DPMI 0.9

Per eseguire applicazioni DOS compilate per il modo protetto, occorrono un server DPMI (DOS Protected Mode Interface) e un DOS Extender, che fa da tramite tra l'applicazione e il sistema operativo, intercettando le chiamate degli interrupt del BIOS e del DOS. In Windows 3.1, ad esempio, il server DPMI e il DOS Extender sono entrambi in DOSX.EXE se si usa il modo standard, in WIN386.EXE se si usa il modo 386 avanzato. Il DOS Extender è il vero e proprio cliente del server DPMI, in quanto se ne avvale sia per emulare gli interrupt BIOS e DOS sia per soddisfare richieste di allocazione di memoria; anche una normale applicazione, tuttavia, può trarre vantaggio dai servigi offerti dal server DPMI e, a volte, non può evitare di farvi ricorso

di Sergio Polini

Quando si esegue un'applicazione DOS compilata con il Pascal 7 per il modo protetto, è necessario che i file RTM.EXE e DPMI16BI.OVL siano presenti nella directory corrente, oppure in una tra quelle assegnate alla variabile PATH dell'environment. Il file EXE comprende, oltre a codice e dati per il modo protetto, anche una sezione di codice per il modo reale, detta *stub*; questa verifica se sono già disponibili un server DPMI - per caricare, ove necessario, il server DPMI16BI.OVL - e il *runtime manager* contenuto in RTM.EXE, per caricarlo da esso se non già presente; il file DPMI16BI.OVL non è necessario se i servizi DPMI sono già disponibili in quanto l'applicazione viene eseguita in una sessione DOS di Windows o di OS/2.

Una volta assicurata la presenza del server DPMI e del runtime manager, lo *stub* passa dal modo reale al modo protetto ed il controllo passa al codice per quest'ultimo; vengono poi caricate le DLL eventualmente necessarie ed il codice e i dati dell'applicazione; infine, il controllo passa al codice dell'applicazione, che può così essere eseguita in modo protetto: si mantiene un accesso ai servizi del BIOS e del DOS, le richieste di allocazione di memoria vengono gestite dal runtime manager in modo da rendere accessibile tutta la memoria estesa installata (fino a 16 megabyte), i segmenti di codice vengono rilocati, scartati e ricaricati automaticamente quando occorrono, rendendo inutile il ricorso agli overlay.

Per molti aspetti, l'applicazione si comporta - e può essere scritta - come una normale applicazione DOS; quando, tuttavia, si desidera lavorare con gli interrupt o gestire tutta la memoria disponibile, occorre avvalersi dei servigi offerti dal server DMPI, implementati come un'ampia serie di funzioni, cui si accede assegnando al registro AX il numero della funzione e chiamando l'INT 31h.

Descrittori nella LDT

Oltre alla tabella dei descrittori «globale» (GDT), esistono tabelle «locali» (LDT) per ogni task; le funzioni da 0000h a 000Bh permettono di allocare e rilasciare descrittori nella LDT, nonché di cambiarne le caratteristiche.

La funzione 0000h alloca un numero di descrittori indicato nel registro CX; come per tutte le altre funzioni, il flag di carry viene settato in caso di errore, altrimenti azzerato. In caso di successo, troveremo nel registro AX il selettore del primo o unico descrittore; se ne sono chiesti più di uno, verrà allocato un array di descrittori contigui; per accedere al secondo, occorrerà aggiungere al selettore del primo l'incremento reso in AX dalla funzione 0003h, e così via.

