

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. 9381

Давыдов Д.С.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

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 МСВ 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.

Описание функций.

Название	Описание
TETR_TO_HEX	Перевод половины байта в символы.
BYTE_TO_HEX	Перевод байта, помещенного в al, в два символа в шестнадцатеричной сс, помещая результат в ah.
WRD_TO_HEX	Перевод числового значения, помещенного в регистр AX, в символьную строку в шестнадцатеричной сс, помещая в регистр di.
BYTE_TO_DEC	Перевод байта, помещенного в AL, в 2 символа в десятичной сс, помещая результат в SI.
WRD_TO_DEC	Перевод слова, помещенного в ah, в последовательность символов в десятичной сс, помещая результат в SI.
PRINT	Осуществляет вывод строки на экран.
PRINT_SYMB	Осуществляет вывод символа на экран (используя функцию DOS 02h, прерывания 21).

Выполнение работы.

1 Шаг.

Был написан и отлажен модуль .com, выводящий информацию о количестве доступной памяти, размере расширенной памяти и выводящий цепочку битов управления памятью.

Результат выполнения программы (рис. 1.1):

```
O:\LAB3>LB3_S1.COM
Amount of available memory:    648912 b
Size of extended memory:      15360 Kb
List of memory control blocks:
MCB type: 4Dh   PSP address: 0008h   Size:    16 b
MCB type: 4Dh   PSP address: 0000h   Size:    64 b
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      LB3_S1
```

рис. 1.1

2 Шаг.

Далее программа была дополнена: память, которая не была использована, освобождается с помощью функции 4Ah прерывания 21H.

Результат выполнения программы (рис. 1.2):

```
Amount of available memory:    648912 b
Size of extended memory:      15360 Kb
List of memory control blocks:
MCB type: 4Dh   PSP address: 0008h   Size:    16 b
MCB type: 4Dh   PSP address: 0000h   Size:    64 b
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      LB3_S2
MCB type: 5Ah   PSP address: 0000h   Size:  648080 b      |      =!70L=
```

рис. 1.2

3 Шаг.

На третьем шаге программа была также изменена. После освобождения памяти, программа запрашивает 64Кб памяти функцией 48H прерывания 21H.

Результат выполнения (рис 1.3):

```
Amount of available memory: 648912 b
Size of extended memory: 15360 Kb
List of memory control blocks:
MCB type: 4Dh PSP address: 0008h Size: 16 b
MCB type: 4Dh PSP address: 0000h Size: 64 b
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: 864 b LB3_S3
MCB type: 4Dh PSP address: 0192h Size: 65536 b LB3_S3
MCB type: 5Ah PSP address: 0000h Size: 582480 b %s exce
```

рис. 1.3

4 Шаг.

Для прохождения 4 шага поменяем местами добавленные ранее участки кода, тем самым запросив память, после чего очистим память, которую не занимает программа.

Результат выполнения программы (рис. 1.4):

```
0:\LAB3>lb3_s4.com
Amount of available memory: 648912 b
ERROR! Memory can not be allocated!
Size of extended memory: 15360 Kb
List of memory control blocks:
MCB type: 4Dh PSP address: 0008h Size: 16 b
MCB type: 4Dh PSP address: 0000h Size: 64 b
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: 864 b LB3_S4
MCB type: 5Ah PSP address: 0000h Size: 648032 b T0≤½1F0H
```

рис. 1.4

Ответы на контрольные вопросы:

1) Что означает "доступный объем памяти"? -

Область основной памяти, выделенная программе.

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 МСВ - блок памяти 64Кб, выделенный программой, далее свободная память.

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

1) Какой размер памяти занимает программа в каждом случае?

Шаг 1 – 648192 байт – вся выделенная память

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

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

Шаг 4 – Выделить 64Кб не смогли, поэтому только 816Б памяти под программу.

Вывод.

