di Pierluigi Panunzi

I «device driver»

sesta parte

Come abbiamo detto la volta scorsa, abbiamo terminato la parte puramente teorica relativa ai «device driver» ed appunto in questa puntata analizzeremo «dal vivo» un driver molto ben noto, quell'«ANSI.SYS» omnipresente nei dischetti di sistema e che serve per permettere la corretta interpretazione e perciò rappresentazione di quelle particolari sequenze di caratteri dette «sequenze ANSI»

Le sequenze ANSI

Con tale nome vengono appunto indicate delle stringhe di caratteri, ideate e standardizzate appunto dall'ANSI, inizialmente ed universalmente dedicate alla gestione di terminali video, in generale connessi a grandi calcolatori.

Fin dalle prime applicazioni i terminali servivano unicamente come una rudimentale interfaccia uomo-macchina, avendo il compito di mostrare pedissequamente sullo schermo tutti i caratteri provenienti dalla linea seriale (a 110 Baud se non alla velocità straordinaria di 300 Baud...) e viceversa di inviare al computer i caratteri digitati tramite la tastiera, previa la loro visualizzazione sul video («eco»), fatto questo che consente il controllo di quanto si sta digitando.

Al di là dei caratteri alfanumerici i primi terminali rispondevano «intelligentemente» ad una manciata di cosiddetti caratteri di controllo, i quali non venivano rappresentati su video, ma viceversa servivano ad eseguire qualcosa di importante, quale, come ben noto, il mandare il cursore all'inizio della linea («Carriage Return»), oppure nella linea successiva («Line Feed»), oppure per cancellare il carattere precedentemente inviato («BackSpace») oppure ancora per far emettere un «beep» per richiamare l'attenzione dell'operatore («Bell»).

Sono nate poi man mano esigenze maggiori, quali quella di poter posizionare e successivamente muovere il cursore in un punto qualunque dello schermo, di poter cancellare parti di schermo, di definire attributi ai caratteri stessi (il lampeggio, l'elevata intensità, la sottolineatura, ecc.) oppure infine di poter effettuare altre operazioni particolari (ad esempio definizione di finestre e di tipi di scrolling, ecc.).

Ecco che dunque parallelamente fiorivano varie «famiglie» di terminali, che raggruppavano terminali (di case differenti) i quali potevano rispondere ad un ben determinato set di tali funzioni aggiuntive: basta citare i terminali «VT52», per poi passare ai «VT100» ed ai «VT220», ecc.: l'opera di standardizzazione è avvenuta appunto grazie all'ANSI, che ha definito una serie di sequenze di caratteri ognuna relativa ad un particolare comportamento del terminale.

Perciò definito un set di sequenze per i «VT52», ecco che tutti i terminali che le rispettavano potevano dirsi appartenenti ad un'unica famiglia, quella dei «VT52».

Con l'avvento poi dei personal computer, è nata successivamente l'esigenza di connettere tali apparati ai mainframe per mezzo di una porta seriale e di un apposito programma di comunicazione: se tale programma faceva sì che fossero rappresentate correttamente le sequenze ANSI di un certo terminale (ad esempio il VT100), allora si dice che il personal era diventato un «emulatore VT100».

A proposito, con il miglioramento dei terminali e l'emulazione tramite personal computer, i 300 baud oramai non sono che un lontanissimo ricordo, potendo oggigiorno lavorare (quasi uno standard) tranquillamente a 9600 Baud: mentre prima per riempire completamente uno schermo di 2000 caratteri ci voleva (con un bit di start e due di stop) più un minuto, a 9600 Baud tale tempo si è ridotto a poco più di 2 secondi.

Il nostro PC non poteva essere da meno e perciò è fiorita tutta una serie di programmi di comunicazione (ne citiamo un paio a caso; il «Cross-talk» e lo «Smart-term») ognuno rispondente (e perciò «emulatore») ad un certo set di sequenze ANSI oppure addirittura a seconda di come è programmato: proprio lo «Smart-Term» ha la possibilità di emulare più di un protocollo ANSI, a scelta dell'operatore.

Questo per quanto riguarda i veri e proprio programmi di comunicazione: infatti in generale, grazie al «driver» ANSI.SYS il nostro fedele PC è in grado di per sé di riconoscere un certo insieme di sequenze ANSI, in particolare un sottoinsieme di quelle relative al terminale «VT100».

In tal modo, ponendo come ben noto il comando

```
DEVICE = ANSI.SYS
```

all'interno del file CONFIG.SYS, ecco che il nostro computer risponderà a

particolari sequenze ANSI, delle quali abbiamo già parlato la scorsa puntata.