Allocare uno o più descrittori non vuol dire allocare memoria; i descrittori allocati con la funzione 0000h, infatti, hanno base e limite entrambi nulli, come si può verificare agevolmente con la funzione 0006h e con l'istruzione assembler LSL *(Load Segment Limit)*; per associare un descrittore allocato con la funzione 0000h ad un'area di memoria (magari allocata con la funzione 0501h),

```
AX = 0000h: Alloca CX descrittori dalla LDT (in AX il primo selettore)
AX = 0001h: Rilascia il selettore BX
AX = 0002h: Converte il segmento reale BX in selettore (in AX)
AX = 0003h: Ritorna in AX il valore da aggiungere ad un selettore per
            accedere al successivo (cfr. AX = 0000h)
AX = 0004h: Riservata (vedi testo)
AX = 0005h: Riservata (vedi testo)
AX = 0006h: Ritorna in CX:DX l'indirizzo lineare della base del segmento
            indicato dal selettore in BX
AX = 0007h: Cambia in CX:DX l'indirizzo lineare della base del segmento
            indicato dal selettore in BX
AX = 0008h: Cambia in CX:DX il limite del segmento indicato dal selet-
            tore in BX
AX = 0009h: Cambia in CL il byte di accesso ed in CH il tipo del segmen-
            to indicato dal selettore in BX
AX = 000Ah: Ritorna in AX il selettore di un segmento dati con base e
            limite uguali a quelli del segmento codice indicato dal se-
            lettore in BX
AX = 000Bh: Copia il descrittore indicato dal selettore in BX in un buf-
            fer di 8 byte con indirizzo ES:(E)DI
AX = 000Ch: Copia il buffer di 8 byte con indirizzo ES:(E)DI nel de-
            scrittore indicato dal selettore in BX
AX = 000Dh: Alloca un descrittore dalla LDT con selettore in BX
```

Le funzioni per l'allocazione, il rilascio e la modifica di descrittori nella LDT (Local Descriptor Table) *del task corrente.*

```
AX = 0100h: Alloca BX paragrafi dalla memoria DOS, rendendo in AX il
            segmento reale iniziale e in DX il primo selettore; in caso
            d'errore, in BX la dimensione in paragrafi del più ampio
            blocco allocabile
AX = 0101h: Rilascia la memoria allocata con la funzione 0100h; in DX il
            primo selettore
AX = 0102h: Cambia la dimensione del blocco allocato con la funzione
            0100h; in BX la nuova dimensione in paragrafi, in DX il se-
            lettore del blocco; in caso d'errore, in BX la dimensione
            massima del blocco in paragrafi
```

Le funzioni per l'allocazione di memoria gestita dal DOS (il primo megabyte).

si possono attribuire al descrittore la base ed il limite di questa con le funzioni 0007h e 0008h. È possibile, d'altra parte, ottenere con la funzione 0002h il selettore di un descrittore per un segmento «reale» di memoria, in modo da rendere questo accessibile da programmi eseguiti in modo protetto.

La funzione 000Bh consente di copiare un descrittore in un buffer di otto byte, per esaminarlo ed eventualmente modificarlo, ricopiando il buffer nel descrittore mediante la funzione 000Ch. Se è sufficiente cambiare solo i diritti di accesso o il tipo, peraltro, si può ricorrere alla funzione 009Ah.

Non è consigliabile, comunque, mutare il tipo di un segmento codice al fine di modificarlo; se si intende variare il contenuto di un segmento codice ovvero, per dirla in altro modo, se si desidera scrivere codice automodificante in modo protetto, è preferibile allocare un descrittore per un segmento dati (quindi accessibile anche in scrittura) con base e limiti uguali al segmento codice, mediante la funzione 000Ah.

La API di Windows comprende funzioni analoghe, cui si può accedere anche da un programma DOS per il modo protetto usando la unit WINAPI; in particolare, *AllocSelector* (0000h con CX = 1), *FreeSelector* (0001h), *GetSelectorBase* (0006h), *SetSelectorBase* (0007h), *GetSelectorLimit* (istruzione LSL), *SetSelectorLimit* (0008h); si dispone anche di una funzione non documentata *AllocSelectorArray* (contenuta in KERNEL.EXE con indice 206 ed equivalente alla funzione 0000h, con un parametro di tipo word per CX) e di una costante __AHINCR, anch'essa non documentata (in KERNEL.EXE con indice 113), il cui valore è quello che verrebbe reso dalla funzione 0003h. Del pari non

```
AX = 0200h: Ritorna in CX:DX l'indirizzo reale (segmento:offset) del ge-
            store dell'interrupt indicato in BL
AX = 0201h: Imposta in CX:DX l'indirizzo reale (segmento:offset) del ge-
            store dell'interrupt indicato in BL
AX = 0202h: Ritorna in CX:(E)DX il selettore e l'offset del gestore
            dell'eccezione indicata in BL
AX = 0203h: Imposta in CX:(E)DX il selettore e l'offset del gestore
            dell'eccezione indicata in BL
AX = 0204h: Ritorna in CX:(E)DX il selettore e l'offset del gestore
            dell'interrupt indicato in BL
AX = 0205h: Imposta in CX:(E)DX il selettore e l'offset del gestore
            dell'interrupt indicato in BL

AX = 0900h: Disabilita gli interrupt virtuali
AX = 0901h: Abilita gli interrupt virtuali
AX = 0902h: Ritorna in AL 0 se gli interrupt virtuali sono disabilitati,
            1 se sono abilitati
```

