

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

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

Студент гр. 7383

\_\_\_\_\_

МЕДВЕДЕВ И.С.

Преподаватель

\_\_\_\_\_

ЕФРЕМОВ М.А.

Санкт-Петербург

2019

## Цель работы

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

## Ход работы

В ходе работы были написаны некоторые функции, которые описаны в табл. 1.

Таблица 1 – Используемые функции

Название функции	Выполняемая задача
TETR_TO_HEX	Перевод числа из 2-ой в 16-ричную систему счисления.
BYTE_TO_HEX	Осуществляет перевод байта, помещенного в AL, в два символа в шестнадцатеричной системе счисления, помещая результат в AX.
WRD_TO_HEX	Переводит числа размером 2 байта в 16-ричную систему счисления. В AX – число, в DI – адрес последнего символа.
BYTE_TO_DEC	Переводит в 10-ную систему счисления, SI – адрес поля младшей цифры.
Print	Вывод сообщения на экран.

Также управляющая программа хранит в памяти строки, выводящиеся для пояснения действий программы:

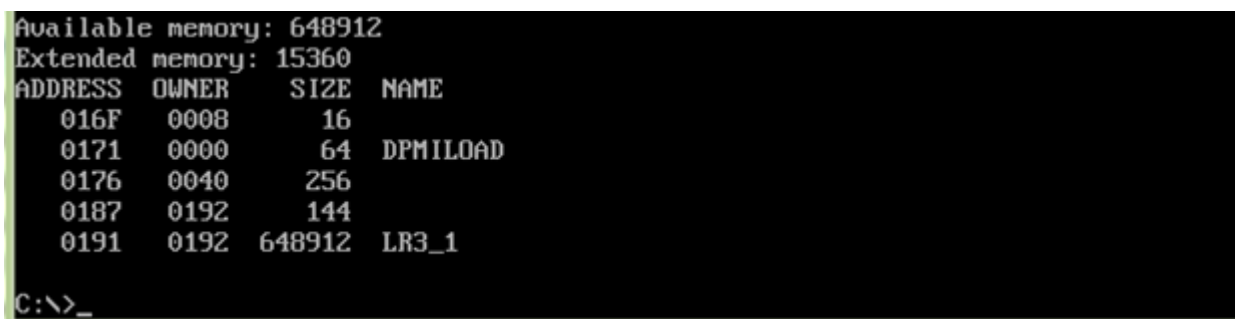
available	db	"Available memory: ", 0dh, 0ah, '\$'
extended	db	"Extended memory: ", 0dh, 0ah, '\$'
header	db	"ADDRESS OWNER SIZE NAME" ,0Dh,0Ah,'\$'
data	db	' \$'

```

endline      db    0dh, 0ah, '$'
error        db    "Allocation error!", 0dh, 0ah, '$'

```

Также в ходе работы был создан .COM файл, который выводит на экран количество доступной памяти, расширенной памяти и цепочку блоков управления памятью. Также были написаны 3 модификации для данной программы. Первая – очищает память, которую программа не использует, вторая – после очищения памяти, программа запрашивает 64Кб памяти, а третья – запрашивает 64Кб памяти, обрабатывает завершение функции ядра, освобождает память. Результаты работы представлены на рис. 1-4.

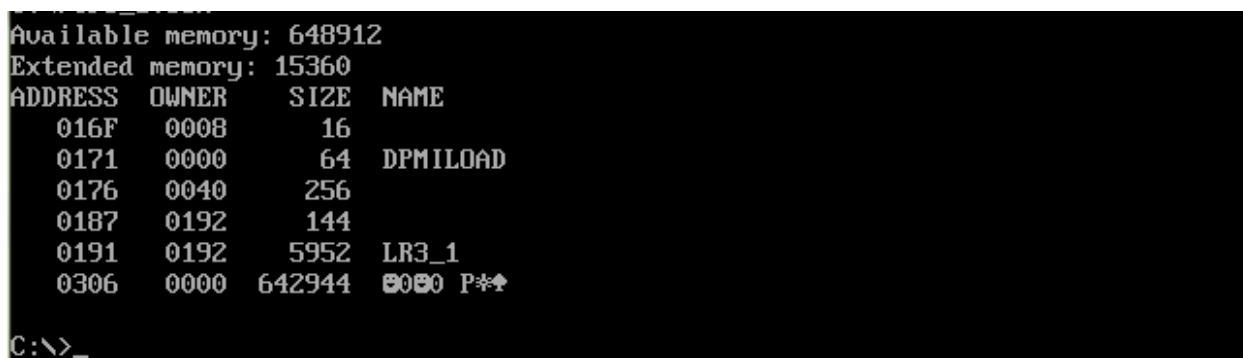


```

Available memory: 648912
Extended memory: 15360
ADDRESS  OWNER   SIZE  NAME
016F     0008    16
0171     0000    64  DPMILOAD
0176     0040   256
0187     0192   144
0191     0192 648912  LR3_1
C:\>_

```

Рисунок 1 – Результат работы первоначальной программы.



```

Available memory: 648912
Extended memory: 15360
ADDRESS  OWNER   SIZE  NAME
016F     0008    16
0171     0000    64  DPMILOAD
0176     0040   256
0187     0192   144
0191     0192   5952  LR3_1
0306     0000 642944  0000 P*↑
C:\>_

```

Рисунок 2 – Результат работы после первой модификации.

```

Available memory: 648912
Extended memory: 15360
ADDRESS  OWNER    SIZE  NAME
016F     0008      16
0171     0000      64  DPMILOAD
0176     0040     256
0187     0192     144
0191     0192    5952  LR3_1
0306     0192   65536  LR3_1
1307     0000  577392  Ys2y60-L&
C:\>

```

Рисунок 3 – Результат работы после второй модификации.

```

Available memory: 648912
Extended memory: 15360
Allocation error!
ADDRESS  OWNER    SIZE  NAME
016F     0008      16
0171     0000      64  DPMILOAD
0176     0040     256
0187     0192     144
0191     0192    5952  LR3_1
0306     0000  642944  0000 P*+
C:\>

```

Рисунок 4 – Результат работы после третьей модификации

## Вывод

В ходе выполнения лабораторной работы была исследована структура данных Memory Control Block, и работа функций ядра по выделению и освобождению памяти.

