60

MEMORY RESIDENCY

by

Z. Nazoordeen,
BSc(Eng), AMIE(SL)
Engineer, Project Division, Fentons Limited

Abstract

A major limitation in MS-DOS is its lack of support for multi-tasking. That is executing several tasks simultaneously. Usually most of the computing power (several Million Floating Point Operations per second. MFLOPS) of ever improving microcomputers is left unharnessed.

To develop resident programs, it is required to study BIOS and DOS interrupts, together with some undocumented features in little depth. By installing and selecting proper applications of TSR (Terminate and Stay Resident) the single user, single processor MS-DOS can be converted into a multitasking environment. In this way the RAM area can be fully utilised for application programs, rather than having multitasking operating systems which consumes a larger area of RAM.

In this paper we will be discussing.

- the problems associated in developing a TSR
- necessary, documented and undocumented BIOS and DOS services
- the possibility of making MS-DOS a multitasking operating system
- advantages of TSR over conventional application programs.

1.0 Introduction

MS-DOS is designed to be a single user, single processor operating system. IF you want a multi-tasking environment you have to go for some other operating system like XENIX, OS/2 or MS DOS with Microsoft Windows. Taking advantage of the undocumented features of DOS, we are able to make DOS an actual multi-tasking operating environment. The type of software that makes DOS multi-tasking should be in the form of TSR (Terminate and Stay Resident).

2.0 What actually is a TSR

A TSR is a program that loads itself into memory and returns control to DOS, but remains dormant in the background. While you are running your application software in the DOS, the TSR is still resident in the memory waiting to be activated or already executing some other pre-assigned task. This way you can go for multi-tasking by sharing the CPU. Multiple number of TSRs can be installed, as far as they co-exist peacefully with each other and the DOS.

2.1 What can a TSR do?

Once a TSR is installed, it stays resident in the memory. When the user presses a designated key combination (the hot key), the TSR comes to life, immediately interruping whatever application is currently running, and, allowing instant access to the services it offers. Or the user may be running his application program, while a distant PABX (Private Branch Automatic Exchange) interrupts the computer to transfer data to TSR without disturbing the user.

MS-DOS provides a resident program PRINT.COM which allows you to print files in the background. It was only from this program that the world came to know about the existence of TSR in MS-DOS. Microsoft officially maintains that TSRs are not possible.

The TSR can interrupt the foreground application by

- the keyboard, usually the key combination is called a 'Hot key'
- the system timer interrupt, which can be invoked once in every 54.9 milliseconds
- communication port interrupts (COM1 & COM2), by arrival of data or using handshake signals
- parrallel centronics port (printer port), by out of paper, not ready signals

The Interrupt Vector Table (IVT) in fig. 1 shows these hardware interrupts which directly get attention of the microprocessor 80x86 via. their dedicated connections. (IORQO - IORQ7). Apart from that there are software interrupts that BIOS and DOS services use. These software interrupts can also be used in a TSR, but then the TSR has to wait till the foreground program uses the particular service. This is the technique Computer Virus, a misused offspring of TSR, effectively employs.

There are several advantages in using TSR, instead of conventional programs that are loaded from the DOS command line.

*** Speed**

Utilities such as notepads, calculators, spell checkers and outline can be rapidly accessed without having to forego the current application, type a DOS command and then restart the program. The speed saving is specially significant when a large and complex application is being run in the foreground.

*** Multitasking**

Accessing communication ports to capture/transmit data to/from a file in the disk, sending large files or documents to the printer, or sorting out a huge data-base etc. You can do all these tasks and many more in the background while you are running your software in DOS in foreground, all simultaneously.

*** Data exchange**

Many TSR's, especially notepad editors, macroprocessors, and outliners, allow the importing and exporting of data between the utility and the foreground application. These programs thus provide conveniet channels for textual and graphic data exchange between dissimilar programs and files of heterorgeneous format.

*** Modularity**

Loading a selected set of resident utilities, rather than running a single integrated package, allows the user to design an optimal computing environment. For an example a favourite word processor or a program editor can be run in the foreground, and combined with a favourite memory-resident thesaurus, spell checker, outliner, notepad etc.

3.0 The challenge of Writing a TSR

Developing a TSR is an adventure. The primary challenge is to write a memory-resident program that can peacefully coexist with other TSRs, with the operating system, with the interrupted foreground application. The next step is to tailor it to your requirements.

