Organizarea memoriei secundare (virtuale) în PC. Funcţii de management al directoarelor şi fişierelor

Scopul lucrării

a) Studiul principalelor tipuri de memorie secundară.
b) Managementul directoarelor şi fişierelor utilizând funcţii de nivel înalt.

4.1. Tipuri de memorie secundară

Memoria principală a unui sistem de calcul trebuie să fie accesibilă direct şi cât mai rapid posibil. Totuşi, preţul ridicat al acesteia face necesară utilizarea unei memorii de mai mare capacitate (*de masă*), care să fie mai ieftină. Această memorie, numită *secundară* sau *virtuală* are dezavantajul unei viteze mai mici de acces la informa□ii decât memoria principală.

Benzile magnetice

Istoric vorbind, benzile magnetice sunt primele memorii secundare ale sistemelor de calcul. Benzile magnetice informatice sunt asemănătoare celor utilizate de către magnetofoane. Principiul de funţionare al unităţilor de bandă magnetică este deplasarea unei benzi cu viteză constantă în faţa unui cap de citire/scriere. Pentru scrierea unei informaţii pe bandă este de ajuns să se varieze curentul electric în capul magnetic, ceea ce provoacă o magnetizare locală a părţii de bandă care se găseşte sub cap.

Figura 1 arată organizarea tipică a informaţiei pe o bandă magnetică. Cel mai adesea informaţiile sunt înregistrate sub forma unor cuvinte de un caracter succesive, pe un anumit număr de piste adiacente, fiecare caracter cuprinzând în general 8 biţi plus un bit de paritate, pentru a ameliora fiabilitatea înregistrării. Densitatea tipică a înregistrării este de 1600 bpi (*byte per inch*). Aceasta corespunde înregistrării unui cuvânt pe cca. 0,03 mm. Alte densităţi utilizate sunt 800 bpi şi 6250 bpi.

După terminarea scrierii unei înregistrări fizice, adică a unui bloc de date, se lasă un *spaţiu liber de înregistrare* (un *gap*) pe bandă înainte de a se scrie un alt bloc. Acest spaţiu permite benzii să atingă şi să anuleze viteza nominală a citirii/scrierii pe durata pornirii şi respectiv a opririi defilării benzii prin faţa capului de citire. Dacă se scriu date de mică dimensiune pe bandă, se obţine un număr mare de *gap*-uri, banda fiind utilizată ineficient.

Benzile magnetice sunt memorii secundare cu *acces secvenţial*. Atunci când banda este poziţionată la început, citirea blocului de date *n* necesită citirea secvenţială prealabilă a blocurilor de la 1 la n-1. Dacă informaţia dorită este aproape de finalul benzii, programul trebuie să citească aproape toată banda înainte de obţinerea informaţiei, ceea ce poate lua câteva minute. A face un procesor care poate executa milioane de instrucţiuni pe secundă să aştepte câteva sute de secunde este o aberaţie. De aceea banda magnetică e utilizată pentru aplicaţii secvenţiale sau pentru arhivare.

