Capitolul 8
Interfaţa cu sistemul de operare DOS

În acest capitol vor fi prezentate elementele de bază ale interacţiunii programelor scrise în ASM cu cele două componente software de bază dintr-un calculator personal, subsistemul BIOS şi sistemul de operare DOS, fără a se prezenta o listă exhaustivă a funcţiilor de sistem sau BIOS. Asemenea liste de funcţii sunt disponibile în manuale specifice ale sistemului de operare DOS.

8.1 Componentele de bază ale sistemului DOS

Componentele sistemului de operare DOS sunt:

- **DOS-BIOS.** Această componentă este memorată pe disc într-un fişier cu numele IBMBIO.COM sau IBMIO.SYS sau IO.SYS, depinzând de furnizor. Ea conţine driverele pentru dispozitivele CON (tastatură şi display), PRN (imprimantă), AUX (interfaţă serială) şi CLOCK (ceas de timp real). Deasemenea, modulul DOS-BIOS conţine drivere pentru discurile flexibile şi pentru discul hard. Accesul la aceste drivere va implica apelul unor rutine din ROM-BIOS (portiunea de BIOS aflată în ROM, care conţine procedurile dependente de hardware).
- **DOS-Kernel (nucleul DOS)**. Această componentă este memorata pe disc în fişierul IBMDOS.COM sau MSDOS.SYS, depinzând de furnizor şi conţine funcţii de acces la fişiere, funcţii de intrare/ieşire la nivel de caracter etc. Aceste funcţii operează într-un mod independent independent de hardware (e vorba de funcţiile DOS apelate prin instrucţiuni INT 21H).
- **Interpretorul de comenzi (Shell) standard**. Este memorat pe disc în fişierul COMMAND.COM. Interpretorul este cel care afişează prompterul DOS la consolă, acceptând şi executând comenzi de la tastatură. Interpretorul de comenzi este la rândul său compus din 3 părţi:
  - **porţiunea rezidentă**, care conţine handlere (rutine de tratare) pentru acţiunile CTRL-Break (CTRL-C) de la tastatură si pentru erori critice (erori la citirea sau scrierea la dispozitive periferice, cum ar fi discurile sau imprimanta). Erorile critice conduc la afişarea la consolă a unor mesaje de forma:
    Error on device ... Abort, Retry, Fail ?
  - **porţiunea tranzitorie**, care afişează prompter-ul la consolă (de exemplu C:\>), citeşte comanda de la consolă şi o execută. Porţiunea tranzitorie conţine comenzile DOS interne (de genul DIR, TYPE, COPY etc.). Când un program executabil îşi încheie execuţia, controlul revine porţiunii rezidente a interpretorului de comenzi, care verifică o sumă de control, detectând astfel dacă s-a modificat cumva porţiunea tranzitorie din memorie. În caz afirmativ, porţiunea tranzitorie se reîncarcă de pe disc;
  - **rutina de iniţializare**, care este încărcată în memorie o dată cu încărcarea sistemului DOS de pe disc. Când operaţia de încărcare s-a terminat, această componenntă nu mai este necesară şi spaţiul respectiv de memorie va fi folosit în alte scopuri.

Comenzile acceptate de interpretorul de comenzi standard sunt de 3 tipuri:

- comenzi interne (cum ar fi COPY, DIR, RENAME etc.), care nu au corespondent în fişiere executabile pe disc;
- comenzi externe, conţinute în fişiere disc de tip .EXE sau .COM, care sunt încărcate în memorie în zona TPA (zona de procese tranzitorii);
- fişiere de comenzi indirecte (.BAT), care sunt fişiere text ce conţin comenzi interne, externe sau alte invocări de fişiere de comenzi indirecte.

Interpretorul testează întâi dacă linia introdusă de la tastatură este o comanda internă. Dacă nu este, se caută un fişier cu extensia .COM, .EXE

sau .BAT (în această ordine), întâi în directorul curent, apoi în toate directoarele specificate în comanda PATH. Dacă nu se găseşte nici un fişier adecvat, se afişează la consolă mesajul:

```
Bad command or file name
```

Dacă s-a identificat un fişier cu extensia .COM sau .EXE, interpretorul încarcă fişierul executabil în memorie utilizând funcţia EXEC (4BH) a sistemului de operare şi îi dă controlul. Funcţia EXEC construieşte în memorie un bloc special numit PSP (Program Segment Prefix), în zona de procese tranzitorii TPA, la prima adresă disponibiă. Se încarcă apoi programul executabil, imediat după PSP şi se execută operaţiile de relocare, adică se ajustează valorilor simbolilor care conţin referinţe la memorie, conform cu poziţia fizică din memorie unde a fost încărcat programul.

- **ROM-BIOS**. Această componentă este rezidentă în memoria ROM a calculatorului şi conţine partea de autotestare de la punerea sub tensiune (numită şi POST, de la **Power-On Self Test**), programul primar de încărcare de pe disc (**bootstrap**), rutinele de intrare/ieşire de nivel scăzut (dependente de hardware) etc. În general, această componentă nu rezidă numai pe placa de bază a calculatorului, anumite porţiuni fiind localizate pe plachetele de interfeţe furnizate de producător.

8.2 Încărcarea sistemului DOS

La punerea sub tensiune, procesorul începe să execute instrucţiuni de la adresa fixă 0FFFF0H (care este situată în memoria de tip ROM). La această adresă se găseşte o instrucţiune JMP FAR la o porţiune de program din ROM-BIOS, numită **POST** (**Power-On Self Test**). Acest cod execută teste hardware pentru unitatea centrală, memoria internă, dispozitivele periferice etc., înscriind totodată vectorii de întrerupere pentru întreruperile hardware externe.

Se trece apoi la execuţia unei zone de cod numită **ROM-Bootstrap** (încărcător primar), care testează daca există dischete în unităţile de disc flexibil. Dacă nu, se trece la examinarea discului hard. Se citeşte apoi primul sector de pe pista 0 a discului (fie de pe discul flexibil, fie de pe discul hard). Acest sector este numit sector de boot (**Boot Secto**r) şi conţine un alt program încărcător (numit **Disk-Bootstrap**) precum şi informaţii despre organizarea discului. Programul **Disk-Bootstrap** este citit în memorie la nivel fizic şi i se dă controlul. Unele calculatoare de generaţie mai nouă permit specificarea ordinii de examinare a unităţilor de disc.

Trebuie remarcat că, în acest moment, sistemul de operare nu este încărcat în memorie, deci nu se dispune de nici o funcţie DOS. Harta memoriei la acest moment este ilustrată în Figura 8.1.

Programul încărcător citeşte primul sector al directorului rădăcină, determinând dacă primele doua fişiere din acest director sunt, în ordine: **IO.SYS** (sau **IBMBIO.COM** sau **IBMIO.SYS**) şi **MSDOS.SYS** (sau I**BMDOS.COM**), care conţin componentele **DOS-BIOS** şi **DOS-Kernel** ale sistemului de operare. Dacă aceste fişiere nu sunt identificate, se afişează la consolă mesajul:

```
Non-System disk or disk error
Replace and strike any key when ready
```

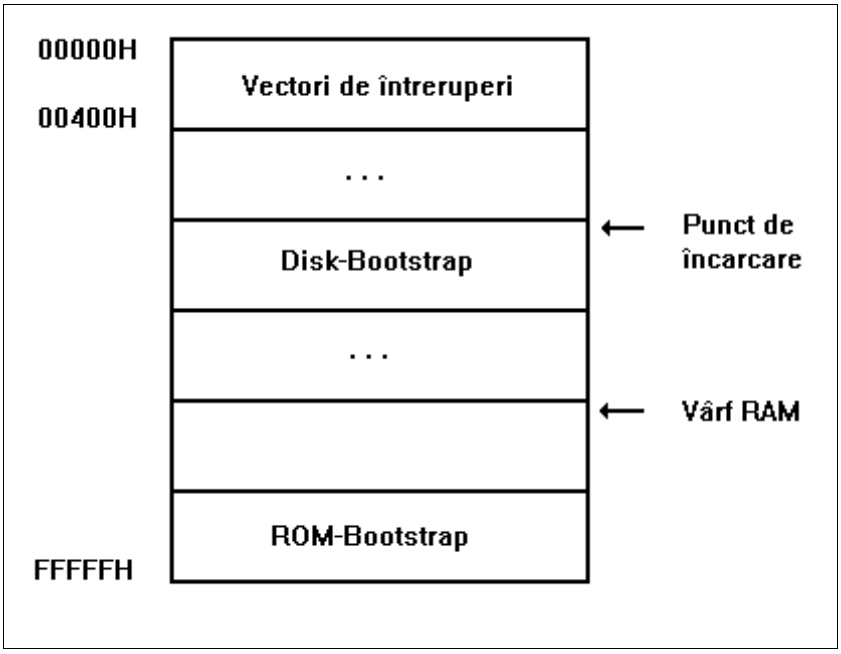

Figura 8.1 Harta memoriei imediat după încărcarea programului Disk-Bootstrap

Dacă fişierele sunt identificate corect, programul Disk-Bootstrap le citeşte în memorie şi transferă controlul la punctul de start al modulului IO.SYS.

Componenta DOS-BIOS constă din doua module: BIOS-ul propriu-zis şi modulul SYSINIT. Harta memoriei din acest moment este ilustrată în Figura 8.2.

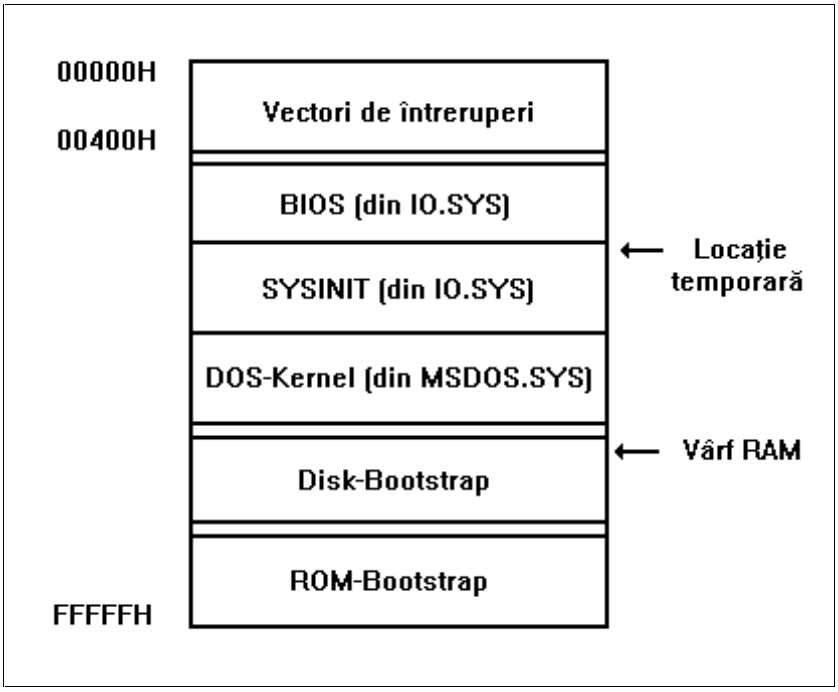

Figura 8.2 Harta memoriei după încărcarea modulelor IO.SYS şi MSDOS.SYS

Încărcarea celor două fişiere se face la adrese mici de memorie, deoarece nu se cunoaşte încă dimensiunea memorie. Modulul SYSINIT, apelat de către partea de iniţializare a BIOS-ulu, determină dimensiunea memoriei continue şi se auto-realocă (adica se mută) la o adresă mare de memorie (către coada memoriei). În continuare, modulul SYSINIT mută modulul MSDOS.SYS din locul unde a fost încărcat iniţial în poziţia sa finală din memorie, suprascriind

vechea copie a modulului SYSINIT (încarcat iniţial la adrese mici) precum şi alte porţiuni iniţiale de cod conţinute în IO.SYS şi care nu mai sunt necesare.

