di Pierluigi Panunzi

Il Mouse

quinta parte

In questa puntata parleremo di una sola delle funzioni ancora rimaste da analizzare: si tratta di una funzione particolarmente importante in quanto apre la strada ad infinite possibilità da parte dell'utente

La funzione 0CH: Set User-defined subroutine

Si tratta in parole povere di una funzione che consente di agganciare una routine scritta dall'utente alle routine del driver del mouse, facendo in modo che venga eseguita allorché il mouse venga mosso oppure ne vengano premuti i pulsanti.

A stabilire quale sia l'azione sul mouse che debba attivare anche la nostra routine, serve il parametro rappresentato dal registro CX: dalla figura 2 vediamo che si tratta di una «call mask» e cioè una maschera di attivazione della chiamata (alla nostra subroutine), rappresentata da un insieme di bit che, opportunamente settati, attivano o meno la routine. In particolare, facendo riferimento alla figura 3, vediamo che il bit meno significativo (LSB) decide o meno l'attivazione della routine se il mouse viene mosso: tale bit posto ad 1 attiverà la nostra subroutine tutte le volte che il mouse verrà mosso, mentre tale bit posto a 0 inibirà la chiamata quando il mouse si muove.

AX = 0H	Mouse Reset
AX = 1H	Cursor Enable
AX = 2H	Cursor Disable
AX = 3H	Get Mouse Position and Button Status
AX = 4H	Set Mouse Position
AX = 5H	Get Button Press Information
AX = 6H	Get Button Release Information
AX = 7H	Set min & max horizontal position
AX = 8H	Set min & max vertical position
AX = 9H	Set Graphic Cursor Block
AX = 0AH	Set Text Cursor
AX = 0BH	Read Motion Counters
AX = 0CH	Set User-defined subroutine
AX = 0DH	Enable Light Pen Emulation
AX = 0EH	Disable Light Pen Emulation
AX = 0FH	Set Mickey/Pixel Ratio
AX = 10H	Window Conditional Off
AX = 12H	Set Large Graphic Cursor
AX = 13H	Set Speed Threshold

Tabella 1 - Questa tabella riassume le funzioni di gestione di un mouse per mezzo dell'INT 33H: ad eccezione della funzione 12H, si tratta di funzioni standard.

Analogamente i bit successivi servono ad abilitare (se settati) la nostra routine allorché uno od entrambi i pulsanti del mouse sono premuti oppure rilasciati: ricordiamo a questo punto che è di fondamentale importanza poter sfruttare l'evento «pulsante premuto» («button pressed») in contrapposizione all'evento «pulsante rilasciato» («button released»).

Infatti di solito i menu vengono attivati alla pressione di uno dei due pulsanti, trascurando poi il successivo rilascio del pulsante stesso in quanto in genere il tempo intercorso tra i due eventi è molto piccolo (in gergo in questi casi il pulsante è stato «click-ato»), mentre un'altra situazione operativa è quella in cui si deve premere un pulsante, lo si tiene premuto magari muovendo il mouse in un'altra posizione e poi lo si rilascia in corrispondenza di questa posizione (in gergo questa manovra prende il nome di «dragging»).

Basti pensare, per questo secondo tipo di manovra, a come si disegnano i rettangoli in un qualsiasi programma di grafica: si preme il pulsante in corrispondenza di un vertice del rettangolo, si tiene premuto il pulsante spostando contemporaneamente il mouse in un'altra posizione (e nel frattempo il rettangolo viene continuamente ridisegnato dovunque ci spostiamo) e non appena rilasciamo il pulsante allora si ottiene il definitivo tracciamento del rettangolo.

Tornando alla figura 3, vediamo che in CX dovremo porre il valore 0AH se desideriamo l'attivazione della nostra routine all'atto della pressione di uno dei due pulsanti.

Inoltre all'atto della chiamata alla funzione 0CH in esame, dovremo fornire in ES:BX l'indirizzo completo (segment in ES ed offset in BX) della nostra routine, che deve essere obbligatoriamente di tipo **FAR**, altrimenti non si avrebbe un corretto ritorno al programma chiamante, che viceversa aveva posto nello stack un indirizzo di ritorno completo di offset e segment.

