Memoria Virtuale

Con questo termine si intende la possibilità di gestire la memoria in modo molto più strutturato ed ordinato rispetto all'idea originale di Von Neumann. Se da un lato la disponibilità di un'unica RAM nella quale si possono mischiare a piacere dati e codice è concettualmente semplice e teoricamente molto potente, in pratica un uso disordinato della memoria da parte dei programmi è fonte inesauribile di errori di programmazione oltre che un grave impedimento per la possibilità di ottimizzare l'uso della memoria in modo automatico.

La memoria virtuale realizza la tale funzionalità: fa "vedere" al programma in esecuzione un insieme di moduli di memoria distinti e con diverse caratteristiche di accesso ai dati, supportate da dispositivi fisici specializzati, mentre continua ad utilizzare i soliti moduli RAM a livello fisico per questioni di economia di realizzazione.

*Segmentazione*

Tecnica più semplice per realizzare un meccanismo di memoria virtuale.

viene definito un insieme di segmenti, che possono essere immaginati come moduli di memoria distinti, ciascuno in grado di supportare modi di accesso specializzati e ciascuno caratterizzato dal suo proprio "spazio di indirizzamento" indipendente.

Esempio: definiamo i segmenti:

- Segmento Codice: destinato a contenere la codifica binaria delle istruzioni del programma da eseguire (quindi la CPU farà accesso a questo segmento, intuitivamente, solo nella fase di fetch, e solo in lettura)
- Segmento Stack: destinato a contenere celle di memoria da usarsi per la realizzazione dello stack, memorizzazione di dati locali di una procedura (routine), passaggio dei parametri,  salvataggio dei valori contenuti nei registri della CPU, ecc. La CPU vi potrà accedere solo nella fase di esecuzione di alcune istruzioni, sia in lettura che scrittura.
- Segmento Dati statici: destinato a contenere tutti gli altri dati del programma non memorizzati nello stack, e la CPU potrà sensatamente accedere a questo segmento solo durante la fase di esecuzione di altre istruzioni (per esempio le istruzioni Load/Store con indirizzamento diretto).
- Ci può essere anche il segmento HEAP (nei moderni sistemi POSIX)

**Segmentazione implicita**

i segmenti e la loro utilizzazione sono definiti una volta per sempre a livello di macchina convenzionale. Questo fa sì che non ci sia bisogno, nella maggior parte dei casi, di dire esplicitamente a quale segmento ci si riferisce: é la natura stessa dell'operazione che stiamo eseguendo che implicitamente definisce anche il segmento su cui dobbiamo operare.

**Segmentazione esplicita**

In questo caso a livello di macchina convenzionale specifichiamo solo che possiamo usare un insieme di segmenti dicendo qual é il massimo numero di segmenti che vogliamo usare. Sarà cura del programmatore stabilire quanti e quali segmenti usare effettivamente e quali modalità di accesso consentire o vietare per i diversi segmenti. Quindi bisognerà dichiarare esplicitamente il numero del segmento al quale si vuole accedere, tipicamente si suddivide la rappresentazione binaria dell'indirizzo di una cella in due sottoinsiemi di bit: un sottoinsieme rappresenta il numero del segmento al quale si vuole accedere, l'altro rappresenta il vero e proprio indirizzo della cella di memoria all'interno del tale segmento.

La struttura dei moduli RAM a livello fisico non cambia, quindi per realizzare un meccanismo di segmentazione (implicita o esplicita) occorre aggiungere dei dispositivi in grado di tradurre indirizzi virtuali (detti anche indirizzi logici) usati dal programma in esecuzione sulla CPU in indirizzi fisici usati sul bus per accedere alla RAM.

Nel caso della segmentazione implicita l'unico problema da risolvere è trovare un metodo semplice per tradurre un qualunque indirizzo compreso tra 0 e n-1 (con n dimensione del segmento considerato) in un indirizzo fisico di RAM.

Soluzione adottata: coppia di registri associata ad ogni segmento, registro base e registro limite. Il registro base contiene l'indirizzo della cella di RAM corrispondente alla cella di indirizzo logico 0 del segmento. Il registro limite contiene l'indirizzo della cella di RAM corrispondente alla cella di indirizzo logico N-1 del segmento (ossia l'ultima).