Se dă apoi controlul porţiunii de iniţializare a modului DOS-Kernel. Acesta iniţializează tabelele interne şi zonele de lucru ale sistemului DOS, setează vectorii de întrerupere de pe nivelele 20H...2FH (adică întreruperile soft specifice funcţiilor DOS) şi iniţializează driverele rezidente ale dispozitivelor periiferice periferice (discuri, consolă etc.). Controlul revine apoi modulului SYSINIT.

În acest moment, sistemul DOS este operaţional, deci modulul SYSINIT poate apela funcţii DOS. SYSINIT încearcă să deschidă fişierul CONFIG.SYS, care conţine diverse comenzi specifice prin care se poate configura sistemul. În acest fişier se pot specifica drivere suplimentare care să fire încărcate (aşa-numitele drivere instalabile), se poate specifica numărul de buffere pentru disc, numărul maxim de fişiere care pot fi deschise simultan, numele fişierului care conţine interpretorul de comenzi etc. Dacă fişierul CONFIG.SYS este identificat în directorul rădăcină, el este încărcat în memorie şi se execută fiecare comandă specificată. Harta memoriei din acest moment este ilustrată în FIgura 8.3.

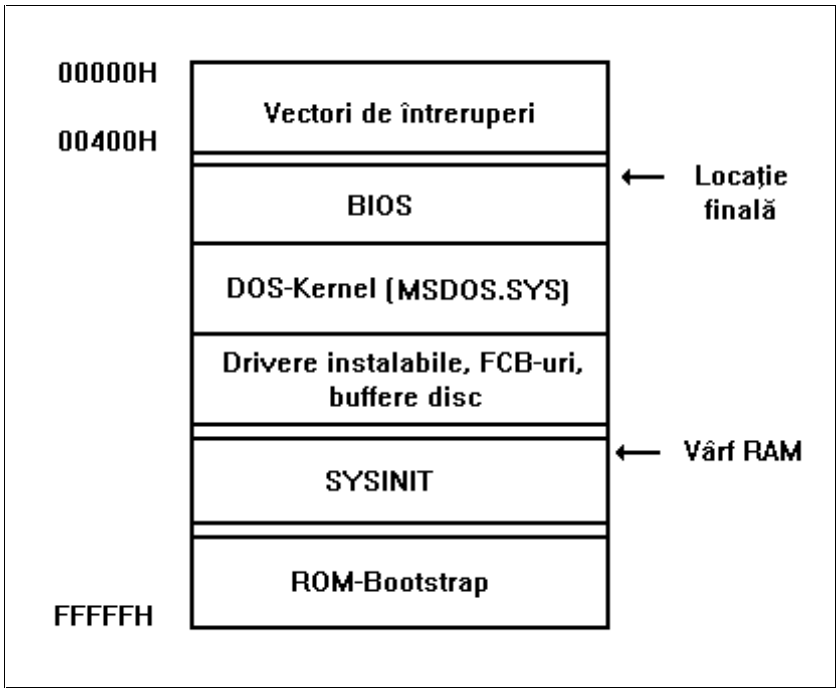

Figura 8.3 Harta memoriei înainte de încărcarea interpretorului de comenzi

După ce toate driverele instalabile au fost încarcate, SYSINIT închide toate fişierele (dispozitivele) folosite şi apoi redeschide dispozitivele consolă (CON), imprimantă (PRN) şi interfaţă serială (AUX), asociindu-le handler-ele de fişiere 0 (standard input), 1 (standard output), 2 (standard error), 3 (standard list) si 4 (standard auxiliary). Handler-ele 0, 1, 2 sunt asignate la dispozitivul CON, handler-ul 3 este asignat la dispozitivul PRN iar handler-ul 4 la dispozitivul AUX. Aceste handler-e pot fi reasignate prin apeluri de funcţii sistem adecvate.

În final, modulul SYSINIT apelează funcţia EXEC a sistemului de operare, pentru a încărca şi executa interpretorul standard de comenzi. Odată ce interpretorul a fost încărcat, el afişează la consolă prompterul specific şi aşteaptă introducerea de comenzi de la tastatură. Porţiunea tranzitorie din

COMMAND.COM este încarcată peste SYSINIT, care nu mai este necesar. În final, Imaginea memoriei este cea din Figura 8.4.

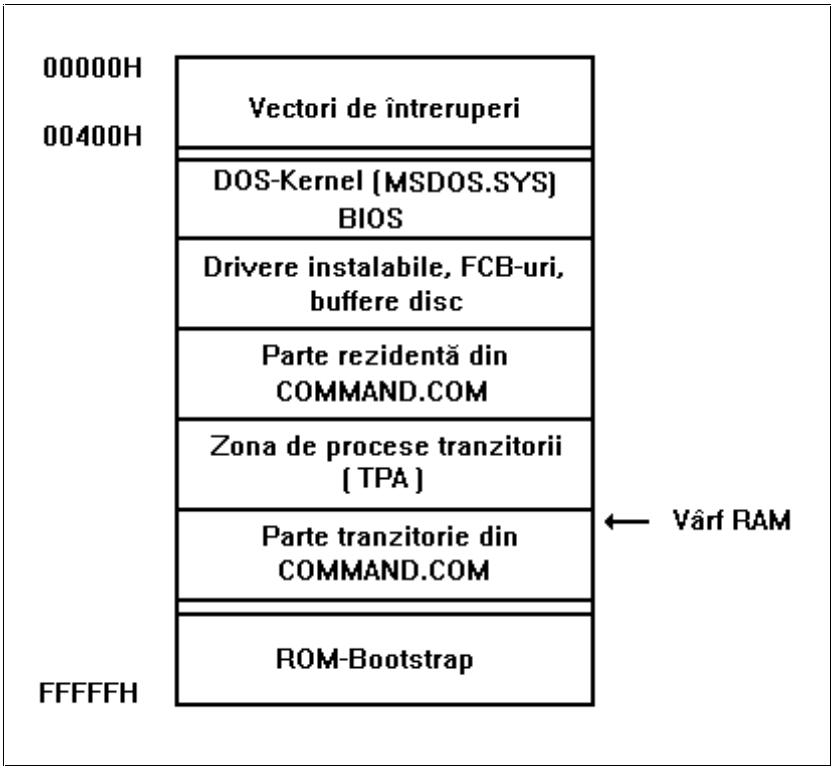

Figura 8.4 Imaginea finală a memoriei

8.3 Strucura Blocului PSP (Program Segment Prefix)

Blocul PSP este caracteristic ambelor categorii de programe executabile (.EXE şi .COM), fiind o zonă de 256 de octeţi care precede imaginea din memorie a programului executabil. La lansarea în execuţie a unui program, adresele DS:0000 şi ES:0000 indică începutul blocului PSP asociat programului. PSP-ul este construit de către funcţia EXEC la încărcarea programului de pe disc. Structura PSP-ului este ilustrată în Figura 8.5.

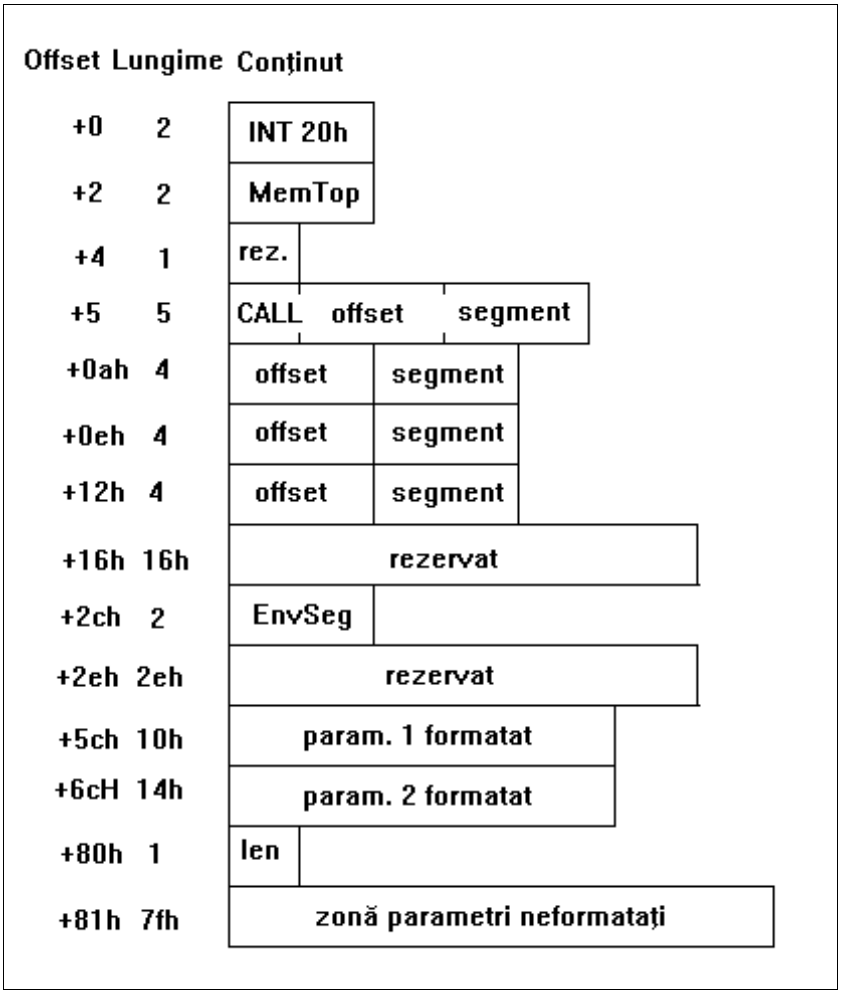

Figura 8.5 Structura blocului PSP

Semnificaţia câmpurilor din Figura 8.5 este următoarea:

- *octeţii 0-1* - conţin codul unei instrucţiuni INT 20H (ieşire în DOS), care permite unui program executabil să se încheie cu o instrucţiune JMP sau RET la adresa PSP:0; această tehnică nu mai este actuală, pentru terminarea unui program utilizându-se funcţia DOS 4CH;
- *octeţii 2-3* - conţin dimensiunea memoriei de bază continue (vârful memoriei), în paragrafe de câte 16 octeţi); această informaţie este folosită în procesul de încărcare a programului executabil;
- *octetul 4* - rezervat;
- *octeţii 5-9* - conţin codul unei instrucţiuni CALL de tip far la dispecerul de funcţii DOS;
- *octeţii 0AH-0DH* - conţin adresa far a rutinei de încheiere (accesibilă prin INT 22H); întreruperea 22H este o altă modalitate perimată de a încheia programele executabile; actualmente se utilizează funcţia DOS 4CH;
- *octeţii 0EH-10H* - conţin adresa far a rutinei de tratare a apăsării tastelor Ctrl-Break (accesibilă prin INT 23H); Ctrl-Break provoacă terminarea forţată a programelor în curs de execuţie;
- *octeţii 12H-15H* - conţin adresa far a rutinei de tratare a erorilor critice (accesibilă prin INT 23H); (erorile critice sunt de genul: nu există dischetă în unitatea de disc flexibil şi se dă o comandă de genul DIR A:);
- *octeţii 16H-2BH* - rezervaţi;
- *octeţii 2CH-2DH* - conţin adresa de segment a blocului de environment; acesta este o zonă de memorie în format text, care conţine informaţii de genul: căii (path) active, specificarea interpretorului de comenzi, specificarea prompterului de comenzi etc; întrea altele, în blocul de environment se găseşte calea completă a fişierului care conţine

programul executabil; în felul acesta, se poate şti în program numele fişierul executabil din care a fost încărcat programul; acesta este suportul fizic al argumenului argv[0] al funcţiei main de la programele C;

