CAPITOLUL 12

12. Facilități de nivel scăzut ale limbajului C

Rutinele utilizate de sistemul de operare MS-DOS pentru tratarea operațiilor de intrare sau ieşire şi gestiunea resurselor sistemului pot fi apelate şi dintr-un program scris intr-un limbaj de nivel înalt cum este C. Limbajul permite şi legarea cu module scrise in limbaj de asamblare.

Rutinele DOS sunt apelate prin întreruperi software. Întreruperea 21h (INT 21h) este serviciul pentru apelul funcțiilor DOS. Pentru apelul unei funcții aceste registre (nu toate) se pregătesc cu diverse valori:

AH conține numărul funcției
AL conține numărul subfuncției (dacă este cazul)

Celelalte registre se încarcă diferit de la caz la caz. Se generează, apoi, întreruperea INT 21H.

Similar cu funcțiile DOS se pot realiza şi apeluri de funcții BIOS pregătind registrele intr-un mod convenabil. Sunt astfel accesibile majoritatea facilităților pe care programatorul le avea numai la nivelul limbajului de asamblare.

Vom descrie in continuare câteva din funcțiile cuprinse in biblioteca C ale căror prototipuri se afla in fisierele dos.h, bios.h, setjmp.h, process.h.

12.1. Funcții pentru acces direct la memorie

```
unsigned FP_OFF(farpointer);
unsigned FP_SEG(farpointer);
void far *MK_FP(unsigned seg, unsigned ofs);
```

Aceste trei macrouri sunt folosite pentru a returna:

-offsetul unui pointer far;
-segmentul unui pointer far;
-un pointer far cu segmentul seg si ofsetul ofs.

Exemplu:

```
#include <dos.h>
#include <stdio.h>
    void main(void)
        {
         char far *ptr;
         unsigned seg,off;

         ptr=MK_FP(0x8000,0);
         seg=FP_SEG(ptr);
```

```
    off=FP_OFF(ptr);
    printf("farptr=%Fp, segment=%04x, offset=%04x\n",
                                              ptr,seg,off);
}
```

```
int peek(unsigned segment,unsigned offset);
char peekb(unsigned segment,unsigned offset);
void poke(unsigned segment,unsigned offset,int value);
void pokeb(unsigned segment,unsigned offset,char value);
```

Cele patru funcții de mai sus realizează citirea (peek, peekb) respectiv scrierea (poke, pokeb) unui cuvânt sau caracter de la sau la adresa de memorie segment:offset.

12.2. Funcții pentru accesul la porturile de intrare sau ieşire.

```
int inp(int portid);
int outp(int portid, int value);
int inport(int portid);
unsigned char inportb(int portid);
void outport(int portid,int value);
void outportb(int portid,unsigned char value);
```

Toate cele şase funcții de mai sus au drept scop citirea (inport) respectiv scrierea (outport) a unui cuvânt, sau caracter (inp, inportb, outp, outportb) de la, sau la portul de intrare sau ieşire specificat prin variabila portid.

12.3. Funcții pentru gestiunea intervalelor de timp

```
void delay(unsigned miliseconds);
```

Funcția suspendă execuția programului în intervalul specificat (în milisecunde).

```
void sleep(unsigned seconds);
```

Funcția suspendă execuția programului curent pe durata seconds secunde.

12.4. Functii pentru controlul sistemului de întreruperi

```
void disable(void);
```

Această funcție invalidează întreruperile, lăsând validă numai întreruperea nemascabilă NMI. Este echivalentă cu instrucțiunea CLI din limbajul de asamblare.

```
void enable(void);
```

Această funcție validează întreruperile oferind programatorului controlul flexibil al întreruperilor hardware. Este echivalentă cu instrucțiunea STI din limbajul de asamblare.

12.5. Funcții pentru alocarea memoriei (prin DOS)

*int allocmem(unsigned size, unsigned *segp);*

Funcția apelează DOS 0x48 pentru a aloca o zonă de memorie liberă de dimensiune *size**16 octeți. Ea returnează adresa de segment a blocului alocat în locația pointată de *segp*. În caz de succes returnează -1. În caz contrar se setează _doserrno si variabila globală errno.

int setblock(unsigned segx, unsigned newsize);

Acestă funcție modifică dimensiunea alocată unui bloc de memorie, de la adresa de segment *segx* rămânând alocați doar *newsize**16 octeți. În caz de succes funcția returnează -1. Funcția face apelul DOS 0x4A.

int freemem(unsigned segx);
Funcția eliberează un bloc de memorie alocat cu allocmem. Parametrul segx este adresa de segment returnată de aceasta din urmă. În caz de succes funcția returnează -1. Funcția face apelul DOS 0x44.

