UNIUNEA EUROPEANĂ
GUVERNUL ROMÂNIEI
Instrumente Structurale 2007-2013

Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Programare în limbaj de asamblare

58. Programe rezidente (TSR). Instalarea, activarea şi dezinstalarea acestor programe.

Programe rezidente în memorie (TSR)

TSR este prescurtarea de la „Terminate and Stay Resident“. Deci, este vorba despre programe care rămân în memoria RAM după ce s-a terminat execuția lor. Aceste programe pot fi invocate fie utilizând anumite taste, fie de către alte programe, care lansează o întrerupere software adecvată. De obicei aceste programe sunt de tip .COM.

Să considerăm ce s-ar întâmpla dacă, în timp ce un program rulează, el ar fi copiat adresa sa „top of memory“ în pointerul special, ce memorează locația de memorie unde DOS încarcă programele pentru execuție. În acest caz, DOS va fi păcălit prin încărcarea fiecărui program ulterior în zona de memorie situată imediat deasupra programului original. În consecință, programul original ar rămâne în memorie pe durata executării tuturor programelor ulterioare. Exact la fel se întâmplă într-un program TSR. Acesta înlocuiește pointerul de început al programului cu pointerul „top of memory“. Fiecare program executat după acest program TSR este încărcat în zona de memorie superioară acestuia şi deci nu se suprapune peste memoria utilizată de programul TSR. Când se încarcă mai multe programe rezidente, fiecare dintre ele utilizează această tehnică.

Un program TSR conține două părți: o parte de inițializare (tranzitorie) şi o parte rezidentă. Partea tranzitorie este responsabilă de încărcarea părții rezidente în memoria RAM şi de instalare a unei rutine de întrerupere care să determine cum este invocat (apelat) programul TSR. Dacă programul TSR este invocat de o întrerupere software, porțiunea plasează adresa părții rezidente a codului în vectorul de întrerupere corespunzător. Dacă TSR-ul va fi invocat printr-o tastă (hotkey) porțiunea de inițializare trebuie să modifice manipularea întreruperilor DOS, pentru tastatură (de fapt pentru tasta respectivă).

Când porțiunea tranzitorie este executată, la sfârşitul acesteia ea invocă o funcție DOS care permite ca o porțiune din fişierul executabil să rămână rezidentă în memoria RAM după încheierea execuției acestuia, de unde şi expresia „Termină şi Stai Rezident“. Porțiunea de inițializare a TSR-ului cunoaşte dimensiunea porțiunii rezidente, precum şi locația în memorie a porțiunii rezidente, şi transmite această informație sistemului DOS. Ulterior, DOS nu va afecta blocul de memorie specificat, dar este liber să încarce programe în zona de memorie neprotejată.

Deoarece sistemul de operare DOS este un sistem monoacces, monoprogramare şi nereentrant, apar o serie de restricții si limitări privind operațiile ce pot fi efectuate de codul rezident al programelor din această categorie. De aceea este necesar să cunoaştem structura sistemului de operare, anumite module care afectează un program rezident, precum şi modul de lucru al unor dispozitive (ca de exemplu tastatura).

Programele TSR pot fi de două tipuri, după modul de activare: *active* şi *pasive*.

Un program TSR pasiv se execută ca parte normală a fluxului unui program; el este executat în urma unui apel explicit dintr-o altă aplicație. TSR-urile pasive sunt aproape întotdeauna rutine de servire a întreruperii sau extind apeluri DOS sau BIOS.

Un program TSR activ este executat constant, la anumite intervale de timp, sau la apăsarea unei anumite combinații de taste care nu are semnificație pentru aplicația curentă. Un program TSR activ, de obicei, va suspenda execuția programului curent. TSR-urile active îndeplinesc una din două funcții: fie îndeplinesc direct o întrerupere hardware, fie sunt activate periodic, fără un apel explicit de la o aplicație.

Un TSR poate conține atât componente active, cât şi componente pasive. Astfel, anumite rutine pot fi activate de o întrerupere hardware, iar altele să fie apelate explicit de alte aplicații.