- *octeţii 2EH-5BH* - rezervaţil;
- *octeţii 5CH-6BH şi 6CH-7FH* - reprezintă aşa numiţii parametri formataţi; sunt utilizaţi pentru două blocuri de control fişiere (FCB-uri), care reprezintă o modalitate perimată de lucru cu fişiere disc; actualmente se utilizează funcţii de prelucrare a fişierelor orientate pe handler-e (vezi 8. ???;
- *octetul 80H* - reprezintă lungimea (numărul de caractere) conţinute în zona de parametri neformataţi;
- *octeţii 81H-0FFH* - reprezintă zona de parametri neformataţi; această zonă conţine caracterele introduse în linia de comandă după numele fişierului executabil (aşa numita coadă a comenzii); prin această zonă, un program executabil are acces la parametrii din linia de comandă (acesta este suportul oferit de DOS pentru parametrii numiţi generic argc şi argv de la funcţia main a programelor C); parametrii din linia de comandă nu cuprind paratrii de redirectare a dispozitivelor standard de intrare şi de ieşire; primii 43 de octeţi de la offsetul 80H mai sunt utilizaţi implicit şi ca Zonă de Transfer Disc (DTA); această zonă este folosită de unele funcţii de acces la disc şi se poate poziţiona într-un spaţiu rezervat de către utilizator prin funcţia DOS 1AH; de asemenea, adresa curentă a zonei DTA poate fi citită cu funcţia DOS 2FH;

8.4 Structura programelor de tip COM

Programele de tip COM corespund modelului de memorie tiny, în care există un singur segment şi toate salturile şi apelurile de proceduri sunt de tip near. Fişierul disk conţine exclusiv modulul care se încarcă, fără alte informaţii suplimentare, ca în cazul fişierelor de tip EXE.

Imaginea memoriei unui program COM este prezentată în Figura 8.6.

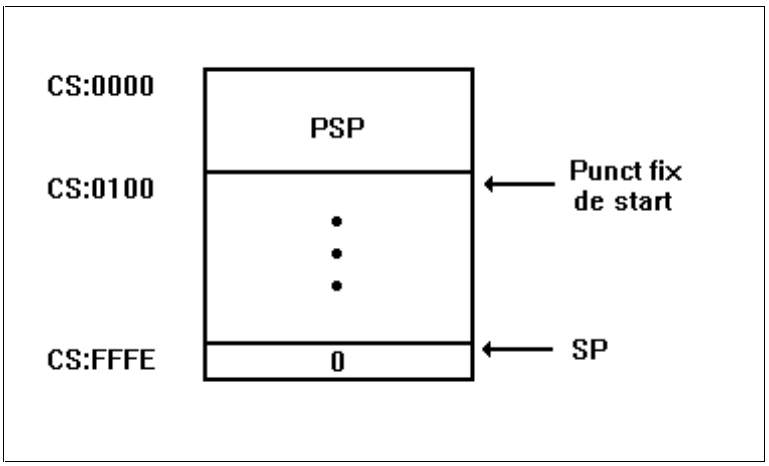

Figura 8.6 Imaginea memoriei ocupate de un program COM

Programele COM au ca punct de start adresa near 100H, adică imediat după blocul PSP. Indicatorul SP este plasat la adresa 0FFFEH, deci pe ultimul cuvânt adresabil. Acest cuvint e initializat cu 0.

Un program COM se poate încheia în unul din următoarel moduri:

- prin apel al funcţiei standard de terminare (funcţia DOS 4CH)
- prin apel al întreruperii soft INT 20H
- prin execuţia unei instrucţiuni RETN, dacă registrul SP conţine valoarea iniţială 0FFFEH. Cuvântul de la adresa 0FFFEH fiind 0, instrucţiunea RETN va da controlul la adresa near 0, adică la primul octet din PSP. Acolo insa se găseşte codul unei instrucţiuni INT 20H, care provoacă revenirea în DOS.

Programele COM nu necesită relocare, deoarece toţi simbolii relativi sunt deplassamente în cadrul unui segment unic. Simpla încărcare a registrului CS este suficientă.

Sablonul de dezvoltare pentru programele COM este:

```
        _TEXT       segment para 'CODE'
        ORG         100H
        ASSUME      CS:_TEXT, DS:_TEXT, ES:_TEXT, SS:_TEXT
start:
        jmp     init
                ;
                ; Aici se pun eventuale definiţii de date
                ;
init:
                ;
                ; Aici se pune codul propriu-zis
                ;
                mov     ax, 4C00H               ; Apel funcţie DOS
                int     21H                     ; pentru terminare program
_TEXT ENDS
end start                                       ; Etichetă de start
```

Programele de tip COM (ca de altfel şi cele de tip EXE) îşi rezervă toată memoria disponibilă. În unele situaţii e necesară reducerea la minim a memoriei rezervate, care se poate realiza printr-un apel al funcţiei DOS 4AH. Şablonul de dezvoltare al programelor COM va fi următorul:

```
        _TEXT       segment para 'CODE'
        ORG         100H
        ASSUME      CS:_TEXT, DS:_TEXT, ES:_TEXT, SS:_TEXT
start:
        jmp init
        ;
        ; Aici se pun definiţii de date
        ;
main proc near
        ;
        ; Aici se pune codul propriu-zis
        ;
                mov     ax, 4C00H
                int     21H
                main    endp
        ;
        ; Aici se pun alte proceduri (daca e cazul)
        ;
init:
                mov     ah, 4AH                 ; Funcţie redimensionare memorie
                mov     bx, offset end_mem      ; Limita pina unde se pastreaza
                mov     cl, 4                   ; Dimensiunea memoriei se
                shr     bx, cl                  ; specifică în paragrafe de 16 octeţi
                inc     bx                      ; Mai punem unul de siguranţă
                int     21H                     ; Apel propriu-zis
                mov     sp, offset end_mem      ; Iniţializare sp
                jmp     main
                ; Rezervăm o stivă de 512 octeţi, care să cuprindă şi spaţiul dintre etichetele
                ; init şi init_mem, zonă ce se poate suprascrie după încărcacrea programului
init_mem:
                db (512 - (init_mem - init)) dup (?)
                ;
```

```
                ; Până aici rezervăm memoria
                ;
end_mem:
_TEXT ENDS
end start
```

8.5 Structura şi încărcarea fişierelor EXE

Programele de tip EXE pot fi încărcate în orice locaţie de memorie şi se compun din mai multe segmente. Acest fapt impune ajustarea tuturor instrucţiunilor care conţin adrese de segment, cum ar fi:

- JMP/CALL FAR PTR nume
- MOV reg, SEG nume;
- MOV reg_seg, valoare.

Acest proces se numeşte **relocare** şi se execută la încărcarea programului în memorie. Punctul de start poate fi oarecare şi este stabilit prin directiva END.

Este evident că un fişier de tip EXE trebuie să conţină, pe lângă programul propriu-zis, şi informaţii despre simbolii relocabile, adresa de start, adresa segmentului de stivă etc. Aceste informaţii sunt grupate în prima parte a fişierului, numită **header al fişierului EXE**. Structura acestui header este ilustrată în Tabelul 8.7.

Offset	Dim.	Conţinut		Semnificaţie
0	2	4DH 5AH		Semnătură fişier EXE ('MZ')
2	2	PartPag		Lungime fişier modulo 512
4	2	PageCnt		Lungime în pagini de 512 octeţi, inclusiv header
6	2	ReloCnt		Număr de elemente din tabela de relocare
8	2	HdrSize		Dimensiune header în paragrafe de 16 octeţi
0AH	2	MinMem		Necesar minim de memorie peste sfârşitul programului (în paragrafe)
0CH	2	MaxMem		Necesar maxim de memorie peste sfârşitul programului (în paragrafe)
0EH	2	ReloSS		Deplasament segment stivă
10H	2	ExeSP		Valoare SP la execuţie
12H	2	ChkSum		Sumă de control
14H	2	ExeIP		Valoare IP (adresă de start)
16H	2	ReloCS		Deplasament segment de cod
18H	2	TablOff		Poziţia în fişier a tabelei de relocare (uzual 1CH)
1AH	2	Overlay		Indicator overlay (0 pentru modulele de bază)
?	4*	offs	seg	Tabela

		... offs	... seg	de relocare
?	?			Caractere până la limita de paragraf

Tabelul 8.7 Structura header-ului fişierelor .EXE

Primii doi octeţi identifică tipul fişierului. Următorii 4 octeţi conţin lungimea fişierului (inclusiv header-ul), în pagini de 512 octeţi şi, respectiv, modulo 512. Suma de control reprezintă suma cu semn schimbat a tuturor cuvintelor din fişier, permiţând un control al validităţiii fişierului.

Valorile **ReloSS** şi **ReloCS** reprezintă deplasamentele segmentelor de stivă şi cod faţă de adresa de segment de încărcare (segmentele pot fi în orice ordine). Valorile **ExeSP** şi **ExeIP** reprezintă conţinutul registrelor **SP** şi **IP** la intrare în execuţie. Aceste valori sunt deduse din dimensionarea segmentului de stivă şi din directiva END a modulului de program principal (care precizează eticheta de start).

Tabela de relocare cuprinde adresele tuturor cuvintelor care trebuie ajustate, adrese precizate prin offset în cadrul segmentului curent şi prin deplasamentul segmentului respectiv faţă de adresa de segment iniţială. Tabela de relocare are **ReloCnt** (vezi [6]) elemente, începe la poziţia **TablOff** (vezi [18H]) în fişier şi ocupă **ReloCnt*4** octeţi. Header-ul este completat cu octeţi fără semnificaţie, până la o limită de paragraf.

Relocarea programului este realizată de funcţia **EXEC** a sistemului de operare (funcţia 4BH) şi constă în următorul algoritm:

- Se creează un PSP cu funcţia DOS 26H;
- Se citesc 1CH octeţi din fişierul .EXE (aşa-numita porţiune formatată a header-ului), într-o zona locală de memorie;
- Se determină lungimea modulului = ((PageCnt * 512) - (HdrSize * 16)) - PartPag;
- Se determină deplasamentul în fişier al modulului = (HdrSize * 16);
- Se selecteaza o adresă de segment START_SEG (uzual PSP + 10H);
- Se citeşte modulul de program în memorie la adresa START_SEG:0000;
- Se setează poziţia de citire din fişier la începutul tabelei de relocare (TablOff);
- Pentru fiecare element al tabelei de relocare (ReloCnt):
  - se citeşte elementul din tabela ca 2 cuvinte (I_OFF, I_SEG);
  - se determină adresa actuală de memorie a elementului care trebuie relocat RELO_SEG = (START_SEG + I_SEG);
  - se citeşte elementul care trebuie relocat, deci cuvântul de la adresa (RELO_SEG:I_OFF);
  - se adună START_SEG la acest cuvânt (se ajusteaza adresa de segment);
  - se depune valoarea ajustată înapoi la adresa (RELO_SEG:I_OFF);
- Se alocă memorie suplimentară pentru program, conform MaxMem şi MinMem;
- Se iniţializează registrele semnificative şi se dă controlul programului:
  - ES = DS = PSP;
  - SS = START_SEG + ReloSS;
  - SP = ExeSP;
  - CS = START_SEG + ReloCS;
  - IP = ExeIP.

Poziţionarea registrelor CS şi IP se realizează print-o secvenţă de forma:

```
PUSH  START_SEG + ReloCS
PUSH  ExeIP
RETF                                    ; Intrare în execuţie
```

8.6 Programe de tip TSR (Terminate and Stay Resident)

Programele de tip EXE sau COM obişnuite sunt încărcate în memorie în zona de procese tranzitorii (TPA) şi ocupă memoria până la încheierea execuţiei. Mai concret spus, la execuţia funcţiei DOS 4CH, memoria ocupată de program este eliberată şi oferită sistemului DOS.

