Abbiamo visto due tecniche di virtualizzazione della memoria: la tecnica di segmentazione, che ci permette di differenziare aree di memoria specializzandole per determinati compiti, e quella di impaginazione, che ci permette di usare efficientemente la memoria RAM tramite l’assegnazione di gruppi di celle contigue dette “pagine” della cui dimensione i segmenti devono essere multipli. Nella versione esplicita, su una rappresentazione a 32 bit i primi 2 bit individuano il segmento, i successivi 20 il numero di pagina e gli ultimi 10 l’offset. Questo ci permette di avere 4 tipi di segmenti di dimensione arbitraria purché multiplo di 2^10 (1024).

La quantità di pagine che si può far stare nella RAM è uguale alla dimensione totale della RAM meno la porzione dedicata alle tabelle di impaginazione (che permettono la traduzione tra numero di pagina virtuale e numero di pagina fisica) diviso la dimensione di una pagina.

Il bit di offset deve essere uguale tra indirizzo virtuale e indirizzo fisico. Il numero di bit dedicati al numero di pagina fisico è vicino al numero di quelli dedicati al virtuale, ma non è detto che sia lo stesso, dipende dal sistema di calcolo.

Abbiamo visto che le tabelle consistono in:

una tabella dei segmenti, che contiene i tipi di segmenti che possono essere indirizzati (nel nostro esempio sono 4) e per ciascuno di essi si ha una tabella delle pagine. Quindi in ogni tabella dei segmenti, in ogni segmento sono presenti la dimensione della tabella delle pagine e un puntatore al punto della RAM in cui essa si trova. Nella tabella delle pagine si va poi a cercare il numero di pagina fisica usando come indice il numero di pagina virtuale.

Aggiungiamo qualche piccolo dettaglio. Nella tabella dei segmenti ogni elemento, oltre a contenere le dimensioni della tabella e il suo puntatore, può avere al suo interno informazioni interiori che rendano più efficiente il meccanismo di traduzione: vengono aggiunti i bit di Permesso di Accesso. Questo tipo di bit rappresentano per esempio il permesso di accesso in lettura o scrittura, quindi se viene violato tale permesso si sollevano delle trap, un esempio si può avere per quanto riguarda la protezione delle pagine dedicate al codice contro la modifica in scrittura. Tipicamente i permessi sono: Il permesso di READ, di WRITE, e di EXEC (quest’ultimo però vuol dire “poter fare il fetch”) L’MMU può tenere conto di un bit che esce dalla CPU durante la fase di fetch per differenziare i due tipi di accesso da parte del processore.

Ci potrebbero essere ulteriori bit che diano altre informazioni, a supporto di un meccanismo di caching. Sempre per motivi di efficienza un’altra informazione potrebbe essere associata alla tabella delle pagine, che oltre al campo per la pagina fisica potrebbe avere dei bit di informazione aggiuntiva che permetteranno di agevolare l’implementazione efficiente, da parte dell’MMU, del meccanismo di impaginazione. Infatti come si fa ad organizzare un sistema operativo in modo da manipolare e aggiustare le dimensioni del segmento se non si rivelassero sufficienti? Abbiamo già visto che è possibile farlo con il segmento Stack, attraverso il meccanismo delle trap (che anziché causare un aborto del programma fan sì che il trap handler aumenti la memoria dedicata allo Stack). Per poter fare ciò, e quindi avere una gestione dinamica, è necessario avere i bit di informazioni aggiuntive nelle tabelle d’impaginazione. Innanzitutto, un bit potrebbe indicare se una pagina è stata effettivamente usata (o in lettura o in scrittura), un altro potrebbe indicare (se ci sono stati degli accessi) se c’è stato almeno un accesso in scrittura, questo perché il meccanismo di impaginazione può essere associato anche a una memoria di massa (per esempio le unità a disco), oltre che alla RAM.

Una caratteristica delle unità a disco è la capacità di immagazzinare molti più dati rispetto a una memoria RAM. Un’idea sarebbe quella di estendere la quantità di memoria disponibile per i nostri programmi e quindi di utilizzare una certa porzione dell’unità a Disco per contenere le Pagine e di riportare in memoria RAM un sottoinsieme di tali Pagine. In tal modo si potrebbero lanciare in esecuzioni più programmi che in totale richiedano di occupare molto più dello spazio presente nella memoria RAM e di tenere temporaneamente nell’unità a Disco le Pagine che non possono/non devono ancora essere utilizzate e caricarle in RAM solo quando si libera abbastanza spazio. Quest’area del disco è detta SWAP AREA.

