

**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 ПАМЯТИ**

Студентка гр. 0382

Здобнова К.Д.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

### **Цель работы.**

Для исследования организации управления памятью необходимо ориентироваться на тип основной памяти, реализованный в компьютере и способ организации, принятый в ОС. В лабораторной работе рассматривается нестраничная память и способ управления динамическими разделами. Для реализации управления памятью в этом случае строится список занятых и свободных участков памяти. Функции ядра, обеспечивающие управление основной памятью, просматривают и преобразуют этот список.

В лабораторной работе исследуются структуры данных и работа функций управления памятью ядра операционной системы.

### **Задание.**

**Шаг 1.** Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа **.COM**, который выбирает и распечатывает следующую информацию:

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

Адреса при выводе представляются шестнадцатеричными числами. Объем памяти функциями управления памятью выводится в параграфах. Необходимо преобразовать его в байты и выводить в виде десятичных чисел. Последние восемь байт MSB выводятся как символы, не следует преобразовывать их в шестнадцатеричные числа.

Запустите программу и внимательно оцените результаты. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

**Шаг 2.** Измените программу таким образом, чтобы она освобождала память, которую она не занимает. Для этого используйте функцию 4Ah прерывания 21h (пример в разделе «Использование функции 4AH»). Повторите эксперимент, запустив модифицированную программу. Сравните выходные

данные с результатами, полученными на предыдущем шаге. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

**Шаг 3.** Измените программу еще раз таким образом, чтобы после освобождения памяти, программа запрашивала 64Кб памяти функцией 48H прерывания 21H. Повторите эксперимент, запустив модифицированную программу. Сравните выходные данные с результатами, полученными на предыдущих шагах. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

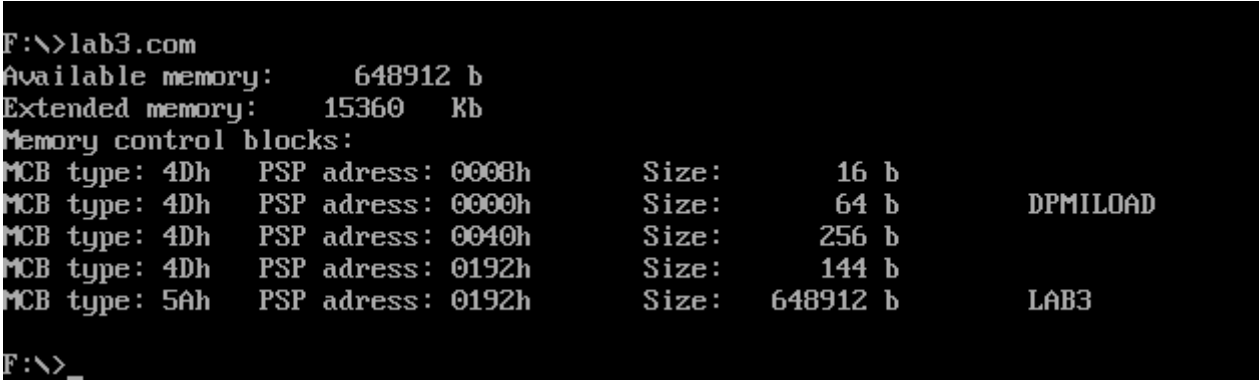
**Шаг 4.** Измените первоначальный вариант программы, запросив 64Кб памяти функцией 48H прерывания 21H до освобождения памяти. Обязательно обрабатывайте завершение функций ядра, проверяя флаг CF. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

**Шаг 5.** Оцените результаты, полученные на предыдущих шагах. Ответьте на контрольные вопросы и оформите отчет.

### **Выполнение работы.**

Была разработана программа для первого пункта работы – написан и отлажен модуль .com, выводящий информацию о количестве доступной памяти, размере расширенной памяти и выводящий цепочку блоков управления памятью.

Результат выполнения:



```
F:\>lab3.com
Available memory:      648912 b
Extended memory:      15360  Kb
Memory control blocks:
MCB type: 4Dh  PSP address: 0008h      Size:      16 b
MCB type: 4Dh  PSP address: 0000h      Size:       64 b      DPMILOAD
MCB type: 4Dh  PSP address: 0040h      Size:     256 b
MCB type: 4Dh  PSP address: 0192h      Size:     144 b
MCB type: 5Ah  PSP address: 0192h      Size:   648912 b      LAB3
F:\>
```

Рисунок 1.