Programele de tip TSR (care sunt tot programe de tip EXE sau COM) rămân în memorie o durată nedeterminată de timp (sunt **rezidente**). Ele sunt **activate** de către evenimente externe, cum ar fi acţionarea unei taste special destinate (**hot-key**) sau expirarea unui interval de timp. Pentru ca să rămână rezident, un program trebuie să-şi încheie execuţia cu un apel al funcţiei DOS 31H, în care se precizează (în registrul DX) numărul de paragrafe de memorie care se păstrează rezidente.

Se deosebesc deci următoarele noţiuni:

- **activarea unui program TSR** - reprezintă întreruperea programului care se rulează în mod curent (nu neapărat un program utilizator), cu salvarea corespunzătoare a întregului context şi transferul controlului către programul TSR;
- **dezactivarea unui program TSR** - reprezintă refacerea contextului programului întrerupt şi transferul controlului către acesta;
- **instalarea unui program TSR** - reprezintă încărcarea de pe disc şi încheierea execuţiei prin apelul funcţiei 31H;
- **dezinstalarea unui program TSR** - reprezintă eliberarea memoriei ocupate de un program TSR instalat anterior.

8.6.1 Activarea şi dezactivarea programelor TSR

Activarea se face de obicei ca urmare a acţionării unei taste special destinate. Se utilizează întreruperea pe nivelul 9 (asociat tastaturii), care se produce la fiecare apăsare şi ridicare a unei taste.

Interacţiunea cu întreruperea 9 se face prin modificarea ("furtul") vectorului de întrerupere cu o rutină proprie de tratare, care apelează rutina veche, testând dacă s-a apăsat tasta specială de activare a TSR-ului. Rutina veche trebuie apelată, deoarece ea converteşte codul de scanare produs de tastatură într-un cod ASCII, actualizează buffer-ul tastaturii, gestionează starea tastelor Ctrl, Shift, Alt etc. De asemenea, este posibil să fie instalate mai multe TSR-uri "unul peste altul", activate prin taste speciale diferite, care au modificat la rândul lor vectorul 9 de întrerupere. Prin apelul rutinei vechi, se iau în considerare şi aceste TSR-uri instalate anterior.

Apelul rutinei vechi se face simulând o instrucţiune INT, prin secvenţa:

```
pushf
call    dword ptr cs:oldint
```

unde oldint este o locaţie în care s-a salvat adresa vechii rutine de tratare.

Trebuie observat faptul esenţial că, la activarea unui TSR, dintre registrele de segment nu se poate conta decât pe conţinutul registrului CS. Acesta este poziţionat în urma invocării rutinei de tratare pe nivelul 9, în timp ce restul

registrelor de segment sunt poziţionate conform programului care a fost întrerupt. De aceea, programele TSR se dezvoltă de obicei sub forma unui uni c segment, adresabil prin registrul CS. Aceasta explică prezenţa prefixului de segment din instrucţiunea CALL de mai sus. Dacă este cazul, se pot încărca şi alte registre de segmet cu valori adecvate, dar aceste valori trebuie memorate la etapa de instalare, obligatoriu în locaţii din segmentul adresabil prin registrul CS.

Secvenţa de parcurgere a mai multor programe TSR instalate este ilustrată în Figura 8.8.

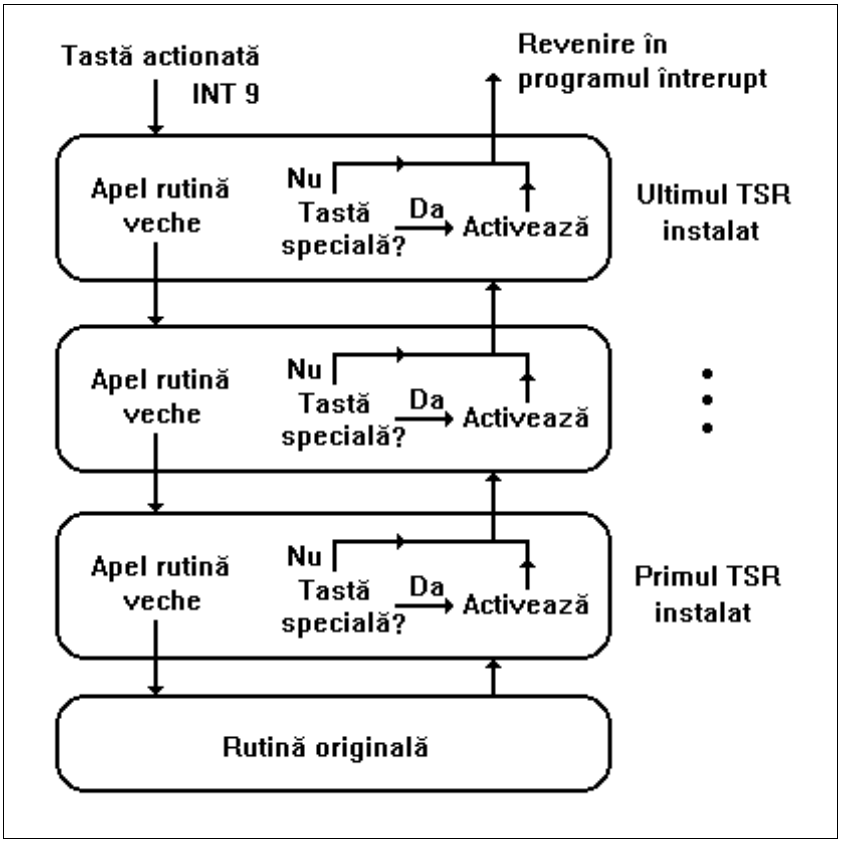

Figura 8.8 Parcurgerea succesivă a mai multor rutine de tratare pe nivelul 9

Non-reentranţa funcţiilor DOS

Problemele speciale legate de activarea TSR-urilor sunt în principal datorate faptului că sistemul DOS **nu este reentrant**. Aceasta înseamnă că, dacă se întrerupe execuţia unei funcţii DOS şi din rutina de întrerupere se apelează o altă funcţie DOS, este posibil ca funcţionarea sistemului de operare să fie compromisă.

Soluţia constă în activarea TSR-urilor numai în situaţii "sigure", adică, teoretic, atunci când nu se execută funcţii DOS. Se utilizează o combinaţie de doua tehnici:

- Prima tehnică se bazează pe un flag de sistem, numit uzual flag INDOS, care menţine un contor al nivelurilor de adâncime ale apelurilor de funcţii DOS. Adresa flagului INDOS poate fi obţinută prin funcţia DOS 34H. Teoretic, ar trebui să activăm TSR-ul doar atunci când acest contor este nul. Din păcate, flagul INDOS este diferit de zero mai tot timpul, datorită interpretorului de comenzi care aşteapă introducerea comenzilor de la tastatură prin execuţia funcţiei DOS 0AH (citire şir de caractere).

- A doua tehnică se bazează pe întreruperea soft 28H, care este apelată periodic de către sistem atunci când interpretorul aşteaptă comenzi de la consolă. Se consideră că, în momentul apelului acestei întreruperi, este relativ sigur să se activeze programul TSR.

În loc de a activa programul TSR imediat după ce a apărut cererea de acttivare, se va memora această cerere într-o variabilă, urmând ca ulterior, când sunt îndeplinite condiţiile de activare, să se facă activarea propriu-zisă. Concret, se va modifica rutina de tratare pe nivelul 28H, astfel încât să se testeze variabila care memorează cererea de activare, şi, dacă există cerere, să se activeze TSR-ul.

Acţiuni critice în timp

O altă categorie de probleme o reprezintă **acţiunile critice în timp**. Acestea sunt activităţi ale sistemului de operare sau ale BIOS-ului, a căror întrerupere ar putea duce le funcţionări defectuoase. Concret, este vorba despre rutinele de nivel scăzut pentru operaţiile cu discurile, realizate prin întreruperea de BIOS 13H şi de operaţiile de nivel scăzut asupra display-ului, realizate prin întreruperea BIOS 10H.

Problema se rezolvă prin modificarea întreruperilor 13H şi 10H, adăugându-se poziţionarea la 1 a unui flag propriu, pe durata cât aceste rutine de tratare sunt active. Programul TSR trebuie activat doar dacă flagul respectiv este nul. Structura noilor rutine de tratare va fi:

```
new13 proc    far
        mov   flag_bios, 1              ; Marchează operaţii critice în curs
        pushf                           ; Simulare execuţie
        call  dword ptr cs:old13        ; apel INT 13H
        mov   flag_bios, 0              ; Marchează sfârşit operaţii critice
        retf  2                         ; Revenire cu flagurile întoarse de
                                        ; apelul lui old13
new13 endp
```

în care old13 este o variabilă DWORD care memorează adresa vechii rutine de tratare, iar flag_bios este o variabilă rezervată în segmentul adresabil prin CS. Se observă încheierea procedurii prin RETF 2 şi nu prin IRET (deşi este procedură de tratare a unei întreruperi), ceea ce menţine flagurile aşa cum au fost întoarse de apelul vechii rutine de pe nivelul 13H.

Evitarea activării recursive

O problemă de bază la implementarea programelor TSR o reprezintă **evitarea activării recursive**. Dacă programul TSR a fost activat şi se află în execuţie, este posibil să apară o nouă cerere de activare (de exemplu, se apasă din nou tasta specială de activare). În această situaţie, programul TSR nu trebuie reactivat. Problema se rezolvă prin menţinerea unui alt flag propriu, care este 1 pe durata cât TSR-ul este activ şi 0 în rest.

Comutarea contextului la activare şi dezactivare

Programele TSR pot fi destul de complexe, în sensul că pot apela orice funcţie DOS, lucra cu fişiere disc etc. Aceasta presupune ca activarea să fie însoţită de o salvare cât mai completă a contextului programului întrerupt şi de o comutare a contextului către programul TSR. Similar, ldezactivarea TSR-ului, trebuie refăcut complet contextul programului întrerupt.

Problemele care trebuie avute în vedere sunt următoarele:

- salvarea şi restaurarea registrelor programului întrerupt;
- salvarea şi restaurarea unor structuri de date specifice programului întrerupt şi comutarea pe structurile specifice programului TSR:
  - salvarea adresei PSP-ului programului întrerupt şi marcarea ca fiind activ a PSP-ului programului TSR;
  - salvarea zonei DTA şi comutarea pe zona DTA a TSR-ului;
- comutarea pe stiva proprie a programului TSR;
- salvarea unei porţiuni (uzual 64 d eocteţi) din stiva programului întrerupt într-o zonă proprie;
- salvarea conţinutului ecranului.

Toate operaţiile de mai sus trebuie realizate în sens invers la cedare controlului către programul întrerupt.

Pentru citirea şi marcarea PSP-ului şi a zonei DTA sunt disponibile funcţii DOS specifice. Salvarea unei porţiuni din stiva programului întrerupt este necesară deoerece sistemul DOS lucrează cu stive proprii alocate la adrese fixe şi este posibil ca întreruperea să fi apărut atuci când registrele SS:SP erau poziţionate pe o asemenea stivă. Un apel al unei alte funcţii DOS va utiliza poate aceeaşi stivă DOS, ceea ce va distruge conţinutul stivei apelului anterior.

