Ascolta

3.7 Segmentazione

Nonostante la paginazione, la memoria virtuale illustrata finora è monodimensionale poiché gli indirizzi virtuali vanno da 0 a un certo indirizzo massimo, uno dopo l'altro. Per molti problemi può essere molto meglio disporre di due o più spazi degli indirizzi virtuali anziché uno solo. Per esempio, un compilatore ha tante tabelle costruite man mano che procede la compilazione, includendo eventualmente:

1. il testo sorgente salvato per la stampa dei listati (sui sistemi batch);
2. la tabella dei simboli, contenente i nomi e gli attributi delle variabili;
3. la tabella contenente tutte le costanti intere e a virgola mobile utilizzate;
4. l'albero di parsing, contenente l'analisi sintattica del programma;
5. lo stack usato per le chiamate di procedura nel compilatore.

Con il procedere della compilazione, le prime quattro tabelle crescono continuamente, mentre l'ultima cresce e si riduce in modi imprevedibili. In una memoria monodimensionale queste cinque tabelle dovrebbero essere allocate in parti contigue dello spazio virtuale degli indirizzi, come nella **Figura 3.30**.

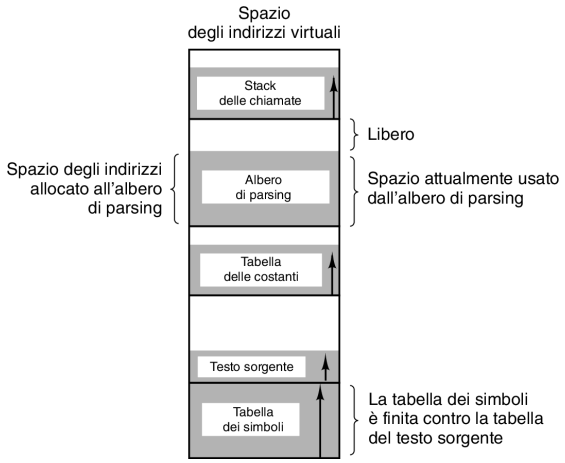

Figura 3.30 In uno spazio di indirizzi monodimensionale con tabelle che crescono, una tabella può cozzare contro un'altra.

Consideriamo che cosa accade se un programma ha un numero di variabili molto più grande del solito e una normale quantità di tutto il resto. La parte dello spazio degli indirizzi allocata per la tabella dei simboli può riempirsi, ma può esservi molto spazio nelle altre tabelle. Ci serve un modo per non obbligare il programmatore a gestire l'espansione e la contrazione delle tabelle, nello stesso modo in cui la memoria virtuale elimina il fastidio di organizzare i programmi in overlay.

Una soluzione diretta e molto generica è fornire la macchina di molti spazi degli indirizzi completamente indipendenti, chiamati **segmenti**. Ogni segmento consiste di una sequenza lineare di indirizzi, da 0 a un certo massimo. La lunghezza di ciascun segmento può essere qualsiasi, da 0 al massimo consentito. Segmenti diversi possono avere lunghezze diverse, e solitamente le hanno, e inoltre la lunghezza dei segmenti può variare durante l'esecuzione. La lunghezza del segmento dello stack può essere aumentata ogni volta che viene inserito qualcosa sullo stack e diminuita ogni volta che qualcosa ne è estratto.

Poiché ogni segmento costituisce uno spazio degli indirizzi separato, segmenti diversi possono crescere o ridursi indipendentemente, senza influenzarsi l'un l'altro. Se uno stack in un certo segmento ha bisogno di altro spazio degli indirizzi per crescere, può averlo poiché nel suo spazio non c'è niente altro contro cui andare a sbattere. Naturalmente un segmento può riempirsi, ma solitamente i segmenti sono molto grandi e quindi capita raramente.

?

Per specificare un indirizzo in questa memoria segmentata o bidimensionale, il programma deve fornire un indirizzo a due parti, un numero di segmento e un indirizzo nel segmento. La **Figura 3.31** illustra una memoria segmentata utilizzata per le tabelle del compilatore discusse in precedenza. Sono mostrati cinque segmenti indipendenti. È bene sottolineare che il segmento è un'entità logica, di cui il programmatore è a conoscenza e che utilizza come tale. Un segmento può contenere una procedura, un array, uno stack oppure una raccolta di variabili scalari, ma solitamente non contiene un mix di tipologie diverse.

