ARHITECTURA SISTEMELOR DE CALCUL
LUCRAREA DE LABORATOR NR. 2

Organizarea memoriei în calculatoarele personele (PC). Funcţii de managenement al memoriei .

Scopul lucrării

a) Descrierea organizării memoriei în calculatoarele PC. Definiţii ale tipurilor de memorie.
b) Funcţii C pentru managementul memoriei. Exemple.
c) Programe rezidente în memorie. Exemple.

1. Memoria în calculatoarele PC

Primele calculatare personale puteau accesa doar 64ko de memorie. Apariţia procesoarelor mai performante (Intel 8086) a permis adresarea unui spaţiu de memorie de 1Mo.

Acest spaţiu de memorie este împarţit astfel , într-un calculator IBM PC (sau combatibil) : o zonă de 640ko - începînd de la adresa 0 - care se numeşte *memoria DOS inferioară* (*low DOS memory*) şi o zonă în continuarea primei - de 384ko - denumita *memoria DOS superioară* (*high DOS memory*) sau *memorie convenţională*.

Sistemul de operare, programe pentru controlul dispozitivelor I/O (*device handlers* sau *drivers*) , programe rezidente (ce rămîn în memorie) ocupă o parte din zona memoriei DOS inferioare.

Uzual , zona de memorie DOS inferioară este constituită din circuite RAM.

Zona de memorie DOS superioară conţine 64ko de memorie ROM (ultimii 64ko) ce reprezintă sistemul de intrare / ieşire (*Basic Input Output System - BIOS*). Se mai numeşte *UMB - Upper memory Block*.

Zona de memorie DOS superioară conţine şi dispozitive de intrare / ieşire : plăci video , plăci de reţea.

Programele de aplicaţie utilizează zona de memorie DOS inferioară (programe care nu lucrează în modul de lucru protejat al procesoarelor I80286,I80386 sau I80486).

Managementul de memorie pentru calculatoarele PC trebuie să asigure două cerinţe :

- eliberarea unei zone cît mai mari din zona de memorie DOS inferioară (pentru rularea aplicaţiilor în mod *real* sub sistemul de operare DOS)
- accesul la memoria peste limita de 1Mo (pentru a exploata avantajele procesoarelor evoluate - I80286,I80386, I80486 - ce pot adresa spaţii de adrese superioare limitei de 1Mo , atunci cînd lucrează în modul *protejat* *) cu păstrarea compatibilităţii cu sistemul de operare DOS.

---

*)

Modul *protejat* modifică radical adresarea memoriei :se utilizează o sdresă logică formată dintr-un *selector* (care determină *adresa de bază a segmentului* - pe 32 biţi - pentru I80486 , *limita segmentului* şi *dreputurile asociate segmentului* ; selectorul este un index într-un *tabel de descriptori de segmente*) şi un offset (pe 32 de biţi - pentru I80486). Adresa fizică , obţinută prin adunarea adresei de bază cu offsetul , este de 32 biţi , ceea ce permite adresarea a maxim 2^{32} octeţi = 4Go.

ARHITECTURA SISTEMELOR DE CALCUL
LUCRAREA DE LABORATOR NR. 2

1.1. Memoria expandată (Expanded memory)- EMS

Memoria expandată este memoria peste limita de 1Mo , dar care apare în interiorul spațiului de adrese asociate primului Mo. Există o metodă (specificație) ce face posibilă accesarea unei zone de memorie de adresă peste 1Mo în spațiul de adrese din interiorul primului Mo. Această specificație se nunește *Expanded Memory Specification - EMS* . Calculatorul trebuie să posede un hardware special (placă de memorie EMS) ; acest hardware dedicat face ca memoria de la adrese peste 1Mo să se "vadă " din interiorul unei ferestre de memorie din spațiul de adresare de pînă la 1Mo (figura 1).