Salvarea şi restaurarea ecranului apare ca necesră în situaţia în care TSR-ul afişează ceva la consolă (ceea ce se întâmplă destul de frecvent). Salvarea resupune un transfer între memoria video şi o zonă proprie. De obicei, programele TSR lucrează în mod text, deci se va utiliza memoria video de la adresa 0B0000H (la ecrane monocrom) şi 0B8000H la ecrane color. Funcţie de modul video curent, se salvează 25 sau 43 de linii de ecran, fiecare a 80 de cuvinte (un simbol pe ecran este codificat prin caracterul ASCII extins şi printr-un octet de atribut, care precizează culoarea sau efecte speciale de afişare). Este indicat să se salveze şi modul video curent precum şi poziţia cursorului pe ecran. Dacă este instalat un driver de mouse, trebuie salvată poziţia cursorului mousului şi inhibată afişarea mousului pe durata cât TSR-ul este activ. Dacă este cazul, se poate comuta pe o rutină (handler) proprie de tratare a evenimentelor de la mouse.

Penru citirea şi setarea cursoarelor, a modurilor video etc., se utilizează funcţii BIOS specifice.

8.6.2 Instalarea şi dezinstalarea programelor TSR

Instalarea unui program TSR înseamnă încărcarea lui în memorie şi pregătirea condiţiilor pentru activare. Dezinstalarea unui TSR înseamnă eliberarea memoriei ocupate la instalare. Trebui remarcat că dezinstalarea trebui făcută de către un alt program decât cel rezident (eventual o altă instanţă a aceluiaşi program).

Evitarea instalării repetate

Programul TSR este memorat într-un fişier executabil. Nimic nu ne împiedică să lansăm în execuţie acest program de mai multe ori. Totuşi, nu se poate accepta ca un TSR să fie instalat de mai multe ori, deoarece s-ar consuma în mod inutil memorie. Pe de altă parte, secvcenţa de activare din Figura 8.8 ar fi compromisă (care instanţă se va activa ?)

Un program TSR trebuie deci să testeze dacă nu este instalat deja în memorie şi să refuze instalarea în acest caz.

Testarea se face definind o secvenţă particulară de caractere (o semnătură) la un offset fix în cadrul unicului segment care compune programul TSR. Identificarea locului în care programul TSR este eventual rezident se poate face numai prin examinarea vectorului de întrerupere care declanşează activarea (uzual, INT 9). Citind acest vector, identificăm adresa de segment a rutinei de tratare, adică adresa de segment a programului TSR care este eventual instalat. Comparând semnătura programului curent cu semnătura din segmentul indicat de vectorul INT 9, se determină dacă TSR-ul este deja instalat (vezi Figura 8.9).

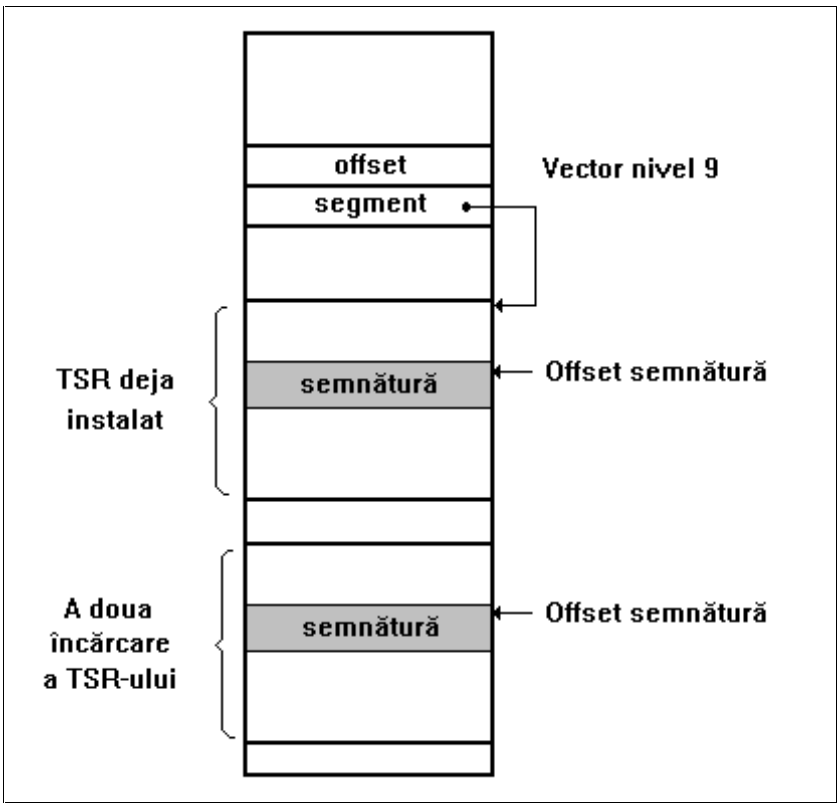

Figura 8.9 Tehnica de testarea a instalării

Această tehnică este posibilă numai dacă TSR-ul deja instalat este ultimul din lanţul de TSR-uri care au modificat vectorul de întrerupere pe nivelul 9.

O altă soluţie, mai generală dar mai lentă, este parcurgerea tuturor blocurilor de memorie alocate de către sistem. Aceste blocuri sunt structuri de date specifice DOS care conţin informaţii depre deţinătorul blocului, deci cărui modul de program i-a fot alocat blocul respecti. Astfel, se caută de fapt în memorie numele fişierului executabil care conţine programul TSR. În acest capitol, vom utiliza prima tehnică de identificare.

Instalarea trebuie să salveze o serie de date necesare procesului de activare, cum ar fi adresa de PSP şi a zonei DTA a programului TSR, adresa stivei programului TSR etc. De asemenea, trebuie rezervat spaţiu pentru stiva TSR, pentru salvarea ecranului, pentru copiile vectorilor de întrerupere modificaţi etc.

Instalarea TSR-ului mai presupune şi modificarea vectorilor de întrerupere 9, 10H, 13H etc. cu adresele propriilor rutine de tratare. În final, se calculează dimensiune spaţiului de memorie care trebuie să rămână rezident şi se încheie execuţia cu apelul fincţiei DOS 31H.

Dezinstalarea TSR-urilor

La identificarea situaţie de TSR deja instalat, multe programe se mulţumesc să afişeze la consolă un mesaj de genul:

```
Program XXXXX already installed!
```

şi apoi să-şi încheie execuţia în mod normal.

O variantă mai evoluată este ca, la a doua invocare a programului TSR, să se realizeze dezinstalarea primei instanţe, după care a doua instanţâ să se încheie cu ieşire normală în DOS.

Dezinstalarea trebuie să elibereze spaţiul de memorie alocat primei instanţe. Trebuie mai întâi eliberat blocul de environment (mediu) al TSR-ului instalat. Adresa blocului de environment se găseşte la un offset fix în cadrul PSP-ului (vezi 8.3). La instalare, este necesară memorarea adresei blocului PSP într-o variabilă aflată la un offset fix. După ce s-a eliberat blocul de environment, se eliberează memoria propriu-zisă cu funcţia 49H şi se încheie execuţia cu funcţia DOS 4CH. Figura 8.10 ilustrează acest proces.

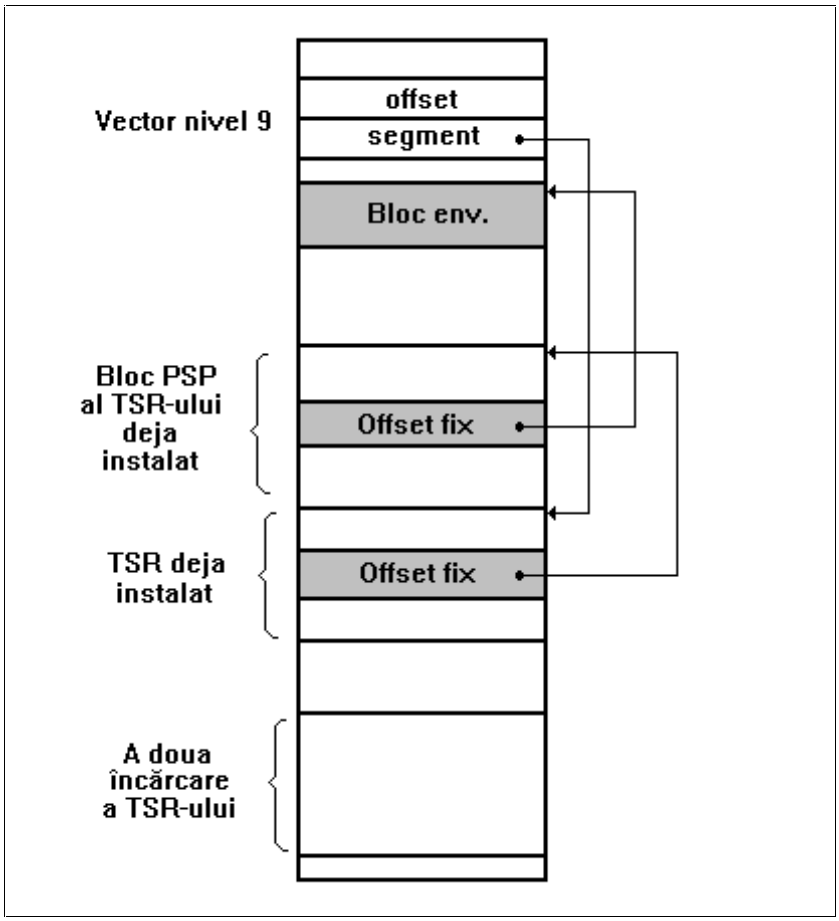

Figura 8.10 Tehnica de dezinstalare

De observat că, funcţie de testul asupra instalării TSR-ului, se va face instalarea instanţei curente sau dezinstalarea instanţei anterior instalate. Tot la

dezinstalare se refac vectorii 9, 10H, 13H şi 28H, care au fost modificaţi la instalare.

8.6.3 Şablon de dezvoltare pentru programe TSR

Următorul program reprezintă un şablon de dezvoltare pentru aplicaţiile TSR uzuale, fiecare acţiune fiind comentată intensiv. Trebuie observat accesul la buffer-ul tastaturii, realizat prin pointerii de la adresele absolute 400:1AH şi 400:1CH, care indică coada cele două capete ale buffer-ului. Buffer-ul tastaturii se află în spaţiul de adrese 400:1EH-400:3CH, fiind organizat circular. Buffer-ul este vid când cei doi pointeri sunt egali şi este plin când pointerul de introducere este cu o poziţie (2 octeţi) în urma pointerului de extragere. Capacitatea logică este de 30 de octeţi (15 taste) iar cea fizică de 32 de octeţi (16 taste).

Acest model de program realizează doar şteregerea ecranului, afişarea unui mesaj şi aşteptarea apăsării unei taste oarecare. Pentru afişări la consolă se folosesc direct funcţii DOS, ceea ce explică terminatorul '$' pentru şiruri de caractere. Tasta de activare este F1. La o a doua execuţie, programul se dezinstalează.

