

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МОЭВМ**

**ОТЧЕТ**  
**по практической работе № 3**  
**по дисциплине «Операционные системы»**  
**Тема: Исследование организации управления основной памятью**

Студент гр. 8383

Степанов В.Д.

Преподаватель

Губкин А.Ф.

Санкт-Петербург

2020

### **Цель работы.**

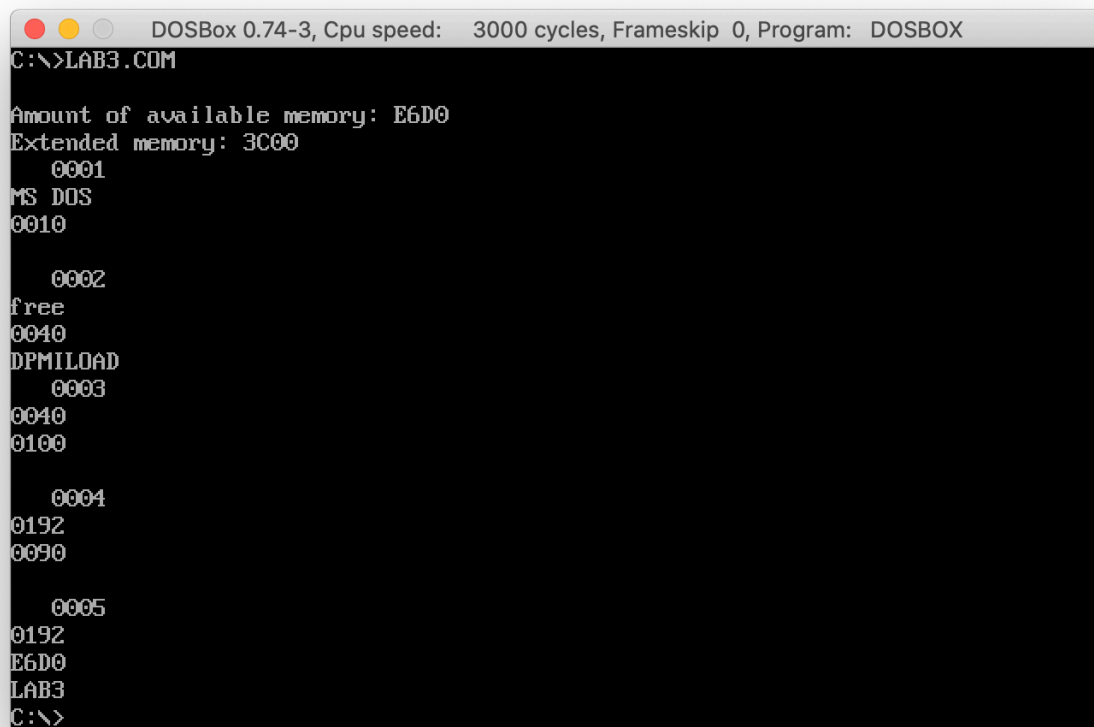
Исследовать структуры данных и работа функций управления памятью ядра операционной системы.

### **Выполнение работы.**

1. Был написан и отлажен программный модуль типа .COM, который выбирает и распечатывает следующую информацию:

- 1) Количество доступной памяти.
- 2) Размер расширенной памяти.
- 3) Выходит цепочку блоков управления памятью.

Адреса при выводе переставляются шестнадцатичными числами. Объем памяти функциями управления памятью выводится в параграфах. Пример вывода программы 1 приведен на рисунке 1.