Traduzione:

Prima si somma all'indirizzo logico il contenuto del registro base, ottenendo l'indirizzo fisico corrispondente; poi si confronta l'indirizzo fisico col contenuto del registro limite, e se l'indirizzo fisico é maggiore del limite allora si genera una condizione di errore (si segnala una Trap).

Tipicamente quando si adotta la segmentazione esplicita nella definizione di una macchina convenzionale si dedica un numero congruo di bit alla rappresentazione del numero di segmento, in modo da non porre troppi vincoli al programmatore e dargli la possibilità di definire anche centinaia o migliaia di segmenti.

La traduzione da indirizzo logico a indirizzo fisico può essere realizzata in modo del tutto analogo a come abbozzato per la segmentazione implicita, con l'unica differenza che si preferisce normalmente organizzare le coppie di "registri" (base e limite) in tabelle ordinate per numero di segmento, per facilitare la gestione di un numero normalmente maggiore di segmenti. (Dovrò avere allocata da qualche parte in memoria una tabella di 4 elementi che mi indichi tutte le informazioni utili riguardanti i segmenti, come gli indirizzi minimi e massimi, la dimensione, ecc…, chiamata Tabella dei segmenti.)

Per effettuare una traduzione da virtuale a fisico quindi controlliamo prima se, secondo questa tabella, l'indirizzo fornito dalla CPU sia valido, utilizzando un comparatore interno all'MMU che ci confronti il valore dato dalla CPU con gli indirizzi di confine del segmento.

Se il confronto è positivo, utilizzo un sommatore interno all'MMU per fare la somma tra indirizzo virtuale e indirizzo di partenza del segmento, producendo l'indirizzo fisico.

Se invece il confronto fosse negativo, l'MMU andrebbe a filtrare l'indirizzo virtuale, bloccandolo

## Paginazione

La RAM viene "suddivisa" in 2^n pagine di dimensione predefinita. L'indirizzo virtuale e fisico sono suddivisi in due parti: offset (bit meno significativi dell'indirizzo) e numero di pagina (bit più significativi dell'indirizzo).

Tale suddivisione può essere operata sia per l'indirizzo logico, sia per l'indirizzo fisico corrispondente. Nel primo caso si parla quindi di "numero di pagina logica", nel secondo di "numero di pagina fisica".

Il numero di pagina servirà per individuare la pagina corrispondente nella RAM, l'offset invece è la distanza tra l'indirizzo della prima cella di questa pagina e quello della cella a cui vogliamo accedere.

La traduzione da indirizzo logico ad indirizzo fisico avviene cambiando il numero di pagina (al numero di pagina logica si sostituisce il numero di pagina fisica corrispondente), facendo riferimento ad una funzione, definita mediante un vettore (array) solitamente chiamato "tabella delle pagine" (page table).

L'offset viene ricopiato tale e quale dall'indirizzo logico all'indirizzo fisico nella fase di traduzione, quindi non si usa fare nessuna distinzione di nome tra offset dell'indirizzo logico e offset dell'indirizzo fisico.

Un meccanismo di paginazione come quello delineato non viene solitamente utilizzato da solo, ma come complemento per la realizzazione efficiente di una memoria virtuale a segmentazione.

L'accoppiamento di segmentazione e paginazione permette infatti di realizzare una memoria virtuale più semplice da gestire in modo efficiente, nella quale i segmenti sono sempre costituiti da un insieme di pagine complete.

Il vantaggio derivante dall'uso della paginazione é che le pagine fisiche libere da aggiungere al segmento possono essere associate a numeri di pagina logica consecutivi a quelli già esistenti nel segmento anche se i numeri di pagina fisica non lo sono, senza costose operazioni di "spostamento" in memoria dei dati contenuti in pagine usate.

Implementare paginazione e segmentazione contemporaneamente è la scelta che migliora di più l'efficienza.

Per fare ciò utilizzeremo indirizzi virtuali divisi in 3 parti, ovvero offset, numero di pagina e numero di segmento.