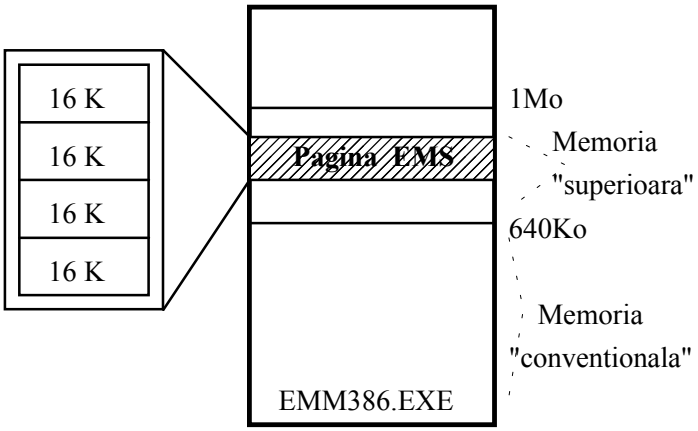

Figura 1. Memoria EMS

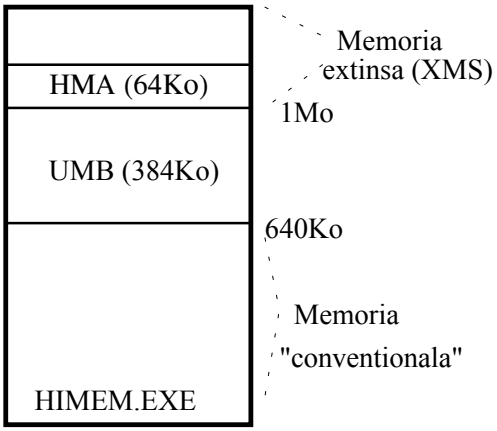

Figura 2. Memoria XMS

Un program special (manager de memorie) numit EMM386.EXE trebuie încărcat pentru a avea acces la memoria expandată. Orice procesor (I8086 - I80486) poate avea acces la memoria expandată , chiar şi în modul real de lucru.

Dezavantajul memoriei expandate este acela că utilizează un hardware complex.

1.2. Memoria extinsă (Extended memory) - XMS

Memoria extinsă este memoria peste limita de 1Mo , dar care este accesată *direct* , atunci cînd procesorul lucrează în modul de lucru protejat (nu mai este necesar un hardware specializat pentru a *mapa* memoria superioară într-un spațiu de adrese de pînă la 1Mo (figura 2).

Primii 64ko peste limita de 1Mo poartă denumirea de *High Memory Area* şi pot să fie *accesați de către procesor (I80286-I80486) în modul real.*

Un program special (manager de memorie) denumit HIMEM.SYS trebuie încărcat pentru a asigura accesul al memoria extinsă (eXtended Memory Specification - XMS).

Programul EMM386.EXE poate utiliza memoria XMS pentru a emula memoria EMS (dacă nu există placa de memorie expandată în sistem).

Memoria EMS nu trebuie confundată cu memoria XMS. Memoria EMS a fost dezvoltată pentru a creşte performanțele sistemului pentru modul real , iar memoria XMS a fost concepută pentru lucrul pe modul protejat. Memoria expandată nu ocupă o zonă specifică de adrese ci este asociată cu un mecanism special de mapare ce face ca această memorie să poată fi accesată, în

pagini, din spațiul de adrese specifice modului real (pînă la 1 Mo). Memoria extinsă apare în mod natural pentru modul de lucru protejat.

2. Funcţii C pentru managementul memoriei

malloc **<ALLOC.H, STDLIB.H>**

Declaratie: void *malloc(size_t size);

Functia malloc aloca un bloc de date de dimensiune size octeti din memoria nealocata (heap). Se permite alocarea explicita de memorie (prin specificarea directa a unei dimensiuni de memorie).

Memoria heap este utilizata pentru alocarea dinamica;tot spatiul de memorie de la sfirsitul segmentului de date pina la virful stivei programului (cu exceptia unui mic spatiu utilizat de DOS plus un spatiu de rezerva) este disponibil pentru alocare dinamica.

Spatiul de memorie alocat asfel constituie parte integranta din program si nu este recunoscut ca bloc de date DOS de catre sistemul de operare.

Valoare intoarsa

- operatie cu succes : un pointer catre blocul alocat
- operatie cu eroare : pointerul NULL (daca nu exista spatiu in heap)

Daca argumentul size=0 functia malloc intoarce NULL.

free **<STDLIB.H, ALLOC.H>**

Declaratie: void free(void *block);

Functia free elibereaza blocuri de memorie alocate cu functia malloc.

Valoare intoarsa : nici una.

allocmem **<DOS.H>**

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

Functia allocmem utilizeaza functia DOS 0x48 pentru a aloca un bloc de memorie DOS caracterizat print-un antet de 16 octeti astfel :

- octetul 0 : este 'M' sau 'Z'
- octetii 1 si 2 : reprezinta adresa de segment a blocului alocat (0000 daca blocul a fost eliberat)
- octetii 3 si 4 : reprezinta lungimea blocului in multiplii de 16 octeti
- octetii 5 - 15 : neutilizati

Semnificatia parametrilor este :

Parametru	Semnificatie
size	Numarul de grupuri de 16 biti cerute
segp	Pointer catre un segment alocat

NOTA: Functia malloc nu poate coexista cu funcția allocmem.