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\>LAB3.COM

Amount of available memory: E6D0
Extended memory: 3C00
  0001
MS DOS
0010

  0002
free
0040
DPMILOAD
  0003
0040
0100

  0004
0192
0090

  0005
0192
E6D0
LAB3
C:\>
```

Рисунок 1 – Вывод программы 1

4. Программа была изменена так, чтобы она освобождала память, которую не занимает. Вывод программы 2 представлен на рисунках 2 и 3.

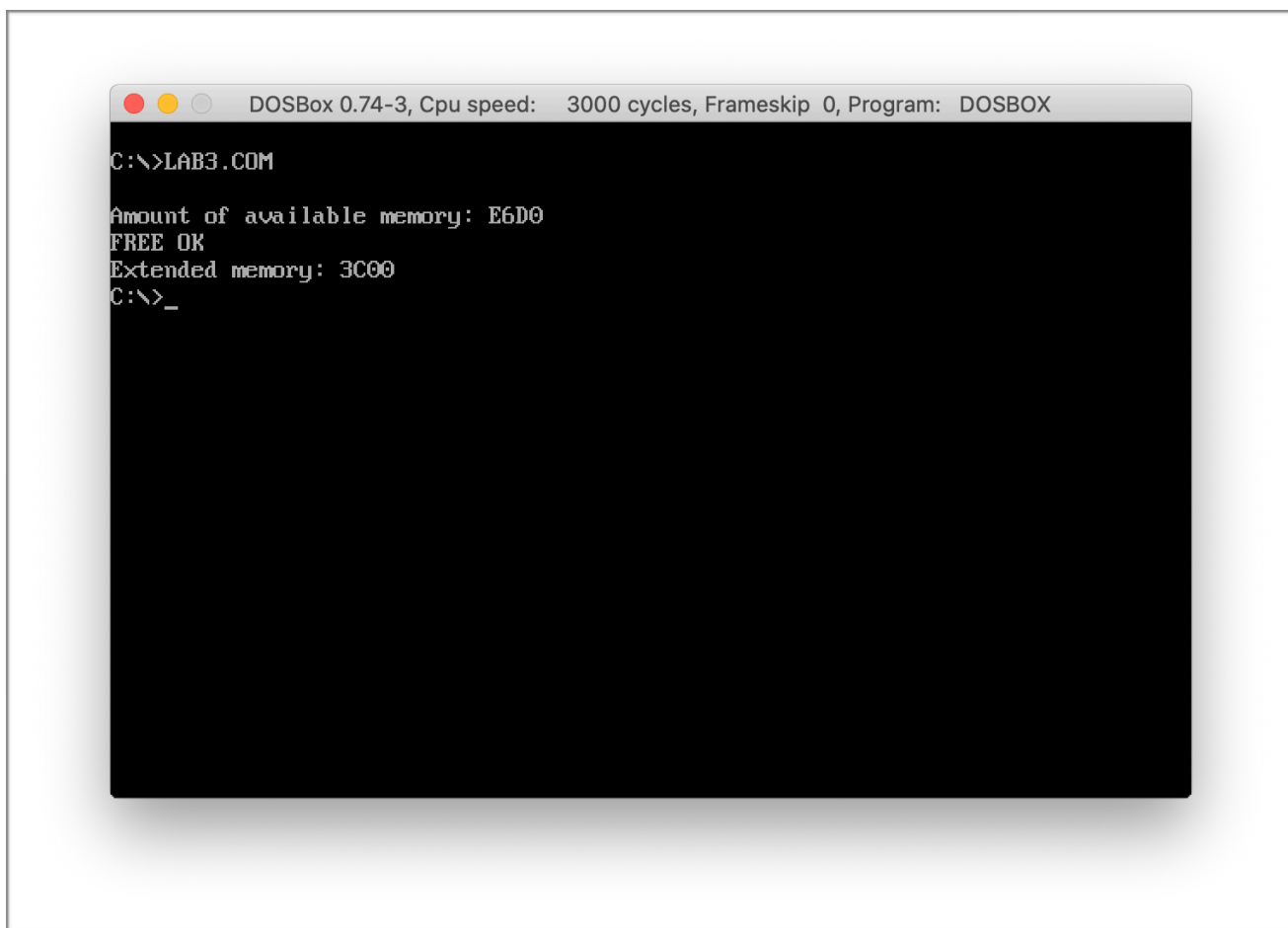


Рисунок 2 – Вывод программы 2, без вывода блоков управления памятью

Из рисунков 1, 2 и 3 видно, что освобожденная память относится к шестому блоку управления памятью.

2. Программа была изменена так, чтобы после освобождения памяти, она запрашивала 64Кб памяти функцией 48H прерывания 21h. Вывод программы 3 приведен на рисунках 4 и 5. Из рисунков 1 – 5 видно, что выделенная память относится к шестому блоку, а освобожденная к седьмому блоку управления памятью.

3. Программа была изменена так, чтобы она запрашивала выделение памяти до освобождения. Вывод программы 4 представлен на рисунках 6 и 7. Из рисунков 1 – 7 видно, память не была выделена.

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
0001
MS DOS
0010

0002
free
0040
DPMILOAD
0003
0040
0100

0004
0192
0090

0005
0192
05F0
LAB3
0006
free
E0D0
atabase
C:\>4
```