Ricordiamo a tal proposito che una sequenza ANSI è sempre riconoscibile per il fatto che i suoi primi due caratteri sono rispettivamente «escape» (1BH) e «[», seguiti da uno o più valori numerici (separati nel caso da un carattere «;») e termina infine con una lettera che specifica la funzione associata alla sequenza stessa: in definitiva, a parte il carattere «escape», tutte le informazioni successive sono espresse in caratteri ASCII.

Così ad esempio un valore numerico pari a 10 verrà espresso con i due caratteri «1» e «0».

Per non ripeterci troppo, rimandiamo i lettori interessati al significato della sequenza ANSI alla scorsa puntata: in questa per comodità riportiamo di nuovo la tabella riassuntiva, sempre utile, che consigliamo magari di fotocopiare ed incollare su di un cartoncino, per averla sempre a portata di mano all'occorrenza.

Il driver vero e proprio: ANSI.SYS

Eccoci dunque ad analizzare finalmente del driver di cui abbiamo parlato moltissime volte: la nostra indagine si calerà all'interno del programma, dal punto di vista dell'Assembler e per questo motivo ci siamo avvalsi di mezzi che praticamente tutti i possessori di PC e compatibili hanno a loro disposizione e cioè il ben noto DEBUG, presente in tutti i dischetti di sistema.

L'analisi è in partenza molto semplificata dal fatto che il file in esame (ANSI.SYS) è di appena 1651 byte, nella versione DOS 3.10, che scendono a 1647 nella versione DOS 3.3: comunque noi ci occuperemo solo in parte e non certo nei minimissimi dettagli, della prima, già dicendo che non ci dovrebbero essere differenze enormi tra una versione e le altre (anche quelle non citate).

Lanciato dunque il DEBUG possiamo subito riconoscere ANSI.SYS come un device driver: infatti i primissimi byte del file stesso sono i seguenti:

FF FF FF FF 13 80 A2 00 AD 00 43 4F 4E 20 20 20 20 20

che non sono altro che i cinque campi presenti nell'header di un device driver.

In particolare si ha:

— il puntatore all'header dell'eventuale driver successivo: nel nostro caso il valore del pointer, espresso come coppia di word, è FFFF:FFFF (relativo ad un valore pari a «−1» se indicato come doubleword), così come vuole la regola nel caso in cui il programma contenga un solo driver.

— L'attributo del driver espresso nella word 8013 (questi valori sono sempre ed ovviamente esadecimali!!) viene rappresentato in binario da

1000 0000 0001 0011

dove cioè sono settati i bit 15,(4),1 e 0, relativi rispettivamente al fatto che il driver è di tipo «character» e rappresenta il dispositivo corrente e standard sia di input che di output: sul bit 4 nulla è dato sapere, anzi c'è da dire che, facendo parte di un gruppo di bit riservati al DOS, dovrebbe essere in realtà posto a 0.

— Il puntatore alla «device strategy routine» è pari a 00A2 ed infatti a tale offset troviamo la routine che si occupa di salvare in memoria l'indirizzo del «Request Header» e cioè

```
MOV CS:[009E],BX
MOV CS:[00A0],ES
RETF
```

Con tale piccola routine infatti viene posto in memoria (agli indirizzi 009E e 00A0, all'interno del Code Segment) il contenuto della coppia di registri ES:BX, contenente appunto l'indirizzo della tabella «request header», che poi il driver stesso andrà a leggere per svolgere la funzione richiesta dal DOS.

— Il puntatore alla «device interrupt routine» invece è pari a 00AD, dove troveremo la routine di gestione del «request header», della quale parleremo nel seguito.

— Infine l'ultimo campo, rappresentante il nome dell'unità è dato dalla stringa «CON », rappresentante appunto che si tratta di un device strettamente connesso con il dispositivo «CON:», la console per antonomasia.

Detto dunque della «device strategy routine», parliamo ora della «device interrupt routine».

La «device interrupt routine»

Si tratta dunque della routine che, in base al valore del «command code», esegue la corrispondente parte di programma relativa alla funzione desiderata: nel nostro caso l'«interrupt routine» è riportata in figura 2.