Valoare intoarsa

Operatie cu succes : -1
Operatie cu eroare : lungimea blocului de dimensiune maxima
(dar mai mica decit cea solicitata, si un cod de eroare DOS)

heapwalk **<ALLOC.H>**

Declaratie: int heapwalk(struct heapinfo *hi);

Este utilizată pentru afisarea nodurilor heapu-lui.
Functia heapwalk receptioneaza un pointer catre o structura de tip heapinfo:

```
struct heapinfo {
void *ptr;
unsigned int size;
int in_use;
 };
```

unde : ptr - pointer intors catre heapwalk (indica nodul in heap)
hi.size - reprezinta lungimea blocului in octeti
hi.in_use - este un flag careeste setat daca blocul este utilizat

Inainte de primul apel al functiei heapwalk trebuie ca hi.ptr=NULL.

Valoare intoarsa

Operatie cu succes : _HEAPEMPTY (= 1) heap vid;
_HEAPOK (= 2) heap corect (OK)
_HEAPEND (= 5) sfirsit heap

Operatie cu eroare : valoare <0

coreleft **<ALLOC.H>**

Declaratie: unsigned coreleft(void);

Functia coreleft intoarce dimensiunea memoriei RAM neutilizata.

3. Programe rezidente

Aceste programe se incarcă apoi se termina, dar rămîn în memoria calculatorului (se mai numesc programe *TSR - Terminate and Stay Resident*). Programele TSR sînt activate de o tastă prestabilită; de obicei aceste programe utilizează intreruperea asociată citirii tastaturii sau alte intreruperi ale sistemului. Aceste programe modifică tabela vectorilor de intrerupri astfel încît . la aparitţia unei interuperi să poată fi eventual activate (se apelează codul rămas rezident in memorie).

Programele rezidente se încheie prin apelul unei funcţii ce apelează o funcţie DOS specială (terminate and stay resident).

Pentru limbajul C aceasta este funcţia :

void **keep (***unsigned char* cod_iesire, *unsigned* dimensiune_rezervata) , cu antetul în DOS.H.

Eliminarea programelor TSR trebuie să se efectueze în două etape :

- restabilirea tabelei vectorilor de întrerupere
- eliberarea zonelor de memorie alocate programului TRS

Un exemplu de program utilitar care elimină programe TSR (fără reincarcarea sistemului de operare) este RELEASE.EXE . Acest program foloseste informaţii de la programul MARK.EXE.

Programul MARK.EXE marchează zona de memorie unde se va incarca programul TSR şi memorează starea sistemului de intreruperi (în general starea calculatorului - ce poate fi modificată de programul TSR).

Înainte de a se instala un program rezident se va da comanda MARK (acesta este el însuşi un program rezident).

Comanda RELEASE elimină programul rezident pînă la MARK inclusiv.

Exemplu : Se instalează două programe rezidente P1 şi P2 astfel :

```
MARK
P1
MARK
P2
```

Prima comandă RELEASE elimină programul P2 , iar următoarea comandă RELEASE elimină programul P2.

4. Exemple de programe pentru managementul memoriei

```
/**********************************/
// aloc1.c - aloca memorie in heap
/**********************************/

#include <stdio.h>
#include <string.h>
#include <alloc.h>
#include <process.h>

void main(void)
{
  int n;
  char *s;

  n=10*1024;

  clrscr();
  printf("Heap:\t%u\n",coreleft());

  //aloca memorie in heap

  if ((s = (char *) malloc(n)) == NULL)
  {
    printf("Memorie insuficienta\n");
    exit(1);  /* iesire din program */
  }
 printf("Memoria a fost alocata\n");
 printf("Heap:\t%u\n",coreleft());
 system("mem /m aloc1"); // apel comanda DOS
 system("mem /f");             // apel comanda DOS
 getch();
 free(s);
 printf("Memory was released\n");
 printf("Heap:\t%u\n",coreleft());
 system("mem /m aloc1");
 system("mem /f");
 getch();

}

/**************************************/
//heapw.c - afiseaza nodurile heap-ului
/**************************************/

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