Le funzioni per l'intercettazione di interrupt ed eccezioni.

documentata è la funzione *SelectorAccessRights* (196 in KERNEL.EXE), che, passato un selettore come primo parametro, legge o scrive i byte di accesso e di tipo secondo il valore del secondo parametro (0 per la lettura, 1 per la scrittura); in caso di scrittura, i valori da assegnare sono passati nel terzo parametro, di tipo word. Vedremo in un prossimo appuntamento come chiamare da un programma Pascal le funzioni non documentate della API di Windows.

Quanto alla creazione di «alias», la funzione *AllocDStoCSAlias* crea un descrittore per un segmento codice che sia alias del descrittore di un segmento dati, consentendo di eseguire codice che sia stato scritto da un'applicazione durante la sua esecuzione. Non sono documentate, invece, le funzioni *AllocAlias* e *AllocCStoDSAlias* (rispettivamente 172 e 170 in KERNEL.EXE) che, corrispondendo alla funzione 000Ah, consentono di scrivere codice automodificante; va rilevato che, se due descrittori si riferiscono entrambi allo stesso segmento, variazioni nell'uno non si riflettono automaticamente nell'altro e, quindi, può accadere che un segmento dati venga scartato, spostato, scritto da altre applicazioni, senza che il descrittore che guarda ad esso come se fosse un segmento codice se ne avveda. Prima di creare un alias per un segmento codice, quindi, è necessario bloccare quest'ultimo mediante la funzione *GlobalPageLock*, che usa la funzione DPMI 0004h, non documentata (funzioni come *LockSegment* o *GlobalLock* non garantiscono che il segmento rimanga allo stesso indirizzo lineare, ma solo che non venga scartato; il selettore bloccato con la funzione 0004h può essere sbloccato con la funzione 0005h).

Sono presenti anche le funzioni *ChangeSelector* e *PrestoChangoSelector* (quest'ultima non documentata, con indice 177 in KERNEL.EXE), che, dato il selettore di un segmento codice, lo convertono in selettore di un segmento dati e viceversa; il nuovo selettore, però, deve essere stato preventivamente allocato e, quindi, conviene usare di-

```
AX = 0300h: Simula l'interrupt reale indicato in BL; in BH 01h per re-
            settare il PIC e la linea A20, altrimenti 00h; in CX il nu-
            mero di word per eventuali parametri; in ES:(E)DI selettore
            e offset di una "struttura di chiamata del modo reale"
AX = 0301h: Chiama una procedura in modo reale che termini con un RET
            FAR; registri come per 0300h
AX = 0302h: Chiama una procedura in modo reale che termini con un IRET;
            registri come per 0300h
AX = 0303h: Ritorna in CX:DX un indirizzo reale da cui sarà possibile
            chiamare una procedura protetta con selettore e offset in
            DS:(E)SI; in ES:(E)DI selettore e offset di una "struttura
            di chiamata del modo reale"
AX = 0304h: Libera l'indirizzo reale predisposto con la funzione 0303h,
            indicato in CX:DX
AX = 0305h: Ritorna in BX:CX segmento e offset e in SI:(E)DI selettore e
            offset di due procedure per salvare/ripristinare lo stato
            dei registri prima/dopo un cambio di modo mediante la fun-
            zione 0306h
AX = 0306h: Ritorna in BX:CX l'indirizzo cui saltare per passare da modo
            reale a modo protetto e in SI:(E)DI l'indirizzo per passare
            da modo protetto a modo reale
```