3.1 How to begin writing a TSR?

First of all you have to decide on the language which is to be used to develop the TSR. Turbo C is a good candidate, but you should get enough literature to handle BIOS interrupts, DOS interrupts, IO port access. Turbo C has supporting commands like getvect(), setvect(), keep() to start writing TSRs. Microsoft C also can be used similarly.

Assembly language is the one chosen to develop TSRSDS. BIOS and DOS interrupts, hardware ports, IVT, can directly and easily be accessed by assembly language.

If you have Microsoft Macro assembler, 80x86 Assembly language instruction set, BIOS & DOS service details, you can directly start writing a TSR.

3.2 Software development

Developing a TSR can be effected as follows.

Initialisation code

a) Saving the current hardware interrupt vector address using INT 21h service 35h

> In the program TSRSDS lines 157-159 get the timer interrupt vector, and from line 167 - 175 it is saved at the address 'com_normal'.

b) Enable the particular interrupt you intend to use and change the interrupt vector address to point the Interrupt handler of our TSR using INT 21h, service 25h

> Lines 184-187 of TSRSDS shows the new address 'new' is placed in the IVT (Interrupt Vector Table). Interrupt handler of TSRSDS begins from this address 'new' (line 1).

c) Make the program to Terminate and Stay Resident using INT 21h service 31h.

> Making the program resident is done from lines 188-191, reserving one kilobyte of RAM.

Interrupt handler

d) Write your software beginning from the address pointed by (b) and at the end instruct it to go back to old address saved by (a).

> The rest of the program TSRSDS, line 1-147 is the Interrupt handler.

As such whenever the hardware interrupt occurs, the CPU stops the current process and looks for the IVT and jumps to designated new address. This is where our TSR stays. Now the CPU executes our program. At the end it had been instructed to jump back to the old interrupt vector address. This will in turn direct the CPU to return back to the interrupted process.

The example Resident Program TSRSDS has been written to capture call information data coming from the PABX via the serial port and to save it in the hard disk.

The system timer interrupt is employed to activate TSRDS. For every 52.9 millisccond interval CPU checks whether the PABX is ready to send call information by checking the status of DSR, CTS of RS232 C serial port. If the PABX is ready TSRSDS initiates the process of capturing data. If not, returns back to the interrupted process.

In the meanwhile the user may be writing a C programme or running Lotus 123, or using the Word processor or playing Chess with the computer. He may not notice the background process.

4.0 Coexisting with MS-DOS

The next perhaps greatest challenge in designing a TSR is to make it compatible with MS-DOS operating system so that the Memory resident program can freely use DOS interrupt service. The majority of MS-DOS kernal is not 're-entrant'; meaning it cannot be called by an interrupt handler if it was already executing at the time when the interrupt occured.

For example, invoking DOS request for a file function by a TSR while DOS is already performing a file function. In other words in TSRSDS we cannot store the captured data in the hard disk because DOS interrupts need to be invoked in this process.

What is the solution to this problem?

There are two possible approaches;

The first approach would be to totally eliminate the use of DOS functions from within the memory resident code. You would have to write all routines using BIOS services and lower level techniques. Writing assembly routines for file creation, file opening, getting file handlers, writing files, and closing files, are definitely going to be tedious.

The second approach is to postpone the activation of the TSR until MS-DOS is inactive. In this way all DOS functions can be invoked freely. How do we know whether the DOS is currently active or not? MS-DOS itself provides a solution through undocumented but well-known interrupt 21h and function number 34h. This interrupt returns a pointer to a flag maintained by DOS which indicates whether the DOS code is currently active. If this Indos flag is 0, then DOS is inactive and if it is greater than 0, DOS is active.

This Indos function should be called in the initialization process (Indos function itself is a DOS interrupt) and the double word pointer to the indos flag saved where it may be subsequently accessed by the interrupt handler of the TSR.

There is one more troublesome spot. That is when the COMMAND.COM displays its prompt and waits for user input, a DOS service being invoked by the COMMAND.COM. In this case the indos flag does not permit us to activate the interrupt handler, while the computer is almost idling. What do we do? There is again a solution provided by MS-DOS. At this idle state COMMAND.COM continuously invokes interrupt 28h to read the key board. This DOS interrupt is an exceptional one. It is relatively safe to invoke a DOS service within this particular interrupt.