Următorul program este un exemplu de TSR care furnizează atât o rutină activă (*MyInt9h*), cât şi una pasivă (*MyInt16h*). Programul utilizează vectorii de întrerupere ai tastaturii:

09h şi *16h*. De câte ori apare o întrerupere pentru tastatură, rutina *MyInt9h*, activă pe acest nivel (*INT 09h*), va incrementa un contor. Deoarece tastatura generează, de obicei, două întreruperi la o tastare (una la apăsarea şi alta la revenirea tastei), valoarea va fi împărțită la 2, pentru a aproxima numărul de apăsări ale tastelor. Rutina pasivă pe nivelul 16h, *MyInt16h*, returnează numărul de apăsări ale tastelor către programul apelant. Codul următor furnizează cele două programe, TSR-ul (*tsr_tast.asm*) şi aplicația (*tsr_tapl.asm*) care afişează numărul de taste apăsate de când s-a lansat TSR-ul.

```
.model tiny
; tsr_tast.asm - TSR activ, numără într. tastatură, după activare
; pt. a det. nr. de taste apăsate între două apeluri succesive
; prog. permite şi dezinstalarea lui, la cererea utilizatorului
; Se va genera program de tipul .COM: tlink/t tsr_tast
.code
        org 100h
start:  jmp test_Instalare
ID_tsr db 'TSR_numara_taste_apasate_INT_9h_&_16h'
lung_ID equ $-ID_tsr
KeyCont dw 0 ; contor întreruperi de la tastatură
OldInt9h dd ?          ; salvarea vechilor vectori de întrerupere
OldInt16h dd ?
MyInt9h proc far
; comp. activa, rutina de într. tastatură, apelată de sistem la
; fiecare apăsare de tastă, incrementează contorul KeyCont, şi
; transferă controlul la întreruperea originală, Int 9.
        inc cs:KeyCont      ; incr. contor de întreruperi tastatură
        jmp cs:OldInt9h     ; apelul vechii într., salvată la OldInt9h
MyInt9h endp
MyInt16h proc far
; componenta pasivă a TSR-ului apelată prin apelul INT 16h.
; Dacă registrul AH=0FFh, întoarce nr-ul de întreruperi
; de la tastatură, în AX. Dacă AH conține altă valoare, atunci
; rutina transmite controlul la întreruperea originală INT 16h.
        cmp ah, 0FFh; se solicită valoarea contorului ?
        je ReturnCont       ; dacă da, se returnează valoarea lui
        jmp cs:OldInt16h    ; altfel apelează întreruperea originală
ReturnCont:
        mov ax, cs:KeyCont
        mov cs:KeyCont, 0 ; reinițializare contor la fiecare apel
        iret
MyInt16h endp
test_Instalare:
        push es             ; salvare ES, DS
        push ds
        push cs
        pop ds          ; init. DS cu CS, şi poate lipsi la .COM
```

```
        mov ax, 0
        mov es, ax              ; init. ES cu adr. Seg. a TVI (0000H)
; se verifică dacă nu este deja instalat acest TSR:
; se verifică dacă au fost instalate cele două întreruperi în TVI
        mov ax, es:[9h*4]       ; se comp. offseturile pt. vectorii din tabelă
        cmp ax, offset MyInt9h  ; dacă nu sunt găsiți cei doi vectori
        jne inst                ; se vor instala
        mov ax, es:[16h*4]
        cmp ax, offset MyInt16h
        jne inst
; aici se ştie că sunt instalați 2 noi vectori de întrerupere
; pe nivelurile 9h şi 16h, cu offseturile noastre, dar pt a fi siguri
; că sunt chiar aceştia vom comp. şi id. TSR-ului curent
; cu cel din memorie (din TVI)
        mov ax, es:[9h*4+2]
        mov es, ax
        mov si, offset cs: ID_tsr   ; se compară identificatorii
        mov di, si              ; offset celor două 'nume_TSR' în SI/DI
        mov cx, lung_ID         ; lungimea numelui vectorului nou
        cld                     ; direcția de parcurgere, crescătoare
        repe cmpsb      ; se compară cele doua nume
        jnz inst                ; nume diferite -> se instalează
        lea dx, mes_inst        ; afişare mesaj program deja instalat
        mov ah, 9               ; şi solicitare utilizator pentru dezinstalare
        int 21h
        mov ah, 1
        int 21h         ; cit. răsp. utilizator, dacă doreşte dezinst
        push ax                 ; salvare răspuns
        mov ah, 9               ; avans pe linie nouă pe ecran
        lea dx, CRLF
        int 21h
        pop ax          ; refacere răspuns utilizator
        and al, 0DFH ; se transformă caracterul în literă mare
        cmp al, 'D'             ; se compară cu caracterul 'D'
        je dezinst              ; daca este 'D' se va dezinstala programul
        pop es          ; refacere ES, DS
        pop ds
        mov ax, 4c00h           ; dacă nu, revine în DOS, fără instalarea
        int 21h         ; codului, care este deja instalat (rezident)
dezinst:
        xor ax, ax              ; se refac vectorii inițiali salvați în codul
        mov ds, ax              ; rezident; se pun în TVI (adr. seg. 0)
        mov es, ds:[9*4+2] ; ES = adr. seg. prg. TSR, încărcat ant.,
        mov ax, word ptr es:[OldInt9h]      ; diferă de cel actual (TVI)
        mov ds:[9*4], ax        ; reface vectorul 9: offsetul
        mov ax, word ptr es:[OldInt9h+2]
```

```
        mov ds:[9*4+2], ax    ; şi adresa de segment
        mov ax, word ptr es:[OldInt16h] ; reface vectorul 16h:
        mov ds:[16h*4], ax              ; offsetul
        mov ax, word ptr es:[OldInt16h+2]
        mov ds:[16h*4+2], ax          ; şi adresa de segment
        pop ds          ; refacere ES, DS
        pop es
        push cs                 ; dezinstalarea se face complet, cu
        pop es          ; eliberarea spațiului de memorie
        mov ah, 49h   ; alocat programului TSR
        int 21h
        push cs
        pop ds
        lea dx, mes_dezi        ; afişare mesaj 'Program Dezinstalat'
        mov ah, 9
        int 21h
        mov ax, 4C00h
        int 21h
; se memorează vechile valori pentru într. INT 9 şi INT 16
; direct în variabilele OldInt9h, şi respectiv OldInt16h
inst:
        cli                     ; dezactivare întreruperi
                                ; secțiune critică: modificare vect. într.
        mov ax, es:[9*4]        ; citire offset rutină pentru Int 9h şi
        mov word ptr OldInt9h, ax ; salvare la OldInt9h
        mov ax, es:[9*4 + 2]  ; cit. Seg. rutină pentru Int9h şi
        mov word ptr OldInt9h[2], ax        ; salvare la OldInt9h+2
        mov word ptr es:[9*4], offset MyInt9h      ; înlocuire vector
        mov word ptr es:[9*4+2], cs ; vechi cu cel nou
        mov ax, es:[16h*4]    ; aceleaşi operații pentru Int 16h
        mov word ptr OldInt16h, ax
        mov ax, es:[16h*4 + 2]
        mov word ptr OldInt16h[2], ax
        mov word ptr es:[16h*4], offset MyInt16h
        mov word ptr es:[16h*4+2], cs
        sti                     ; reactivare întreruperi
; se termină partea de instalare cu lăsarea rezidentă a codului
        mov ah, 9               ; tipărire mesaj de instalare cod TSR
        lea dx, mesaj
        int 21h
        pop ds          ; refacere DS, ES
        pop es
        mov dx, offset test_instalare + 15 ; calcul spațiul de memorie
        push cx                 ; salvare CX
        mov cl, 4               ; ocupat de codul rezident, ca nr paragrafe
        shr dx, cl
```

```
        pop cx          ; refacere CX
        mov ax, 3100h       ; apelează funcţia ce lasă rezident codul
        int 21h         ; din seg. curent, până la 'test_instalare'
mesaj  db      'S-a instalat programul ce numara ''Numarul de                  taste''
apasate.'
CRLF db      0Dh, 0Ah, '$'
mes_inst db   'Programul este deja instalat in memorie!!!',
                    0Dh, 0Ah
        db      'Doriti dezinstalarea lui (D/d-DA, orice alta tasta pentru NU): $'
mes_dezi db   'Programul a fost dezinstalat !!!', 0Dh, 0Ah, '$'
end start
```