Рисунок 3 – Вывод программы 2, с выводом блоков управления памятью

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\>LAB3.COM

Amount of available memory: E6D0
FREE OK
ALLOCATE OK
Extended memory: 3C00
C:\>
```

Рисунок 4 – Вывод программы 3, без вывода блоков управления памятью

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
0002
free
0040
DPMILOAD
0003
0040
0100

0004
0192
0090

0005
0192
05F0
LAB3
0006
0192
0000
LAB3
0007
free
E0C0
C:\>
```

Рисунок 5 – Вывод программы 3, с выводом блоков управления памятью

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\>LAB3.COM

Amount of available memory: E6D0
ALLOCATE NOT OK
FREE OK
Extended memory: 3C00
C:\>
```

Рисунок 6 – Вывод программы 4, без вывода блоков управления памятью

A screenshot of a DOSBox window titled "DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX". The window displays a list of memory management blocks in a text-based format. The blocks are numbered 0001 through 0006. Each block contains several lines of text, including "MS DOS", "free", "DPMILOAD", "LAB3", and "atabase". The window has a standard macOS-style title bar with red, yellow, and green buttons on the left.

```
0001
MS DOS
0010

0002
free
0040
DPMILOAD
0003
0040
0100

0004
0192
0090

0005
0192
05F0
LAB3
0006
free
E0D0
atabase
C:\>_
```

Рисунок 7 – Вывод программы 4, с выводом блоков управления памятью

### Контрольные вопросы.

- 1) Что означает “доступный объем памяти”?

Память доступная для выполнения программы.

- 2) Где MCB блок Вашей программы в списке?

Из рисунков 1 – 7 видно, что это 5 блок, так же в программе 5 это еще и шестой блок.

- 3) Какой размер памяти она занимает в каждом случае?

Программа 1 – E6D0h (59088) байт.

Программа 2 – 05F0h (1520) байт.

Программа 3 – 05F0h (1520) + E0C0h (57536 память выделения для программы) байт.

Программа 4 – 05F0h (1520) байт.

**Выводы.**

В ходе лабораторной работы были исследованы структуры данных и работа функций управления памятью ядра операционной системы.

## ПРИЛОЖЕНИЕ А

### КОД ДЛЯ .COM МОДУЛЯ

```
LAB1  SEGMENT
      ASSUME  CS:LAB1, DS:LAB1, ES:NOTHING, SS:NOTHING
      ORG 100H

      START: JMP BEGIN

;-----
TYPE_OF_PC db "Type of PC: $"
PC db "PC$"
PC_XT db "PC/XT$"
AT db "AT$"
PS2_30 db "PS2 model 30$"
PS2_50 db "PS2 model 50 or 60$"
PS2_80 db "PS2 model 80$"
PCjr db "PCjr$"
PC_Con db "PC convertible$"
UNKNOWN_TYPE db " Unknown$"
OUT_UNKNOWN db "  $"
VER_NUM db 13, 10, "Version number: $"

OUT_VER_NUM db "  .  $"

OEM_NUM db 13, 10, "OEM: $"
OUT_OEM db "      $"
SER_NUM db 13, 10, "Serial number: $"
OUT_SER_NUM DB "          $"
;-----