Le funzioni per la chiamata di codice protetto da codice reale e viceversa.

```
AX = 0400h: Ritorna in AH:AL la versione DPMI; in BX flag che indicano
            se si tratta di una versione per 80386 (bit 0), se si torna
            al modo reale - invece che al modo V86 - per gli interrupt
            (bit 1), se c'e memoria virtuale (bit 2); in CL il processo-
            re (2=80286, 3=80386, 4=80486); in DH e DL i valori del
            primo interrupt per i PIC master e slave
```

La funzione che ritorna informazioni sulla versione del server DPMI installato e su alcune sue caratteristiche.

rettamente *AllocDStoCSAlias* o *AllocCStoDSAlias*.

Allocazione di memoria

Le funzioni 0100h, 0101h e 0102h possono essere usate per allocare memoria nel primo megabyte, accessibile sia in modo reale mediante un valore da assegnare ad un registro di segmento, sia in modo protetto mediante un selettore (se si chiedono più di 64 KByte, vengono allocati più descrittori e, per accedere a quelli oltre il primo, occorre aggiungere al selettore l'incremento reso dalla funzione 0003h); la memoria così allocata può essere utilizzata, come vedremo, per far comunicare tra loro un'applicazione eseguita in modo reale (quale un programma residente) ed un'altra eseguita in modo protetto.

Le funzioni da 0500h a 0503h, invece, consentono di allocare memoria estesa. La funzione 0500h ritorna in ES:DI (o ES:EDI se si lavora a 32 bit) il selettore e l'offset di un buffer di 48 byte contenente, nell'ordine, i seguenti valori, tutti di 4 byte: la dimensione del blocco più grande che può essere allocato, il numero di pagine che possono essere allocate, il numero di pagine che possono essere allocate e bloccate *(locked)*, la dimensione totale in pagine della memoria (compresa quella già allocata), il numero di pagine non bloccate (sia quelle in uso che potrebbero essere «parcheggiate» su disco, sia quelle non usate), il numero delle pagine non usate, il numero totale delle pagine gestite dal server DPMI, la dimensione in pagine della memoria non allocata, la dimensione in pagine del file usato per la paginazione (seguono 12 byte riservati).

Solo il primo valore è sempre valido; gli altri, infatti, possono essere -1 se si opera in un ambiente che con supporta la memoria virtuale mediante paginazione. Per avere informazioni sull'ambiente, si può usare la funzione 0400h, che setta il bit 2 del registro BX se la memoria virtuale è disponibile. In questo caso, la funzione 0604h ritorna in BX:CX l'ampiezza in byte di una pagina di memoria.

La funzione 0501h alloca BX:CX byte di memoria, rendendo nella stessa coppia di registri l'indirizzo lineare del blocco allocato e in SI:DI un *handle* da utilizzare per rilasciare il blocco (funzione 0502h) o per cambiarne le dimensioni (funzione 0503h). La funzione non alloca anche un descrittore, per il quale si può provvedere con la funzione 0000h.