To summarize, our overall strategy is;

- In the initilization code use both system timer interrupt as well as DOS Interrupt 28h, and save the Indos flag pointer.

  - In TSRSDS line 153 to 156 shows how to retrieve the Indos flag pointer. This pointer is saved at the address vector, 'indos-ptr' (segment address 'indos-seg', offset address 'indos-off')

  - Saving and installing the DOS idle interrupt INT 28h is executed in lines 179 to 187. This is same as saving the system timer interrupt. Old interrupt address is saved in address 'old-int28' and then 'new-int28' address is installed. The 'new-int28' address points our new interrupt handler.

- Once inside the Interrupt handler check the Indos flag.

  Is the previous process interrupted by the system timer?

  Exit or execute accordingly.

  If it was interrupted from the DOS service 28h, there is no need to check the indos flag. Interrupt handler can be executed straight away.

  - Accordingly the interrupt handler of TSRSDS follows these steps

First of all when it enters the interrupt handler, checks whether the PABX is ready by scanning the line signals CTS & and DSR (lines 22 - 28). In case PABX not ready goes back to the foreground application.

If PABX is ready then it checks the status flags dos-idle flag, indos flag in lines 29 - 38. If it is unsafe, activation of the next part of interrupt handler is postponed, and instructed to exit to foreground application.

If the flags permit the TSRSDS enters the critical part of interrupt handler, and

- .. initialize the temporary buffers (lines 39 - 41)
  .. set serial port parameters (lines 42 - 48)
  .. signal tPABX to send data by DTR & RTS (lines 46 - 48)
  .. capture data to a temporary buffer ('data-buf') from the serial port (lines 49 - 74)
  .. save temporary buffer to the PABX.DAT file in Hard disk by calling subroutine 'save-dat' (lines 103 - 146)
  .. finally return back to the foreground by calling old interrupt address (lines 90 - 102)

Concluding there are two more points in TSRSDS which need to be mentioned.

- First one is the setting up a 'busy' flag (lines 19 - 21) to prevent recursive entrance to the interrupt handler. That is while executing interrupt handler, hardware interrupts may occur again forcing it to re-enter into the interrupt handler.

- - Next one is adding a fault recovery routine to breakaway from a infinite loop situation inside TSRSDS. (lines 49- 56). Due to problems in handshake signals (CTS,DSR,DTR,RTS) or conflicts in RS232 protocols (baud rate, parity, stop bit, data length, end of transmission indication) the infinite loop may occur.

5.0 Conclusion

The DOS platform which does not support multi - tasking operations can be converted to a multi - tasking, time sharing environment by employing TSRs. TSRs have to be properly designed and tailored to suit the requirements.

The limitations in this technique are,

- Inability to use DOS services freely inside a TSR. We have discussed the methods to overcome this situation. Still you might encounter two concurrent real time applications with critical timings trying to access DOS services simultaneously.

- Another limitaion is the trade off between the Computing Power and no. Of multi - tasks being employed. The Computing power of the computer has to be shared by all active multi - tasks and the environment program. The power of the micro - computer is measured by the number of Million Floating Point Operations per Second (MFLOPS) whereas in a SuperComputer it is in the range of Giga FLOPS. With the advancement of Technology it is always in a state of improvement.

6.0 References

1. EFY Enterprises Pvt Ltd. Electronics for you, December 1992

2. Hall V. Douglas. Microprocessors and Interfacing Programming and Hardware, Mc Graw Hill International Edition 1986

3. Micro Soft Corporation, Programmers' Guide Microsoft Macro Assembler 5.1 1987

4. Miller Allen L. Assembly Language Techniques for the IBM PC BPB Publications (First Edition 1988)

5. Mukhi Vijay The 'C' Odessy Advanced Dos - The Uncharted waters, Tech Publications Pte Ltd (First edition 1992)

6. Wyatt Allen L. Using Assembly Languages Micro - Tech Publications (Asian editon 1987)

7. Young Micheal J. MS - DOS Advanced programming.

APPENDIX

Number	Address	Name	Owner
IRQ 00	101C:003C	Timer output	DOS System Area
IRQ 01	101C:0045	Keyboard	DOS System Area
IRQ 02	F000:DBDE	[Cascade]	BIOS
IRQ 03	F000:DBDE	COM2	BIOS
IRQ 04	F000:DBDE	COM1	BIOS
IRQ 05	F000:DBDE	LPT2	BIOS
IRQ 06	101C:00B7	Floppy Diek	DOS System Area
IRQ 07	0070:06F4	LPT1	DOS System area