Se observă şi funcţiile DOS sau BIOS pentru citirea sau poziţionarea diverselor date de sistem.

```
        bios segment at 40h                     ; Segment date BIOS
                org     1AH
                b_out   dw      ?               ; Indicator extragere buffer BIOS
                b_in    dw      ?               ; Indicator introducere buffer BIOS
        bios ends

        cseg segment
        assume cs:cseg
        MONO    equ     0
        F1      equ     3B00H                   ; 59 în zecimal = cod F1
        HOTKEY          equ     F1              ; Tastă activare

        ident           db      '_IDENTIF_'
        lng             equ     $ - ident
        mes_act         db      'TSR activ: Apasati orice tasta '
                        db      'pentru a reveni'
                        db      7, '$'
        mes_inst        db      ' TSR instalat: Activare cu F1', '$'
        mes_dezinst     db      ' TSR dezinstalat', '$'
;
;       Spaţiu pentru memorat vectorii vechi de întrerupere
;
        old9    label dword
                old9_off        dw      ?
                old9_seg        dw      ?
        old10   label dword
                old10_off       dw      ?
                old10_seg       dw      ?
        old13   label dword
                old13_off       dw      ?
                old13_seg       dw      ?
        old28   label dword
                old28_off       dw      ?
                old28_seg       dw      ?
```

```
;
;       Spaţiu pentru adresa flagului INDOS
;
        flag_dos        label   dword
        indos_off       dw      ?
        indos_seg       dw      ?
;
        flag_tsr                dw      0               ; Flag TSR activ
        flag_bios       dw      0                       ; Flag BIOS activ
        cerere          db      0                       ; Memorează cerere viitoare
        cod_scan        dw      ?                       ; Cod tastă activare
        ss_int          dw      ?                       ; Salvare registre stivă
        sp_int          dw      ?                       ; program întrerupt
        ss_tsr          dw      cseg                    ; Stivă locală TSR
                        dw      256 dup (?)
        sp_tsr          dw      sp_tsr
        buff_video      db      4000 dup (?)            ; Buffer salvare ecran
        salv_cursor     dw      ?                       ; Salvare cursor
        dta             db      43 dup (?)              ; DTA local TSR
        psp_tsr         dw      ?                       ; Adresă PSP TSR
        dta_tsr         dd      dta                     ; Adresă DTA TSR
;
        psp_int         dw      ?                       ; Salvare PSP şi
        dta_int         dd      ?                       ; DTA program întrerupt
;
;       Noile rutine de tratare întreruperi 9, 10H, 13H, 28H
;
new10 proc far
        mov     flag_bios, 1
        pushf
        call    dword ptr cs:old10
        mov     flag_bios, 0
        retf    2
new10 endp
new13 proc      far
        mov     flag_bios, 1
        pushf
        call    dword ptr cs:old13
        mov     flag_bios, 0
        retf    2
new13 endp

new9    proc    far
        assume ds:bios
        pushf
        call    cs:old9                         ; Apel rutină veche
        cli                                     ; Dezactivare întreruperi
        cmp     cs:flag_tsr, 0                  ; Test deja activ
        jne     new9_end
        push    ds                              ; Salvări
        push    bx                              ; registre
        push    ax
        mov     ax, bios
        mov     ds, ax                          ; Acces BIOS prin ds
        mov     bx,  b_out                      ; Indicator extr. buffer BIOS
new9_0:
        cmp     bx, b_in                        ; Este buffer-ul BIOS vid ?
        je      new9_1                          ; Da, return
        mov     ax, [bx]                        ; Nu e vid, iau codul
```

```
        mov     cs:cod_scan, ax                 ; şi îl memorez
        cmp     ax, HOTKEY                      ; Este tastă de activare ?
        jnz     new9_1                          ; Nu, return
        add     bx, 2                           ; Incrementez (extrag din buffer)
        cmp     bx, 3EH                         ; Compar cu val. maxima + 2
        jne     new9_11                         ; Nu este = 3EH
        mov     bx,1eh                          ; Este 3EH şi se aduce la val. iniţiala
new9_11:
        mov     b_out, bx                       ; Memorare indicator buffer BIOS
        jmp     new9_2
new9_1:
        pop     ax
        pop     bx
        pop     ds
        iret

assume          ds: NOTHING

new9_2:
        pop     ax
        pop     bx
        pop     ds

        cmp     cs:flag_bios, 0                 ; Test operaţii critice
        je      cont1
        mov     cs:cerere, 1                    ; Marchează cerere
new9_end:
        iret                                    ; Revenire în programul întrerupt
cont1:
        push    ds
        push    bx
        lds     bx, cs:flag_dos                 ; Test INDOS
        cmp     byte ptr [bx], 0
        je      cont2
        mov     cs:cerere, 1                    ; Marchează cerere
        pop     bx
        pop     ds
        iret                                    ; Revenire în programul întrerupt
cont2:
        pop     bx
        pop     ds
        call    activ_tsr                       ; Apel funcţie TSR
        iret
new9    endp
;
new28 proc far
        pushf                                   ; Apel rutină
        call    dword ptr cs:old28              ; veche
        cli
        cmp     flag_tsr, 0                     ; Test TSR activ
        jz      cont3
        iret
cont3:
        cmp     flag_bios, 0                    ; Test operaţii critice
        jz      cont4
cont5:
        ret     2
cont4:
        cmp     cerere, 0                       ; Este cerere ?
```

```
        jz      cont5
        mov     cerere,0
        call    near ptr activ_tsr                      ; Satisface cererea
        ret     2
new28 endp
;
;       Rutina de baza care realizează activarea TSR-ului
;
activ_tsr proc near
        cli
        mov     flag_tsr, 1                             ; Marchează TSR activ
;
;       Comutarea contextului
;
        mov     sp_int, sp                              ; Salvează stiva
        mov     sp, ss                                  ; programului întrerupt
        mov     ss_int, sp
        mov     sp, ss_tsr                              ; Comută pe stiva TSR-ului
        mov     ss, sp
        mov     sp, sp_tsr
;
        push    ax                                      ; Salvări
        push    bx                                      ; registre
        push    cx
        push    dx
        push    si
        push    di
        push    bp
        push    ds
        push    es
;
        mov     cx, 64                                  ; Salvare 64
        mov     ax, ss_int                              ; de octeţi
        mov     ds, ax                                  ; din stiva
        mov     si, sp_int                              ; programului
        cld                                             ; întrerupt
iar1:
        push    [si]                                    ; în stiva
        inc     si                                      ; proprie
        inc     si
        loop    iar1
;
        mov     ah, 62H                                 ; Citeşte PSP-ul
        int     21h                                     ; programului întrerupt
        mov     psp_int, bx                             ; şi salvează
;
        push    es
        mov     ah, 2FH                                 ; Citeşte DTA
        int     21h                                     ; programului întrerupt
        mov     word ptr dta_int, bx                    ; şi
        mov     word ptr dta_int+2, es                  ; salvează
        pop     es
;
        mov     ah,50H                                  ; Marchează PSP-ul
        mov     bx,psp_tsr                              ; TSR-ului
        int     21h                                     ; ca fiind cel activ
;
        push    ds                                      ; Comută
        mov     ah,1ah                                  ; pe zona DTA
```

```
        lds     dx,dta_tsr                      ; a TSR-ului
        int     21h
        pop     ds
;
        sti
        cmp     cod_scan, HOTKEY                ; Test tastă activare
        jz      e1                              ; sau nu
        jmp     end_tsr
e1:
        mov     ah, 3
        mov     bh, 0                           ; Citire
        int     10H                             ; poziţie
        mov     salv_cursor, dx                 ; cursor şi salvare
;
IF MONO
        mov     ax, 0B000H                      ; Adresa ecran monocrom
ELSE
        mov     ax, 0B800H                      ; Adresa ecran color
ENDIF
;
        push    ds
        push    es
        mov     ds, ax                          ; Sursă = B000:0 (B8000:0)
        mov     ax, cs
        mov     es, ax                          ; Destinaţie = cs:buff_video
        cld
        xor     si, si
        mov     di, offset buff_video
        mov     cx, 4000                        ; Copiere 80*25*2
        rep     movsb                           ; octeţi
        pop     es
        pop     ds
;
        mov     bh, 7                           ; Ştergere
        mov     ax, 600H                        ; ecran
        mov     cx, 0                           ; De la linia 0, coloana 0
        mov     dx, 24*256 + 79                 ; până la linia 24, coloana 79
        int     10H
;
;       Aici se pune codul propriu-zis pentru funcţia programului TSR
;       În cazul de faţă, se tipăreşte un mesaj şi se asteaptă apăsarea unei taste
;
        mov     bh, 0                           ; Poziţionare cursor în
        mov     dx, 010CH                       ; linia 1, coloana 0CH
        mov     ah, 2
        int     10H
;
        mov     ax, cs
        mov     ds, ax                          ; DS = segment curent
        mov     ah, 9
        mov     dx, offset mes_act              ; Mesaj TSR activ
        int     21H
        mov     ah, 0                           ; Citire caracter de la tastatură
        int     16H
;
;       Refacerea contextului
;
        mov     dx, salv_cursor                 ; Poziţionare cursor
        mov     bh, 0
```

```
        mov     ah, 2
        int     10H
;
        mov     ax, cs                          ; Refacere
        mov     ds, ax                          ; ecran
        mov     si, offset buff_video
;
IF MONO
        mov     ax, 0B000H
ELSE
        mov     ax, 0B800H
ENDIF
;
        mov     es, ax
        xor     di, di
        cld
        mov     cx, 4000
        rep     movsb
;
end_tsr:
        cli
        mov     ah, 50H                         ; Comutare pe PSP-ul
        mov     bx, psp_int                     ; programului întrerupt
        int     21H
;
        push    ds
        mov     ah, 1aH                         ; Comutare pe DTA a
        lds     dx, dta_int                     ; programului întrerupt
        int     21H
        pop     ds
;
        mov     cx, 64                          ; Refacere conţinut stivă
        mov     ax, ss_int                      ; program
        mov     ds, ax                          ; întrerupt
        mov     si, sp_int                      ; din stiva
        add     si, 128                         ; curentă
iar2:
        dec     si
        dec     si
        pop     [si]
        loop    iar2
;
        pop     es                              ; Refacere
        pop     ds                              ; registre
        pop     bp                              ; din stivă
        pop     di                              ; curenta (TSR)
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
;
        mov     sp, ss_int                      ; Refacere registre stivă
        mov     ss, sp                          ; ale programului
        mov     sp, sp_int                      ; întrerupt
;
        mov     flag_tsr, 0                     ; Marcare TSR inactiv
        sti
        ret
```

```
activ_tsr endp
;
;       Rutina urmatoare reface vectorii de întrerupre
;       şi eliberează memoria alocată pentru programul TSR
;
dezinstproc near
;
;       În ES : segmentul programului rezident
;
        push    ds
        push    es
        cli
;
;       Refacere vectori de intrerupere 9, 10H, 13H, 28H
;
        mov     ax, 2509H                       ; Vector 9
        mov     ds, es:old9_seg
        mov     dx, es:old9_off
        int     21H
;
        mov     ax, 2510H                       ; Vector 10H
        mov     ds, es:old10_seg
        mov     dx, es:old10_off
        int     21H
;
        mov     ax, 2513H                       ; Vector 13H
        mov     ds, es:old13_seg
        mov     dx, es:old13_off
        int     21H
;
        mov     ax, 2528H                       ; Vector 28H
        mov     ds, es:old28_seg
        mov     dx, es:old28_off
        int     21H
        sti
;
;       Prima dată eliberăm blocul de environment, a cărui adresă
;       de segment se găseşte la offset-ul 2CH în PSP
;
        mov     bx, es:psp_tsr
        mov     es, bx                          ; Adresă PSP
        mov     es, es:[2ch]                    ; ES = adresă bloc environment
        mov     ah, 49H                         ; Eliberare memorie alocată
        int     21H                             ; la instalare
;
;       Acum eliberăm memoria deţinută de progamul
;       TSR, începând de la PSP
;
        mov     es, bx                          ; ES = PSP-ul TSR-ului
        mov     ah, 49H                         ; Eliberare
        int     21H
;
        pop     es
        pop     ds
        ret                                     ; Revenire
dezinstendp
;
sfirsit label   byte                            ; Etichetă pentru a păstra codul
                                                ; rezident numai până aici
```

```
;
;       Aici e punctul de start al programului
;
start:
        mov     ax, cseg                        ; Iniţializări registre
        mov     ds, ax                          ; de
        mov     ss, ax                          ; segment
        mov     sp, sp_tsr                      ; şi stivă
;
;       Testăm dacă TSR-ul este deja instalat
;
        mov     ax, 3509H                       ; Citire vector de
        int     21H                             ; întrerupere 9
                                                ; ES = segment program instalat
                                                ; deja (eventual)
        mov     ax, cs
        mov     ds, ax                          ; DS = segment  program curent
        xor     si, si                          ; Offseturi 0
        mov     di, si
        mov     cx, lng                         ; Lungime mesaj
        repz    cmpsb                           ; Comparăm ds:[si] cu es:[di]
        jnz     not_inst                        ; Nu e instalat, salt
;
        call    dezinst                         ; Dezinstalare, apoi
        mov     dx, offset mes_dezinst          ; mesaj dezinstalare
        mov     ah, 9
        int     21H
;
        mov     ax, 4c00H                       ; Ieşire normală în DOS
        int     21H

not_inst:
;
;       Instalare TSR
;
        mov     ah, 62H                         ; Citire PSP şi salvare
        int     21H                             ; pentru comutare context
        mov     psp_tsr, bx                     ; la activări viitoare
;
        mov     ah, 1aH                         ; Similar pentru DTA
        lds     dx, dta_tsr
        int     21H
;
;       Citiri şi salvări vectori vechi
;       (nivelele 9, 10H, 13H, 28H)
;
        mov     ax, 3509H
        int     21H
        mov     cs:old9_off, bx
        mov     cs:old9_seg, es                 ; Vector 9
        mov     ax, 3510H
        int     21H
        mov     cs:old10_off, bx
        mov     cs:old10_seg, es                ; Vector 10H
        mov     ax,3513H
        int     21H
        mov     cs:old13_off, bx
        mov     cs:old13_seg, es                ; Vector 13H
        mov     ax, 3528H
```

```
        int     21H
        mov     cs:old28_off, bx
        mov     cs:old28_seg, es                ; Vector 28H
        mov     ax, cs
        mov     ds, ax                          ; DS = segmentul de cod
;
        push    ds
        push    es
;
;       Poziţionare vectori noi
;
        mov     ax, 2509H
        mov     dx, offset new9
        int     21H                             ; Vector 9 nou
;
        mov     ax, 2510H
        mov     dx, offset new10
        int     21H                             ; Vector 10H nou
;
        mov     ax, 2513H
        mov     dx, offset new13
        int     21H                             ; Vector 13H nou
;
        mov     ax, 2528H
        mov     dx, offset new28
        int     21H                             ; Vector 28H nou
;
;       Citire şi salvare adresă flag INDOS
;
        mov     ax, 3400H
        int     21H
        mov     cs:indos_off, bx
        mov     cs:indos_seg, es
;
        pop     es
        pop     ds
;
        mov     ah, 9
        mov     dx, offset mes_inst             ; Mesaj instalare
        int     21H
;
;       Calcul acoperitor al necesarului de memorie (inclusiv PSP-ul)
;
        mov     dx, offset sfirsit + 100h
        mov     cl, 4                           ; Spaţiul se specifică
        shr     dx, cl                          ; în paragrafe de câte 16 octeţi
        inc     dx
        mov     ax, 3100H                       ; Terminare cu rămânere în memorie
        int     21H

cseg    ends
end     start
```

