATE".
Questo address space avrà una tabella delle pagine che ci dirà dove le zone mappate e le relative pagine sono collocate in memoria fisica, ma per le pagine non mappate la tabella non ci dirà nulla.
Oltre alla tabella, quello che noi abbiamo all'interno di un kernel di un sistema operativo per effettuare la gestione di questo address space, è un'informazione ulteriore che si chiama "*Virtual Memory*"

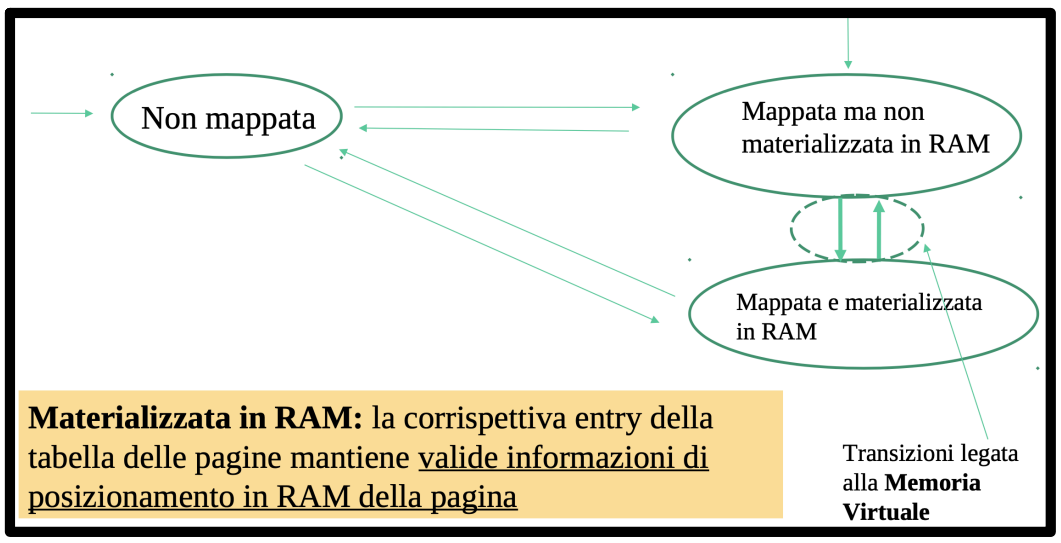
***Area List*".**
È una lista di informazioni che vanno esattamente ad indicare quali sono le zone correntemente valide all'interno dell'address space.

Chiaramente noi potremmo mappare qualche cosa all'interno di questo Address Space, in particolare nella zona originariamente non mappata potremmo richiedere di voler mappare alcune pagine, la richiesta va fatta al software del sistema operativo che, per il processo che fa la richiesta, dovrà cambiare lo stato della Virtual Memory Area List, inserire quindi un'informazione che va ad indicare che c'è un nuovo range di pagine che l'applicazione può voler toccare all'interno dell'address space.



Se l'applicazione tocca una di queste pagine questa cosa dovrà essere permessa, mentre originariamente avevamo che le pagine - non essendo mappate - erano intoccabili dall'applicazione.
Se tocchiamo delle pagine non mappate, generiamo quello che viene chiamato SEGMENTATION FAULT. Mentre invece se tocchiamo la stessa zona, in particolare nella nuova pagina mappata, possiamo avere un esito completamente diverso perché stiamo toccando qualche cosa che correntemente è diventato utilizzabile.

Quello che noi abbiamo su un sistema operativo moderno e su un address space di un'applicazione che è attiva su un S.O moderno, è la seguente cosa: Una pagina all'interno di una address space è classicamente per default NON MAPPATA, poi può diventare mappata e quindi diventare valida per essere utilizzata, ma la mappatura prevede che l'oggetto è utilizzabile all'interno dell'address space, in realtà potrebbe non essere ancora materializzato in RAM. Ma quindi per accedere realmente dobbiamo "materializzare" queste informazioni in RAM.



Però da NON MAPPATA, POSSIAMO ANDARE SUBITO A MATERIALIZZARE E MAPPARE IN RAM. Questo implica dire che la Page Table è stata mandata in set-up per portarci esattamente sul FRAME fisico corrispettivo.
Una pagina NON MAPPATA diventa anche Mappata, ma non (materializzata) presente in memoria fisica, poi magari lo diventerà, ma poi attenzione, la pagina mappata e materializzata può essere di nuovo MAPPATA e quindi utilizzabile ma senza essere materializzata, questo è il classico scenario di uno SWAP OUT di un'applicazione.
Da mappata e materializzata --> mappata ma non materializzata, parliamo di SWAP OUT.

Lo swap out di un'applicazione prevede che noi portiamo fuori memoria tutte le pagine, che erano presenti anche in ram, e vengono portate fuori.