fig. 1. Interrupt Vector Table (IVT) of Hardware interrupts

```
;Program name    - tsrsds
;Purpose         - Resident program to capture serial data
;                - identifies copies of itself
;                - identifies infinite loop
;Interrupt       - System timer, Dos idle,  Indos flag
;Serial port     - COM1  2400 N 1 8
;Designed by     - Ziath Nazoordeen
;Date            - 10th August 1993

CODE            SEGMENT BYTE PUBLIC 'CODE'
                ORG     100h
KEYHARD         PROC    FAR
                ASSUME  CS:CODE,DS:CODE

strt:           JMP     over_data
message         DB      ' TSR installed..   $'
com_normal      DD      0
data_buf        DB      200 DUP(0)
busy            DB      0
dat_len         DB      0
data_file       DB      "C:\pabx.dat",0
file_len        DB      0
indos_ptr       label   dword
indos_off       dw      ?
indos_seg       dw      ?
no_copy         DB      'in background. No copies allowed again.  $'
error_level     DW      0
old_int28       DD      0
dos_idle        DB      0

                ASSUME  DS:NOTHING
 1. new:            MOV     dos_idle,0          ;not dos idle
 2.                 JMP     avoid_int28
 3. new_int28:      MOV     dos_idle,1          ;dos_idle flag set
 4. avoid_int28:    PUSHF
 5.                 PUSH    AX
 6.                 PUSH    CX
 7.                 PUSH    DX
 8.                 PUSH    SI
 9.                 PUSH    BX
10.                 PUSH    DI
11.                 PUSH    DS
12.                 PUSH    ES
13.                 PUSHF
14.                 PUSH    CS
15.                 POP     DS
16.                 JMP     adr1
17.  fin:           JMP     get_out
18. adr1:           CLI

19.                 CMP     CS:busy,1           ;check busy status to sto
20.                 JE      fin                 ;recursion
21.                 MOV     CS:busy,1           ;busy flag set
```

```
22.              MOV     AX,3                    ;test line status
23.              MOV     DX,00
24.              INT     14h
25.              TEST    AL,00100000b            ;is DSR set
26.              JZ      fin
27.              TEST    AL,00010000b            ;is CTS set
28.              JZ      fin
29.              PUSH    DS
30.              PUSH    BX                      ;load pointer
31.              LDS     BX,CS:indos_ptr         ;to indos flag
32.              CMP     BYTE PTR DS:[BX],0      ;test indos flag
33.              POP     BX
34.              POP     DS
35.              JE      adr2
36.              CMP     CS:dos_idle,1
37.              JE      adr2
38.              JMP     fin                     ;sorry either you are indo
                                                 ;or dos not idle, exit
39.  adr2:       MOV     DI,OFFSET data_buf
40.              MOV     CS:dat_len,0
41.              MOV     CS:file_len,0
42.              MOV     DX,0                    ;set serial port
43.              MOV     AX,0                    ;parameters
44.              MOV     AL,10100011b            ;2400 N 1 8
45.              INT     14h

46.              MOV     AX,03h                  ;set DTR &RTS
47.              MOV     DX,03FCh
48.              OUT     DX,AX

49.              MOV     error_level,0
50.  again:      MOV     DX,03FDh                ;is char ready
51.              IN      AX,DX
52.              TEST    AX,01h
53.              JNZ     next                    ;yes go next
54.              INC     error_level
55.              CMP     error_level,0fffh
56.              JNE     again
57.              JP      rts_off

58.  next:       MOV     DX,03F8h                ;read char from serial
59.              IN      AX,DX                   ;port
60.              MOV     error_level,0
61.              INC     DI
62.              INC     dat_len
63.              INC     file_len
64.              MOV     BYTE PTR [DI],AL

65.              CMP     AL,03d                  ;is it line feed
66.              JNZ     again                   ;no
67.  close:      INC     DI
68.              MOV     BYTE PTR [DI],10
69.              INC     file_len
```

```
rts_off:    MOV     AX,01h              ;RTS off
            MOV     DX,03FCh
            OUT     DX,AX
            CMP     file_len,0
            JE      dont_save
            CALL    save_dat            ;save data in hard disk
dont_save:

get_out:    MOV     CS:busy,0
            MOV     AL,20h
            OUT     20h,AL
            STI