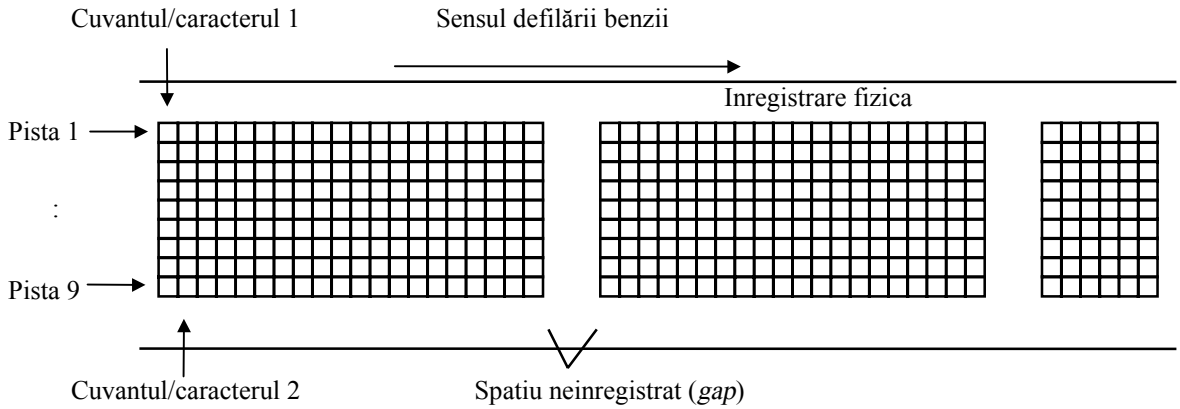

Fig. 1. Organizarea înregistrării pe o bandă magnetică

Discurile magnetice

Un disc magnetic este un platou metalic circular din material non magnetic (ex. cupru, aluminiu) cu diametru de la 15 la 30 cm, care este acoperit pe cele două fețe cu o substanță magnetizabilă (figura 2).

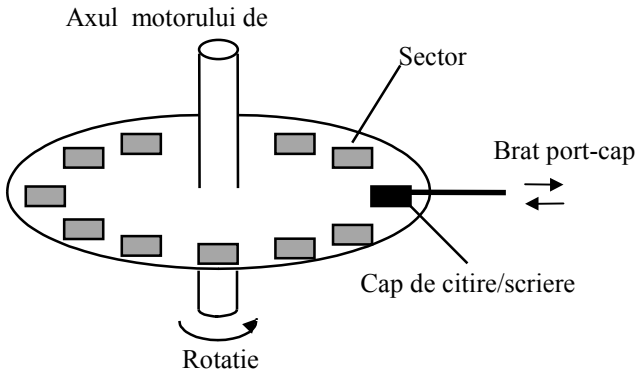

Fig. 2. Disc cu un platou

Informațiile sunt înregistrate pe un anumit număr de *piste* concentrice. Discurile au câteva zeci sau sute de piste pe față. În general, în unitățile de discuri, capetele de citire/scriere sunt plasate pe un braț mobil care se deplasează radial pe disc. Adesea, unitățile de discuri cuprind mai multe discuri dispuse vertical pe acelaşi ax de rotație. În acest caz, brațul are câte un cap pentru fiecare dintre fețele discurilor, deplasarea brațului antrenând simultan ansamblul de capete. Distanța radială a capetelor (distanța de la capete la axul de rotație) defineşte un *cilindru de date*.

O unitate de discuri dotată cu n platouri are $2n$ capete, ceea ce defineşte un cilindru cu $2n$ piste.

Pistele sunt divizate în sectoare, de ordinul zecilor pe o pistă. Un sector constituie unitatea de informație pe un disc, şi are în general capacitatea de 512 octeți. Pentru definirea unui schimb de informații cu discul, un program trebuie să precizeze: *cilindrul* şi *capul* care definesc *pista*, *numărul de sector* la care începe înregistrarea, *numărul de cuvinte* schimbate, *adresa de memorie* care primeşte sau furnizează informația.