Pe lângă problemele legate de apeluri reentrante ale TSR-ului către DOS/ BIOS, se poate ca apelurile DOS ale TSR-ului să afecteze structuri de date utilizate de alte aplicaţii curente: stiva aplicaţiei, zona DTA, etc. Când se execută un TSR el operează în contextul (mediul de lucru) al aplicaţiei principale. De ex., adresa returnată de TSR şi orice valori, pe care TSR-ul le salvează pe stivă, sunt puse pe stiva aplicaţiei. Dacă TSR foloseşte intens stiva, datorită unor apeluri recursive, sau alocării de spaţiu pe stivă pentru variabile locale, atunci TSR-ul trebuie să salveze stiva aplicaţiei (adică SS şi SP) şi să comute pe o stivă locală (proprie).

Dacă TSR-ul execută apelul funcţiei DOS ce returnează adresa PSP, se va returna adresa PSP-ului aplicaţiei principale, şi nu cel al TSR-ului. Dacă nu se comută de la PSP-ul aplicaţiei principale la cel al TSR-ului, şi apare o excepţie (manipulare disc, sau *ctrl-break*), atunci se va executa manipulatorul asociat cu aplicaţia principală (în locul celui asociat TSR-ului). De aceea, când se execută un apel DOS, care poate rezulta într-una din acele condiţii, trebuie comutat PSP-ul. Astfel când TSR-ul returnează controlul la aplicaţia principală, trebuie refăcut PSP-ul. Funcţia *50h* iniţializează adresa PSP a programului curent la valoarea din registrul BX, iar funcţia *51h* determină adresa PSP a programului curent şi o returnează în registrul BX.