Se non ci fosse abbastanza spazio nella RAM al momento del caricamento di una Pagina dall’area di Swap, si cerca una pagina di cui al momento si può fare a meno, copiarla nello Swap e sostituirla con la pagina che si voleva inserire inizialmente. Questo processo può avvenire però SOLO se sulla pagina che si vuole mandare in Area di Swap non è stata fatta neanche un’operazione di scrittura (è “DIRTY”), questo perché se sui due bit di informazione aggiuntiva troviamo 00 allora quella pagina probabilmente è attualmente inutile e la si può sovrascrivere, ma se sono state tutte usate (quindi sono del tipo 10 o 11) si dà priorità a sovrascrivere quelle su cui NON è stata fatta alcuna scrittura, perché così il valore salvato nel Disco (cioè la Vecchia copia) corrisponde ancora al valore contenuto nella pagina attuale. Se si sostituisse una pagina che è stata usata e in modalità scrittura questo richiederebbe di aggiornare la corrispondente pagina nello SWAP prima di poterla sovrascrivere.

Il gestore delle Trap, così come il resto dei programmi dedicati al sistema operativo (e alle tabelle), si trovano in una porzione della RAM che non possono far parte della suddivisione tramite impaginazione. Le Unità a disco sono memorie di tipo Non Volatile, poiché sono in grado di mantenere le informazioni anche in assenza di corrente, a differenza della memoria RAM, che è invece una memoria Volatile.

Possiamo infine aggiungere un terzo bit di informazione che segnala se la pagina è presente in memoria RAM oppure no, tramite questo bit in più possiamo costruire una tabella già completa (che non ha bisogno di aggiustamenti). Se il valore del bit è 0 allora la pagina si trova nell’area di Swap.

La memoria RAM, così, diventa qualcosa di “opzionale”, sebbene sia indispensabile (poiché il processore può accedere solo ai contenuti nella memoria RAM). Tuttavia, la memoria a Disco permette che non tutto il segmento necessario a un programma sia presente contemporaneamente in memoria RAM. Il modo per ottenere tale risultato è utilizzare un particolare metodo di impaginazione, la impaginazione A Richiesta.

Per poter fare ciò si può inizializzare le tabelle come se la memoria RAM fosse vuota, ossia ponendo tutti i bit di presenza in RAM come = 0. Quando la CPU manda per la prima volta un indirizzo alla MMU, questa va a vedere nella tabella di impaginazione e scopre che tale pagina non è presente nella RAM, allora manda una Trap. Viene interrotta l’esecuzione del programma e parte l’esecuzione del gestore delle Trap, che vede che manca la pagina in memoria e per questo motivo ordina che venga caricata la copia principale della Pagina voluta dall’area di Swap a una cella libera della RAM (quindi è un sistema che costruisce una Richiesta Implicita tramite il gestore delle Trap). Il bit di presenza viene quindi impostato a 1 e si riprende l’esecuzione del programma dopo aver eseguito l’istruzione RTT. A questo punto L’MMU trova l’indirizzo di presenza = 1 e può svolgere il suo compito. Quando si esce dalla pagina allocata e si passa a un’altra con bit di presenza = 0 il processo iniziale si ripete e viene caricata (su Richiesta) la Pagina voluta dalla memoria a Disco alla memoria RAM.

Si continua così finché si trovano pagine vuote all’interno della RAM. Dal momento in cui tutte le pagine sono state riempite in poi, si procede utilizzando gli altri due bit (di utilizzo e di “dirty”) per capire quale pagina (eventualmente copiare nell’area di Swap e poi) sostituire con quella voluta.

Sull’unità disco non esistono esattamente le pagine, esiste però il concetto di “blocco” che è analogo a quello di pagina nella RAM. Il blocco può avere la stessa dimensione della pagina o no, questo è a discrezione del sistema operativo: alternativamente uno sarebbe un multiplo dell’altro (1 blocco = 2 pagine o 1 pagina = 2 blocchi ecc.).