8.7 Funcţii de intrare/ieşire orientate pe handlere

Handlerele sunt variabile întregi mai mari sau egale cu zero, întoarse de sistemul DOS sau predefinite, care identifică în mod univoc fişiere disc sau dispozitive. Funcţiile orientate pe handlere au generalitate maximă şi permit schimbarea dispozitivelor de intrare şi/sau ieşire asociate unui program într-o manieră foarte comodă.

Sunt predefinite următoarele handlere:

- 0 - numit şi **stdin** (**Standard Input- Intrare Standard**), uzual asignat la dispozitivul consolă;
- 1- numit şi **stdout** (**Standard Output - Ieşire Standard**), uzual asignat la dispozitivul consolă;
- 2 - numit şi **stderr** (**Standard Error - Ieşire standard pentru mesaje de eroare**), totdeauna asignat la consolă;
- 3 - numit şi **stdaux** (**Standard Auxilliary - Dispozitiv auxiliar standard**), uzual asignat la dispozitivul COM1;
- 4 - numit şi **stdprn** (**Standard Printer - Imprimantă standard**), uzual asignat la dispozitivul LPT1.

Funcţiile orientate pe handlere pot avea acces la orice subdirector al discurilor, pornind de la rădăcină sau de la directorul curent.

Intrarea şi ieşirea standard asociate unui program executabil pot fi redirectate din linia de comandă, cu ajutorul caracterelor < (redirectare stddin) şi > (redirectare stdout). Dacă un program executabil **nume.exe** utilizează funcţii de intrare/ieşire orientate pe handlere, atunci o lansare în execuţie de forma:

```
C:\> nume < file1
```

va face ca toate citirile care au loc de la consolă să fie înlocuite cu citiri din fiişierul **file1**.

Similar, o lansare în execuţie de forma:

```
C:\> nume > file2
```

va face ca toate afişările care au loc la consolă să fie înlocuite cu scrieri în fiişierul **file2**.

În fine, o linie de comandă de forma:

```
C:\> nume < file1 > file2
```

va redirecta atât stdin cât şi stdout.

Interpretorul de comenzi permite şi un simulacru de mecanism **pipe**, prin care ieşirea standard a unui program **nume1** este redirectată către intrarea standard a altui program **nume2**:

```
C:\> nume1 |  nume2
```

Cum sistemul DOS nu poate executa programe în mod concurent, linia de mai sus este echivalentă cu secvenţa:

```
C:\> nume1 > temp
C:\> nume2 < temp
C:\> del temp
```

În continuare, sunt descrise principalele funcţii de intrare/ieşire orientate pe handlere şi parametrii acestora, respectiv valorile întoarse.

- **Create File (Creează sau recreează fişier)**
  - Parametri de intrare:
    AH = 3CH (cod funcţie);
    CX = atribute (Read Only, Hidden, System etc.);
    DS:DX = adresă şir de caractere terminat cu 0 (calea fişierului).
  - Parametri de ieşire:
    CF = 0, AX = handler (dacă operaţia a fost încheiată cu succes);
    CF = 1, AX = cod de eroare (eşec).
  - Descriere:
    Se creează fişierul cu numele specificat; dacă fişierul exista anterior, conţinutul vechi se pierde.
- **Open File (Deschide fişier)**
  - Parametri de intrare:
    AH = 3DH (cod funcţie);
    AL = mod acces (0 = pentru citire, 1 = pentru scriere, 2 = punere la zi etc.);
    DS:DX = adresă şir de caractere terminat cu 0 (calea fişierului).
  - Parametri de ieşire:
    CF = 0, AX = handler (dacă operaţi a fost încheiată cu succes);
    CF = 1, AX = cod de eroare (eşec).
  - Descriere:
    Se deschide fişierul cu numele specificat, pentru operaţia specificată; fişierul trebuie să existe; la deschidere pentru scriere, conţinutul vechi se pierde.
- **Read File (Citeşte din fişier)**
  - Parametri de intrare:
    AH = 3FH (cod funcţie);
    BX = handler (obţinut la deschidere sau predefinit);
    DS:DX = adresă buffer de citire (dimensionat corespunzător);
    CX = număr de octeţi cerut pentru citire.
  - Parametri de ieşire:
    CF = 0, AX = număr de octeţi citiţi efectiv (dacă operaţia a fost încheiată cu succes);
    CF = 1, AX = cod de eroare (eşec);
  - Descriere:
    Se încearcă citirea a CX octeţi din fişier, din poziţia curentă; poziţia este avansată automat; dacă de la poziţia curentă până la sfârşitul fişierului sunt mai puţin de CX octeti, se citesc câţi octeţi există (posibil 0); dacă AX = 0, s-a ajuns la EOF înainte de apelul funcţiei, iar dacă AX < CX, s-a ajuns la EOF în decursul execuţiei funcţiei; ambele situaţii sunt corecte.
- **Write File (Scrie în fişier)**
  - Parametri de intrare:
    AH = 40H (cod funcţie);
    BX = handler (obţinut la deschidere sau predefinit);
    DS:DX = adresă buffer de scriere;
    CX = număr de octeţi cerut pentru scriere.
  - Parametri de ieşire:
    CF = 0, AX = număr de octeţi scrişi efectiv (AX diferă de CX numai în situaţii de eroare);

CF = 1, AX = cod de eroare (eşec).

- Descriere:

  Se încearcă scrierea a CX octeţi în fişier, din poziţia curentă; poziţia este avansată automat; dacă CX < AX, este vorba de o situaţie de eroare.

- **Close File (Închide fişier)**
  - Parametri de intrare:

    AH = 3EH (cod funcţie);
    BX = handler (obţinut la deschidere).
  - Parametri de ieşire:

    CF = 0 (operaţie încheiată cu succes);
    CF = 1, AX = cod de eroare (eşec).
  - Descriere:

    Se închide fişierul, adică se taie legătura logică dintre handle şi fişier, stabilită la deschidere; operaţia este obligatorie, în special la fişierele deschise pentru scriere, deoarece acum se scriu bufferele interne ale sistemului DOS în fişier.
- **Delete File (Şterge fişier)**
  - Parametri de intrare:

    AH = 41H (cod funcţie);
    DS:DX = adresă şir de caractere terminat cu 0 (calea fişierului).
  - Parametri de ieşire:

    CF = 0 (operaţie încheiată cu succes);
    CF = 1, AX = cod de eroare (eşec).
  - Descriere:

    Se şterge fişierul cu numele specificat.
- **Duplicate Handle (Duplică handler)**
  - Parametri de intrare:

    AH = 45H (cod funcţie);
    BX = handler (obţinut la deschidere sau predefinit).
  - Parametri de ieşire:

    CF = 0, AX = handler duplicat (operaţie încheiată cu succes);
    CF = 1, AX = cod de eroare (eşec).
  - Descriere:

    Se obţine o copie a handlerului asociat fişierului; operaţia este necesară înainte de redirectarea unor fişiere sau dispozitive, în situaţia în care dorim şi revenirea la asocierea iniţială.
- **Redirect Handle (Redirectează handler)**
  - Parametri de intrare:

    AH = 46H (cod funcţie);
    CX = handler care trebui redirectat;
    BX = handler (obţinut la deschidere sau predefinit).
  - Parametri de ieşire:

    CF = 0, AX = handler redirectat (operaţie încheiată cu succes);
    CF = 1, AX = cod de eroare (eşec);
  - Descriere:

    Redirectează handlerul CX către handlerul BX; handlerul BX a fost obţinut în urma unei deschideri de fişier sau în urma unei duplicări.

- **Seek File (Poziţionează indicatorul fişierului)**
  - Parametri de intrare:
    AH = 42H (cod funcţie);
    BX = handler (obţinut la deschidere);
    CX:DX = poziţia cerută în fişier (întreg pe 32 de biţi, CX = partea high, DX = partea low);
    AL = punctul de referinţă (0=faţă de începutul fişierului, 1=faţă de poziţia curentă, 2=faţă de sfârşitul fişierului).
  - Parametri de ieşire:
    CF = 0, DX:AX = noua poziţie în fişier (operaţie încheiată cu succes);
    CF = 1, AX = cod de eroare (eşec).
  - Descriere:
    Se poziţionează indicatorul intern al fişierului într-o poziţie absolută. Are sens la exploatarea în acces direct a fişierelor, în care dorim să controlăm explicit poziţia din care se va citi sau în care se va scrie. De exemplu, după ce am făcut operaţii de citire, vrem să revenim la începutul fişierului, pentru o nouă operaţie de citire. Acest lucru se poate realiza cu CX=DX=0, AL=0.
- **Find First (Identifică primul fişier dintr-un nume generic cu * şi/sau ?)**
  - Parametri de intrare:
    AH = 4EH (cod funcţie);
    DS:DX = adresă şir de caractere terminat cu 0 (poate conţine '*' şi '?');
    CX = atribute (0 = normal).
  - Parametri de ieşire:
    CF = 0, zona DTA conţine informaţii despre fişierul identificat (operaţie încheiată cu succes);
    CF = 1, AX = cod eroare (nu s-a identificat fişier).
  - Descriere:
    Se caută primul fişier care verifică numele generic specificat. La offset-ul 1EH în zona DTA se găseşte numele acestui fişier; este necesară obţinerea adresei zonei DTA în prealabil (necesară la funcţia următoare).
- **Find Next (Identifică următorul fişier dintr-un nume generic cu * şi/sau ?)**
  - Parametri de intrare:
    AH = 4FH (cod funcţie);
    DS:DX = adresa zonei DTA sau a unei copii a acestei zone.
  - Parametri de ieşire:
    CF = 0, zona DTA conţine informaţii despre următorul fişier identificat (operaţie încheiată cu succes);
    CF = 1, AX = cod eroare (nu s-a identificat fişier).
  - Descriere:
    Se caută următorul fişier care verifică numele generic specificat la un apel anterior al funcţiei Find First. La offset-ul 1EH în zona DTA se găseşte numele acestui fişier; este necesară obţinerea adresei zonei DTA în prealabil. În general, această funcţie se apelează ciclic, până când se obţine CF = 1 (nu mai sunt fişiere care să verifice numele generice).