Andiamo dunque ad analizzare per grandi linee il funzionamento di tale routine: in essa, dopo il salvataggio nello stack di quasi tutti i registri della CPU, troviamo il caricamento dei registri CX ed AL con dei valori presi dal «request header» e cioè rispettivamente un «byte count» (utile per routine di I/O multiple e cioè relative a più di un byte alla volta) nonché, importantissimo, il «command code», indicante, come sappiamo, quale funzione viene richiesta al driver.

nome	sequenza	significato
CUP	[#];[#]H	"CUrsor Position" : posizionamento del cursore
HVP	[#];[#]f	"Horizontal and Vertical Position" : posizionamento del cursore
CUU	[#]A	"CUrsor Up" : muove il cursore in alto
CUD	[#]B	"CUrsor Down" : muove il cursore in basso
CUF	[#]C	"CUrsor Forward" : muove il cursore a destra
CUB	[#]D	"CUrsor Backward" : muove il cursore a sinistra
DSR	6n	"Device Status Report" : segnala la posizione del cursore
SCP	s	"Save Cursor Position" : salva la posizione del cursore
RCP	u	"Restore Cursor Position" : ripristina la posizione del cursore
ED	2J	"Erase display" : cancella lo schermo (equivale al CLS)
EL	K	"Erase Line" : cancella dal cursore alla fine linea
SGR	#;[#]m	"Set Graphic Rendition" : setta caratteristiche dell'output
SM	#h	"Set Mode" : setta il modo video
RM	#l	"Reset Mode" : resetta il modo video
KKR	#;"s"p	"Keyboard Key Reassignment" : associa la stringa "s" ad un tasto
KKR	"c";"s"p	"Keyboard Key Reassignment" : associa la stringa "s" al tasto "c"

Figura 1

Con tale valore si entrerà all'interno di una «jump table», posta a partire dall'indirizzo 0012, nella quale trovano posto tutti gli indirizzi delle routine che implementano il set di funzioni possibili: in figura 3 possiamo vedere l'elenco delle routine con i relativi «entry point».

In particolare possiamo vedere che vi sono alcune routine che condividono lo stesso «entry point» (00DE), che in realtà è (ci sia concesso il gioco di parole...) un «exit point» del driver stesso, come si può vedere dal listato precedente: infatti tutte queste routine non sono state implementate e non fanno altro che settare il bit «DONE» nella word di stato.

Viceversa nel caso in cui venga richiesta la routine «INPUT IOCTL», allora si avrebbe un salto all'indirizzo 00D8: tornando ad analizzare l'«interrupt routine» vediamo che a tale indirizzo si salta pure nel caso in cui il «command code» supera 0B. Da qui si uscirà dalla routine dopo aver settato, nella word di stato del «request header», i bit «DONE», «ERROR» ed «UNKNOWN COMMAND».

Nel caso invece in cui la funzione richiesta è lecita allora si salta proprio all'entry point il cui indirizzo all'interno della tabella è posto nella word puntata dal registro SI. Infine, al termine dell'esecuzione della routine si passerà per l'indirizzo 00DE, laddove verrà aggiornata la word di stato, verranno ripristinati i registri che erano stati salvati nello stesso stack ed infine si ritornerà al programma chiamante (il DOS stesso).

INDIRIZZO	COM.	ROUTINE
062F	0	INIT
00DE	1	MEDIA CHECK
00DE	2	BUILD BPB
00D8	3	IOCTL INPUT
0200	4	INPUT
0282	5	NONDESTRUCTIVE INPUT NO WAIT
00DE	6	INPUT STATUS
02C6	7	INPUT FLUSH
02E2	8	OUTPUT
02E2	9	OUTPUT WITH VERIFY
00DE	A	OUTPUT STATUS
00DE	B	NOUTPUT FLUSH

Figura 3 Tabella degli «entry point» delle varie routine gestibili con il driver ANSI.SYS: come spiegato nell'articolo, alcune di queste routine non sono in realtà implementate.

```
00AD: 56              PUSH SI
00AE: 50              PUSH AX
00AF: 51              PUSH CX
00B0: 52              PUSH DX
00B1: 57              PUSH DI
00B2: 55              PUSH BP
00B3: 1E              PUSH DS
00B4: 06              PUSH ES
00B5: 53              PUSH BX
00B6: 2EC51E7E00      LDS DX,DWORD PTR CS:[007E]
00BB: 8B4F12          MOV CX,[BX+12]
00BE: 8A4702          MOV AL,[BX+02]
00C1: 98              CBW
00C2: BE1200          MOV SI,0012
00C5: 03F0            ADD SI,AX
00C7: 03F0            ADD SI,AX
00C9: 3C0B            CMP AL,0B
00CB: 770B            JA 00D8
00CD: C47F0E          LES DI,DWORD PTR [BX+0E]
00D0: 0E              PUSH CS
00D1: 1F              POP DS
00D2: FF24            JMP [SI]

00D4: B403            MOV AH,03
00D6: EB08            JMP 00E0

00D8: B003            MOV AL,03
00DA: B481            MOV AH,81
00DC: EB02            JMP 00E0

00DE: B401            MOV AH,01
00E0: 2EC51E7E00      LDS BX,DWORD PTR CS:[007E]
00E5: 894703          MOV [BX+03],AX
00E8: 5B              POP BX
00E9: 07              POP ES
00EA: 1F              POP DS
00EB: 5D              POP BP
00EC: 5F              POP DI
00ED: 5A              POP DX
00EE: 59              POP CX
00EF: 58              POP AX
00F0: 5E              POP SI
00F1: CB              RETF
```

Figura 2 - Disassemblato della «device interrupt routine».