Fatto questo, la nostra routine verrà attivata ogni volta che si verifica un evento segnalato nella «call mask»: vediamo ora quali informazioni fornisce il driver alla nostra subroutine. In particolare in **AX** viene fornita una «condition mask», in pratica una «call mask» nella quale i bit settati stavolta indicano che

AX = 0CH	Set User-defined subroutine
INPUT	CX = call mask ES:BX = subroutine address
OUTPUT	-

Figura 2 - Questa tabellina indica i parametri che devono essere forniti in input alla funzione 0CH per far si che il driver esegua la nostra routine tutte le volte che il mouse viene mosso oppure tutte le volte che premiamo un pulsante.

l'evento a cui si riferiscono si è verificato: così come nella «call mask» si poteva settare più di un bit, può capitare che nella «condition mask» risultino settati più bit ad indicare che si sono verificati più eventi ed allora starà alla nostra routine mascherare l'evento che interessa. Nel registro **BX** invece il driver fornirà il «button status» (una ripetizione, dunque) del quale abbiamo già parlato nelle scorse puntate e sul quale non ritorniamo.

Nei registri **CX** e **DX** invece vengono fornite le coordinate correnti del cursore, rispettivamente quella orizzontale e quella verticale.

Conclusa dunque la teoria, passiamo alla pratica, analizzando il programma presentato.

Un programma di utility che sfrutta il mouse: M2K

Abbiamo dunque realizzato un programmino, tutto scritto in Assembler, per sfruttare al meglio le nozioni che abbiamo sulla gestione di un mouse.

L'idea che ci ha condotti a scrivere questo programma non è nuova: in particolare è un programma che permette di simulare la pressione di alcuni tasti della tastiera allorché il mouse venga mosso oppure ne vengano premuti i pulsanti.

Ancor più in dettaglio, tale programma permette di trasformare il movimento del mouse nelle quattro direzioni (alto, basso, destra, sinistra) nelle corrispondenti pressioni dei tasti di cursore (le frecce del tastierino numerico), con in più la possibilità di associare altri due tasti della tastiera ai pulsanti del mouse stesso.

Dicevamo che l'idea non è nuova in quanto esiste un programma della Microsoft che consente di gestire il cosiddetto MML («Mouse Menu Language»), un linguaggio ad alto livello che permette di assegnare appunto al moto del mouse ed ai suoi pulsanti alcune determinate azioni, quali la pressione dei tasti, la comparsa di scritte varie, nonché la gestione di menu: il tutto si ottiene appunto scrivendo un file contenente questo programma in MML, da dare poi in pasto ad un apposito compilatore.

L'idea nuova è in effetti il fatto di avere un programmino semplice e corto per mezzo del quale, all'atto della chiamata, possiamo decidere quali tasti associare ai pulsanti del mouse, mentre per default vengono associate le frecce al movimento del mouse nelle quattro direzioni.

Iniziamo dunque dal nome: M2K sta per «Mouse to Keyboard» (dove c'è il solito giochetto di parole inglese secondo il quale la preposizione «to» viene sostituita dal numero «2» in quanto dotati della stessa pronuncia) e cioè «dal mouse alla tastiera».

La sua sintassi è

M2K tasto1 tasto2

dove «tasto1» e «tasto2» sono i nomi dei tasti ai quali rispettivamente associare i pulsanti sinisro e destro del mouse. In particolare possono essere scelti tutti i tasti funzione da **F1** e **F10**, l'**ESC**ape, il tasto di **RET**urn, i quattro tasti del tastierino numerico **HOM**e, **END**, **PgUp**, **PgDn**, nonché un qualunque carattere ASCII, da impostare tra apici.

La codifica dei tasti e cioè la sigla da impostare nella linea di comando è rappresentata nella tabella che segue:

Tasto	Codifica
F1	F1
...	...
F10	F10
Escape	ESC
Return	RET
Home	HOM
End	END
PgUp	PU
PgDn	PD
carattere ASCII	tra apici

In generale i tasti più comodi sono l'Escape ed il Return e supponendo di associare l'ESC al pulsante destro ed il Return a quello sinistro, il programma andrà chiamato con

M2K RET ESC

con il che potremo tranquillamente lanciare i nostri programmi preferiti e finora non forniti di interfaccia per il mouse: ogni volta che spostiamo il mouse sarà come se avessimo premuto le frecce, mentre premendo il pulsante a sinistra è come se avessimo premuto il tasto Return.