Se c'è memoria virtuale, la memoria

```
AX = 0500h: Avvalora i campi di una struttura DI 30h byte con selettore
            ES e offset (E)DI contenente informazioni sulla memoria
            disponibile; i primi quattro byte contengono la dimensione
            del più ampio blocco disponibile
AX = 0501h: Alloca BX:CX byte di memoria, ritornandone l'indirizzo li-
            neare in BX:CX e in SI:DI un handle
AX = 0502h: Rilascia il blocco di memoria con handle SI:DI
AX = 0503h: Cambia in BX:CX le dimensioni del blocco di memoria con han-
            dle SI:DI; ritorna in BX:CX e SI:DI il nuovo indirizzo li-
            neare e il nuovo handle

AX = 0600h: Blocca un'area di memoria con inizio in BX:CX (indirizzo li-
            neare) e dimensione SI:DI
AX = 0601h: Sblocca un'area di memoria con inizio in BX:CX (indirizzo
            lineare) e dimensione SI:DI
AX = 0602h: Marca come paginabile un'area di memoria con inizio in BX:CX
            (indirizzo lineare) e dimensione SI:DI
AX = 0603h: Blocca un'area di memoria con inizio in BX:CX (indirizzo li-
            neare) e dimensione SI:DI, già marcata come paginabile dalla
            funzione 0602h
AX = 0604h: Ritorna in BX:CX le dimensioni in byte di una pagina di me-
            moria

AX = 0702h: Marca le pagine con inizio all'indirizzo lineare BX:CX, per
            un totale di SI:DI byte, come candidate per swapping su di-
            sco
AX = 0703h: Scarta il contenuto delle pagine con inizio all'indirizzo
            lineare BX:CX, per un totale di SI:DI byte

AX = 0800h: Ritorna in BX:CX l'indirizzo lineare di un'area di memoria
            oltre il primo megabyte con indirizzo fisico BX:CX ampia
            SI:DI byte
```

Le funzioni per la gestione della memoria estesa e virtuale.

viene allocata come non bloccata; per bloccarla, si può usare la funzione 0600h.

```
AX = 0A00h: Ritorna in ES:(E)DI il punto d'ingresso nella API estesa
            identificata da una stringa con indirizzo DS:(E)SI

AX = 0B00h: Imposta un watchpoint all'indirizzo lineare BX:CX; ne rende
            l'handle in BX
AX = 0B01h: Rimuove il watchpoint con handle BX
AX = 0B02h: Ritorna in AX lo stato (bit 0 = 1 se eseguito) del
            watchpoint con handle BX
AX = 0B03h: Resetta il watchpoint con handle BX
```

Le funzioni per l'accesso ad un'eventuale API estesa e per il debugging.

Interrupt ed eccezioni

Un programma DOS eseguito in modo protetto ha sempre sotto di sé il tradizionale ambiente DOS in modo reale; questo è vero in particolare per gli interrupt hardware e software, che un DOS Extender conforme alle specifiche DPMI deve sempre intercettare ed emulare. Alla tabella degli interrupt posta all'inizio del primo megabyte di memoria (256 coppie segmento:offset contenenti gli indirizzi dei gestori degli interrupt), deve corrispondere una *Interrupt Descriptor Table* (IDT) nel task eseguito in modo protetto. La situazione si complica quando sono presenti anche task in modo V86 (ad esempio, una sessione DOS sotto Windows in modo 386 avanzato); potremmo avere, infatti, la tabella degli interrupt «di sistema» all'inizio della memoria fisica, quella del task V86, la IDT del task eseguito in modo protetto a partire da quest'ultimo. Una simile complicazione può aversi anche quando non sembri essere attivo alcun task V86, in quanto, per emulare un interrupt del BIOS o del DOS, il DOS Extender può scegliere tra due diverse strategie: convertire il processore da modo protetto a modo reale per passare il controllo ad un gestore di stampo tradizionale (anche quello di default), oppure eseguire una routine in un task V86 «dedicato» (la strategia adottata viene resa nota dalla funzione 0400h).

Non solo. Gli interrupt sono abilitati o disabilitati secondo lo stato di uno dei flag del processore; di norma, tuttavia, gli interrupt sono sempre abilitati in modo protetto, per evitare che un task possa disabilitare gli interrupt hardware; allo stesso scopo, ad ogni task è associato un flag «virtuale», attraverso il quale il task può abilitare o disabilitare i suoi interrupt senza interferire con quelli di sistema: dopo un'istruzione CLI, ad esempio, il task non riceverà gli interrupt hardware, ma questi potranno pervenire ad altri task. Rileviamo anche che, eseguendo un'istruzione PUSHF, vengono posti nello stack i veri flag del processore; per esaminare e modificare lo stato del flag virtuale, quindi, occorre usare le funzioni 0900h, 0901h e 0902h.