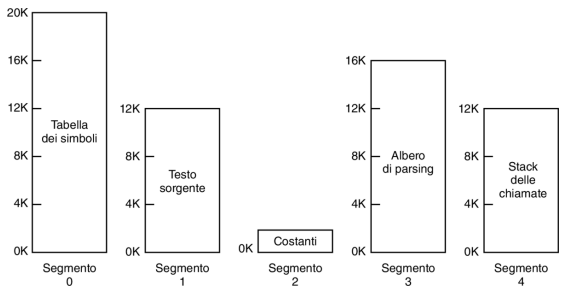

Figura 3.31 Una memoria segmentata consente a ogni tabella di crescere o di ridursi indipendentemente dalle altre tabelle.

Oltre a semplificare il trattamento delle strutture dati che crescono o si riducono, una memoria segmentata ha altri vantaggi. Se ogni procedura occupa un segmento separato, con indirizzo 0 al suo punto di partenza, il linking di procedure compilate separatamente è estremamente semplificato. Dopo che tutte le procedure che compongono un programma sono state compilate ed è stato fatto il linking, una chiamata alla procedura nel segmento *n* userà l'indirizzo a due parti (*n*, 0) per riferirsi alla parola 0 (il punto d'ingresso).

Se la procedura nel segmento *n* è modificata in seguito e ricompilata, nessun'altra procedura deve essere modificata (poiché nessun indirizzo di partenza è stato cambiato), anche se la nuova versione è più grande della vecchia. In una memoria monodimensionale le procedure sono una accanto all'altra, senza spazi degli indirizzi intermedi, quindi modificare la dimensione di una procedura significa influire sull'indirizzo di partenza di altre procedure (non correlate). Questo richiede giocoforza la modifica di tutte le procedure che chiamano una qualsiasi delle procedure spostate per incorporare i loro nuovi indirizzi di partenza. Se un programma contiene centinaia di procedure, questa attività può essere onerosa.

La segmentazione facilita inoltre la condivisione di procedure o di dati fra molti processi. Un esempio comune è offerto dalle librerie condivise. Le stazioni di lavoro moderne che eseguono avanzati sistemi a finestre hanno spesso librerie grafiche molto grandi, compilate in quasi ogni programma. In un sistema segmentato la libreria grafica può essere posta in un segmento e condivisa da più processi, eliminando la necessità di averla nello spazio degli indirizzi di ogni processo. È possibile avere librerie condivise anche nei sistemi paginati puri, ma la cosa è più complicata e in effetti si ottiene simulando la segmentazione.

Ciascun segmento forma un'entità logica, come una procedura o un array o lo stack, di cui il programmatore è a conoscenza, perciò segmenti diversi possono avere tipi diversi di protezione. Un segmento contenente una procedura può essere specificato come di sola esecuzione, proibendo qualsiasi tentativo di lettura o di salvataggio. Un array a virgola mobile può essere specificato come di sola lettura/scrittura, ma non di esecuzione e i tentativi di salto al suo interno saranno intercettati. Questo tipo di protezione è utile per scovare i bug. Nella **Figura 3.32**. sono confrontate paginazione e segmentazione.

Considerazione	Paginazione	Segmentazione
Il programmatore deve sapere che questa tecnica è in uso?	No	Sì
Quanti spazi di indirizzi lineari ci sono?	1	Molti
Lo spazio degli indirizzi totale può superare la dimensione della memoria fisica?	Sì	Sì
Le procedure e i dati possono essere distinti e protetti separatamente?	No	Sì
Le tabelle la cui dimensione varia possono essere disposte facilmente?	No	Sì

La condivisione delle procedure fra utenti è facilitata?	No	Sì
Perché fu inventata questa tecnica?	Per avere uno spazio degli indirizzi lineare grande senza dover acquistare ulteriore memoria fisica	Per consentire a programmi e dati di essere spezzati in spazi degli indirizzi logicamente indipendenti e per facilitare la condivisione e la protezione

Figura 3.32 Confronto fra paginazione e segmentazione.