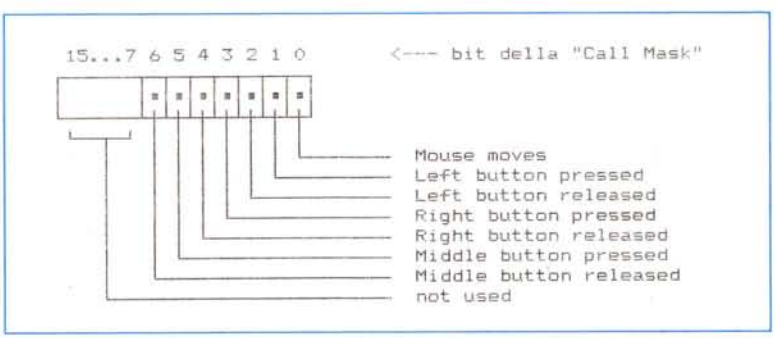

Figura 3 - Rappresentazione schematica della cosiddetta «call mask»: si tratta di una word i cui bit meno significativi servono ad abilitare (se posti ad 1) l'attivazione della funzione in corrispondenza dell'evento a cui si riferiscono. Il «middle button» non è nello standard Microsoft in quanto tutti i mouse di tale casa sono sempre dotati di solo due pulsanti. Inoltre tale maschera coincide pure con la «condition mask» che il driver fornisce (nel registro AX) alla subroutine d'utente: in tal caso i bit settati indicano che l'evento corrispondente si è verificato.

I dettagli tecnici

Il programma presentato è in pratica formato da due parti, una che rimarrà residente in memoria e che è proprio la subroutine d'utente attivata dal driver del mouse, mentre l'altra è una routine che gestisce altre funzioni che ora esaminiamo in dettaglio.

In particolare la routine principale effettua le seguenti funzioni:

— dapprima testa la presenza del driver del mouse e, se non lo trova, interrompe l'elaborazione spedendo un apposito messaggio diagnostico.

— Successivamente gestisce i parametri forniti all'atto della chiamata, andando a vedere se le stringhe in questione

```
;
; M2K.ASM    P.PANUNZI  1989
;
BUFFER_SEGMENT SEGMENT AT 40H
        ORG 1AH
B_HEAD  DW ?
B_TAIL  DW ?
        ORG 80H
B_START DW ?
B_END   DW ?
BUFFER_SEGMENT ENDS

CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE,ES:CODE
        ORG 100H
START:  JMP END_OF_RESIDENT_PART
;
LEFTBUTTON      DW ?
RIGHTBUTTON     DW ?
;
MOUSE   PROC FAR
        TEST AX,1
        JZ BUTTONS
        XOR BL,BL       ;BL = 0 => CARATTERE SPECIALE
        MOV AX,0BH
        INT 33H
        OR CX,CX
        JZ VERT         ;NESSUN MOVIMENTO ORIZZONTALE
        MOV BH,4BH      ;LEFT ARROW
        JS INSERTKEY
        MOV BH,4DH      ;RIGHT ARROW
INSERTKEY:
        MOV AX,BUFFER_SEGMENT
        MOV DS,AX
        MOV ES,AX
        ASSUME DS: BUFFER_SEGMENT,ES:BUFFER_SEGMENT
        CLI
        MOV DI,B_TAIL
        MOV SI,DI
        ADD SI,2
        CMP SI,B_END
        JNE L0
        MOV SI,B_START
L0:     CMP SI,B_HEAD
        JE END1
        MOV AX,BX
        STOSW
        MOV B_TAIL,SI
END1:   STI
ENDMOUSE:
        RET

        ASSUME DS:CODE,ES:CODE
VERT:   OR DX,DX


        JZ ENDMOUSE
        MOV BH,48H      ;UP ARROW
        JS INSERTKEY
        MOV BH,50H      ;DOWN ARROW
        JMP INSERTKEY
BUTTONS:
        MOV BX,CS:LEFTBUTTON
        TEST AX,2       ;LEFT BUTTON
        JNZ INSERTKEY
        MOV BX,CS:RIGHTBUTTON
        TEST AX,8       ;RIGHT BUTTON
        JNZ INSERTKEY
        RET
MOUSE   ENDP

        ASSUME DS:CODE,ES:CODE
END_OF_RESIDENT_PART:
        JMP SKIP
;
NOTPRESENT      DB 'Mouse driver not installed',0DH,0AH,'$'
NOARG           DB 'Syntax is  "m2k key1 key2"',0DH,0AH,'$'
ARGERR          DB 'Error in one argument',0DH,0AH,'$'
DONE            DB 'Done...',0DH,0AH,'$'

TABLE   DB 'F1  ','F2  ','F3  ','F4  ','F5  '
        DB 'F6  ','F7  ','F8  ','F9  ','F10 '
        DB 'RET ','ESC ','HOM ','END ','PU  ','PD  '
CODES   DW 3B00H,3C00H,3D00H,3E00H,3F00H
        DW 4000H,4100H,4200H,4300H,4400H
        DW 1C0DH,011BH,4700H,4F00H,4900H,5100H
;
NUM     EQU ($ - OFFSET CODES) / 2 + 1
;
ERROR1: LEA DX,ARGERR
        JMP SHORT ERROR0
;
CHECK:  LODSB
        CMP AL,'
        JNE CH0
        LEA DX,NOARG
ERROR0: POP AX
ERROREXIT:
        MOV AH,9
        INT 21H
        INT 20H
;
CH0:    CMP AL,27H
        JNE LETTER
        LODSB
        MOV BL,AL
        LODSB
        CMP AL,27H
        JNE ERROR1
        RET
;


LETTER: LEA DI,TABLE
        DEC SI
        XOR BX,BX
        MOV CX,NUM
SCAN:   DEC CX
        JCXZ ERROR1
        PUSH CX
        PUSH SI
        ADD BX,2
        MOV CX,4
        REP CMPSB
        PUSHF
        ADD DI,CX
        POPF
        POP SI
        POP CX
        JNZ SCAN
        MOV AX,CODES[BX - 2]
        RET
;
SKIP:   CLD
        MOV AX,0
        INT 33H
        CMP AX,0FFFFH
        LEA DX,NOTPRESENT
        JNE ERROREXIT
        MOV SI,5DH
        CALL CHECK
        MOV CS:LEFTBUTTON,AX
        MOV SI,6DH
        CALL CHECK
        MOV CS:RIGHTBUTTON,AX
        MOV AX,0CH
        MOV CX,0BH
        LEA DX,MOUSE
        INT 33H
        LEA DX,DONE
        MOV AH,9
        INT 21H
        LEA DX,END_OF_RESIDENT_PART
        INT 27H
        INT 20H
CODE    ENDS
        END START
```