TYPE_PC PROC near

      push ax
      push dx

      mov dx, offset TYPE_OF_PC
      call PRINT

      mov ax, 0F000h
      mov es, ax
      mov al, es:[0FFFEH]

      mov al, 7h
```



```

    cmp al, 0FFh
        je W_PC
    cmp al, 0FEh
        je W_PC_XT
    cmp al, 0FBh
        je W_PC_XT
    cmp al, 0FCh
        je W_AT
    cmp al, 0FAh
        je W_PS2_30
    cmp al, 0FCh
        je W_PS2_50
    cmp al, 0F8h
        je W_PS2_80
    cmp al, 0FDh
        je W_PCjr
    cmp al, 0F9h
        je W_PC_Con

    mov di, offset OUT_UNKNOWN
    call BYTE_TO_HEX
    mov [di], ax
    mov dx, offset OUT_UNKNOWN
    CALL PRINT

    mov DX, offset UNKNOWN_TYPE
    jmp ESC_PROC

W_PC:
    mov DX, offset PC
    jmp ESC_PROC

W_PC_XT:
    mov DX, offset PC_XT
    jmp ESC_PROC

W_AT:
    mov DX, offset AT
    jmp ESC_PROC

W_PS2_30:

```

```

        mov DX, offset PS2_30
        jmp ESC_PROC

W_PS2_50:
        mov DX, offset PS2_50
        jmp ESC_PROC

W_PS2_80:
        mov DX, offset PS2_80
        jmp ESC_PROC

W_PCjr:
        mov DX, offset PCjr
        jmp ESC_PROC

W_PC_Con:
        mov DX, offset PC_Con

ESC_PROC:
        call PRINT

pop dx
pop ax

ret
TYPE_PC ENDP

;-----

PRINT PROC near

push ax
sub ax, ax
mov ah, 9h
int 21h
pop ax

ret
PRINT ENDP

;-----

WRD_TO_HEX PROC near

```

```

        push    BX
        mov     BH,AH
        call    BYTE_TO_HEX
        mov     [DI],AH
        dec     DI
        mov     [DI],AL
        dec     DI
        mov     AL,BH
        call    BYTE_TO_HEX
        mov     [DI],AH
        dec     DI
        mov     [DI],AL
        pop     BX
        ret
WRD_TO_HEX ENDP

```

;-----

```

TETR_TO_HEX  PROC  near
        and     AL,0Fh
        cmp     AL,09
        jbe     NEXT
        add     AL,07
NEXT:        add     AL,30h
        ret
TETR_TO_HEX  ENDP

```

;-----

```

BYTE_TO_HEX  PROC  near
        push    CX
        mov     AH,AL
        call    TETR_TO_HEX
        xchg    AL,AH
        mov     CL,4
        shr     AL,CL
        call    TETR_TO_HEX ;â AL ñòàðøàÿ öèôðà
        pop     CX          ;â AH ìëääøàÿ
        ret
BYTE_TO_HEX  ENDP

```

;-----

```

BYTE_TO_DEC    PROC    near

                push     CX
                push     DX
                xor      AH,AH
                xor      DX,DX
                mov      CX,10
loop_bd:        div      CX
                or       DL,30h
                mov      [SI],DL
                dec      si
                xor      DX,DX
                cmp      AX,10
                jae      loop_bd
                cmp      AL,00h
                je       end_1
                or       AL,30h
                mov      [SI],AL

end_1:          pop      DX
                pop      CX
                ret

BYTE_TO_DEC    ENDP

```

;-----

```

SYSTEM_VERSION PROC

    push ax
    push bx
    push cx
    push dx

    sub ax, ax
    mov ah, 30h
    int 21h

    mov dx, offset VER_NUM
    call PRINT

    push ax
    mov si, offset OUT_VER_NUM
    inc si
    call BYTE_TO_DEC

```

```

pop ax
mov al, ah
add si, 3
call BYTE_TO_DEC

mov dx, offset OUT_VER_NUM
call print

mov DX, offset OEM_NUM
call PRINT

mov si, offset OUT_OEM
add si, 2
mov al, bh
call BYTE_TO_DEC
mov dx, OFFSET OUT_OEM
call PRINT