12.6. Întreruperi software. Accesul la registrele procesorului.

void geninterrupt(int intr_num);

Acest macrou generează întreruperea software intr_num. Starea registrelor după apel depinde de întreruperea apelată. Accesul la registrele procesorului se face prin variabilele pseudoregistru prezentate mai jos:

_AX, _BX, _CX, _DX, _SI, _DI, _BP, _SP, _DS, _CS, _ES, _SS, _FLAGS

care le copie starea. Aceste variabile trebuie folosite cu precauție. Se recomandă a se evita alterarea variabilelor _DS, _SS, _CS si _SP. În orice caz, se va urmări codul generat şi se va depana cu atenție programul.

*int int86(int intno, union REGS *inregs, union REGS *outregs);*

Aceasta funcție execută întreruperea software specificată în intno. Înaintea execuției întreruperii, se copie în registre valorile specificate în reuniunea pointată de inregs. După execuție, valorile registrelor sunt copiate în outregs; de asemenea, şi starea bitului Carry în x.cflag (1=de obicei indică o eroare). De notat că outregs poate pointa pe aceeaşi structura ca inregs. Funcția returnează valoarea din AX la terminarea întreruperii. Reuniunea REGS este definită astfel:

```
struct WORDREGS {
    unsigned int ax,bx,cx,dx,si,di,cflag,flags;};
struct BYTEREGS {
    unsigned char al,ah,bl,bh,cl,ch,dl,dh;};


union REGS {
    struct WORDREGS x;
    struct BYTEREGS h; };
```

*int int86x(int intno, union REGS *inregs, union REGS *outregs, struct SREGS *segregs);*

Aceasta funcție este identică ca funcțiune cu cea anterioară, permițând în plus folosirea registrelor de segment pentru transmiterea de parametri (salvează si restaurează automat ES si DS). Returnează aceleaşi valori ca int86. Structura SREGS este definită astfel:

```
struct SREGS {
    unsigned int es;
    unsigned int cs;
    unsigned int ss;
    unsigned int ds;};
```

*void intr(int intno, struct REGPACK *preg);*

Funcția realizează o alternativă pentru interfațarea cu întreruperile software, numărul întreruperii fiind specificat în variabila intno. Structura REGPACK este definită astfel:

```
struct REGPACK {
    unsigned r_ax,r_bx,r_cx,r_dx;
    unsigned r_bp,r_si,r_di,r_ds,r_es,r_flag;};
```

*void segread(struct SREGS *segp);*

Funcția returnează în structura pointata de segp valorile registrelor de segment.

12.7. Funcții pentru citirea sau modificarea vectorilor de întrerupere

*void interrupt (*getvect(int interruptno))();*

Fiecare procesor din familia 80x86 include un set de vectori de întrerupere, numerotați de la 0 la 255. Valoarea pe 4 octeți din fiecare vector este o adresă reprezentând adresa rutinei de tratarea a întreruperii. getvect citeşte valoarea vectorului de întrerupere specificat de interruptno şi returnează această valoare ca un pointer far la o rutină de tratare a întreruperii.

*void setvect(int interruptno, void interrupt (*isr)());*

Această funcție setează valoarea vectorului de întrerupere specificat prin interruptno cu o valoare nouă, isr, care este un pointer far la adresa altei funcții de tratare a întreruperii. Obligatoriu aceasta nouă funcție trebuie sa fie declarată de tip interrupt (cu modificatorul interrupt). Exemplu:

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

void interrupt (*oldvect)();
void interrupt get_out();
void capture_prtscr(void interrupt (*func)());

int looping=1;

void main(void)
   {
    puts("Pentru a termina apasati <Shift><PrtSc>");
    capture_prtscr(get_out);
    while(looping);
    puts("Succes");
   }

/*noua rutina de tratare intrerupere*/
 void interrupt get_out()
   {
    setvect(5,oldfunc);
    looping=0;
   }

void capture_prtscr(void interrupt (*func)())
   {
      oldfunc=getvect(5);
      setvect(5,func);
   }
```

12.8. Apeluri de funcții DOS.

int bdos(int dosfun, unsigned dosdx, unsigned dosal);

Furnizează accesul direct la multe funcții DOS. Ca argumente:

dosfun = numărul funcției DOS;
dosdx = valoarea registrului DX;
dosal = valoarea registrului AL.

Funcția returnează valoarea registrului AX. Exemplu:

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

char current_drive(void);

main()
   {
    printf("DRIVERUL %c\n",current_drive());
    getch();
   }

char current_drive(void)
   {
    char drive;
    drive = bdos(0x19,0,0);
    return ('A'+drive);
   }
```

*int bdosptr(int dosfun, void *argument, unsigned dosal);*

Funcția furnizează accesul la funcțiile DOS prin argumentele sale:

dosfun = numărul funcției DOS;
argument = pointer cerut de funcție in registrul DS:DX;
dosal = valoarea registrului AL.