Componentele unui program TSR

Programele TSR utilizează, după cum am arătat, în general, întreruperile de la tastatură sau de la ceasul de timp real; ele nu trebuie să interfereze cu întreruperile hardware care aparţin de domeniul driverelor de dispozitiv, excepţie făcând întreruperile de la tastatură şi de la circuitul de ceas.

Circuitul de ceas generează întreruperi hardware la fiecare 1/18,2 secunde (acest interval este dat de faptul că, pe durata unei ore, un registru contor ajunge la valoarea maximă 65535); aceste întreruperi periodice permit actualizarea orei sistemului. Pentru ceasul de timp real există doi vectori de întrerupere, şi anume *08h,* respectiv *1Ch*. Întreruperea de pe nivelul *08h, INT 08h,* actualizează ora şi data sistemului, după care execută instrucţiunea *INT 1Ch*; această rutină constă dintr-o singură instrucţiune IRET, tocmai pentru a permite utilizatorului să-şi insereze aici rutina de tratare a întreruperii, şi nu pe nivelul *08h*, tocmai pentru a nu afecta actualizările de sistem.

Întreruperile de la tastatură, generate atât la apăsarea unei taste, cât şi la eliberarea ei, au fost descrise anterior. Întreruperea de la tastatură se află pe nivelul *09h, INT 09h*; această rutină de tratare a întreruperii citeşte codul generat de tastatură şi în funcţie de starea tastelor de tip „shift“ (CTRL, SHIFT, ALT)- memorată într-un octet de stare la adresa 40h:17h- generează

codul tastei apăsate respective. De asemenea, această procedură verifică şi anumite combinaţii de taste cu semnificaţie deosebită, cum ar fi: CTRL-ALT-DEL (reiniţializare), CTRL-BREAK (suspendare), pentru care se execută instrucţiunile *INT 19h*, respectiv *INT 1Bh*. Caracterele de la tastatură sunt depuse într-un buffer circular, de 16 cuvinte; bufferul tastaturii se află la adresa 40h:1Ah şi conţine un pointer (deplasament) pentru extragere, un pointer (tot de tip cuvânt) pentru introducere şi zona de 16 cuvinte unde se depun codurile tastelor.

Aplicaţiile utilizator vor folosi instrucţiunea *INT 16h* pentru accesul la bufferul tastaturii, care permite citirea caracterelor din buffer şi verificarea stării tastaturii.

Componenta mai simplă a unui program rezident este tocmai cea de iniţializare, care îl lasă rezident. În locul terminării normale, ca până acum, a programului cu funcţia *4Ch, INT 21h*, programul se va termina cu funcţia *31h*, cea care păstrează programul în memorie. Când se va executa rutina de iniţializare, sistemul va rezerva blocul de memorie unde este rezident programul şi va încărca programele următoare în memoria rămasă disponibilă după programul rezident.

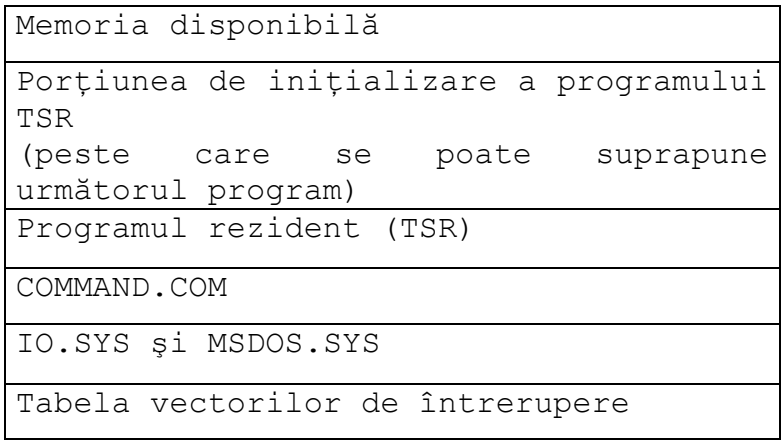
Componenta mai delicată din elaborarea unui program rezident este cea care realizează activarea programului după ce acesta a devenit rezident, deoarece nu este un program intern, de sistem, cum sunt de exemplu CLS, COPY, DIR etc. Metoda cea mai utilizată pentru activare este cea de modificare a tabelei vectorilor de întrerupere, astfel încât programul rezident întrerupe toate tastările, acţionând la o anumită tastă sau combinaţie de taste. Partea rezidentă conţine:

- una sau mai multe proceduri, dintre care una este noua rutină de tratare a întreruperii (RTI), care apelează şi vechea întrerupere; la apariţia unei întreruperi corespunzătoare acestei întreruperi, se va activa RTI din partea rezidentă.
- o zonă de date, care conţine şi spaţiul alocat în care se salvează vectorii de întrerupere ce sunt modificaţi de programul rezident.