```
#define NUM_PTRS  10
#define NUM_BYTES 16
int main( void )
{
  struct heapinfo hi;
  char *array[ NUM_PTRS ];
  int i;

  for( i = 0; i < NUM_PTRS; i++ )
    array[ i ] = (char *) malloc( NUM_BYTES );

  for( i = 0; i < NUM_PTRS; i += 2 )
    free( array[ i ] );

  clrscr();

  hi.ptr = NULL;
  printf( "Adresa nod   Marime   Stare \n" );
  printf( "             ------   ------\n" );
  while( heapwalk( &hi ) == _HEAPOK )
  {
  printf( "%x\:%x",FP_SEG(hi.ptr),FP_OFF(hi.ptr));
// FP_SEG - intoarce segmentul argumentului
// FP_off - intoarce offsetul argumentului
  printf( "%8u    %s\n", hi.size, hi.in_use ? "used" : "free" );
  getch();
  }
  return 0;
}

/****************************************/
//aloc2.c - aloca blocuri de memorie DOS
/****************************************/

#include <dos.h>
#include <alloc.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <process.h>

#define n 53

char str1[]="ALOCAREA SEGMENTULUI 1 *** ABCDEFGHIJKLMNOPQRSTUVXZY";
char str2[]="ALOCAREA SEGMENTULUI 2 *** abcdefghijklmnopqrstuvxzy";

char far *p;
char far *p1;

int main(void)
{
 unsigned int size, segp , segp1;
 int i,stat;

 size = 64;          /* (64 x 16) = 1024 bytes */
```

```
  stat = allocmem(size, &segp);

  if (stat == -1)
    printf("Alocare - OK\n");
  else
    printf("Alocare - Fail\n");

  p=MK_FP(segp,0); // MK_FP - creaza un pointer segp:0
  printf("Adresa segmentului alocat : %x\n",FP_SEG(p));
  printf("Offsetul segmentului alocat : %x\n",FP_OFF(p));

  for (i=0;i<n;i++,p++) {*p=str1[i];}

  system("mem /m aloc2");
  system("debug");
/* comenzi debug :          dxxxx:yy - afisare
                                        exxxx:yy - afisare cu modificare (SPACE pentru avans,
            ENTER pentru iesire
                    q            - exit              */

  stat = allocmem(size * 4,&segp1);

  if (stat == -1)
    printf("Alocare - OK\n");
  else
    printf("Alocare - Fail\n");

  p1=MK_FP(segp1,0);
  printf("Adresa segmentului alocat : %x\n",FP_SEG(p1));
  printf("Offsetul segmentului alocat : %x\n",FP_OFF(p1));

  for (i=0;i<n;i++,p1++) {*p1=str2[i];}

  system("mem /m aloc2");
  system("debug");

  freemem(segp);
  freemem(segp1);

  return 0;
}

/*******************************************************/
//tsr1.c - instaleaza / dezinstaleaza programe rezidente
/*******************************************************/

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

char key;
int cnt;

void main(void)
{
```

```
  while (key!='q')
  {
  system("mem /m vsafe");
  system("mem /m msd");
  cnt=cnt++;
  cnt=cnt%10;
  printf("%d\tTSR1***\n",cnt);;
  printf("(q - Quit, a - activate VSAFE, d - deactivate VSAFE)");
  printf("(A - activate MSD)\n");
  key=getch();
  if (key=='a') system("vsafe");
  if (key=='d') system("vsafe/u");
  if (key=='A') system("msd");

  }
}


/*******************************************/
// tsr2.c - aloca memorie si ramine rezident
/*******************************************/

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

char key;
int n,safety_space;

void main(void)
{

  while (key!='t')
  {
  printf("TSR2---\t k:0-9 - aloca k*16Ko de memorie\n");
  printf("TSR2--- t - quit\n");
  key=getch();
  if ((key>=0x30)&&(key<=0x39)) n=key-0x30;
  safety_space=n*16*1024;
  if (key=='t') {keep(0, (_SS + ((_SP+safety_space)/16) - _psp));}
  }
}

/******************************/
// tsr21.c - instaleaza pe tsr2
/******************************/

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

char key;
```

```
void main(void)
{
  while (key!='q')
  {
  printf("a- activeaza TSR2---\n");
  key=getch();
  if(key=='a') {system("tsr2");}
  system("mem /m tsr2");
  }
}

/*******************************************************/
// tsr.c - program rezident ce utilizeaza intreruperi
/*******************************************************/