## Ответы на контрольные вопросы

1) Что означает «объем доступной памяти»?

Объём памяти, в который можно загружать пользовательские программы. Данный объём в MS DOS составляет всю свободную память до загрузки данной программы.

2) Где MCB блок Вашей программы в списке?

В первом случае мы наблюдаем 2 MCB поля с одним owner-ом 192h. Первый, находящийся по адресу 187h, это блок памяти переменных среды, который передается программе. Второй, находящийся по адресу 191h, является программным блоком, который начинается с PSP.

При первой модификации – предпоследняя строка, т.к. последнюю занимает блок выделенной памяти.

При второй модификации – пятая строка, далее идут блоки выделенной и свободной памяти. Также добавляется еще один MCB поле с owner-ом 192h, находящийся по адресу 306h, это дополнительный блок.

При третьей модификации – предпоследняя строка.

*3) Какой размер памяти занимает программа в каждом случае?*

В первом случае – 648912 байт.

Во втором – 5952 байт (освобождается неиспользуемая память).

В третьем (вместе с выделенной памятью) – 71488 байт.

В четвертом – 5952 байт.

## ПРИЛОЖЕНИЕ А. КОД ПОСЛЕДНЕЙ МОДИФИКАЦИИ

```
.186
testpc segment
    assume cs:testpc, ds: testpc, es:nothing, ss:nothing
    org 100h
start: jmp main
    ;data segment
    available      db    "Available memory:      ", 0dh, 0ah, '$'
    extended      db    "Extended memory:      ", 0dh, 0ah, '$'
    header         db    "ADDRESS  OWNER      SIZE  NAME"
,0Dh,0Ah,'$'
    data           db    '                                $'
    endl ine      db    0dh, 0ah, '$'
    END_OF_PROGRAMM db    0
    error         db    "Allocation error!", 0dh, 0ah, '$'
    ;data ends

Print proc near
    push ax

    mov ax, 0900h
    int 21h

    pop ax
    ret
Print 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
    pop CX
    ret
BYTE_TO_HEX 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

```

```

BYTE_TO_DEC PROC near
    push CX
    push DX
    xor AH,AH
    xor DX,DX
    mov CX,10
    _bd: div CX
    or DL,30h
    mov [SI],DL
    dec SI
    xor DX,DX
    cmp AX,10
    jae _bd
    cmp AL,00h
    je end_
    or AL,30h
    mov [SI],AL
    end_: pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP

```

```

WRD_TO_DEC PROC near
    push CX
    push DX
    mov CX,10
    _b: div CX
    or DL,30h
    mov [SI],DL
    dec SI
    xor DX,DX
    cmp AX,10
    jae _b
    cmp AL,00h
    je endl
    or AL,30h
    mov [SI],AL

```

```

        endl: pop DX
        pop CX
        ret
WRD_TO_DEC ENDP

```

```
main proc near
```

```
;=====
```

```
;Available Mem
```

```

        sub ax, ax
        mov ah, 04Ah
        mov bx, 0FFFFh
        int 21h
        mov ax, 10h
        mul bx
        mov si, offset available
        add si, 017h
        call WRD_TO_DEC
        mov dx, offset available
        call PRINT

```

```
;=====
```

```
;EXTENDED Mem
```

```

        mov si, offset extended
        add si, 015h
        mov al, 30h
        out 70h, al
        in al, 71h
        mov dh, al
        mov al, 31h
        out 70h, al
        in al, 71h
        mov ah, al
        mov al, dh
        xor dx, dx
        call WRD_TO_DEC
        mov dx, offset extended
        call Print

```

```
;=====
```

```
;REQUEST 64kb
```

```

        mov bx, 1000h
        mov ah, 48h
        int 21h

```

```
;=====
```

```
;PROCESSING
```

```

        jnc clear
        mov dx, offset error
        call Print

```

```
;=====
```

```
;CLEARING
```

```
clear:
```

```

        mov ah, 4ah
        mov bx, offset END_OF_PROGRAMM

```



int 21h

```
;=====
;
;MCB Data
    mov dx, offset header
    call Print
    mov ah, 52h
    int 21h
    sub bx, 02h
    mov es, es:[bx]

@while_mcb:

    mov ax, es
    mov di, offset data
    add di, 6
    call WRD_TO_HEX

    mov ax, es:[0001h]
    mov di, offset data
    add di, 13
    call WRD_TO_HEX

    xor si, si

    mov ax, es:[03h]
    xor dx, dx
    mov bx, 16
    mul bx
    mov si, offset data
    add si, 21
    call WRD_TO_DEC

    mov dx, offset data
    call Print

    xor si, si

@for_mcb:
    mov al, es:[si + 08h]
    inc si
    int 29h
    cmp si, 8h
    jne @for_mcb

    mov dx, offset endlne
    call Print
    mov ax, es
    inc ax
    add ax, es:[03h]
    mov bl, es:[00h]
    mov es, ax
```

```
        cmp bl,4Dh
        je @while_mcb
        mov ah, 4ch
        int 21h

        ret
    main endp
testpc ends

end start
```