Un program rezident obişnuit constă din următoarele componente:

1. O componentă care redefineşte locaţiile din tabela vectorilor de întrerupere, adică instalarea unui nou vector de întrerupere.
2. O componentă de iniţializare, care se execută doar la prima rulare a programului şi care realizează:
   - înlocuirea adresei din vectorul de întrerupere cu propria sa adresă (a codului rezident);
   - determină dimensiunea părţii rezidente a programului (la nivel de paragrafe, 16 octeţi);
   - utilizează o întrerupere (31h) care specifică sistemului să termine execuţia programului curent şi să lase rezidentă în memorie porţiunea specificată de program.
3. Componenta (procedura) care rămâne rezidentă şi care va fi activată fie printr-o combinaţie specifică de taste, fie de ceasul de timp real al sistemului.

Ca efect, rutina de iniţializare stabileşte toate condiţiile pentru a lăsa rezident programul şi apoi permite ca ea însăşi să fie ştearsă. Organizarea memoriei, acum, apare astfel:



Un program rezident poate utiliza două funcții DOS, *INT 21h, 35h* şi respectiv *25h*, pentru accesul la tabela vectorilor de întreruperi, deoarece nu există nici o garanție că adresa acestei tabele va începe întotdeauna de la adresa *0000H*. Aceste funcții primesc ca parametru nivelul întreruperii pe care se doreşte instalarea activării codului rezident sau salvarea vechiului vector, deci nu trebuie cunoscută adresa fizică a acestuia.

Instalarea unui nou vector de întrerupere

Partea de instalare a unui program TSR trebuie să realizeze următoarele operații:

- să verifice, în primul rând, dacă nu cumva a fost deja instalat în memorie codul său rezident; dacă programul a fost deja instalat, nu îl va mai instala şi programul se va termina cu funcția *4C00h, INT 21h*.
- să captureze un vector de întrerupere pe care îl înlocuieşte cu unul nou. Înainte de a modifica vectorul de întrerupere, programul rezident (cel ce se instalează), salvează în partea sa de inițializare vectorul curent şi apoi îl înscrie pe cel nou, care reprezintă adresa, de tip FAR a procedurii, pentru tratarea întreruperii definite în partea rezidentă;
- să calculeze dimensiunea memoriei necesare pentru codul rezident şi să apeleze funcția DOS *31h*, care lasă codul părții rezidente în memorie şi termină programul.

Prin capturarea unui vector de întrerupere se înțelege înlocuirea vectorului existent, cu unul nou, care conține adresa noii rutine de tratare a întreruperii, definită în programul TSR. De obicei, noua rutină apelează vechea rutină de tratare a întreruperii.

Înlocuirea unui vector de întrerupere, care constă în salvarea celui vechi şi înscrierea unuia nou, se poate implementa direct de utilizator, utilizând instrucțiuni de transfer (mov), dar este mult mai comodă utilizarea funcțiilor DOS (*INT 21h*) *35h* şi *25h*.

Funcția DOS *35h* returnează în ES:BX vechiul vector de întrerupere, cel care se doreşte a fi înlocuit, al cărui număr a fost transmis în registrul AL, astfel:

```
mov ah, 35h         ; funcția DOS de salvare a vechiului vector
mov al, nr_int      ; nr. Într. pe nivelul căreia se pune TSR-ul
int 21h
mov Vechea_Int, bx          ; salv. vechiului vector de întrerupere
mov Vechea_Int[2], es ; din ES:BX, în zona TSR-rezident
```

Funcția *25h* scrie noul vector de întrerupere, transmis în registrele DS:DX, în tabela vectorilor de întrerupere, în locul vectorului al cărui număr (index din tabelă) este specificat în registrul AL:

```
mov ah, 25h         ; funcția DOS de instalare a noului vector
mov al, nr_int      ; nr. Într. pe nivelul căreia se pune TSR-ul
lea dx, Noua_Int; offsetul RTI, adr. seg. este în DS, întrucât
                    ; toate reg. seg., la programele .COM
                    ; sunt inițializate cu segmentul PSP
int 21h             ; deci nu mai trebuie reinițializat cu
                    ;'seg Noua_Int'
```

Ca efect, la apariția unei întreruperi pe nivelul specificat în AL, se va executa noua procedură rezidentă (TSR), în locul celei vechi.

Salvarea vechiului vector şi instalarea celui nou se poate face şi fără funcțiile DOS, direct în tabela vectorilor de întrerupere (TVI); secvența aceasta trebuie însă executată cu întreruperile dezactivate.