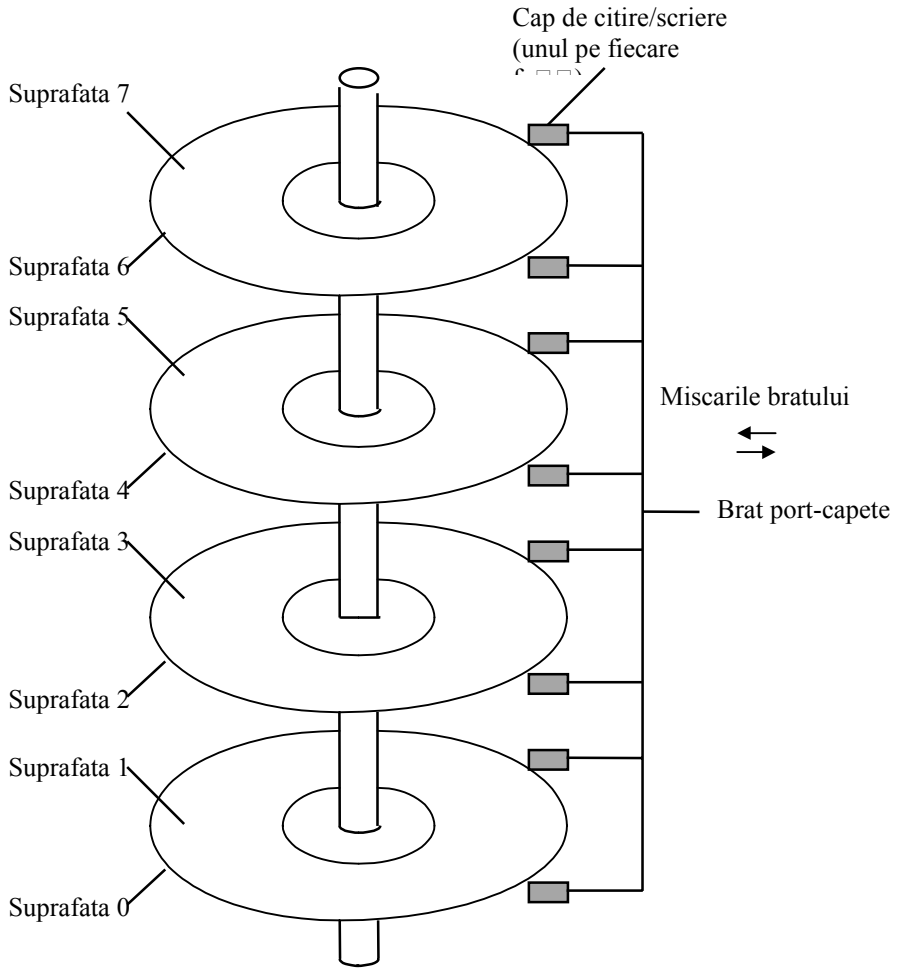

Fig. 3. Disc cu patru platouri

Transferul între disc şi memorie debutează întotdeauna cu începutul unui sector. Când se transferă informații voluminoase care necesită mai multe sectoare de pe piste adiacente ale aceluiaşi cilindru nu se pierde timp cu trecerea de la un cap de citire la altul. Dacă însă transferul necesită trecerea de la un cilindru la altul, este necesară deplasarea brațului port-capete, ceea ce conduce la pierdere de timp (de ordinul ms sau zecilor de ms). Pe de altă parte, viteza de rotație tipică este de 3600 rotații/minut.

Unitățile de disc ale calculatoarelor personale (PC) cuprind frecvent un număr de platouri plasate pe acelaşI ax de rotație (figura 3). Acestea sunt numite discuri dure (*hard-disk*-uri), datorită materialului metalic al suportului utilizat de platouri.

Pentru transportul programelor între PC-uri au fost introduse discuri mici, numite dischete sau discuri suple (*floppy-disk*-uri), datorită materialului plastic al suportului. Pentru a reduce uzura dischetelor, capetele de citire/scriere se retrag şi rotația discului este întreruptă pe durata inactivității, ceea ce duce la întârzieri în accesul la informații. Dimensiunile actuale ale dischetelor sunt de 5,25” (din ce în ce mai rar utilizată) şi 3,5”. Fiecare tip are versiuni cu densitate de înregistrare normală şi mărită. Parametrii acestora sunt prezentați în tabelul 1.

Tab. 1. Caracteristici specifice pentru 4 tipuri de dischete

Dimensiune (inch)	5,25	5,25	3,5	3,5
Capacitate (octeti)	360 k	1,2 M	720 k	1,44 M
Numar de piste	40	80	80	80
Sectoare/pista	9	15	9	18
Numar de capete	2	2	2	2
Viteza de rotatie (rot./min.)	300	360	300	300
Debitul (kb/s)	250	500	250	500
Tipul suportului	suplu	suplu	rigid	rigid

Pentru initializarea discurilor este necesara mai intai o *formatare la nivel fizic*, de nivel jos, care pregateste discul pentru formatarea propriu-zisa.

Formatarea de nivel inalt imparte discul in piste si sectoare. Sistemul de operare DOS utilizeaza comanda FORMAT pentru aceasta operatie. Exista programe utilitare de management al fisierelor si directoarelor pe disc care pot efectua aceasta operatie (Norton Commander, Windows File Manager, etc.).