mov dx, offset SER_NUM
call PRINT

mov di, offset OUT_SER_NUM
add di, 6
mov ax, cx

call WRD_TO_HEX
mov al, bl
call BYTE_TO_HEX
sub di, 2
mov [di], ax
mov dx, offset OUT_SER_NUM
CALL PRINT

pop DX
pop CX
pop BX
pop AX
ret
SYSTEM_VERSION ENDP

```

```

;-----

```

```
PRINT_NUMBER_DEC PROC
```

```
    push AX
```

```
    push BX
```

```
    sub BX, BX
```

```
    mov Bl, 10
```

```
    mov AH, 0
```

```
    div Bl
```

```
    mov DX, AX
```

```
    add DL, '0'
```

```
    sub AX, AX
```

```
    mov AH, 02h
```

```
    int 21h
```

```
    mov DL, DH
```

```
    add DL, '0'
```

```
    sub AX, AX
```

```
    mov AH, 02h
```

```
    int 21h
```

```
    pop BX
```

```
    pop AX
```

```
    ret
```

```
PRINT_NUMBER_DEC ENDP
```

```
;------
```

```
BEGIN:
```

```
    call TYPE_PC
```

```
    call SYSTEM_VERSION
```

```
    xor AX, AX
```

```
    mov AH, 4Ch
```

```
    int 21h
```

```
LAB1 ENDS
```

```
END START
```

## ПРИЛОЖЕНИЕ Б

### КОД ДЛЯ «ХОРОШЕГО» .EXE МОДУЛЯ

```
LAB1STACK SEGMENT STACK
    dw 100h
LAB1STACK ENDS

DATA SEGMENT

TYPE_OF_PC db "Type of PC: $"
PC db "PC$"
PC_XT db "PC/XT$"
AT db "AT$"
PS2_30 db "PS2 model 30$"
PS2_50 db "PS2 model 50 or 60$"
PS2_80 db "PS2 model 80$"
PCjr db "PCjr$"
PC_Con db "PC convertible$"
UNKNOWN_TYPE db " Unknown$"
OUT_UNKNOWN db "    $"
VER_NUM db 13, 10, "Version number: $"

OUT_VER_NUM db "    .  $"

OEM_NUM db 13, 10, "OEM: $"
OUT_OEM db "        $"
SER_NUM db 13, 10, "Serial number: $"
OUT_SER_NUM DB "            $"

DATA ENDS

LAB1CODE SEGMENT
    ASSUME CS: LAB1CODE, DS:DATA, ES:NOTHING, SS:LAB1STACK

;-----

TYPE_PC PROC near

    push ax
    push dx

    mov dx, offset TYPE_OF_PC
```

```

call PRINT

mov ax, 0F000h
mov es, ax
mov al, es:[0FFFEH]

mov al, 7h

cmp al, 0FFh
    je W_PC
cmp al, 0FEh
    je W_PC_XT
cmp al, 0FBh
    je W_PC_XT
cmp al, 0FCh
    je W_AT
cmp al, 0FAh
    je W_PS2_30
cmp al, 0FCh
    je W_PS2_50
cmp al, 0F8h
    je W_PS2_80
cmp al, 0FDh
    je W_PCjr
cmp al, 0F9h
    je W_PC_Con

mov di, offset OUT_UNKNOWN
call BYTE_TO_HEX
mov [di], ax
mov dx, offset OUT_UNKNOWN
CALL PRINT