Pentru a evita reinstalarea aceluiaşi program TSR, se verifică dacă există deja o copie în memorie a codului rezident, deci dacă acesta a fost deja instalat, prin compararea offset-ului noii rutine cu offset-ul vectorului de întrerupere utilizat. În cazul în care cele două offset-uri sunt diferite, înseamnă că programul nu este rezident şi, deci, poate fi instalat. Dacă, însă, cele două offseturi sunt identice, întrucât se poate întâmpla ca programe diferite să utilizeze acelaşi vector de întrerupere şi să aibă acelaşi offset, se vor compara identificatorii celor două programe TSR. Două TSR-uri diferite pot avea acelaşi offset al adresei de lansare dacă zona de date ce precede începutul codului TSR are aceeaşi dimensiune. Din acest motiv trebuie înscris în codul rezident un şir de caractere care să identifice în mod unic acel program; acest identificator de program TSR se plasează înaintea programului TSR propriu-zis.

Pentru a lăsa rezidentă în memorie numai partea rezidentă a programului TSR se va utiliza funcția *31h*, care are ca parametru de intrare dimensiunea de memorie ocupată de partea rezidentă, exprimată în paragrafe (1 paragraf = 16 octeți), transmisă în registrul DX (adresa de segment este cea conținută în DS, care are acelaşi conținut cu toate celelalte registre segment în cazul programelor .COM). Funcția va lăsa rezident în memorie codul de lungime specificată şi realizează ieşirea din program, în DOS. Memoria ocupată de partea de inițializare a programului va fi eliberată de DOS.

Deci un program TSR are următoarea structură:

```
cod_TSR segment
        ; partea rezidentă
        assume cs: cod_TSR
        org 100H
intrare:
        jmp start_init      ; exe. la prima intrare
nume_TSR        db      'prog_TSR_1'    ; ID cod rezident
lung_nume       equ     $ - nume_TSR    ; lungime ID
vect_vechi      dd      ?       ; spațiu pt salvarea vechiului vector
nr_vect equ     . . .   ; vectorul pe care se instalează TSR
        ; definirea noii proceduri de tratare a întreruperii
vect_nou proc   far     ; noua proc. poate apela şi vechea p.
        push resurse            ; salv. resurse utilizate de TSR
```

```
            . . . . . .                         ; noile prelucrări, descrise de noua
            . . . . . .                         ; procedură de tratare a întreruperii
            pop resurse                 ; refacerea resurselor utiliz. de TSR
            jmp cs: vect_vechi                  ; vechea într., fie cu salt fie ca apel:
;           pushf               ; apel de tip întrerupere pentru vechea
;           call cs:vect_vechi  ; procedură de tratare întrerupere
;           iret                        ; revenire în programul întrerupt
vect_nou    endp
            ; partea de inițializare a codului rezident
start_init:
            mov al, nr_vect     ; numărul vechiului vector de întrerupere,
            mov ah, 35h         ; cel înlocuit. Salvare vector vechi.
            int 21h             ; (ES:BX) = vect vechi. Comp. offseturile
            cmp bx, offset vect_nou     ; celor doi vectori: vechi ÷ nou
            jnz instalare       ; Este deja rezident ? Dacă nu, se instalează
            mov si, offset cs: nume_TSR         ; Se compară identificatorii
            mov di, si                  ; offseturile celor două nume_TSR în SI/DI
            mov ax, cs                  ; adresa de segment pentru numele noului
            mov ds, ax                  ; vector
            mov cx, lung_nume           ; lungimea numelui vectorului nou
            cld                         ; direcția de parcurgere, crescătoare
            repe cmpsb          ; se compară cele două nume
            jnz instalare       ; nume diferite -> se instalează
            lea dx, mes_deja_inst       ; afişare: 'Program deja instalat'
            mov ah, 9
            int 21h
; aici se poate insera secvența de dezinstalare a TSR-ului
; se întreabă utilizatorul dacă doreşte dezinstalare:
            lea dx, mes_dezinst
            mov ah, 9
            int 21h
            mov ah, 1                   ; citire răspuns utilizator
            int 21h
            cmp al, 'D'                 ; dacă nu este DA se termină fără dezinst.
            jnz iesire
            xor ax, ax                  ; se reface vectorul din TVI, 'vect_vechi'
            mov ds, ax                  ; în codul rezident; se pun în TVI (seg. 0)
            cli
            mov es, ds:[nr_int*4+2]     ; ES=adr. seg.  TSR, deja încărcat,
            mov ax, word ptr es:vect_vechi      ; care diferă de cea actuală,
            mov ds:[nr_int*4], ax       ; (se reface în TVI) reface vectorul
            mov ax, word ptr es:vect_vechi[2] ; 'nr_int': offsetul
            mov ds:[nr_int*4+2], ax     ; şi adresa de segment
            sti
            push cs             ; dezinstalarea se face complet, cu
            pop es              ; eliberarea spațiului de memorie
```

```
        mov ah, 49h       ; alocat programului TSR
        int 21h
        push cs
        pop ds
        lea dx, mes_dezi  ; afişare mesaj 'Program Dezinstalat'
        mov ah, 9
        int 21h
iesire:
        mov ax, 4c00h     ; se revine în DOS, fără instalarea
        int 21h           ; codului, care este deja rezident
instalare:
        cli                         ; dezactivare sistem întreruperi (INTR)
        mov word ptr vect_vechi, bx       ; se salvează vechiul
        mov word ptr vect_vechi[2], es    ; vector
        mov ah, 25h       ; funcția DOS de instalare a noului
        mov al, nr_vect   ; vector
        lea dx, vect_nou  ; DS a fost inițializat anterior
        int 21h
; secvența care lasă codul vect_nou rezident
; 'start_init' - eticheta de început a părții de inițializare,
; reprezintă şi eticheta de sfârşit a codului rezident
        lea dx, start_init  ; adr. de sfârşit a părții rezidente, care este
        add dx, 15                ; rotunjită la paragraful următor
        mov cl, 4                 ; pentru a determina
        shr dx, cl                ; dimensiunea în paragrafe
        mov ah, 31h       ; funcția DOS, care îl lasă rezident
        sti                       ; reactivare sistem de întreruperi
        int       21h
mes_deja_inst db  0Ah, 0Dh, 'TSR-ul este deja instalat !!!'
                  db      0Ah, 0Dh, '$'
mes_dezinst       db      'Doriti dezinstalare (D pentru DA, orice '
        db        'alta tasta pentru NU): $'
mes_dezi db       0Dh, 0Ah, 'Programul a fost dezinstalat !!!'
             db   0Dh, 0Ah, '$'
cod_TSR ends
end       intrare
```