            POPF
            POP     ES
            POP     DS
            POP     DI
            POP     BX
            POP     SI
            POP     DX
            POP     CX
            POP     AX
            POPF
            CMP     CS:dos_idle,1
            JNE     was_not
            JMP     [old_int28]         ;go back to, from where
was_not:    JMP     [com_normal]        ;came


save_dat    PROC    NEAR                ;open file
            CLI
            IN      AL,21h
            PUSH    AX
            OR      AL,11111111b        ;disable hardware interru
            OUT     21h,AL
            MOV     AL,20h
            OUT     20h,AL
            STI
open:

            PUSH    CS
            POP     DS
            MOV     AH,3Dh              ;to write
            MOV     AL,1
            MOV     DX,OFFSET data_file ;get file name
            INT     21h
            JC      ERR
```

```
119.              MOV     BX,AX                   ;get file handle to bx

120.              MOV     AH,42h                  ;move file pointer
121.              MOV     AL,2                    ;relative to end of file
122.              XOR     CX,CX
123.              SUB     DX,DX
124.              MOV     AH,42h
125.              INT     21h


126.              MOV     AH,40h                  ;write file
127.              PUSH    CS
128.              POP     DS
129.              MOV     CL,file_len             ;data length
130.              MOV     CH,0
131.              MOV     DX,OFFSET data_buf      ;offset address of buffer
132.              INT     21h


133.              MOV     AH,3Eh                  ;close the file
134.              INT     21h
135.              JMP     over

136.  ERR:        MOV     AH,3Ch                  ;create the file as it
137.              MOV     CX,0                    ;is not available
138.              INT     21h
139.              JMP     open

140. over:        CLI
141.              POP     AX                      ;enable hardware inerrupts
142.              OUT     21h,AL                  ;again
143.              MOV     AL,20h
144.              OUT     20h,AL
145.              STI

146.              RET

147.  save_dat   ENDP

    over_data:

148.              MOV     DX,OFFSET message       ;installation message
149.              PUSH    CS
150.              POP     DS
151.              MOV     AH,9h
152.              INT     21h

153.              MOV     AH,34h                  ;call DOS to retrieve
154.              INT     21h                     ;pointer indos flag
155.              MOV     CS:indos_off,BX         ;save it to indos flag
156.              MOV     CS:indos_seg,ES

157.              MOV     AL,08h                  ;get the timer tick vector
158.              MOV     AH,35h
159.              INT     21h
```

```
160.                CMP     BX,003Ch                ;already in background?
161.                JE      go_on                   ;IVT IRQ 00 101C:003C
162.                MOV     AH,9                    ;yes, sorry
163.                MOV     DX,OFFSET no_copy       ;message
164.                INT     21h

165.                MOV     AH,4Ch                  ;please exit
166.                INT     21h


167.go_on:          MOV     SI,OFFSET com_normal    ;save the vector
168.                MOV     [SI],BX
169.                MOV     [SI+2],ES
170.                MOV     AX,CS
171.                MOV     DS,AX
172.                MOV     DX,OFFSET new           ;install the new interru
173.                MOV     AL,08h
174.                MOV     AH,25h
175.                INT     21h

176.                MOV     AL,28h                  ;take the DOS idle inter
177.                MOV     AH,35h
178.                INT     21h

179.                MOV     SI,OFFSET old_int28     ;save it
180.                MOV     [SI],BX
181.                MOV     [SI+2],ES
182.                MOV     AX,CS
183.                MOV     DS,AX
184.                MOV     DX,OFFSET new_int28     ;install the new inerrupt
185.                MOV     AL,28h                  ;address
186.                MOV     AH,25h
187.                INT     21h


188.                MOV     AL,0                    ;tsr
189.                MOV     DX,40h                  ;reserve 1K
190.                MOV     AH,31h
191.                INT     21h

    KEYHARD         ENDP

    CODE            ENDS
    END     KEYHARD
```