#include <dos.h>
#define ATTR 0x0F00 //atribut video (alb pe negru)
/* ATTR = BFFFCCCC00000000 ;
                        B=afisare continua(0)/intermitenta(1)
                        FFF - culoare fundal
                        CCCC - culoare text
*/

/* Intreruperea de la ceasul de timp real - 50ms */
#define INTR 0x1C
/* se reduce marimea stivei si a heap-ului pentru a crea un program redus*/
extern unsigned _heaplen = 1024;
extern unsigned _stklen  = 512;

//rutina de servire a intreruperii veche
void interrupt ( *oldhandler)(void);

typedef unsigned int (far *s_arrayptr);

// noua rutina de servire a intreruperii
void interrupt handler()
{
  s_arrayptr screen[25]; //vector de pointeri catre linii
  int count,cnt;

/* Creaza un pointer catre prima locatie a memoriei video: B800:0000 */
  screen[0] = (s_arrayptr) MK_FP(0xB800,0);

/* incrementeaza count modulo 10 */
  count++;count %= 10;
/*afiseaza count */
  screen[0][79] = count + '0'+ ATTR; //[linie][coloana]

/* apel vechea rutina de intrerupere */
 oldhandler();
}

void main(void)
{

/* preia adresa de servire a intreruperii vechi*/
```

```
oldhandler = getvect(INTR);

/* instaleaza noul vector de intrerupere */
setvect(INTR, handler);

/* _psp este adresa de start a programului in memorie
   Virful stivei este sfirsitul programului
   Utilizind _SS si _SP impreuna se poate determina sfirsitul
   stivei (plus un spatiu de rezerva) :
     (_SS + ((_SP + safety space)/16) - _psp)
*/
//termina programul si ramine rezident
keep(0, (_SS + (_SP/16) - _psp));

}


/*****************************************************************/
// remove.c - elimina programe rezidente (in anumite conditii)
/*****************************************************************/

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

unsigned int seg;
char nume[20];
char command[20];

void free_block(unsigned int s)
{
unsigned char far *p;

p=MK_FP(s,0);                    //pointer catre inceputul blocului

  printf("Adresa segmentului alocat : %x\n",FP_SEG(p));
  printf("Offsetul segmentului alocat : %x\n",FP_OFF(p));

p++;                     // pointer catre al doilea element
*p=0;                     // stergere
p++;                     // pointer catre al treilea element
*p=0;                     // stergere
}

void main(void)
{
   clrscr();
   printf("ATENTIE !!!! ADRESELE DE SEGMENT ALE BLOCURILOR ALOCATE \n");
   printf("PROGRAMULUI TREBUIE INTRODUSE CORECT !!!!\n");
   printf("\nATENTIE !!!! NU INTRODUCETI ALTE ADRESELE DE SEGMENT \n");
   printf("IN AFARA CELOR SPECIFICATE  !!!!\n");
printf("\n");
   printf("ELIMINAREA PROGRAMULUI REZIDENT ESTE CORECTA NUMAI DACA \n");
   printf("ACESTA NU UTILIZEAZA INTRERUPERI \n");
```

```
  printf("\nIesirea din program cu adresa de segment : FFFF \n");

  printf("Numele programului :");
  scanf("%s",&nume);
  strcat(command,"mem /m ");
  strcat(command,nume);
  while (1)
  {
  system(command);
  printf("Adresa de segment a blocului :");
  scanf("%x",&seg);
  printf("\n");
  if(seg!=0xffff) free_block(seg); else exit(0);
  }
}
```

5. Desfăsurarea lucrării

a) Să se studieze tipurile de memorie într-un calculator PC.

b) Să se verifice functionarea comenzilor DOS: MEM.EXE (comanda MEM /?) şi MSD.EXE (utilitar ce poate afisa alocarea memoriei convenționale)

c) Să se studieze exemplele de programe pentru alocarea memoriei

d) Să se observe , cu ajutorul programului MEM , diferența dintre alocarea în heap şi alocarea de blocuri de memorie DOS.

e) Să se verifice functionarea programelor MARK.EXE şi RELEASE.EXE

Temă

Să se realizeze un program care alocă 3 blocuri de memorie DOS , fiecare de cîte 1ko, apoi eliberează unul din aceste blocuri , crează un bloc de memorie DOS de 2ko, copiază informația din cele două blocuri de memorie rămase în acest nou bloc , apoi eliberează blocurile de 1ko. (Programul realizează o compactare a spațiului de memorie). Se vor utiliza , ca exemplu , programele ALOC2.C şi REMOVE.C .