- '*nr_vect*' este numărul vectorului de întrerupere ce se înlocuieşte, de exemplu *1Ch* - vectorul ceasului de timp real, sau *09h* pentru tastatură.
- Partea de inițializare nu va rămâne în memorie, ci numai codul rezident (*vect_nou*), primele (DX) paragrafe, până la eticheta '*start_init*'.
- Spațiul de memorie ocupat de partea de inițializare a programului TSR, va fi eliberat de DOS.

Partea rezidentă a programului TSR

Partea rezidentă defineşte o zonă de date şi prelucrările respective, reprezentate de una sau mai multe proceduri.

Procedurile dintr-un program rezident devin active ca răspuns la un apel direct al unui alt program sau la apariția unei întreruperi hardware. După lansare un lucru nu poate fi aşteptat pentru codul apelant: să inițializeze corespunzător registrele segment. Singurul registru segment cu semnificație pentru codul rezident este registrul *CS*. Pentru acces la variabilele locale -> rutinele vor inițializa *DS* sau alte registre segment. De ex. o rutină care numără de câte ori alte programe o apelează, după ce a devenit rezidentă. Ea ar conține o singură instrucțiune *inc contor,* dar s-ar incrementa variabila de la offsetul *contor* în segmentul referit de *DS*, şi deci nu va incrementa variabila *contor*.

Două soluții la această problemă: 1. toate variabilele în segmentul de cod, şi să se utilizeze prefixul segment (*inc cs:contor*); utilizată doar când avem puține variabile. 2. salvare registrul segment de date, DS, la intrarea în TSR, inițializarea lui cu noua valoare, iar înainte de ieşire din TSR să fie refăcut la valoarea avută.

Activarea unui program TSR presupune întreruperea celui curent executat. În cazul în care codul rezident foloseşte funcții de sistem, el trebuie să testeze, la activarea sa, dacă programul întrerupt executa o funcție *DOS/BIOS*. Codul rezident nu poate reactiva o funcție de sistem, în timp ce se execută o altă funcție *DOS/BIOS,* deoarece DOS-ul nu este un sistem de operare reentrant (accesul la disc: *13h, 25h, 26h,* nu poate fi întrerupt).

Această problemă este doar pentru TSR-uri active, întrucât cele pasive sunt apelate şi se vor executa în cadrul programului apelant, care nu poate executa simultan două apeluri.

Pentru a evita astfel de operații sistemul de operare utilizează două variabile: '*eroare critică*' şi '*secțiune'* sau *'stare critică*', pe care le activează corespunzător stării în care se află. Cele două variabile pot fi testate de codul rezident, în momentul în care se utilizează funcții DOS. Adresele celor două variabile, şi deci starea lor, sunt returnate de funcțiile *34h* (numită indicator *secțiune critică* sau indicator *inDOS*) şi respectiv, *50D6h* (*eroare critică*), în registrele ES:BX. Cele două adrese pot fi memorate în zona de date a părții rezidente, de către partea de inițializare a programului TSR, pentru a le testa direct.