3.7.1 Implementazione della segmentazione pura

L'implementazione della segmentazione differisce dalla paginazione per un motivo fondamentale: le pagine sono di dimensioni fisse, i segmenti no. La **Figura 3.33**(a) mostra un esempio di memoria fisica contenente in principio cinque segmenti. Consideriamo adesso che cosa accade se il segmento 1 è rimosso e il segmento 7, che è più piccolo, viene messo al suo posto: arriviamo alla configurazione di memoria della **Figura 3.33**(b). Fra il segmento 7 e il segmento 2 c'è dello spazio inutilizzato, cioè vuoto. Poi il segmento 4 è sostituito dal segmento 5, come nella **Figura 3.33**(c) e il segmento 3 è rimpiazzato dal segmento 6, come nella **Figura 3.33**(d). Dopo che il sistema è stato in esecuzione per un po', la memoria sarà suddivisa in parti, qualcuna contenente segmenti e altre spazi vuoti. Questo fenomeno, chiamato **checkerboarding** (da checkerboard, scacchiera) o **frammentazione esterna**, spreca la memoria negli spazi vuoti. Può essere risolto tramite la compattazione, come mostrato nella **Figura 3.33**(e).

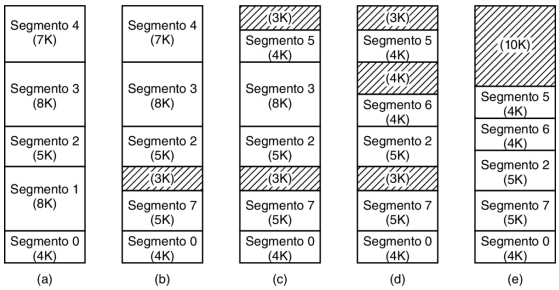

Figura 3.33 (a)-(d) Sviluppo del checkerboarding. (e) Rimozione del checkerboarding tramite la compattazione.

3.7.2 Segmentazione con la paginazione: MULTICS

Se i segmenti sono grandi può essere difficile, se non impossibile, tenerli per intero in memoria principale; questo porta all'idea di paginarli, in modo che siano in circolazione solo le pagine effettivamente necessarie. Vari sistemi significativi hanno supportato la paginazione dei segmenti. In questo paragrafo descriveremo il primo: MULTICS. Il suo design ha avuto una forte influenza sui processori Intel x86, che hanno fornito segmentazione e paginazione fino all'x86-64.

Il sistema operativo MULTICS è stato uno dei sistemi operativi più importanti della storia e ha influenzato fortemente aspetti disparati dell'informatica: da UNIX all'architettura della memoria x86, passando per i TLB e il cloud computing. Nacque come progetto di ricerca presso il M.I.T. e divenne un prodotto operativo nel 1969. L'ultimo sistema MULTICS è stato spento nel 2000, il che significa una durata di 31 anni. Pochissimi altri sistemi operativi hanno avuto durate paragonabili senza subire sostanziali modifiche. I sistemi operativi denominati Windows sono più o meno coetanei, ma Windows 11 ha in comune con Windows 1.0 soltanto il nome e il fatto di essere prodotto da Microsoft. Ma soprattutto, le idee sviluppate in MULTICS sono valide e utili oggi tanto quanto lo erano nel 1965, quando fu pubblicato il primo articolo che ne parlava (Corbató e Vyssotsky, 1965). Per questo motivo ora ci dedicheremo a uno degli aspetti più innovativi di MULTICS: l'architettura della memoria virtuale. Troverete ulteriori informazioni su MULTICS all'URL www.multicians.org.

MULTICS girava sulle macchine Honeywell della serie 6000 e sulle loro discendenti; forniva a ciascun programma una memoria virtuale sino a 218 segmenti (più di 250.000), ognuno dei quali poteva essere lungo sino a 65.536 parole (da 36 bit). Per l'implementazione, i progettisti del MULTICS scelsero di trattare ciascun segmento come memoria virtuale e paginarlo, combinando i vantaggi della paginazione (la dimensione uniforme delle pagine e la possibilità di non tenere in memoria l'intero segmento, ma solo la parte che deve essere usata) con quelli della segmentazione (facilità di programmazione, modularità, protezione e condivisione).