La routine INIT

Come ben sappiamo, questa routine serve per inizializzare il driver, le sue variabili, eventuali contatori, ecc. e viene chiamata solamente una volta all'inizio dei tempi: dato che nel «request header» relativo alla funzione INIT è presente un campo in cui il driver deve porre il suo ultimo indirizzo, a partire dal quale il DOS possa caricare altri programmi, ecco che a questo indirizzo viene di solito posta la routine INIT la quale, dopo la sua attivazione, non serve più ed altrimenti occuperebbe spazio prezioso.

La routine disassemblata riportata in figura 4.

Ma vediamo con calma le operazioni svolte da tale routine:

— dapprima viene analizzata la configurazione hardware del nostro PC, per mezzo della chiamata all'INT 11, grazie alla quale si è in grado di sapere se è presente o meno una scheda monocromatica: in questo caso viene posto il valore B000 in una certa locazione di memoria, che originariamente conteneva il valore B800.

Questi valori non sono altro che i segmenti a cui ha inizio la memoria video rispettivamente per un adattatore monocromatico (anche una scheda Hercules) o per una CGA, in ogni caso «in modo testo» e quando si possono rappresentare su video solo caratteri alfanumerici.

— Successivamente, nel caso in cui il modo video preveda l'output di 40 colonne, invece di 80, ecco che vengono

```
062F CD11           INT 11
0631 2430           AND AL,30
0633 3C30           CMP AL,30
0635 7506           JNZ 063D
0637 C706170100BC   MOV WORD PTR [0117],BC00
063D 3C10           CMP AL,10
063F 770A           JA 064B
0641 C606FF0000     MOV BYTE PTR [00FF],00
0646 C606000127     MOV BYTE PTR [0100],27
064B 33DB           XOR BX,BX
064D 8EDB           MOV DS,BX
064F BB6C00         MOV BX,006C
0652 C707F200       MOV WORD PTR [BX],00F2
0656 8C4F02         MOV [BX+02],CS
0659 BBA400         MOV BX,00A4
065C C707EF02       MOV WORD PTR [BX],02EF
0660 8C4F02         MOV [BX+02],CS
0663 2EC51E9E00     LDS BX,DWORD PTR CS:[009E]
0668 C7470E2F06     MOV WORD PTR [BX+0E],062F
066D 8C4F10         MOV [BX+10],CS
0670 E96BFA         JMP CODE
```

Figura 4
Disassemblato della routine che implementa la funzione INIT.

resettate altre due locazioni di memoria, una delle quali servirà da contatore di colonne per il «wrap around»

— Vengono cambiati i due «interrupt vector» relativi agli INT 1B ed INT 29, il primo dei quali serve per gestire il «break» da tastiera, ottenibile in genere con «Ctrl-C» o «Ctrl-Break» (o meglio «Ctrl-Scroll Lock»), mentre sul secondo non sappiamo dire di più in quanto si tratta di una funzione riservata al DOS: analizzandola si vede che si tratta di un modo alternativo di emettere in output il carattere posto in AL.

— Infine viene posto al valore 062F (l'entry point della routine INIT) il campo di cui abbiamo parlato sopra e relativo all'ultimo indirizzo occupato dal driver, o meglio la prima cella «libera per il DOS».

Le funzioni di INPUT

Sulla funzione INPUT, e su quelle ad essa collegate, non ci fermiamo in quanto poco interessanti per il nostro discorso: in generale sfruttano l'INT 16, che è appunto l'interrupt che serve a sentire se è stato premuto un tasto ed in caso positivo a fornirlo in maniera opportuna al programma chiamante.

In questo caso, in parole povere, viene gestito un buffer nonché una coppia di contatori che istante per istante sanno in quale posizione dello schermo è posto il cursore: come si vede in questo caso particolare l'input da tastiera e l'output su video sono intimamente legati dal fatto che nel 99% dei casi ad un input corrisponde un'eco (output) su video.

Dato che l'analisi delle funzioni di OUTPUT richiederebbero molto spazio, preferiamo terminare così questa puntata, dando l'appuntamento alla prossima, con la quale termineremo l'argomento «ANSI.SYS». MC