**Parte II - Capitolo 9 - Memoria Virtuale**

La ***memoria virtuale*** è una tecnica che permette di eseguire processi che possono anche non essere completamente contenuti in memoria. Il vantaggio principale offerto è quello di permettere che i programmi siano più grandi della memoria fisica; inoltre la memoria virtuale astrae la memoria centrale in un vettore di memorizzazione molto grande e uniforme, separando la memoria logica da quella fisica. La memoria virtuale permette inoltre ai processi di condividere facilmente file e spazi d'indirizzi, e fornisce un meccanismo efficiente per la creazione dei processi. La memoria virtuale si fonda sulla separazione della memoria logica percepita dall'utente. L'espressione ***spazio degli indirizzi virtuali*** si riferisce alla collocazione dei processi in memoria dal punto di vista logico (o virtuale). Il fatto che la memoria virtuale offre ai processi la possibilità di condividere le pagine, comporta diversi vantaggi: le librerie di sistema sono condivisibili associando l'oggetto condiviso a uno spazio d'indirizzi virtuali (***mappatura***), i processi possono condividere la memoria, per mezzo della chiamata di sistema fork() consente che le pagine siano condivise durante la creazione di un processo.

Si consideri il caricamento in memoria di un eseguibile residente su un disco. Una possibilità è quella di caricare l'intero programma nella memoria fisica al momento dell'esecuzione. Il problema che ne risulta è che non sappiamo se abbiamo bisogno di tutto il programma all'avvio. Una strategia alternativa consiste nel caricare le pagine nel momento in cui servono realmente; si tratta di una tecnica, detta ***paginazione su richiesta*** (*demand paging*), comunemente adottata dai sistemi con memoria virtuale. Secondo questo schema, le pagine sono caricate in memoria solo quando sono richieste durante l'esecuzione del programma: ne consegue che le pagine cui non si accede mai non sono mai caricate nella memoria fisica. I processi risiedono in memoria secondaria, quindi, per eseguire un processo, occorre caricarlo in memoria. Tuttavia, anzichè caricare in memoria l'intero processo, si può seguire un criterio d'avvicendamento pigro (*lazy swapping*): non si carica mai in memoria una pagina che non sia necessaria. Con questa tecnica non si manipolano interi processi, ma singole pagine di processi. Nell'ambito della paginazione su richiesta, il modulo del sistema operativo che si occupa della sostituzione della pagine si chiama ***paginatore*** (*pager*). Quando un processo sta per essere caricato in memoria, il paginatore ipotizza quali pagine saranno usate e le trasferisce in memoria. Con tale schema è necessario che l'architettura disponga di un qualche meccanismo che consenta di distinguere le pagine presenti in memoria da quelle nei dischi. A tal fine è utilizzabile lo schema basato sul bit di validità: se il bit è impostato a valido la pagina corrispondente è valida ed è presente in memoria; se il bit è impostato a non valido la pagina non è valida oppure è valida, ma attualmente nel disco. Durante la sua esecuzione, un processo accede alle pagine residenti in memoria. Se, invece, tenta l'accesso a una pagina che non era stata caricata in memoria, l'accesso a una pagina contrassegnata come non valida causa un'***eccezione di pagina mancante*** (*page-fault trap*). Tale eccezione è dovuta a un insuccesso del sistema operativo nella scelta delle pagine da caricare in memoria. La procedura di gestione della pagina mancante corrisponde ai seguenti passi: si controlla la tabella interna per questo processo per stabilire se il riferimento è un accesso valido o non valido alla memoria; se il riferimento è non valido si termina il processo, se il riferimento è valido, ma la pagina non era ancora stata portata in memoria, se ne effettua l'inserimento; si individua un frame libero; si programma un'operazione sui dischi per trasferire la pagina desiderata nel frame appena assegnato; si modifica la tabella interna non appena la lettura del disco è completata; si riavvia l'istruzione interrotta dal segnale di eccezione. È possibile avviare l'esecuzione di un processo senza pagine in memoria. Quando il sistema operativo carica nel contatore di programma l'indirizzo della prima istruzione del processo, questo accusa un'assenza di pagina. Una volta portata la pagina in memoria, il processo continua l'esecuzione subendo assenze di pagine fino a che tutte le pagine necessarie non si trovano effettivamente in memoria. Questo schema è detto ***paginazione su richiesta pura***. La paginazione su richiesta deve essere aiutata nello svolgere la sua funzione dalla tabella delle pagine, che contrassegna un elemento secondo il bit di validità, e la memoria secondaria, che conserva le pagine non presenti in memoria centrale (la sezione usata per questo scopo è detta ***area d'avvicendamento*** o ***area di scambio*** o *swap space*). Per comprendere come la paginazione su richiesta possa avere un effetto rilevante sulle prestazioni di un calcolatore, basta calcolare il ***tempo d'accesso effettivo*** per una memoria che fa uso di questa tecnica. Definendo *ma* come il tempo d'accesso alla memoria e *p* come la probabilità che si verifichi un'assenza di pagina, allora:

Il tempo d'accesso effettivo è direttamente proporzionale alla ***frequenza delle assenze di pagine*** (*page-fault rate*), quindi è importante tenere basso questo valore per non rallentare di molto l'esecuzione di un processo.

La chiamata di sistema fork() crea un processo figlio come duplicato del genitore. Considerando che molti processi figli eseguono subito dopo la loro creazione la chiamata di sistema exec(), questa operazione di copiatura risulta inutile. In alternativa, si può impiegare una tecnica nota come ***copiatura su scrittura*** (***copy on write***), il cui funzionamento si fonda sulla condivisione iniziale delle pagine da parte dei processi genitore e dei processi figli. Le pagine condivise si contrassegnano come pagine da copiare su scrittura: se un processo scrive su una pagina condivisa, il sistema deve creare una copia di tale pagina. È chiaro che si copiano soltanto le pagine modificate da uno dei processi, mentre le altre sono condivisibili dai processi genitore e figlio. Si noti inoltre che solo le pagine modificabili si devono contrassegnare come da copiare su scrittura, mentre quelle che non si possono modificare sono condivisibili dai processi genitore e figlio. Diverse versioni di UNIX offrono una chiamata di sistema vfork() che consente una copiatura su scrittura: il processo genitore viene sospeso e il processo figlio usa lo spazio d'indirizzi del genitore. Se il processo figlio modifica qualche pagina dello spazio d'indirizzi del genitore, le pagine modificate saranno visibili al processo genitore non appena riprenderà il controllo. Questa chiamata è molto efficiente, ma è da utilizzare con molta attenzione.