Ogni programma MULTICS ha una tabella dei segmenti, con un descrittore per segmento. Poiché la tabella poteva contenere potenzialmente più di un quarto di milione di voci, la stessa tabella dei segmenti era segmentata e paginata. Un descrittore di segmento conteneva un'indicazione del fatto che il segmento fosse nella memoria principale o meno. Se una qualunque parte del segmento era in memoria, il segmento era considerato in memoria e la sua tabella delle pagine era in memoria. Se il segmento era in memoria, il suo descrittore conteneva un puntatore a 18 bit alla sua tabella delle pagine, come nella **Figura 3.34**(a). Poiché gli indirizzi fisici erano a 24 bit e le pagine erano allineate su limiti di 64 byte (il che implicava che i 6 bit meno significativi degli indirizzi della pagina fossero 000000), nel descrittore servivano solo 18 bit per salvare un indirizzo della tabella delle pagine. Il descrittore conteneva anche la dimensione del segmento, i bit di protezione e altri oggetti. La **Figura 3.34**(b) illustra un descrittore di segmento del MULTICS. L'indirizzo del segmento nella memoria secondaria non era nel descrittore del segmento ma in un'altra tabella usata dal gestore degli errori di segmentazione.

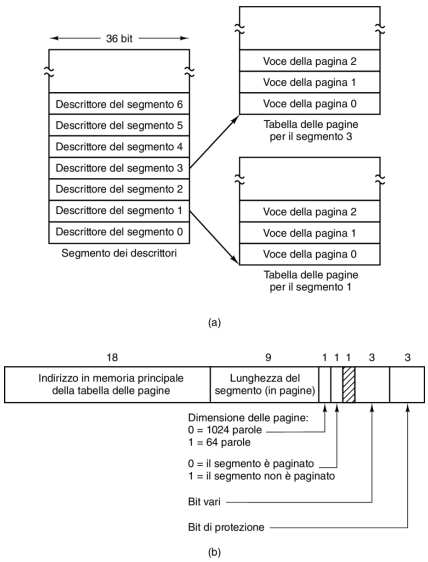

Figura 3.34 Memoria virtuale del MULTICS. (a) Il segmento dei descrittori punta alle tabelle delle pagine. (b) Un descrittore di segmento. I numeri indicano le lunghezze dei campi.

Ogni segmento era uno spazio virtuale degli indirizzi ordinario ed era paginato nello stesso modo della memoria paginata non segmentata descritta in precedenza in questo capitolo. La normale dimensione di pagina era di 1024 parole (pochi piccoli segmenti usati dallo stesso MULTICS non erano paginati o lo erano in unità di 64 parole per risparmiare memoria fisica).

In MULTICS un indirizzo consisteva di due parti: il segmento e l'indirizzo nel segmento. L'indirizzo nel segmento era ulteriormente suddiviso in un numero di pagina e in una parola nella pagina, come illustrato nella **Figura 3.35**. Quando avveniva un riferimento alla memoria veniva messo in atto l'algoritmo descritto di seguito.

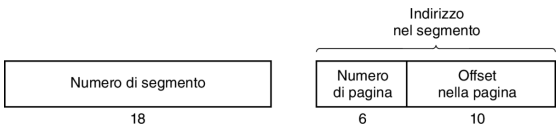

Figura 3.35 Un indirizzo virtuale MULTICS a 34 bit.

1. Il numero del segmento era usato per trovare il descrittore del segmento.
2. Si verificava se la tabella delle pagine del segmento fosse in memoria. Se lo era, veniva localizzata. Se non lo era avveniva un errore di segmentazione (segment fault). Se c'era una violazione della protezione, si verificava un errore (trap).
3. Veniva esaminata la voce della pagina virtuale richiesta nella tabella delle pagine. Se la pagina non era in memoria veniva generato un page fault; se lo era, dalla voce della tabella delle pagine veniva estratto l'indirizzo dell'inizio della pagina nella memoria principale.
4. Si otteneva l'indirizzo nella memoria principale in cui era localizzata la parola aggiungendo l'offset all'origine della pagina.
5. Finalmente avveniva la lettura o il salvataggio.

Il processo è illustrato nella **Figura 3.36**. Per semplificare, è stato omesso che il segmento del descrittore fosse esso stesso paginato. Quello che realmente accadeva è che un registro (il registro base del descrittore) era usato per localizzare la tabella delle pagine del segmento del descrittore, e la pagina puntava alle pagine del segmento del descrittore. Una volta trovato il descrittore del segmento necessario, l'indirizzamento procedeva come mostrato nella figura.