mov DX, offset UNKNOWN_TYPE
jmp ESC_PROC

W_PC:
    mov DX, offset PC
    jmp ESC_PROC

W_PC_XT:
    mov DX, offset PC_XT

```



```

        jmp ESC_PROC

W_AT:
        mov DX, offset AT
        jmp ESC_PROC

W_PS2_30:
        mov DX, offset PS2_30
        jmp ESC_PROC

W_PS2_50:
        mov DX, offset PS2_50
        jmp ESC_PROC

W_PS2_80:
        mov DX, offset PS2_80
        jmp ESC_PROC

W_PCjr:
        mov DX, offset PCjr
        jmp ESC_PROC

W_PC_Con:
        mov DX, offset PC_Con

ESC_PROC:
        call PRINT

pop dx
pop ax

ret
TYPE_PC ENDP

;-----

PRINT PROC near

push ax
sub ax, ax
mov ah, 9h
int 21h
pop ax

```

```
    ret
PRINT ENDP
```

;-----

```
WRD_TO_HEX    PROC    near
                push    BX
                mov     BH,AH
                call    BYTE_TO_HEX
                mov     [DI],AH
                dec     DI
                mov     [DI],AL
                dec     DI
                mov     AL,BH
                call    BYTE_TO_HEX
                mov     [DI],AH
                dec     DI
                mov     [DI],AL
                pop     BX
                ret
WRD_TO_HEX ENDP
```

;-----

```
TETR_TO_HEX    PROC    near
                and     AL,0Fh
                cmp     AL,09
                jbe     NEXT
                add     AL,07
NEXT:           add     AL,30h
                ret
TETR_TO_HEX    ENDP
```

;-----

```
BYTE_TO_HEX    PROC    near
                push    CX
                mov     AH,AL
                call    TETR_TO_HEX
                xchg    AL,AH
                mov     CL,4
                shr     AL,CL
```

```

        call    TETR_TO_HEX ;â AL ñòàððøàÿ öèôðà
        pop     CX          ;â AH ìèàäøàÿ
        ret
BYTE_TO_HEX ENDP

```

```

;-----

```

```

BYTE_TO_DEC PROC near
        push    CX
        push    DX
        xor     AH,AH
        xor     DX,DX
        mov     CX,10
loop_bd:  div     CX
        or      DL,30h
        mov     [SI],DL
        dec     si
        xor     DX,DX
        cmp     AX,10
        jae     loop_bd
        cmp     AL,00h
        je      end_1
        or      AL,30h
        mov     [SI],AL

end_1:    pop     DX
        pop     CX
        ret
BYTE_TO_DEC ENDP

```

```

;-----

```

```

SYSTEM_VERSION PROC

        push    ax
        push    bx

        sub     ax, ax
        mov     ah, 30h
        int     21h

        mov     dx, offset VER_NUM

```

```

call PRINT

push ax
mov si, offset OUT_VER_NUM
inc si
call BYTE_TO_DEC
pop ax
mov al, ah
add si, 3
call BYTE_TO_DEC

mov dx, offset OUT_VER_NUM
call print

mov DX, offset OEM_NUM
call PRINT

mov si, offset OUT_OEM
add si, 2
mov al, bh
call BYTE_TO_DEC
mov dx, OFFSET OUT_OEM
call PRINT

mov dx, offset SER_NUM
call PRINT

mov di, offset OUT_SER_NUM
add di, 6
mov ax, cx

call WRD_TO_HEX
mov al, bl
call BYTE_TO_HEX
sub di, 2
mov [di], ax
mov dx, offset OUT_SER_NUM
CALL PRINT

pop bx
pop ax

```

```
    ret
SYSTEM_VERSION ENDP
```

-----

```
PRINT_NUMBER_DEC PROC
```

```
    push AX
```

```
    push BX
```

```
    sub BX, BX
```

```
    mov Bl, 10
```

```
    mov AH, 0
```

```
    div Bl
```

```
    mov DX, AX
```

```
    add DL, '0'
```

```
    sub AX, AX
```

```
    mov AH, 02h
```

```
    int 21h
```

```
    mov DL, DH
```

```
    add DL, '0'
```

```
    sub AX, AX
```

```
    mov AH, 02h
```

```
    int 21h
```

```
    pop BX
```

```
    pop AX
```

```
    ret
```

```
PRINT_NUMBER_DEC ENDP
```

-----

```
MAIN PROC FAR
```

```
    push AX
```

```
    sub AX, AX
```

```
    mov ax, data
```

```
    mov ds, ax
```

```
    POP AX
```

```
    call TYPE_PC
```

```
    call SYSTEM_VERSION
```

```
sub AX, AX
mov AH, 4Ch
int 21h

MAIN ENDP
LAB1CODE ENDS
END MAIN
```