Întreruperea multiplexată, INT 2FH

Când se instalează un program TSR care are şi componente pasive, trebuie ales un vector de întrerupere pe care să fie instalat, astfel ca alte programe să poată comunica cu rutinele sale, pasive. Se poate alege un vector aproape la întâmplare dar aceasta poate conduce la anumite probleme de compatibilitate. Ce se întâmplă dacă altcineva utilizează deja acest vector de întrerupere ?

În multe situații alegerea este clară, cum ar fi în situația în care codul pasiv al TSR-ului extinde anumite servicii (tastatură, video). În celelalte situații, cum ar fi crearea unui nou driver pentru un nou dispozitiv, care să nu interfere funcțiile furnizate pentru acest dispozitiv, cu alte întreruperi, alegerea la întâmplare a unui vector de întrerupere este riscantă.

Pentru a rezolva aceste situații DOS-ul furnizează soluția utilizării unei întreruperi multiplexate, *INT 2FH*, care permite instalarea, testarea prezenței şi a comunicării cu un program TSR.

Pentru a utiliza această întrerupere, o aplicație plasează o valoare de identificare în *AH* şi un număr de funcție în *AL*, şi apoi execută *INT 2FH*. Fiecare TSR din lanțul *INT 2FH* va compara valoarea din *AH* cu propria sa valoare unică de identificare. Dacă valoarea sa coincide cu cea din *AH,* atunci TSR-ul respectiv va prelucra comanda specificată de valoarea din registrul *AL*. Dacă

nu se identifică, prin valoarea din *AH,* atunci TSR-ul respectiv transmite controlul la următorul manipulator din lanțul *INT 2FH*. Aceasta simplifică cumva problema, dar nu o elimină. De obicei, trebuie ales numărul întreruperii înainte de proiectarea TSR-ului. Cum, însă, o aplicație va ști ce valoare să încarce în *AH*, dacă se asignează dinamic această valoare, când TSR-ul este rezident ?

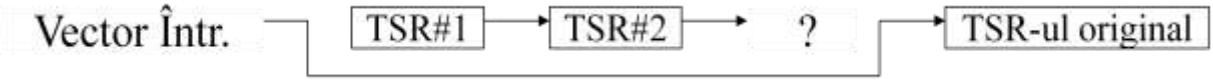
Se permite oricărei aplicații interesate să determine identificatorul (ID) TSR-ului. Prin convenție, funcția zero (*AL=0*) este un apel ce determină prezența TSR-ului. Acest apel se va realiza înainte de solicitarea oricărui serviciu TSR, pentru a determina dacă programul TSR este prezent în memorie.

Dezinstalarea unui program TSR

La dezinstalarea unui TSR trebuie făcute trei lucruri:

- *oprirea/ suspendarea* oricăror activități nerezolvate sau în așteptare (adică, un TSR poate avea anumiți indicatori inițializați pentru a lansa anumite activități ulterioare);
- *refacerea* tuturor vectorilor de întrerupere la valorile lor inițiale;
- *eliberarea* spațiului de memorie ocupat, adică cedarea spațiului de memorie rezervat înapoi către DOS, astfel ca alte aplicații să-l poată utiliza;

Dacă dezinstalarea TSR-ului constă doar în refacerea valorilor vectorilor de întrerupere, se va crea o problemă, dacă există alte TSR-uri care utilizează aceeaşi vectori de întrerupere. Să presupunem că avem următorul lanț de TSR-uri, pe un anumit nivel de întrerupere:



Aceasta va dezactiva efectiv TSR-urile înlănțuite din program, iar aceste TSR-uri vor avea efecte neașteptate, în aceste condiții. O soluție ar fi tipărirea unui mesaj de eroare, care să informeze utilizatorul că nu se poate dezinstala acest TSR, până nu se dezinstalează toate TSR-urile instalate anterior acestuia.

Problema va fi în a determina TSR-urile înlănțuite în întrerupere. Pentru aceasta ştim că vectorii de întrerupere ar trebui încă să refere rutina noastră, dacă acesta este ultimul TSR care se execută. Atunci, trebuie să comparăm vectorii de întrerupere instalați cu adresele rutinelor de servire a întreruperii respective. Dacă toate acestea se potrivesc, atunci se poate elimina în siguranță TSR-ul din memorie. Dacă, însă, doar unul dintre ei nu se potrivește, atunci nu putem elimina TSR-ul din memorie.