Nei sistemi descritti finora, la frequenza delle assenze di pagine non è stata un problema grave, perchè ogni pagina poteva essere assente al massimo una volta. Questa rappresentazione non è molto precisa. Se durante l'esecuzione di un processo viene aumentato il grado di multiprogrammazione, si può osservare che la memoria di sovrassegna. La ***sovrallocazione*** (*over-allocation*) è un problema che si può illustrare come segue: se durante l'esecuzione di un processo utente si verifica l'assenza di una pagina, il sistema operativo deve determinare la locazione del disco in cui risiede la pagina desiderata. A questo punto scopre che la lista dei frame liberi è vuota, cioè tutta la memoria è in uso. A questo punto il sistema operativo può scegliere tra diverse possibilità: terminare il processo utente (non è ottimale in quanto la paginazione su richiesta è fatta per migliorare l'utilizzo e la produttività del sistema di calcolo) oppure scaricare dalla memoria l'intero processo utente, liberando tutti i suoi frame e riducendo il livello di multiprogrammazione. Analizziamo ora una possibilità più interessante. La ***sostituzione delle pagine*** (*page replacement*) segue il seguente criterio: se nessun frame è libero, si può liberarne uno attualmente inutilizzato. È possibile liberarlo scrivendo il suo contenuto nell'area di avvicendamento e modificando la tabella delle pagine per indicare che la pagina non si trova più in memoria. Il frame liberato si può usare per conservare la pagina che ha causato l'eccezione. La procedura di servizio dell'eccezione di pagina mancante viene modificata in tal modo: si individua la locazione nel disco della pagina richiesta, si cerca un frame libero (se esiste si usa, altrimenti si sceglie una ***vittima*** che si scrive nel disco), si scrive la pagina richiesta nel frame appena liberato e si riavvia il processo utente. Si può notare che il tempo effettivo d'accesso aumenta in quanto devono essere effettuati due trasferimenti di pagina. Questo sovraccarico si può ridurre usando un ***bit di modifica*** (*modify bit* o *dirty bit*) che è associato a ogni pagina (o a un frame) e indica se questa è stata modificata. Quando si esamina una pagina da sostituire si esamina il suo bit di modifica: se è attivo la pagina è stata modificata e quindi deve essere scritta nel disco; se non è attivo la pagina non è stata modificata quindi è già presente nel disco e non deve essere scritta nuovamente. Per realizzare la paginazione su richiesta è necessario risolvere due problemi importanti: occorre realizzare un ***algoritmo di allocazione dei frame*** e un ***algoritmo di sostituzione delle pagine***. Valutiamo ora diversi algoritmi per la sostituzione delle pagine. L'algoritmo di sostituzione delle pagine più semplice è un ***algoritmo FIFO***. Questo algoritmo associa a ogni pagina l'istante di tempo in cui quella pagina è stata portata in memoria. Se si deve sostituire una pagina, si seleziona quella presente in memoria da più tempo. Quando si carica una pagina in memoria, la si inserisce come ultimo elemento in coda. Occorre notare che il numero delle assenze di pagine per n frame può essere maggiore delle assenze di pagine per n-1 frame. Questo inatteso risultato è noto come ***anomalia di Belady*** e riflette il fatto che, con alcuni algoritmi di sostituzione delle pagine, la frequenza delle assenze di pagine può aumentare con l'aumentare del numero di frame assegnati. In seguito a questa osservazione, la ricerca si è diretta verso un ***algoritmo ottimale di sostituzione delle pagine***. Tale algoritmo è quello che fra tutti gli algoritmi presenta la minima frequenza di assenza di pagine e non presenta mai l'anomalia di Belady. Questo algoritmo è stato chiamato OPT o MIN, e semplicemente sostituisce la pagina che non si userà per il periodo più lungo di tempo. Sfortunatamente questo algoritmo non è realizzabile perchè bisognerebbe conoscere la successione futura degli eventi. Dato che l'algoritmo ottimale non è realizzabile, si cerca di realizzarne un'approssimazione. L'***algoritmo LRU*** (*least recently used*) sostituisce la pagina che non è stata usata per il periodo più lungo. Questo algoritmo associa ad ogni pagina l'istante in cui è stata usata per l'ultima volta. Il problema principale di quest'algoritmo riguarda la realizzazione della sostituzione, che può richiedere una notevole assistenza da parte dell'architettura del sistema di calcolo. Il problema consiste nel determinare un ordine per i frame definito secondo il momento dell'ultimo uso. Si possono realizzare due soluzioni. Nel caso più semplice, a ogni elemento della tabella delle pagine si associa un campo del momento d'uso, e alla CPU si aggiunge un *contatore* che si incrementa a ogni riferimento alla memoria; ogni volta che si fa un riferimento a una pagina, si copia il contenuto del registro contatore nel campo del momento d'uso della tabella relativa a quella specifica pagina. Un metodo alternativo prevede la presenza di una *pila* dei numeri delle pagine: ogni volta che si fa riferimento a una pagina, la si estrae dalla pila e la si colloca in cima a quest'ultima. Esiste una classe di algoritmi di sostituzione delle pagine chiamati ***algoritmi a pila*** che non presenta l'anomalia di Belady. In questi algoritmi è possibile mostrare che l'insieme delle pagine in memoria per n frame è sempre un sottoinsieme dell'insieme delle pagine che dovrebbero essere in memoria per n+1 frame. Dato che l'algoritmo LRU è molto oneroso, molti sistemi offrono un algoritmo di sostituzione delle pagine che ***approssima LRU***. Viene fornito un ***bit di riferimento*** (*bit reference*) a una pagina, impostato automaticamente dall'architettura del sistema, ogni volta che si fa un riferimento a quella pagina. Se si inizia l'esecuzione di un processo, l'architettura del sistema imposta a 1 il bit associato a ciascuna pagina cui si fa riferimento. Dopo qualche tempo è possibile stabilire quali pagine sono state usate esaminando il bit di riferimento, senza però conoscere l'ordine d'uso. Ulteriori informazioni sull'ordinamento si possono ottenere registrando i bit di riferimento a intervalli regolari: è possibile conservare in una tabella in memoria una serie di bit per ogni pagina; a intervalli regolari, un segnale d'interruzione del timer trasferisce il controllo al sistema operativo che sposta il bit di riferimento della pagina nel bit più significativo della sequenza, traslando gli altri a destra. Le successioni di bit possono essere interpretate come interi senza sengo e, dopo 8 periodi di tempo, la pagina cui è associato il numero minore è la pagina LRU e può essere sostituita. L'algoritmo di sostituzione ***con seconda chance*** (algoritmo orologio) è di tipo FIFO e funziona nel seguente modo: dopo aver selezionato una pagina si controlla il bit di riferimento, se il suo valore è 0 si sostituisce la pagina, se è 1 si da una seconda chance alla pagina e la selezione passa alla successiva pagina FIFO. Quando una pagina riceve una seconda chance, si azzera il suo bit di riferimento e si aggiorna il suo istante d'arrivo al momento attuale. In questo modo, una pagina cui si offre una seconda chance non viene mai sostituita finchè tutte le altre pagine non sianostate sostituite, oppure non sia stata offerta loro una seconda chance. Un metodo per realizzare un algoritmo con seconda chance è basato sull'uso di una coda circolare. Questo algoritmo può essere migliorato considerando il bit di riferimento e modifica come una coppia ordinata, quindi, in ordine dal migliore da sostutire, abbiamo: (0,0) nè usato, nè modificato di recente; (0,1) non usato recentemente, ma modificato; (1,0) usato recentemente, ma non modificato; (1,1) usato recentemente e modificato. Esistono molti altri algoritmi per la sostituzione delle pagine. Ad esempio si può utilizzare un contatore del numero di riferimenti fatti a ciascuna pagina in modo da poter avere due schemi differenti: ***algoritmo di sostituzione delle pagine meno frequentemente usate*** (*least frequently used*, *LFU*), che sostituisce la pagina con conteggio più basso, e ***algoritmo di sostituzione delle pagine più frequentemente usate*** (*most frequently used*, *MFU*), basata sul fatto che la pagina con il contatore più basso è stata appena inserita, quindi non è stata ancora usata. Questi algoritmi non sono molto comuni perchè la loro realizzazione è molto onerosa. Oltre a uno specifico algoritmo, per la sostituzione delle pagine si usano spesso altre procedure. Quando si verifica un'assenza di pagina, dopo aver scelto il frame vittima, ma prima che questa sia scritta in memoria secondaria, si trasferisce la pagina richiesta in un frame libero del gruppo (*pool of free frames*). Questa procedura permette al processo di ricominciare l'esecuzione al più presto, senza aspettare che la vittima venga scritta in memoria secondaria. Alcuni sistemi operativi permettono a certi programmi di utilizzare una partizione del disco come un array sequenziale di blocchi logici, senza ricorrere alle strutture dati del file system. Un simile array è detto ***disco di basso livello*** (***raw disk***), e il relativo I/O è denominato ***I/O di basso livello*** (***raw I/O***).