Returnează aceleaşi valori ca bdos.

*int intdos(union REGS *inregs, union REGS *outregs);*

Funcția apelează întreruperea INT 21h pentru a realiza funcția sistem specifică în registrul AH (inregs->h.ah). Structura outregs este actualizată setându-se in caz de eroare indicatorul carry.

*int intdosx(union REGS *inregs, union REGS *outregs,*
*struct SREGS *segregs);*

Este similară funcției intdos, permițând în plus şi transmiterea parametrilor prin registrele de segment (DS,ES). Aceste registre sunt restaurate la terminarea apelului. Exemplu:

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

```
int delete_file(char far *filename);

main()
    {
     int err;
     err=delete_file("Temp1.$$$");
     printf("S-a sters fisierul Temp1.$$$:%s\n",(!err)?"DA":"NU");
    }

int delete_file(char far *filename)
    {
    union REGS regs;
    struct SREGS sregs;
    int ret;

    regs.h.ah=0x41;     /*delete file*/
    regs.x.dx=FP_OFF(filename);
    sregs.ds=FP_SEG(filename);
    ret=intdos(&regs,&regs,&sregs);
    return(regs.x.cflag?ret:0);
   }
```

12.9. Funcții pentru terminarea sau lansarea programelor

void exec(int code);

Funcția determină terminarea normală a programului şi furnizarea unui cod de retur (code) la ieşire. Acest cod poate fi interpretat ulterior în fişierele de comenzi. Este echivalentă cu funcția DOS 0x4C.

void abort(void);

Provoacă terminarea anormală a programului, cu afişarea unui mesaj corespunzător pe consola sistemului.

void keep(unsigned char status,unsigned size);

Apelul acestei funcții permite terminarea programului curent cu transmiterea stării procesului părinte în variabila status, precum şi setarea lui ca program TSR (Terminate and Stay Resident) de dimensiune size paragrafe. Restul memoriei este eliberat. Funcția este similară apelului DOS 0x31.

*int system(const char *cmd);*

Lansează COMMAND.COM şi execută o comandă internă, sau o comandă externă, sau un fişier de comenzi. Şirul de caractere, folosit ca parametru, conține numele complet al comenzii şi toate argumentele necesare execuției.

unsigned getpsp(void);

Funcția returnează segmentul prefix (Program Segment Prefix) al programului curent. Ea realizează functia DOS 0x62. În locul acestei funcții însă se poate folosi direct valoarea variabilei globale _psp.

int bioskey(int cmd);

Pot fi executate serviciile specifice INT 16h. Cu ajutorul acestei funcții se pot citi codurile de scanare, codurile extinse ale tastelor, se pot citi caractere fără extragere din bufferul tastaturii, sau se poate afla starea tastelor speciale (SHIFT, CTRL, ALT):

- cmd=0 citeşte codul de scanare si scoate caracterul din bufferul tastaturii; dacă nu mai sunt caractere se aşteaptă apăsarea unei noi taste.
- cmd=1 citeşte codul de scanare fără a scoate caracterul din bufferul tastaturii; dacă nu mai sunt caractere funcția returnează 0.
- cmd=2 citeşte "BIOS Shift State" (starea tastelor speciale SHIFT, CTRL, ALT).

int biosprint(int cmd, int abyte, int lpt_port);

Reuneşte toate funcțiile specifice INT 17h (accesul la portul paralel LPTxx). Funcția permite citirea stării, trimiterea unui caracter (abyte), sau inițializarea imprimantei:
- cmd=0 trimite caracterul abyte la portul paralel cu numărul dat de lpt_port (0=LPT1, 1=LPT2, etc).
- cmd=1 inițializează imprimanta, conectată pe lpt_port, cu valorile din abyte.
- cmd=2 citeşte starea imprimantei conectate pe lpt_port.

int bioscom(int cmd, int abyte, int com_port);

Reuneşte toate funcțiile specifice INT 14h (accesul la portul serial COMxx). Funcția permite citirea stării, trimiterea unui caracter (abyte) pe comunicație, citirea unui caracter de pe comunicație, sau setarea unor controale şi parametri de funcționare:

- cmd=0 inițializarea portului de comunicație com_port (0=COM1, 1=COM2) cu parametrii dați de abyte.
- cmd=1 trimite caracterul abyte pe linia de comunicație data de com_port.
- cmd=2 citeşte caracterul recepționat pe linia de comunicație dată de com_port.
- cmd=3 citeste starea portului de comunicație com_port.

long biostime(int cmd, long newtime);

Funcția permite citirea (cmd=0), sau modificarea (cmd=1) timpului sistem, exprimat în incrementi de 55ms, de la pornirea sistemului.