Далее программа была дополнена кодом, освобождающим память, которую она не занимает.

Результат выполнения:

```

F:\>lab3.com
Available memory:      648912 b
Extended memory:      15360  Kb
Memory control blocks:
MCB type: 4Dh  PSP adress: 0008h      Size:      16 b
MCB type: 4Dh  PSP adress: 0000h      Size:       64 b      DPMILOAD
MCB type: 4Dh  PSP adress: 0040h      Size:      256 b
MCB type: 4Dh  PSP adress: 0192h      Size:       144 b
MCB type: 4Dh  PSP adress: 0192h      Size:       784 b      LAB3
MCB type: 5Ah  PSP adress: 0000h      Size:     648112 b      801 80F:

```

Рисунок 2.

Для выполнения третьего шага, дополним программу кодом, запрашивающим память.

Результат выполнения:

```

Turbo Assembler Version 3.1 Copyright (c) 1988, 1992 Borland International

Assembling file:  lab3.ASM
Error messages:   None
Warning messages: None
Passes:          1
Remaining memory: 471k

F:\>tlink /t LAB3.obj
Turbo Link Version 5.1 Copyright (c) 1992 Borland International

F:\>lab3.com
Available memory:      648912 b
Extended memory:      15360  Kb
Memory control blocks:
MCB type: 4Dh  PSP adress: 0008h      Size:      16 b
MCB type: 4Dh  PSP adress: 0000h      Size:       64 b      DPMILOAD
MCB type: 4Dh  PSP adress: 0040h      Size:      256 b
MCB type: 4Dh  PSP adress: 0192h      Size:       144 b
MCB type: 4Dh  PSP adress: 0192h      Size:       816 b      LAB3
MCB type: 4Dh  PSP adress: 0192h      Size:     65536 b      LAB3
MCB type: 5Ah  PSP adress: 0000h      Size:     582528 b      à??

```

Рисунок 3.

Поменяем местами дополненные участки кода, тем самым запросив память, после чего очистим память, которую не занимает программа.

Результат выполнения программы:

```

Assembling file:   lab3.ASM
Error messages:   None
Warning messages: None
Passes:           1
Remaining memory: 471k

F:\>tlink /t LAB3.obj
Turbo Link Version 5.1 Copyright (c) 1992 Borland International

F:\>lab3.com
Available memory: 648912 b
ERROR!

Extended memory: 15360 Kb
Memory control blocks:
MCB type: 4Dh  PSP address: 0008h      Size:      16 b
MCB type: 4Dh  PSP address: 0000h      Size:      64 b      DPMILOAD
MCB type: 4Dh  PSP address: 0040h      Size:     256 b
MCB type: 4Dh  PSP address: 0192h      Size:     144 b
MCB type: 4Dh  PSP address: 0192h      Size:     816 b      LAB3
MCB type: 5Ah  PSP address: 0000h      Size:    648080 b

```

Рисунок 4.

Ответы на контрольные вопросы:

1) Что означает "доступный объем памяти"?

Доступный объем памяти – это область памяти, которая не занята процессами системы и может выделиться для использования.

2) Где MCB блок Вашей программы в списке?

Шаг 1 – в конце списка, программа была загружена в память в последнюю очередь, имеет все свободную ранее память

Шаг 2 – предпоследний, последний – блок освобожденной программой памяти.

Шаг 3 – после блока MCB блок памяти 64Кб, выделенный программой, далее свободная память.

Шаг 4 – Выделение памяти не свершилось – завершилось неудачей => ситуация, аналогичная Шагу 2.

3) Какой размер памяти занимает программа в каждом случае?

Шаг 1 – 648192Б – вся выделенная память

Шаг 2 – 784Б – только память, занимаемая программой.

Шаг 3 – 816Б – память, занимаемая программой и 64Кб – выделенная

Шаг 4 – Выделить 64Кб не удалось => только память под программу, 816Б

### **Вывод**

В результате выполнения данной работы были исследованы структуры данных и работа функций управления памятью ядра операционной системы.

# ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab3.asm

```
PCinfo SEGMENT
    ASSUME cs:PCinfo, ds:PCinfo, es:nothing, ss:nothing
    ORG    100h

start:
    jmp     main

    ;data
    available db 'Available memory:          b$'
    extended  db 'Extended memory:          Kb$'
    mcb       db 'Memory control blocks:$'
    MCBtype   db 'MCB type: 00h$'
    PSPadr    db 'PSP address: 0000h$'
    s         db 'Size:          b$'
    error     db 'ERROR!'
    endl     db 13, 10, '$'
    tab       db 9, '$'

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
bloop:
    div     cx
    or      dl, 30h
    mov     [si], dl
    dec     si
    xor     dx, dx
    cmp     ax, 10
    jae     bloop
    cmp     al, 00h
    je      bend
    or      al, 30h
    mov     [si], al
bend:
    pop     dx
    pop     cx
    ret
BYTE_TO_DEC endp

```

```

WRD_TO_DEC proc near
    push    cx
    push    dx
    mov     cx, 10
wloop:
    div     cx
    or      dl, 30h
    mov     [si], dl
    dec     si
    xor     dx, dx
    cmp     ax, 10
    jae     wloop
    cmp     al, 00h
    je      wend
    or      al, 30h
    mov     [si], al
wend:
    pop     dx
    pop     cx
    ret
WRD_TO_DEC endp

```

```

PRINT proc near
    push    ax
    push    dx
    mov     ah, 09h
    int     21h
    pop     dx
    pop     ax
    ret
PRINT endp

```

```

PRINT_SYMBOL proc near
    push    ax
    push    dx
    mov     ah, 02h
    int     21h
    pop     dx
    pop     ax
    ret
PRINT_SYMBOL endp

```



```

main:

;available
    mov     ah, 4Ah
    mov     bx, 0ffffh
    int     21h
    xor     dx, dx
    mov     ax, bx
    mov     cx, 10h
    mul     cx
    mov     si, offset available+27
    call    WRD_TO_DEC
    mov     dx, offset available
    call    PRINT
    mov     dx, offset endl
    call    PRINT

;allocate
    xor     ax, ax
    mov     ah, 48h
    mov     bx, 1000h
    int     21h
    jnc     mem
    mov     dx, offset error
    call    PRINT
    mov     dx, offset endl
    call    PRINT
mem:

;free
    mov     ax, offset SegEnd
    mov     bx, 10h
    xor     dx, dx
    div     bx
    inc     ax
    mov     bx, ax
    mov     al, 0
    mov     ah, 4Ah
    int     21h

;extended
    mov     al, 30h
    out     70h, al
    in      al, 71h
    mov     bl, al ;младший байт
    mov     al, 31h
    out     70h, al
    in      al, 71h ;старший байт
    mov     ah, al
    mov     al, bl
    mov     si, offset extended+24
    xor     dx, dx
    call    WRD_TO_DEC
    mov     dx, offset extended
    call    PRINT
    mov     dx, offset endl
    call    PRINT

;mcb
    mov     dx, offset mcb
    call    PRINT
    mov     dx, offset endl
    call    PRINT

```

```

mov     ah, 52h
int     21h
mov     ax, es:[bx-2]
mov     es, ax

```

```

first_check:
;MCBtype
    mov     al, es:[0000h]
    call    BYTE_TO_HEX
    mov     di, offset MCBtype+10
    mov     [di], ax
    mov     dx, offset MCBtype
    call    PRINT
    mov     dx, offset tab
    call    PRINT

;PSPadr
    mov     ax, es:[0001h]
    mov     di, offset PSPadr+15
    call    WRD_TO_HEX
    mov     dx, offset PSPadr
    call    PRINT
    mov     dx, offset tab
    call    PRINT

;Size
    mov     ax, es:[0003h]
    mov     cx, 10h
    mul     cx
    mov     si, offset s+13
    call    WRD_TO_DEC
    mov     dx, offset s
    call    PRINT
    mov     dx, offset tab
    call    PRINT

;Last
    push    ds
    push    es
    pop     ds
    mov     dx, 08h
    mov     di, dx
    mov     cx, 8

```

```

second_check:
    cmp     cx, 0
    je      third_check
    mov     dl, byte PTR [di]
    call    PRINT_SYMBOL
    dec     cx
    inc     di
    jmp     second_check

```

```

third_check:
    pop     ds
    mov     dx, offset endl
    call    PRINT

;if ends
    cmp     byte ptr es:[0000h], 5ah
    je      quit

;go to next
    mov     ax, es

```

```
    add    ax, es:[0003h]
    inc    ax
    mov    es, ax
    jmp    first_check

quit:
    xor    ax, ax
    mov    ah, 4ch
    int    21h

SegEnd:
PCinfo ENDS

        END    START
```