В ходе выполнения лабораторной работы было исследована организация управления памяти, структур памяти и работа функций управляющие памятью ядра ОС. Была написана, протранслирована и протестирована программа на языке Ассемблера, печатающая информацию о количестве доступной памяти, размере расширенной памяти, а также выводящая цепочку блоков управления памятью.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb3_4.asm

```
MAIN SEGMENT
    ASSUME CS:MAIN, DS:MAIN, ES:nothing, SS:nothing
    org 100h

start:
    jmp begin

;data
availableMemory db 'Amount of available memory:          b$'
extendedMemory db 'Size of extended memory:             Kb$'
mcb db 'List of memory control blocks:$'
typeMCB db 'MCB type: 00h$'
adressPSP db 'PSP address: 0000h$'
size_s db 'Size:          b$'
endl db 13, 10, '$'
tab db 9, '$'
_error db 'ERROR! Memory can not be allocated!$'

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

;Байт в al переводится в два символа 16-ричного числа в ax
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 ;В al старшая цифра, в ah младшая
    pop cx
    ret
    byte_to_hex endp

;Перевод в 16 ss 16-ти разрядного числа
;ax - число, di - адрес последнего символа
wrд_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

;Перевод в 10 сс, si - адрес поля младшей цифры
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_l
    or      al, 30h
    mov     [si], al

end_l:
    pop     dx
    pop     cx
    ret
    byte_to_dec endp

wrd_to_dec proc near
    push    cx
    push    dx
    mov     cx, 10

wloop_bd:
    div     cx
    or      dl, 30h
    mov     [si], dl
    dec     si
    xor     dx, dx
    cmp     ax, 10
    jae     wloop_bd
    cmp     al, 00h
    je      wend_l
    or      al, 30h
    mov     [si], al

wend_l:
    pop     dx
    pop     cx
    ret
    wrd_to_dec endp

;ВЫВОД строки
print proc near
    push    ax

```



```

    push    dx
    mov     ah, 09h
    int     21h
    pop     dx
    pop     ax
    ret
    print endp

;ВЫВОД СИМВОЛА
print_symb proc near
    push    ax
    push    dx
    mov     ah, 02h
    int     21h
    pop     dx
    pop     ax
    ret
    print_symb endp

begin:
;количество доступной памяти
    mov     ah, 4Ah
    mov     bx, 0ffffh
    int     21h

    xor     dx, dx
    mov     ax, bx
    mov     cx, 10h
    mul     cx

    mov     si, offset availableMemory+37
    call    wrd_to_dec

    mov     dx, offset availableMemory
    call    print
    mov     dx, offset endl
    call    print

;запрос памяти
    xor     ax, ax
    mov     ah, 48h
    mov     bx, 1000h
    int     21h
    jnc     mem_ok
    mov     dx, offset _error
    call    print
    mov     dx, offset endl
    call    print

mem_ok:

;освобождение памяти
    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

;размер расширенной памяти
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 extendedMemory+34
xor     dx, dx
call    wrd_to_dec

mov     dx, offset extendedMemory
call    print
mov     dx, offset endl
call    print

;цепочка блоков управления памятью
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

;тип MCB
tag1:
mov     al, es:[0000h]
call    byte_to_hex
mov     di, offset typeMCB+10
mov     [di], ax

mov     dx, offset typeMCB
call    print
mov     dx, offset tab
call    print

;сегментный адрес PSP владельца участка памяти
mov     ax, es:[0001h]
mov     di, offset adressPSP+15
call    wrd_to_hex

mov     dx, offset adressPSP
call    print
mov     dx, offset tab
call    print

;размер участка в параграфах
mov     ax, es:[0003h]
mov     cx, 10h
mul     cx

```

```

    mov     si, offset size_s+13
    call    wrd_to_dec
    mov     dx, offset size_s
    call    print
    mov     dx, offset tab
    call    print

;последние 8 байт
    push    ds
    push    es
    pop     ds

    mov     dx, 08h
    mov     di, dx
    mov     cx, 8

tag2:
    cmp     cx, 0
    je      tag3
    mov     dl, byte PTR [di]
    call    print_symb
    dec     cx
    inc     di
    jmp     tag2

tag3:
    pop     ds
    mov     dx, offset endl
    call    print

;проверка, последний блок или нет
    cmp     byte ptr es:[0000h], 5ah
    je      quit

;адрес следующего блока
    mov     ax, es
    add     ax, es:[0003h]
    inc     ax
    mov     es, ax
    jmp     tag1

quit:

    xor     ax, ax
    mov     ah, 4ch
    int     21h
SegEnd:
MAIN ENDS

                END      START

```