Per effettuare la traduzione, l'MMU avrà bisogno di riferirsi a due tabelle quindi, quella dei segmenti e quella delle pagine, divisi in due livelli. Infatti l'idea è quella di avere delle macro aree, i segmenti, che siano divise in tante parti, ovvero le pagine. Sulla tabella dei segmenti avremo quindi la dimensione dei segmenti, ovvero il numero di pagine al loro interno, che sarà anche il numero di elementi della tabella delle pagine, e un'informazione che ci farà individuare la tabella delle pagine.

Una realizzazione inefficiente della traduzione degli indirizzi avrebbe un impatto devastante sulla velocità di esecuzione dei programmi con indirizzamento virtuale

rispetto a quelli con indirizzamento fisico. Un dispositivo che ci aiuterà molto nel processo di traduzione è l'MMU (unità di gestione della memoria).

All'interno della MMU viene introdotta una memoria associativa, la presenza di essa consente di "copiare" parte delle informazioni contenute nelle tabelle di paginazione e di segmentazione contenute in RAM.

Si parte "a freddo" con la MMU che non contiene alcuna informazione; la prima volta che viene richiesta la traduzione di un indirizzo, la MMU accede alle tabelle di segmentazione e paginazione per recuperare i dati necessari dalla RAM; questi dati vengono inseriti nella memoria associativa interna sotto forma di associazione tra i bit più significativi dell'indirizzo virtuale e i bit di numero di pagina fisica corrispondente accoppiati con i bit di specifica dei modi di accesso consentiti per il segmento che contiene quella pagina (che verranno usati come dati utili a supporto della traduzione); traduzioni successive di indirizzi facenti riferimento alla stessa pagina potranno così essere completate con un solo accesso alla memoria associativa interna alla MMU anziché con due accessi alle tabelle in RAM. Notare che la "ritraduzione" di un indirizzo facente riferimento alla stessa pagina di un indirizzo già tradotto correttamente non richiede più la verifica sulla dimensione del segmento (che è già stata portata a termine con successo la volta precedente).

## Virtualizzazione della CPU e istruzioni privilegiate

Con tale sistema cerchiamo di risolvere un problema molto importante nell'uso di sistemi di calcolo: quello di garantire l'integrità e la riservatezza delle informazioni che un programma sta trattando, anche in presenza di altri programmi eseguiti sullo stesso sistema che, per errore oppure per deliberato intento aggressivo, attentassero alla integrità e riservatezza di altri programmi.

Una caratteristica molto desiderabile per un sistema operativo é infatti quella di consentire l'accesso a più utenti diversi (contemporaneamente oppure in tempi successivi) agli stessi dispositivi.

Il meccanismo della memoria virtuale a segmentazione é uno dei principali "ingredienti" che consentono al progettista di un sistema operativo di predisporre un ambiente di lavoro sicuro ed affidabile per più utenti che condividano l'accesso allo stesso sistema.

### confinamento

La definizione di segmenti assegnati in uso esclusivo ad un programma e non ad altri consente a tali programmi di far affidamento sul fatto che i dati memorizzati in tali segmenti non potranno essere ne' letti ne' tantomeno cambiati a seguito

dell'esecuzione di altri programmi. La garanzia a tempo di esecuzione viene fornita dall'uso del dispositivo MMU, il quale segnala una trap in caso di tentativi di violazione "dei confini" stabiliti da parte di un qualsiasi programma.

Ciò che si desidera fare è non permettere a “chiunque” di eseguire istruzioni di accesso alle tabelle di segmentazione e paginazione con indirizzamento fisico, questo “sentiero” deve rimanere aperto per consentire al sistema di inserire i valori giusti in tali tabelle, prima di passare lui stesso all’indirizzamento virtuale, è ovvio che la stessa possibilità non deve essere concessa anche ad un programma “utente”, visto che in questo caso il sistema non potrebbe garantire che, l’utente una volta garantitagli la possibilità di accedere alle tabelle, non possa compromettere i dati all’interno di esse o in qualche modo recare danno al sistema.