Figura 4 - Listato del programma proposto, che sfrutta la funzione 0CH di attivazione di subroutine d'utente ogni volta che il mouse viene mosso o i suoi pulsanti vengono premuti.

combaciano con una di quelle che possiede al suo interno e che formano la tabella di indirizzo TABLE.

Nel caso in cui uno o entrambi i parametri siano stati introdotti erroneamente oppure addirittura non introdotti, allora provvederà ad emettere un messaggio d'errore.

— In caso positivo di parametri buoni, calcola un valore (una word) che la subroutine «d'utente» andrà in seguito a porre nel buffer di tastiera per simulare la pressione di un certo tasto.

— Infine aggancia la routine «d'utente» al driver del mouse per mezzo della chiamata alla funzione 0CH, emette un messaggio e si «eclissa» rendendo residente solo la routine che interessa.

In tal modo il driver del mouse, ogni volta che accade un evento di quelli visti in precedenza, attiverà la nostra routine, il cui nome è (con poca fantasia...) MOUSE.

Tale routine effettua le seguenti funzioni:

— testa il registro AX («condition mask») per vedere quale evento ha innescato la routine.

— Se è stato premuto un pulsante allora semplicemente carica in BX il codice corrispondente al tasto definito all'atto della chiamata al programma e poi salta alla routine INSERTKEY della quale parliamo nel seguito.

— Se è stato mosso il mouse, allora effettua la chiamata alla funzione 0BH la quale, lo ricordiamo, fornisce in CX e DX dei valori che rappresentano di quanto si è spostato il mouse rispetto all'ultima volta che tale routine è stata chiamata: in particolare ricordiamo che tali valori sono dotati di segno e consentono di stabilire in quale delle quattro direzioni è stato spostato il mouse.

Trovata dunque la direzione, in BX verrà posto il codice corrispondente alla relativa freccia, dopodiché si passa alla routine di inserimento nel buffer di tastiera.

— Avuto dunque in BX il codice del tasto da simulare, inserisce tale valore nella coda circolare di tastiera, secondo meccanismi lievemente complessi e sui quali ora non intendiamo entrare nel dettaglio se non per dire che questa operazione è fatta ad interrupt disabilitati, per evitare ingerenze dall'esterno.

— La routine «d'utente» termina con una RET, che ricordiamo deve essere di tipo FAR: nel nostro caso ciò si ottiene definendo la procedura MOUSE di tipo far con la direttiva:

```
MOUSE PROC FAR
```

Come si vede dunque il programma in esame è molto semplice, anche se utilizza particolari trucchi di programmazione che sfruttano al meglio le risorse fornite dal sistema operativo: ad esempio per il controllo dei parametri viene sfruttato il fatto che il DOS spezza la stringa contenente i due parametri forniti all'atto della chiamata del programma, andandoli a porre in due zone ben determinate della memoria, evitando al programma di accollarsi il compito alquanto gravoso di scomporre la linea di comando in più campi da testare poi singolarmente.

Con ciò abbiamo terminato e diamo l'appuntamento alla prossima puntata, nella quale proseguiremo l'analisi delle rimanenti funzioni del driver di gestione del mouse. MC