In cazul discurilor hard, formatarea de nivel inalt poate fi urmata de *partitionarea discului fizic* in mai multe *discuri virtuale*. Sistemul de operare DOS utilizeaza comanda FDISK pentru aceasta operatie.

4.2. Managementul directoarelor şi fişierelor

Spatiul pe disc al PC-urilor este alocat fisierelor in unitati de alocare numite *cluster*-e. Dimensiunea cluster-elor variaza in functie de tipul discului, formatul si dimensiunea lui. Floppy-disk-urile au in general dimensiunea clusterelor de 1k octeti, pe cand dimensiunea clusterelor hard-disk-urilor poate varia intre 2k si 16k, depinzand de capacitatea discului. Pe un hard-disk pot exista zeci de mii de cluster-e. Comanda FORMAT determina aceasta dimensiune.

Cluster-ele sunt numerotate si un fisier pe disc este format din unul sau mai multe astfel de unitati de alocare. In *tabela de alocare a fisierului* (FAT = *File Allocation Table*) se pastreaza numerele si ordinea clusterelor apartinand unui fisier, data si momentul crearii sale.

Fragmentarea este consecinta naturala a crearii si stergerii fisierelor. Deoarece un fisier este format dintr-unul sau mai multe cluster-e a caror ordine este cunoscuta, un fisier poate fi plasat oriunde pe disc. Un procesor de texte poate de exemplu sa resalveze un fisier modificat scriind in alta parte pe disc, eliberand apoi spatiul utilizat initial pentru alte fisiere.

Pentru a citi din fisier sau scrie in fisier, sistemul de operare DOS urmeaza pur si simplu lista curenta a numerelor de cluster-e din FAT. Nu conteaza cate parti sunt si nici de cat de dispersate sunt pe disc. Sarind pe toata suprafata discului pentru a citi sau scrie biti se reduce viteza de acces. Daca partile unui fisier nu sunt plasate alaturat, fisierul este numit *fragmentat*, iar operatiile de refacere in caz de distrugere, stergere sau pierdere sunt ingreunate.

Utilitarul TMAP prezinta nivelul fregmentarii tuturor fisierelor de pe disk.

4.3. Exemple de funcţii C pentru managementul directoarelor şi fişierelor

getfat : Obtinerea informatiilor FAT

Declaratie: void getfat(unsigned char drive, struct fatinfo *dtable);

Parametru *Semnificatie*

drive specifica drive-ul ale carui informatii sunt obtinute (0 = implicit, 1 = A, 2 = B, etc.).
dtable adresa structurii fatinfo

```
struct fatinfo {
  char  fi_sclus;        /* sectoare pe cluster */
  char  fi_fatid;        /* identificatorul FAT */
  int   fi_nclus;                /* numarul de cluster */
  int   fi_bysec;        /* octeti pe sector */
};
```

_dos_getdiskfree : Obtinerea spatiului liber pe disc

Declaratie: unsigned _dos_getdiskfree(unsigned char drive, struct diskfree_t *dtable);

Parametru *Semnificatie*

drive specifica drive-ul ale carui informatii sunt obtinute (0 = implicit, 1 = A, 2 = B, etc.).
dtable adresa structurii dfree

```
struct dfree {
  unsigned df_avail;  /* clustere disponibile */
  unsigned df_total;  /* total clustere */
  unsigned df_bsec;   /* octeti pe sector */
  unsigned df_sclus;  /* sectoare pe cluster */
};
```

getdisk : Obtinerea numarului driveului curent

setdisk : Setarea numarului driveului curent

Declaratii: int getdisk(void);
int setdisk(int drive);

Parametru *Semnificatie*
drive specifica drive-ul setat (0 = implicit, 1 = A, 2 = B, etc.).

mkdir : creaza un director

rmdir : sterge un director de fisiere DOS

Declaratii: int mkdir(const char *path);
int rmdir(const char *path);

Parametru *Semnificatie*
path sir de caractere specificand calea

Obs: Calea pentru *rmdir* trebuie sa nu fie directorul curent sau directorul radacina, iar directorul trebuie sa fie vid.