Funcţiile de duplicare şi de redirectare sunt utilizate de interpretorul de comenzi, pentru redirectarea din linia de comandă a intrării şi ieşirii standard. Redirectări asemănătoare pot fi făcute şi din programele utilizator. Să considerăm următorul program, care execută următoarele acţiuni:

- scrie un mesaj la ieşirea standard;
- creează un fişier disc;
- duplică handlerul 1 (ieşirea standard);
- redirectează ieşirea standard către fişierul anterior creat;
- scrie un mesaj la ieşirea standard (scrierea se va face de fapt în fişier);
- redirectează ieşirea standard la handlerul duplicat (revenire la situaţia iniţială);
- scrie un mesaj la ieşirea standard.

Programul est elistat în continuare:

```
        .model large
        include io.h
        .data
                nume        db      'fis.dat', 0
                h1          dw      ?
                h2          dw      ?
                mes_1       db      'Acest mesaj se va tipari la consola', cr, lf
                lng_1       dw      $ - mes_1
                mes_2       db      'Acest mesaj se va tipari in fis.dat', cr, lf
                lng_2       dw      $ - mes_2
                mes_3       db      'Acest mesaj se va tipari din nou la consola', cr, lf
                lng_3       dw      $ - mes_3
         .stack 1024
        .code
        start:
                init_ds_es
        ;
                mov     ah, 40H                     ; Scriere la
                mov     bx, 1                       ; standard output
                lea     dx, mes_1
                mov     cx, lng_1
                int     21H
        ;
                mov     ah, 45H                     ; Duplicare
                mov     bx, 1                       ; standard output
                int     21H
                mov     h2, ax                      ; Handler duplicat in h2
        ;
                mov     ah, 3CH                     ; Deschidere 'fis.dat'
                mov     cx, 0                       ; pentru scriere
                lea     dx, nume
                int     21H
                mov     h1, ax                      ; Handler in h1
;
                mov     ah, 46H                     ; Redirecteaza
                mov     cx, 1                       ; standard output
                mov     bx, h1                      ; la h1, adica la 'fis.dat'
                int     21H
;
                mov     ah, 40H                     ; Scriere la
                mov     bx, 1                       ; standard output redirectat
                lea     dx, mes_2
                mov     cx, lng_2
```

```
                int     21H
        ;
                mov     ah, 46H                 ; Redirecteaza
                mov     cx, 1                   ; standard output
                mov     bx, h2                  ; la h2, adica la cel original
                int     21H
        ;
                mov     ah, 40H                 ; Scriere la
                mov     bx, 1                   ; standard output original
                lea     dx, mes_3
                mov     cx, lng_3
                int     21H
        ;
                exit_dos
        end     start
```

Mesajele mes_1 şi mes_3 vor fi afişate pe ecran, iar mesajul mes_2 va fi scris în fişîeruyl fis.dat. Programul de mai sus (să presupunem că se numeşte redi.exe) se poate acum rula şi cu redirectare din linia de comandă:

```
        C:\> redi > fis_nou.dat
```

ceea ce va duce la scrierea mesajelor mes_1 şi mes_2 în fişierul fis_nou.dat. Se observă că redirectărie efectuate prin program se referă la asignarea curentă a dispozitivului respectiv, nu la asignarea implicită.

O problemă importantă la scrierea fişierelor executabile este accesul la parametrii din linia de ccomandă. La descrierea blocului PSP, am văzut că în PSP există un contor al caracterelor din linia de comandă, iar "coada" propriu-zisă a comenzii se găseşte începând de la offsetul 81H.

Să considerăm un program care acceptă în linia de comandă un nume generic de fişier (cu '*' sau '?') şi afişează la consolă toate fişierele care corespund numelui generic. Accesul la fişiere se va face prin funcţiile Find First şi Find Next, iar parametrii din linia de comandă se citesc din PSP.

Se pune problema corectitudinii numărului de parametri (cuvinte). Vom utiliza procedura count, dezvoltată în subcapitolul 8.9, care primeşte un şir de caractere şi întoarce numărul de cuvinte din acel şir. Este necesară o uşoară modificare, în sensul de a accepta ca delimitatori de cuvinte numai spaţiul şi caracterul Tab. Pentru apelul funcţiilor DOS se utilizează macroinstrucţiunile definite în fişierul io.h (descris în Anexa A).

Implementarea programului este următoarea:

```
.model          large
include io.h
        extrn count : far
.stack  512
;
;       Zonă proprie DTA
;
dta     struc
                        db      30      dup ()
        f_name          db      13      dup ()          ; Nume fişier identificat
dta     ends
;
        stdout equ      1
;
```

```
.data
        file_struc      dta     <>
        buf_name        db      128     dup (0)
        gen_name        db      80      dup (0)
        f_hand          dw      ?
        nr_bytes        dw      ?
        buf             db      4096    dup (?)
        dim_buf                 equ     $ - buf
        help_err        db      cr, lf , 'Sintaxa este: dump <file>',0
        mess_no_file    db      cr, lf, 'dump: Nu exista fisier: ',0
        open_err        db      cr, lf, 'dump: Eroare deschidere fisier ',0
        read_err        db      cr, lf, 'dump: Eroare citire fisier ',0
.code
start:
        cld
        push    ds                              ; Avem nevoie de adresa PSP
        pop     ax
        init_ds_es
        mov     si, 80H                         ; Offset contor caractere
        lea     di, buf_name
        push    ds
        mov     ds, ax
        mov     cl, [si]
        xor     ch,ch                           ; CX = număr caractere, fără CR, LF
        inc     si                              ; Primul caracter
        jcxz    help_ds                         ; Dacă CX = 0, mesaj de help
;
        push    cx
        rep     movsb                           ; Transfer din PSP in buf_name
        mov     byte ptr es:[di], 0             ; Terminator şir
        pop     cx
        pop     ds
        lea     si, buf_name
ch_nxt:
        cmp     byte ptr [si], ' '              ; Sărim peste spaţiile albe
        jz      skip
        cmp     byte ptr [si], tab              ; şi peste Tab-uri
        jz      skip
        jmp     first_ch
skip:
        inc     si                              ; De la cap la coadă
        dec     cx                              ; şi contorizăm
        jmp     ch_nxt
;
first_ch:
        lea     di, gen_name                    ; Transfer în gen_name (nume generic)
        jcxz    help                            ; Dacă acum CX = 0, nu există parametri
        rep     movsb                           ; Abia acum transfer
ch_last:
        dec     di                              ; Pe ultimul caracter
        cmp     byte ptr [di], ' '              ; Eliminăm spaţiile albe şi de la sfârşit
        jz      skip1
        cmp     byte ptr [di], tab              ; La fel cu Tab-urile
        jz      skip1
        jmp     last_char
skip1:
        dec     di                              ; De la coadă la cap
        dec     cx                              ; şi contorizăm
        jmp     ch_last
```

```
last_char:
        inc     di                      ; Am decrementat o dată în plus
        mov     byte ptr [di], 0        ; Punem terminator
        lea     si, gen_name            ; Adresă şir curăţat de spaţii albe
                                        ; iniţiale şi finale
        call    far ptr count           ; Numărăm cuvintele
        cmp     ax, 1                   ; AX = numărul de cuvinte
        je      ok                      ; Dacă e unul singur, e bine
        jmp     help                    ; Altfel, mesaj help
;
help_ds:
        pop     ds                      ; Help cu POP DS
help:
        puts    help_err                ; Mesaj de asistenţă
        exit_dos                        ; şi ieşire în DOS
;
ok:
        set_dta file_struc              ; Setăm DTA pe zona noastră
        find_first gen_name             ; Apel Find First
        jnc     rel                     ; S-a găsit ceva
        jmp     gata_1                  ; Nu s-a găsit nimic
rel:
        o_read file_struc.f_name, f_hand ; Deschide fişierul cu numele
                                        ; raportat de Find First în DTA
        jnc     no_err                  ; Test eroare
        puts    open_err                ; Mesaj de eroare
        puts    file_struc.f_name       ; Afişare nume fişier
        putsi   <cr, lf>                ; CR, LF
        jmp     next                    ; Şi trecem la următorul fişier
no_err:
        putsi   <cr, lf, 'Fisier: '>    ; Nume
        puts    file_struc.f_name       ; fişier identificat
        putsi   <cr, lf, cr ,lf>
iar:
        f_read f_hand, buf, dim_buf     ; Citire dim_buf octeţi din fişier
                                        ;  în bufferul buf
        jc      error                   ; Test eroare
        test    ax, ax                  ; Test sfârşit de fişier (s-au citit
                                        ; 0 octeţi)
        jz      next                    ; Salt la următorul fişier
        mov     nr_bytes, ax            ; Memorăm cât s-a citit de fapt
        f_write stdout, buf, nr_bytes   ; Afişăm ce sa- citit
        cmp     nr_bytes, dim_buf       ; Dacă s-au citit mai puţini octeţi
        jb      close                   ; decât am cerut, salt la închidere
        jmp     iar                     ; Allfel, reluăm bucla de citire
close:
        f_close f_hand                  ; Închidere fişier
next:
        find_next file_struc            ; Următorul fişier
        jc      gata                    ; CF = 1 : nu mai sunt
        jmp     rel
error:
        puts    read_err                ; Meaj de eroare
        puts    file_struc.f_name       ; Nume fişier
        putsi   <cr, lf>
        jmp     next                    ; Următorul
gata_1:
        puts    mess_no_file            ; Nu există nici un fişier cu numele dat
        puts    gen_name
```

```
        putsi   <cr, lf>
gata:
        exit_dos                                ; Ieşire în DOS
end start
```

Programul accesează PSP-ul, copiind "coada" comenzii într-un buffer propriu. Se sare apoi peste spaţiile albe de la începutul şirului şi de elimină şi spaţiile albe de la sfârşitul şirului (ptin mutarea terminatorului mai la stânga).

Apoi se copiază şirul "curăţat" în bufferul gen_name şi se apelează procedura count, care întoarce numărul de cuvinte din şir. Dacă numărul de parametri este diferit de 1, se afişează un mesaj de help. Altfel, se setează zona DTA pe o zonă proprie şi se apelează Find First. La un offset specificat în zona DTA, se va găsi numele primului fişier (dacă CF = 0).

Se deschide fişierul pentru citire şi se intră într-o buclă de citire-afişare, care se încheie dacă numărul de octeţi citiţi este strict mai mic decât cel cerut sau este zero.

Se apelează Find Next, cu parametrul zona proprie DTA, care întoarce numele următorului fişier identificat. Toată procedura se reia în buclă, până când Foind Next raportează (prin CF = 0), că nu mai sunt fişiere care să corespundă numelui generic specificat.