A questo punto bisogna stabilire un criterio per l'***allocazione*** della memoria libera ai diversi processi. Le strategie di allocazione dei frame sono soggette a parecchi vincoli. Non si possono assegnare più frame di quanti siano disponibili; inoltre è necessario assegnare almeno un numero minimo di frame. In numero minimo di frame per ciascun processo è definito dall'architettura, mentre il numero massimo è definito dalla quantità di memoria fisica disponibilie. Il modo più semplice per dividere m frame per n processi è quello per cui a ciascuno di da una parte uguale, m/n frame. Questo schema è detto ***allocazione uniforme***. Un'alternativa consiste nel riconoscere che diversi processi hanno bisogno di quantità di memorie diverse a seconda della propria dimensione. Questo schema è detto ***allocazione proprorzionale***. Si può notare che sia con allocazione uniforme, sia con allocazione proporzionale, un processo a priorità elevata è trattato come un processo a bassa priorità anche se, per definizione, si vorrebbe che al processo con alta priorità venga allocata più memoria per accelerarne l'esecuzione. Un altro fattore che riguarda il modo in cui si assegnano i frame ai vari processi è la sostituzione delle pagine. Gli algoritmi si possono classificare in due modi: ***sostituzione globale***, l'algoritmo permette che per un processo si scelga un frame per la sostituzione dall'insieme di tutti i frame, anche se quel frame è correntemente allocato a un altro processo; ***sostituzione locale***, l'algoritmo richiede che per ogni processo si scelga un frame solo dal proprio insieme di frame. Con questa strategia il numero di blocchi di memoria assegnati a un processo non cambia, cosa che invece avviene per la sostituzione globale. Per quanto riguarda i sistemi con processori multipli, i tempi di accesso alla memoria variano in modo significativo tra i vari processori, perchè vi sono differenze di interconnessione tra processori e memoria all'interno del sistema stesso. Questi sistemi vengono detti ***sistemi con accesso non uniforme alla memoria*** (***NUMA***, ***non-uniform memory access***) e, senza eccezioni, sono più lenti dei sistemi nei quali la memoria e i processori risiedono sulla stessa scheda madre. Le prestazioni migliori degli algoritmi di allocazione dei frame sono date dall'allocare il più vicino possibile al processore il processo in esecuzione. I cambiamenti negli algoritmi consistono nel fatto che lo scheduler tiene traccia dell'ultimo processore sul quale ciascun processo è stato eseguito. La questione diventa ancora più complessa con l'aggiunta dei thread. Solaris risolve il problema creando un'entità ***Igroup*** nel kernel, che raccoglie i processori e la memoria vicini tra loro.