Valoare returnata: mkdir returneaza 0 daca directorul a fost creat
rmdir returneaza 0 daca directorul a fost sters
mkdir si rmdir returneaza -1 si seteaza *errno* astfel:
EACCES = operatie nepermisa
ENOENT = cale sau fisier inexistent

getcurdir : Obtine directorul curent pentru un drive specificat

Declaratie: int getcurdir(int drive, char *directory);

Parametru *Semnificatie*

drive specifica drive-ul setat (0 = implicit, 1 = A, 2 = B, etc.).
directory adresa unei zone de memorie (de lungime MAXDIR) in care va fi plasat un nume
de director terminat cu null.

Valoare returnata: in caz de suces, returneaza 0
in caz de suces, returneaza -1

chdir : Schimba directorul curent

Declaratie: int chdir(const char *path);

Parametru *Semnificatie*

path sir de caractere specificand calea noua

Valoare returnata: in caz de suces, returneaza 0
in caz de suces, returneaza -1 si seteaza *errno* astfel:
ENOENT = cale sau nume de fisier inexistente

_dos_creat : Creeaza fisier sau suprascrie unul existent (utilizand functia DOS 0x3C)

Declaratie: unsigned _dos_creat(const char *path, int attrib, int *handlep);

Parametru	*Semnificatie*
attrib	permisiune de acces (atribut DOS), una din constantele: FA_NORMAL fisier normal FA_RDONLY fisier read-only FA_HIDDEN fisier ascuns FA_SYSTEM fisier sistem
handlep	adresa identificatorului logic al fisierului (*handle*)
path	calea si numele fisierului (calea implicita este directorul curent)

_dos_write : Scrie intr-un fisier (utilizand functia DOS 0x40)

Declaratie: unsigned _dos_write(int handle, void far *buf, unsigned len, unsigned *nwritten);

Parametru	*Semnificatie*
handle	identificatorului logic al fisierului
nwritten	adresa numarului de biti curent scrisi
buf	adresa buffer-ului din care functia scrie octetii
len	numarul de octeti pe care functia se asteapta ii scrie

_dos_close : Inchide un fisier asociat cu un identificator logic de fisier (utilizand functia DOS 0x3E)

Declaratie: unsigned _dos_close(int handle);

_dos_open : Deschide un fisier pentru citire sau scriere (utilizand functia DOS 0x3D)

Declaratie:
```
#include <fcntl.h>
#include <share.h>
#include <dos.h>
unsigned _dos_open(const char *filename, unsigned oflags, int *handlep);
```

Parametru	*Semnificatie*
filename	adresa numelui fisierului deschis
oflags	modul de deschidere, constante simbolice definite in FCNTL.H si tipul deschiderii, constante simbolice definite in SHARE.H
handlep	adresa identificatorului logic al fisierului (*handle*)

_dos_read : Scrie intr-un fisier (utilizand functia DOS 0x3F)

Declaratie: unsigned _dos_read(int handle, void far *buf, unsigned len, unsigned *nread);

Parametru	*Semnificatie*
handle	identificatorului logic al fisierului
nread	adresa numarului de biti curent cititi

buf	adresa buffer-ului in care functia citeste octetii
len	numarul de octeti pe care functia se asteapta ii citeasca