12.10 Dezvoltarea de cod într-un limbaj mixt (C şi ASM)

Este specifică mediului de dezvoltare Borland C care oferă posibilitatea inserării de cod ASM în programe sursă C. Fiecare instrucțiune ASM trebuie precedată de cuvântul cheie asm sau trebuie inclusă într-un bloc de tip ASM, de forma:

```
asm {
; instrucțiuni ASM
    }
```

Acest model de dezvoltare este foarte comod, deoarece vizibilitatea obiectelor definite în C se extinde şi asupra blocurilor ASM. Astfel putem folosi nume de parametri formali ai funcțiilor etc., fără a mai scrie caracterul "_" în fața numelor respective. Se evită astfel secvențele de intrare şi revenire din funcții, care sunt realizate automat de compilatorul C.

Există restricții de utilizare a etichetelor. Mai precis, etichetele trebuie definite ca etichete C, deci în afara blocurilor ASM.

Ca şi la dezvoltarea pe module separate, programatorul trebuie să gestioneze explicit modelele de memorie, în sesnsul definirii precise a adreselor far sau near. Pentru aceasta, se utilizează modelele de memorie C şi/sau cuvintele cheie near şi far.

Pentru exemplificare să discutăm implementarea unei funcții de căutare folosind atât metoda clasică cât şi metoda cu cod de asamblare.

Funcția respectivă scrisă în C este prezentată mai jos:

```
int my_search(int *x, size_t n, int y)
    {
int i;
for(i=0;i<n;i++)
    if(y==x[i])return i;
    return –1; // nu am găsit întregul y  în tabloul x
}
```

Funcția rescrisă folosind limbaj de asamblare :

```
int my_search(int near *x, size_t n, int y)
{
    asm {
     push si
     push cx
        push dx
        mov si,x
        mov cx,n
        jcxz nu
        mov dx,x
        sub si,2
        }
```

```
et: asm {
      add si,2
      cmp [si], dx
      loopnz et
      jnz nu
          mov ax,n
          sub ax,cx
          dec ax
          jmp m_end
         }

}

nu:
      asm mov ax,-1
m_end:
     asm {
      pop dx
      pop cx
      pop si
      }
return _AX
}
```

Se observă că indicele găsit a fost calculat ca diferență dintre dimensiunea n a tabloului şi valoarea incrementată a contorului Cx la ieşirea din buclă. Revenirea din buclă se face folosind valoarea stocată în registrul AX la care am acces prin preudoregistrul _AX.

Metoda are dezavantajul scăderii portabilității la recompilarea sub alte compilatoare.

12.11 Tratarea dispozitivelor bus asincrone

Să presupunem că se dezvolta un cod pentru controlul unui dispozitiv hardware (un driver de exemplu). Acest lucru este realizat prin depunerea de diverse valori în registre hard la adrese absolute (existente fizic independent de modul virtual de manipulare a memoriei pus la dispoziție de SO) .

Să presupunem că acest dispozitiv are 2 registre, fiecare fiecare având lungimea de 16 biți la adrese de memorie crescătoare.

Metoda clasică de accesare este ca mai jos :

```
struct devregs
        {
        unsigned short csr   // control si stare
        unsigned short data  // portul de date
        }
```

```
#define  ER  0x1
#define  RDY  0x2
#define  RES  0x4

// adresa absoluta a dispozitivului

#define DEVADR  (( struct dev regs*)ox  ffff0000

// numărul de dispozitive de acest tip din sistem

#define NDEVS 4
```

// o funcție ce citeşte un octet din dispozitivul n folosind protocol de tip busy/wait.

Aşteaptă pâna când apare semnal de RDY (ready) sau ER(error); dacă nu este eroare şi un ready activ înseamnă că pot citi un octet, altfel va anula eroarea dar voi întoarce 0xffff.

```
unsigned int read_dev ( unsigned devno)
  {
   unsigned  devregs *dvp=DEVADR + devno;
   if ( devno >= NDEVS ) return (0xffff)
                    // nu există un asemenea dispozitiv în lot
   while ((  dvp -> csr& (RDY | ER )) == 0);
                    // bucla vector care aşteaptă prima una din condiții este validată
   if  dvp -> csr & ER    // a fot eroare
           {
           dvp -> csr= RES   // reset
           return 0xffff;
           }
return (dvp-> data) & 0xffff;   // nu  a fost eroare şi dispozitivul e gata să emită octetul
  }
```