La soluzione al problema è molto semplice: si definiscono a livello di macchina convenzionale due o più modi di esecuzione dei programmi. Uno di questi modi viene normalmente chiamato "privilegiato", l’altro modo di esecuzione viene normalmente chiamato "non privilegiato" (o “utente”).

Il modo di esecuzione viene memorizzato in uno o più bit dedicati del registro di stato della CPU, quindi viene definita una partizione dell’insieme delle istruzioni in due sottoinsiemi: istruzioni privilegiate e istruzioni non privilegiate.

A livello di realizzazione della macchina si farà in modo che , mentre le istruzioni non privilegiate possano essere sempre eseguite dalla CPU nel solito modo, le istruzioni privilegiate possano essere eseguite dalla CPU solo quando il registro di stato indica il modo di esecuzione privilegiato. Al tentativo da parte di un programma eseguito in modo non privilegiato di eseguire una istruzione privilegiata il sistema controbatte con una Trap (la cui gestione solitamente termina immediatamente il programma).

Ovviamente si provvede poi a definire come privilegiate le istruzioni della macchina convenzionale che usano l'indirizzamento fisico, e come non privilegiate quelle che usano l'indirizzamento virtuale.

Ultimo dettaglio, al momento dell'accensione, il registro di stato viene forzato a contenere l'indicazione di modo di esecuzione privilegiato, in modo da consentire al sistema operativo di eseguire tutte le istruzioni della macchina convenzionale.

Inizialmente, infatti, la RAM è vuota, e le info di cui ha bisogno la CPU per partire devono essere caricate in memoria, dunque nella fase di Bootstrap è necessario che la CPU acceda direttamente alla RAM usando indirizzi fisici, utilizzando appunto alcune delle istruzioni privilegiate. Appena terminata la Bootstrap e caricati i programmi di avvio del sistema, cambiamo il bit di stato della CPU per far sì che le

istruzioni non privilegiate vengano eseguite unicamente usando l'indirizzamento virtuale attraverso l'MMU. La fase di Bootstrap è dunque estremamente delicata in quanto la macchina non è ancora protetta dall'OS attraverso meccanismi di Trap e interruzioni.

Virtualizzazione vera e propria del processore:

nasce a fine anni '60 per esigenze di carattere pratico: l'esigenza che si aveva di far sostenere alla macchina l'esecuzione di più programmi contemporaneamente.

Si basa sull'idea di far finta di avere più macchine a disposizione, ovvero che partendo da una macchina fisica si potesse arrivare a più Macchine Virtuali (MV), identiche all'originale se non per la quantità di risorse utilizzate, suddivise tra le macchine arbitrariamente. Implementa l'utilizzo dei meccanismi di segmentazione e paginazione per più programmi contemporaneamente, e il meccanismo di Trap/Interrupt, che serve in caso le due zone di memoria adibite ai due programmi dovessero interagire fra loro. Ciò ha portato alla tecnologia degli Ipervisori, il meccanismo usato per realizzare macchine virtuali più piccole all'interno di una macchina fisica più grande. Questa progettazione permette che le macchine virtuali siano indipendenti fra loro e dalla macchina fisica, se non per l'uso della memoria, e consente anche l'installazione di OS diversi. Quando facciamo partire una Bootstrap in una macchina virtuale, dunque a macchina fisica già avviata, chiediamo alla CPU di eseguire un'istruzione privilegiata quando si trova in modalità non privilegiata. La Trap che consegue questa incongruenza fa sì che l'ipervisore controlli che il programma che sta venendo avviato non esca dalla zona di memoria consentita, e, nel caso affermativo, simula il comportamento che la CPU ha per la macchina fisica, per eseguire le istruzioni privilegiate per eseguire la Bootstrap della macchina virtuale. L'istruzione non è stata quindi eseguita dal processore fisico, ma simulata dall'ipervisore, questa simulazione tuttavia rallenta i programmi, anche se consente l'esecuzione simultanea di più programmi contemporaneamente.

L'ipervisore dunque prende il ruolo del "sistema operativo" e anche della cpu, per la macchina virtuale, esso "fa vedere" al sistema operativo una macchina virtuale che è del tutto identica alla macchina fisica.