4.4. Exemple de programe

```
{FAT1.C - Obtinerea spatiului total pe disc}

#include <stdio.h>
#include <conio.h>
#include <dos.h>

void main(void)
{
struct fatinfo diskinfo;
unsigned int u;         //unitatea de disc dorita
unsigned int nc;        //nr de unitati de alocare (cluster)
unsigned int ns;        //nr de sectoare pe cluster
unsigned int no;        //nr de octeti pe sector
long unsigned int dim;        //dimensiune disc

clrscr();
printf("Obtinerea informatiilor FAT\n");
printf("Unitate de disc : (0-implicita,1-A:,2-B:3-C:....) : ");
scanf("%d",&u);
getfat(u, &diskinfo);
printf("\n");
nc=diskinfo.fi_nclus;
ns=diskinfo.fi_sclus;
no=diskinfo.fi_bysec;

printf("Numar de unitati de alocare (clustere) :  %5u\n",nc);
printf("Numar de sectoare pe unitatea de alocare: %5u\n",ns);
printf("Numar de octeti pe sector :               %5u\n",no);
dim = (long) nc*ns*no;
printf("\nNumar de octeti pe unitatea ");
if (u!=0) printf("%c:",0x40+u);else printf("C:");
printf( "%lu\n",dim);
printf("Identificator FAT : %x\n",diskinfo.fi_fatid);
getch();
}

{FAT2.C - Obtinerea spatiului disponibil pe disc}

#include <stdio.h>
#include <conio.h>
#include <dos.h>

void main(void)
{
struct diskfree_t free;
unsigned int u;               //unitatea de disc dorita
unsigned int nt;        //nr total de clustere
unsigned int nd;        //nr diponibil de clustere
unsigned int ns;        //nr de sectoare pe cluster
unsigned int no;        //nr de octeti pe sector
long unsigned int dim;        //dimensiune totala disc
long unsigned int avail;      //dimensiune disc liber

clrscr();
printf("Obtinerea spatiului liber pe disc\n");
printf("Unitate de disc : (0-implicita,1-A:,2-B:3-C:....) : ");
scanf("%d",&u);
if (_dos_getdiskfree(u, &free) != 0) {
```

```
printf("Eroare la apelul _dos_getdiskfree()\n");
exit(1);
  }

printf("\n");
nt=free.total_clusters;
nd=free.avail_clusters;
ns=free.bytes_per_sector;
no=free.sectors_per_cluster;

printf("Numar total de clustere :             %5u\n",nt);
printf("Numar de clustere libere :            %5u\n",nd);
printf("Numar de sectoare libere pe cluster : %5u\n",ns);
printf("Numar de octeti liberi pe sector :    %5u\n",no);
dim = (long) nt*ns*no;
avail = (long) nd*ns*no;

printf("\nNumar total de octeti pe unitatea ");
if (u!=0) printf("%c:",0x40+u);else printf(" C : ");
printf( "%10lu\n",dim);
printf("\nNumar de octeti liberi pe unitatea ");
if (u!=0) printf("%c:",0x40+u);else printf("C : ");
printf( "%10lu\n",avail);
getch();
}

{FAT3.C - Obtinerea tipului dischetei}

#include <stdio.h>
#include <dos.h>

void main(void)
{
struct fatinfo diskinfo;
int flag = 0;

printf("Introduceti un disc in drive-ul A\n");
getchar();

getfat(1, &diskinfo);
/* obtine informatiile discului */
printf("\nDrive-ul A este: ");
switch((unsigned char) diskinfo.fi_fatid)
   {
case 0xF9:
printf("720K - Double Density\n");
break;

case 0xF0:
printf("1440K - High Density\n");
break;

default:
printf("Neformatat\n");
flag = 1;
   }

if (!flag)
   {
printf(" Numar de clustere   %5d\n", diskinfo.fi_nclus);
printf(" Sectoare pe cluster %5d\n", diskinfo.fi_sclus);
printf(" Octeti pe sector    %5d\n", diskinfo.fi_bysec);
   }

getch();
}

{DRIVE.C - Determinarea drive-urilor logice disponibile}
```

```
#include <stdio.h>
#include <conio.h>
#include <dir.h>

void main(void)
{
   int save, disk, disks;

   clrscr();
   /* Salvarea drive-ului original */
   save = getdisk();

   /* Afisarea numarului de drive-urilor logice */
   disks = setdisk(save);
   printf("Sistemul are %d drive-uri logice\n\n", disks);

   /* Afisarea drive-urilor disponibile */
   printf("Drive-uri disponibile:\n");
   for (disk = 0;disk < 26;++disk)
   {
      setdisk(disk);
      if (disk == getdisk())
       printf("  drive-ul %c este disponibil\n", disk + 'A');
   }
   setdisk(save);
   getch();
}