Si consideri un qualsiasi processo che disponga di un numero di frame sufficiente. Anche se tecnicamente si può ridurre al valore minimo il numero di frame allocati, esiste un certo numero di pagine in uso attivo. Se non dispone di questo numero di frame, il processo accusa immediatamente un'assenza di pagina e si deve, quindi, sostituire una pagina. Poichè tutte le pagine sono in uso attivo, si deve sostituire una pagina che sarà immediatamente necessaria e, di conseguenza, si verificano subito parecchie assenze di pagine. Il processo continua a subire assenze di pagine, facendo sostituire pagine che saranno immediatamente trattate come assenti e dovranno essere riprese. Questo fenomeno è detto ***paginazione degenere*** (***thrashing***) e si verifica quando si spende più tempo per la paginazione che per l'esecuzione dei processi. Osserviamo le cause del fenomeno. Per effettuaro il caricamento e lo scaricamento delle pagine per nuovi processi si deve utilizzare il dispositivo di paginazione. Mentre si mettono i processi in coda per il dispositivo di paginazione, la coda dei processi pronti per l'esecuzione si svuota, quindi l'utilizzo della CPU diminuisce. Lo scheduler della CPU avverte questa riduzione e aumenta il grado di multiprogrammazione: si tenta quindi di avviare un nuovo processo sottraendo pagine ai processi in esecuzione, causando ulteriori assenze di pagine e allungando la coda per il dispositivo di paginazione. La multiprogrammazione aumenta e l'utilizzo della CPU diminuisce ancora: i processi non svolgono lavoro poichè si sta spendendo tutto il tempo nell'attività di paginazione. Per evitare il verificarsi di queste situazioni, occorre fornire a un processo tutti i frame di cui necessita. Per sapere quanti frame servono al processo si impiegano diverse tecniche. Il ***modello di località*** d'esecuzione del processo stabilisce che un processo, durante la sua esecuzione, si sposta di località in località. Una località è un insieme di pagine usate attivamente. Si supponga di allocare a un processo un numero di frame sufficiente per sistemare le sue località attuali. Finchè tutte queste pagine non si trovano in memoria, si verificano assenze di pagine relative a tali località; quindi, finchè le località non vengono modificate, non hanno luogo altre assenze di pagine. Se si assegnano meno frame rispetto alla dimensione della località attuale, la paginazione del processo degenera, poichè non si possono tenere in memoria tutte le pagine che il processo sta usando attivamente. Il ***modello dell'insieme di lavoro*** (*working set model*) è basato sull'ipotesi della località. Questo modello usa un parametro Δ per definire la ***finestra dell'insieme di lavoro***. L'idea consiste nell'esaminare i più recenti Δ riferimenti alle pagine: l'insieme di pagine nei più recenti Δ riferimenti è *l'insieme di lavoro*. Se una pagina è in uso attivo si trova nell'insieme di lavoro, invece, se non è più usata, esce dall'insieme di lavoro. Il modello dell'insieme di lavoro riscuote un discreto successo, e la sua conoscenza può servire per la paginazione, ma appare un modo alquanto goffo per controllare il thrashing. La strategia basata sulla ***frequenza delle assenze di pagine*** (*page fault frequency*, *PFF*) è più diretta. Con questo metodo si tenta di controllare l'alta frequenza dell'assenza delle pagine. Se la frequenza è eccessiva significa che il processo ha bisogno di più frame; analogamente se la frequenza è bassa, il processo potrebbe disporre di troppi frame. Si può quindi fissare un limite inferiore e superiore per la frequenza desiderata dell'assenza di pagine: se la frequenza effettiva delle assenze di pagine per un processo oltrepassa il limite superiore, occorre allocare a quel processo un altro frame. Come nel caso dell'insieme di lavoro può essere necessaria la sospensione di un processo.