Oltre alla impaginazione A Richiesta si potrebbero usare algoritmi di rimpiazzamento un pochino più avanzati: all’interno di memorie Cache, infatti, vengono usati questi algoritmi alternativi.

Presentiamo l’algoritmo LRU (LRU = Usata meno recentemente): I programmi hanno un insieme di caratteristiche di tipo statistico che possono essere misurate, una di queste viene chiamata “località nel tempo”. Se guardo gli indirizzi che un processore genera, questi indirizzi non sono casuali, quindi può capitare che per certi periodi di tempo una pagina non venga utilizzata (e questo è il motivo per cui l’impaginazione a Richiesta/ Demand Paging funziona), ma dopo un po’ può succedere che le cose cambino e che venga generato l’indirizzo di una pagina che prima non era mai stata utilizzata: secondo l’impaginazione a richiesta questa pagina viene caricata e dopo essere stata caricata, se gode della caratteristica della località nel tempo, verrà riutilizzata a breve e più volte (perché una pagina che viene usata verrà probabilmente usata anche nel futuro immediato) quindi è conveniente tenerla lì. Poi il processore cambia pagina e quella pagina rimane inutilizzata per una certa quantità di tempo. Se a questo punto la memoria è piena e bisogna caricare una nuova pagina, per scegliere quale pagina sostituire si usa l’algoritmo LRU, che trova la pagina che non è stata usata per più tempo (e che quindi è meno probabile che sia riusata nei prossimi minuti) e rimpiazza quella. C’è quindi quest’idea del Working Set, secondo cui tra tutte le pagine di un programma il nostro processore sta lavorando solo su un sottoinsieme di esse, c’è pertanto stabilità nell’uso della memoria, e se si riesce a determinare quale è tale sottoinsieme basta mantenere caricato in RAM quello. L’algoritmo LRU è l’algoritmo che si comporta meglio dal punto di vista empirico, ma è TREMENDAMENTE COSTOSO da realizzare, perché anziché uno o due bit bisognerebbe inserire come aggiuntivi nelle tabelle di impaginazione anche 30/40 bit PER OGNI PAGINA che contengano la data e l’ora dell’ultimo utilizzo. Si può utilizzare quindi un’approssimazione che anziché usare tutti i bit necessari per una data si sfrutti soltanto il bit di uso: si programma un timer che di tanto in tanto invia una interrupt al processore che fa sì che il gestore delle interruzioni (dopo aver capito che si tratta dell’interruzione del timer) va ad azzerare tutti i bit di utilizzo all’interno della tabella. Da quel punto in avanti quel bit verrà riportato a 1 solo e quando si avrà un accesso in lettura o scrittura a quella pagina. Se ci sono delle pagine che non sono state utilizzate per un po’ di tempo tipicamente si troverà il bit di uso a 0. Non è detto però che una pagina con bit di uso a 0 non stia venendo utilizzato: potrebbe essere letteralmente appena partita l’interrupt del timer. Però gestendosi bene i tempi, si può massimizzare la probabilità di scegliere pagine non in uso. Ovviamente questo algoritmo, poiché è un’approssimazione, può sbagliare: in quel caso pazienza, basta che ci azzecchi nella maggior parte dei casi, dato che è sufficiente ricaricare la pagina in RAM.

Questo è uno dei casi in cui il bit Used può essere 0 mentre il bit Dirty è = 1 : ciò segnala che la pagina non è stata usata per un po’ di tempo, ma che è stata scritta almeno una volta, quindi prima di sovrascriverla il corrispondente blocco/blocchi nella memoria a Disco vanno aggiornati (facendo una copia della pagina).

Si può avere una situazione di TRASHING se si sta perdendo un sacco di tempo per fare copie da RAM a Disco e viceversa (Ossia l’algoritmo ti dice che nessuna pagina è da rimpiazzare, perché dall’ultimo azzeramento del timer sono state referenziate tutte, ma tu ne devi continuare ad aggiungere) : In genere accade se non si ha abbastanza RAM per sostenere tutti i Working Set (o c’è un solo Working Set ma che è troppo grande per essere gestito dalla macchina) che ho caricato in memoria. Una contromisura che si può applicare è lo “Swap-Out”, che consiste nel buttare fuori uno o più programmi in esecuzione (dopo averne copiato le pagine nell’area di Swap) bloccandone l’esecuzione.