{DIR1.C - Crearea si stergerea un director}

#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <dir.h>
#define DIRNAME "a:\\testdir.$$$"

void main(void)
{
   int stat;

   stat = mkdir(DIRNAME);
   if (!stat)
        printf("Director creat\n");
   else
   {
      printf("Nu se poate crea directorul\n");
      exit(1);
   }
   getch();
   system("dir/p a:");
   getch();
   stat = rmdir(DIRNAME);
   if (!stat)
        printf("\nDirector sters\n");
   else
   {
        perror("\nNu se poate sterge directorul\n");
      exit(1);
   }
   getch();
}

{DIR2.C - Schimbarea directorului}

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
```

```
#include <dir.h>

char old_dir[MAXDIR];
char new_dir[MAXDIR];

void main(void)
{
   clrscr();
   if (getcurdir(0, old_dir))
   {
      perror("getcurdir()");
      exit(1);
   }
   printf("Directorul curent este: \\%s\n", old_dir);
   system("dir");
   getch();

   if (chdir("\\"))
   {
      perror("chdir()");
      exit(1);
   }

   if (getcurdir(0, new_dir))
   {
      perror("getcurdir()");
      exit(1);
   }
   printf("\nDirectorul curent este acum: \\%s\n", new_dir);
   system("dir");
   getch();

   printf("\nRevenire la directorul original: \\%s\n", old_dir);
   if (chdir(old_dir))
   {
      perror("chdir()");
      exit(1);
   }
   system("dir");
   getch();
}


{FILE1.C - Crearea unui fisier si scrierea in el}

#include <dos.h>
#include <string.h>
#include <stdio.h>

#define n 100
int main(void)
{
  unsigned count;
  int handle;                // identificator de fisier
  char buf[n];                 // buffer fisier
  char nume[8];              // nume fisier
  char nume0[n], *nume1="a:\\";

  /* citeste numele fisierului */
  clrscr();
  printf("Introduceti numele fisierului: ");
  strcpy(nume,nume1);
  scanf("%s",nume0);
  strcat(nume,nume0);

    /* creaza fisierul */
    if (_dos_creat(nume, _A_NORMAL, &handle) != 0)  // fisier normal
  {
```

```
    perror("Nu se poate crea fisierul !");
    return 1;
  }

  /* citeste datele */
  clrscr();
  printf("Introduceti datele: ");
  scanf("%s",buf);

  /* scrie in fisier */
  if (_dos_write(handle, buf, strlen(buf), &count) != 0)
  {
    perror("Nu se poate scrie in fisier !");
    return 1;
  }

  /* inchide fisierul */
  _dos_close(handle);
  return 0;


}

{FILE2.C - Deschiderea unui fisier si citirea lungimii lui}

#include <dos.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>

#define n 100
int main(void)
{
  unsigned count;
  int handle;                 // identificator de fisier
  char buf[n];                  // buffer fisier
  char nume[n];               // nume fisier
  char nume0[n], *nume1="a:\\";


  /* citeste numele fisierului */
  clrscr();
  printf("Introduceti numele fisierului: ");
  strcpy(nume,nume1);
  scanf("%s",nume0);
  strcat(nume,nume0);

  /* deschide fisierul */
  if (_dos_open(nume, O_RDWR, &handle) != 0) {
    perror("Nu se poate deschide fisierul !");
    return 1;
  }

  /* citeste din fisier */
  if (_dos_read(handle, buf, 10, &count) != 0) {
    perror("Nu se poate citi din fisier");
    return 1;
  }
  else
    printf("_dos_read: %d octeti cititi\n",count);
  return 0;

}
```

4.5. Desfăşurarea lucrării

a) Să se studieze tipurile de memorie secundara si functiile de management al directoarelor si fisierelor.

b) Să se studieze exemplele de la punctul 4.4. Să se verifice corectitudinea funcționării programelor.

c) Să se ruleze utilitarele *Coretest*, *Dmi*, *Tmap* si *NDD*.

Tema 1: Sa se scrie un program care sa creeze un subdirector al directorului curent, sa schimbe directorul curent cu cel creat, sa creeze un fisier, sa scrie in fisier si sa il inchida.

Tema 2: Sa se scrie un program care sa schimbe directorul curent cu un subdirector dat, sa deschida un fisier din acest director, sa citeasca lungimea fisierului si sa il inchida.