Si consideri la lettura sequenziale di un file sul disco per mezzo delle consuete chiamate di sistema. Grazie a una soluzione nota come ***mappatura dei file in memoria***, una parte dello spazio degli indirizzi virtuali può essere associata logicamente al file. La mappatura di un file in memoria si realizza associando un blocco del disco a una o più pagine residenti in memoria. L'accesso iniziale al file avviene tramite una normale richiesta di paginazione, che causa un errore di pagina mancante. Tuttavia, una porzione del file, che è pari a una pagina, è caricata dal file system in una pagina fisica. Ogni successiva lettura o scrittura del file è gestita come accesso ordinario alla memoria, semplificando così l'accesso al file e il suo utilizzo, in quanto si permette al sistema di manipolare i file attraverso la memoria anzichè appesantire il sistema stesso con le chiamate read() e write(). Allo stesso modo, dal momento che il file I/O è creato nella memoria anche l'accesso al file è molto più veloce. Alcuni sistemi prevedono un'apposita chiamata di sistema per la mappatura dei dati; altri sistemi, invece, possono mappare un file in memoria, anche senza un'esplicita richiesta (Solaris considera tutto l'I/O relativo al file come mappato in memoria). La prassi generale per la configurazione di una regione di memoria condivisa nella API Win32, consiste nel ***mappare il file*** interessato dall'operazione, per poi stabilire una vista (*view*) del file mappato nelo spazio degli indirizzi virtuali di un processo. Un secondo processo, a questo punto, può aprire e creare una sua vista sul file mappato nel proprio spazio degli indirizzi virtuali. Il file mappato è l'oggetto condiviso tramite il quale non può svolgersi la comunicazione tra i processi. Per quanto riguarda l'I/O, ogni controllore è dotato di registri contenenti istruzioni e i dati in via di trasferimento. Per rendere agevole l'accesso ai dispositivi di I/O, molte architetture di elaboratori forniscono la ***mappatura in memoria dell'I/O***: gli indirizzi di memoria compresi in certi intervalli devono essere riservati alla mappatura dei registri del dispositivo. Le operazioni in lettura e scrittura su questi indirizzi di memoria fanno si che i dati siano trasferiti fra i registri dei dispositivi e la memoria, o viceversa. La mappatura in memoria dell'I/O è conveniente per altri dispositivi, come le porte seriali e parallele usate per connettere i modem e le stampanti all'elaboratore.