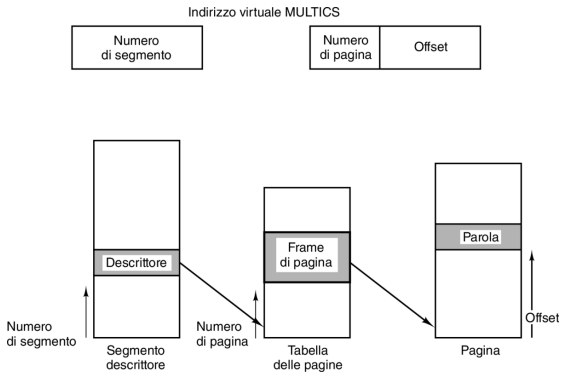

Figura 3.36 Conversione di un indirizzo MULTICS a due parti in un indirizzo di memoria principale.

Come sicuramente avete intuito, se l'algoritmo precedente fosse stato effettivamente eseguito dal sistema operativo a ogni istruzione, i programmi sarebbero stati lenti e gli utenti non tropo entusiasti. In realtà l'hardware del MULTICS conteneva un TLB ad alta velocità a 16 parole, illustrato nella **Figura 3.37**, che poteva ricercare parallelamente una determinata chiave in tutte le sue voci. Quando al computer veniva presentato un indirizzo, l'hardware di indirizzamento prima controllava se l'indirizzo virtuale fosse nel TLB. Se c'era, prendeva direttamente dal TLB il numero di frame e formava l'indirizzo effettivo della parola a cui era stato fatto riferimento senza dover guardare nel segmento del descrittore o nella tabella delle pagine. Fu il primo sistema dotato di TLB, oggi presente in tutte le architetture moderne.

Campo di confronto					Questa voce è usata?
Numero di segmento	Pagina virtuale	Frame	Protezione	Età	
4	1	7	Lettura/scrittura	13	1
6	0	2	Solo lettura	10	1
12	3	1	Lettura/scrittura	2	1
					0
2	1	0	Solo esecuzione	7	1
2	2	12	Solo esecuzione	9	1

Figura 3.37 Una versione semplificata del TLB del MULTICS. L'esistenza di due dimensioni della pagina rende il TLB reale più complicato.

Nel TLB erano tenuti gli indirizzi delle 16 pagine con i riferimenti più recenti. I programmi con un set di lavoro più piccolo del TLB arrivavano a regime con gli indirizzi dell'intero set di lavoro nel TLB, quindi erano eseguiti efficacemente; in caso contrario si verificavano degli errori del TLB.

3.7.3 Segmentazione con paginazione: l'Intel x86

Fino all'uscita dell'Intel x86-64, la memoria virtuale degli x86 ricordava abbastanza da vicino quella di MULTICS, inclusa la presenza di segmentazione e paginazione. Mentre il MULTICS aveva 256.000 segmenti indipendenti, ciascuno con un massimo di 64.000 parole a 36 bit, l'x86 ha 16.000 segmenti indipendenti, ognuno contenente fino a 1 miliardo di parole a 32 bit. I segmenti sono meno, ma la dimensione maggiore è molto più importante: pochi programmi hanno bisogno di più di 1000 segmenti, mentre molti programmi necessitano di segmenti grandi. Dall'x86-64 in poi la segmentazione è considerata obsoleta e non è più supportata, se non in modalità legacy. Anche

se nella modalità nativa x86-64 sono rimaste alcune vestigia dei vecchi meccanismi di segmentazione, soprattutto per questioni di compatibilità, non hanno più lo stesso ruolo né offrono più un'autentica segmentazione. L'x86-32, invece, dispone ancora di entrambi i meccanismi.

Ma allora perché Intel ha eliminato una variante del degnissimo modello della memoria di MULTICS che aveva supportato per quasi trent'anni? Forse la ragione principale è che né UNIX né Windows l'hanno mai usato, nonostante la sua efficienza (eliminava le chiamate di sistema, trasformandole in fulminee chiamate a procedura all'indirizzo pertinente entro un segmento protetto del sistema operativo). Nessuno sviluppatore di sistemi UNIX né Windows voleva adottare un modello di memoria specifico per x86: avrebbe sicuramente compromesso la portabilità verso altre piattaforme. Visto che il software non utilizzava la caratteristica, Intel decise di smettere di consumare superficie del chip per supportarla, e così la eliminò dalle CPU a 64 bit.

Tutto sommato bisogna dar credito ai progettisti dell'x86. Considerati i conflitti nell'implementazione contemporanea di paginazione pura, segmentazione pura e segmenti paginati, mantenendo allo stesso tempo la compatibilità con il 286 e realizzando il tutto in modo efficiente, il design risultante è sorprendentemente semplice e pulito.