Le funzioni 0200h e 0201h consentono di leggere o impostare l'indirizzo reale (segmento:offset) del gestore di un interrupt, mentre le funzioni da 0202h a 0205h consentono di leggere o impostare l'indirizzo protetto (selettore:offset) del gestore di un'eccezione o di un interrupt.

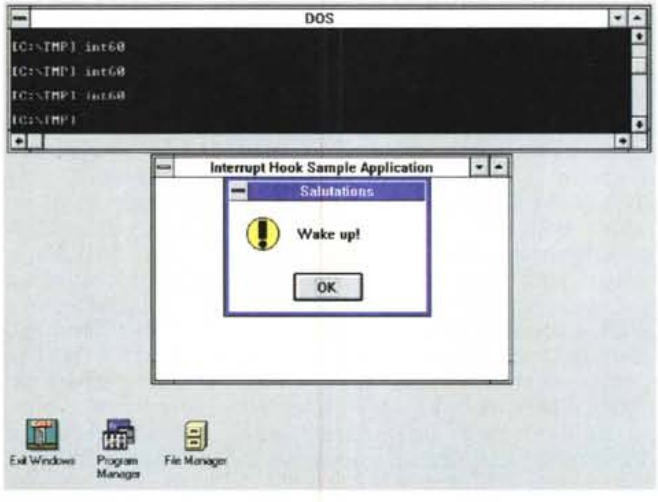
Queste, a volte, non sono necessarie o non sono sufficienti; potrebbe occorrere, infatti, eseguire un interrupt BIOS o DOS il cui gestore di default sia adeguato, ma non raggiungibile mediante il DOS Extender e il server DPMI. Il DOS Extender, infatti, non sempre emula quegli interrupt il cui gestore richiede il passaggio di registri di segmento (anche come primo membro di una coppia segmento:offset), in quanto i registri di segmento, in modo protetto, contengono selettori che non avrebbero senso per un gestore scritto per il modo reale ed in esso eseguito.

Per emulare tali interrupt, si dispone quindi della funzione 0300h, che va chiamata dopo aver assegnato ai registri ES e DI (o EDI) il selettore e l'offset di una struttura che esamineremo in dettaglio il mese prossimo.

Terminiamo invece questa rassegna accennando alle funzioni da 0301h a 0306h, che offrono varie soluzioni per codice che debba essere eseguito parte in modo reale e parte in modo protetto. Alcune di queste funzioni possone essere utilizzate per realizzare una qualche comunicazione tra applicazioni DOS e Windows, come illustrato in un articolo apparso sul numero di febbraio 1992 del *Dr. Dobb's Journal* («Using DPMI to hook interrupts in Windows 3», di Walter Oney). I sorgenti illustrati nell'articolo e i relativi eseguibili sono disponibili su MC-link, nel CD-ROM Simtel20, file DDJ9202.ZIP.

Si tratta di tre file, INT60.EXE, INT61.EXE e INTHOOK.EXE; si deve eseguire INT61.EXE da DOS prima di caricare Windows, poi, caricato Windows, si fa partire INTHOOK.EXE e si apre una sessione DOS: ogni volta che si esegue in questa INT60.EXE, INTHOOK.EXE reagisce aprendo una message box. L'effetto è notevole, ma, soprattutto, i sorgenti costituiscono un'ottima esemplificazione dell'uso di funzioni come 0303h e 0304h.

Chi volesse prendere diretta conoscenza delle specifiche della versione 0.9 della DPMI, cui sono conformi i server sia del Pascal 7 che di Windows, può trovarne il testo nel file DPMI_TXT.ZIP, disponibile su MC-link nel CD-ROM Cica-Win. MS



Un esempio di comunicazione tra una sessione DOS e un'applicazione Windows: quando si esegue INT60.EXE nella prima, l'applicazione INTHOOK.EXE apre una message box; il tutto è reso possibile dalla funzione DPMI 0304h.

Sergio Polini è raggiungibile tramite MC-link alla casella MC1166 e tramite Internet all'indirizzo MC1166@mclink.it.