Quando un processo eseguito in modalità utente necessita di memoria aggiuntiva, le pagine sono allocate dalla lista dei frame disponibili che il kernel mantiene. Per allocare la propria memoria, il kernel attinge spesso a una riserva di memoria libera differente dalla lista usata per soddisfare i processi ordinari in modalità utente. Questo avviene principalmente per due motivi: il kernel deve fare un buon uso della memoria perchè la richiede per strutture dati di dimensioni variabili e le pagine allocate ai processi in modalità utente devono necessariamente essere contigue alla memoria fisica. Il ***sistema buddy*** (*sistema gemellare*) utilizza un segmento di grandezza fissa per l'allocazione della memoria, contenente pagine fisicamente contigue. La memoria è assegnata mediante il cosiddetto ***allocatore-potenza-di-2***, che alloca memoria in unità di dimensione pari a potenze di 2. Le differenti quantità richieste sono arrotondate alla successiva potenza di 2. Questo sistema offre il vantaggio di poter congiungere rapidamente buddy adiacenti per formare segmenti più grandi tramite la tecnica nota come ***fusione*** (*coalescing*). Lo svantaggio di questo sistema è che l'arrotondamento per eccesso a una potenza di 2 può facilmente generare frammentazione all'interno dei segmenti allocati. Una seconda strategia per assegnare la memoria del kernel è detta ***allocazione a lastre*** (*slab allocation*). Una *lastra*è composta da una o più pagine fisicamente contigue. Una *cache* consiste di una o più lastre. Ogni cache è popolata da oggetti e ogni cache rappresenta una categoria di struttura dei dati del kernel. Quando si crea una cache, un certo numero di oggetti che dipende dalla grandezza della lastra associata alla cache, inizialmente dichiarati liberi, viene associato alla cache. Al principio tutti gli oggetti nella cache sono contrassegnati come *liberi*. Quando una struttura dati del kernel ha bisogno di un oggetto può selezionare dalla cache qualunque oggetto libero che viene contrassegnato come *usato*. In Linux una determinata lastra può essere in stati diversi: *piena*, se tutti gli oggetti della lastra sono contrassegnati come usati, *vuota*, se tutti gli oggetti della lastra sono contrassegnati come liberi, o *parzialmente occupata*, se contiene oggetti liberi e usati. L'allocatore delle lastre è un descrittore di processi: in primo luogo tenta di estrarre un oggetto libero da una lastra parzialmente occupata; se non ne esistono, assegna un oggetto libero da una lastra vuota; in mancanza di lastre vuote disponibili, crea una nuova lastra da pagine fisiche contigue e le alloca in una cache. L'allocatore delle lastre offre due vantaggi: annulla lo spreco derivante da frammentazione e rende più veloce il tempo di esaurimento delle richieste.

Facciamo ora altre considerazioni. Una caratteristica ovvia per un sistema di paginazione su richiesta pura consiste nell'alto numero di assenze di pagine che si verificano all'avvio di un processo. La ***prepaginazione*** rappresenta un tentativo di prevenire un così alto livello di paginazione iniziale. Alcuni sistemi operativi (Solaris) applicano la prepaginazione ai frame dei file di dimensioni modeste. In un sistema che usi il modello dell'insieme di lavoro, a ogni processo si associa una lista delle pagine contenute nel suo insieme di lavoro. La prepaginazione può essere vantaggiosa in alcuni casi, anche se si pone un problema riguardante il costo, che deve essere inferiore al costo per l'assistenza delle corrispondenti mancanze di pagina. Un altro problema si pone nella scelta delle dimensioni delle pagine, quando si deve progettare un nuovo calcolatore. Come si può intuire non esiste un'unica soluzione migliore delle altre, poichè vi sono più fattori a sostegno delle diverse dimensioni. Si deve tenere conto della dimensione della tabella delle pagine, in quanto diminuendo la dimensione delle pagine aumenta il numero delle pagine stesse e quindi aumenta la dimensione della tabella delle pagine; d'altra parte più le pagine sono piccole, meno rischio c'è che si presenti il problema della frammentazione; un altro problema legato alla dimensione della pagine è il tempo richiesto per leggere o scrivere che è direttamente proporzionale alla grandezza della pagina; tuttavia con pagine piccole si riduce l'I/O totale perchè si migliorano le caratteristiche di località; con pagine di piccole dimensioni è possibile avere una migliore risoluzionen, che permette di isolare la memoria effettivamente necessaria; d'altra parte bisogna notare che con pagine molto grandi si verificano assenze di pagina molto spesso. Non esiste, come già detto, un metodo ottimale: storicamente si tende ad incrementare le dimensioni delle pagine. Un parametro importante che occorre considerare è la ***portata della TLB*** (*TLB reach*) che esprime la quantità di memoria accessibile dalla TLB, ed è data dal numero di elementi moltiplicato per la dimensione delle pagine. La portata della TLB può essere aumentata incrementando il numero di elementi, in modo da contenere un insieme di lavoro, oppure incrementando la dimensione delle pagine, che, come visto in precedenza, può portare alla frammentazione. Quando si usa la paginazione su richiesta, talvolta occorre permettere che alcune pagine si possano ***vincolare*** alla memoria (*locked in memory*). Una situazione di questo tipo si presenta quando l'I/O si esegue verso o dalla memoria utente (virtuale).

Esempi tra i sistemi operativi (pag. 389)

Windows XP

Solaris