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. 9382 _____ Дерюгин Д.А.

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 MCB 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.

Необходимые сведения для составления программы

Учет занятой и свободной памяти ведется при помощи списка блоков управления памятью MCB (Memory Control Block). MCB занимает 16 байт (параграф) и располагается всегда с адреса кратного 16 (адрес сегмента ОП) и находится в адресном пространстве непосредственно перед тем участком памяти, которым он управляет.

Смещение	Длина поля (байт)	Содержимое поля
00h	1	тип MCB: 5Ah, если последний в списке, 4Dh, если не последний
01h	2	Сегментный адрес PSP владельца участка памяти, либо 0000h - свободный участок, 0006h - участок принадлежит драйверу OS XMS UMB 0007h - участок является исключенной верхней памятью драйверов 0008h - участок принадлежит MS DOS FFFAh - участок занят управляющим блоком 386MAX UMB FFFDh - участок заблокирован 386MAX FFFEh - участок принадлежит 386MAX UMB
03h	2	Размер участка в параграфах
05h	3	Зарезервирован
08h	8	"SC" - если участок принадлежит MS DOS, то в нем системный код "SD" - если участок принадлежит MS DOS, то в нем системные данные

MCB имеет следующую структуру:

По сегментному адресу и размеру участка памяти, контролируемого эгим MCB можно определить местоположение следующего MCB В списке.

Адрес первого MCB хранится во внутренней структуре MS DOS, называемой "List of Lists" (список списков). Доступ к указателю на эгу структуру можно получить используя функцию 52h "Get List of Lists" int 21h. В результате выполнения этой функции ES:BX будет указывать на список списков. Слово по адресу ES:[BX—2] и есть адрес самого первого MCB.

Размер расширенной памяти находится в ячейках 3011, 3111 CMOS. CMOS это энергонезависимая память, в которой хранится информация о конфигурации ПЭВМ. Объем памяти составляет 64 байта. Размер расширенной памяти в Кбайтах можно определить обращаясь к ячейкам CMOS следующим образом:

```
mov AL,30h ; запись адреса ячейки CMOS
out 70h,AL
in AL,7lh ; чтение младшего байта
mov BL,AL ; размера расширенной памяти
mov AL,3lh ; запись адреса ячейки CMOS
out 70h,AL
in AL,7lh ; чтение старшего байта
; размера расширенной памяти
```

Результат работы.

1) Результат работы исходной программы

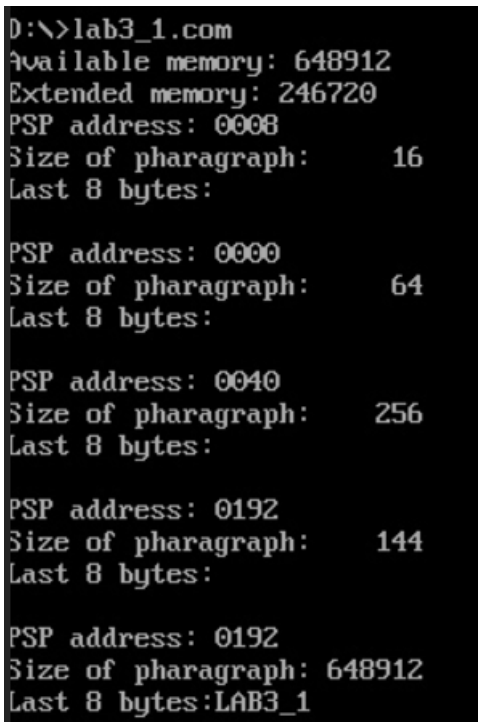

Рис.1 Результат lab3_1.com

2)

На данном рисунке результат программы, которая освобождает память, которую не занимает.

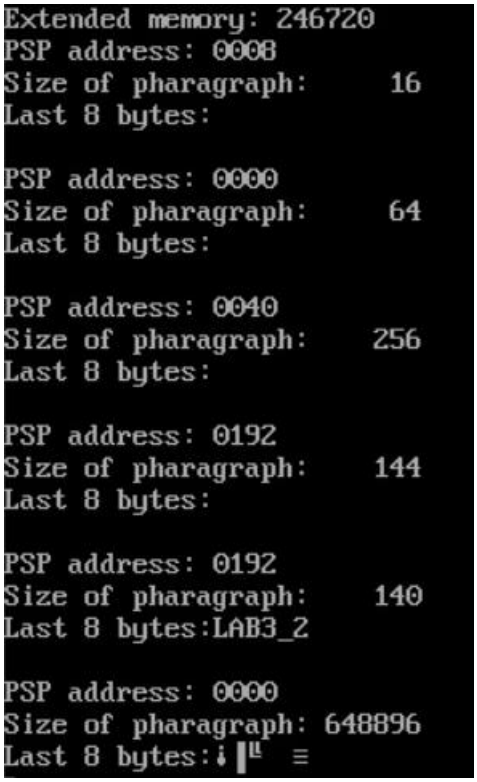

Рис.2 Результат lab3_2.com

3) На данном рисунке показан результат работы программы, которая запрашивает 64Кб памяти после освобождения.

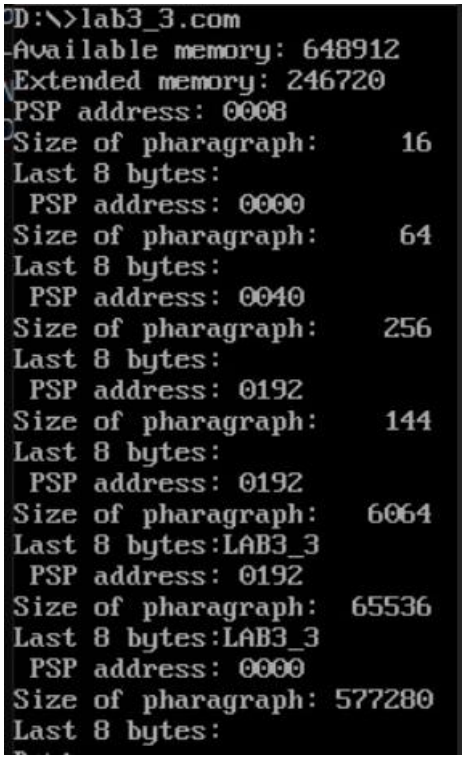

Рис.3 Результат lab3_3.com

4) В данной модификации программы, она запрашивает 64Кб памяти до освобождения памяти, но память не выделилась и процедура проверки выделения памяти выдала ошибку

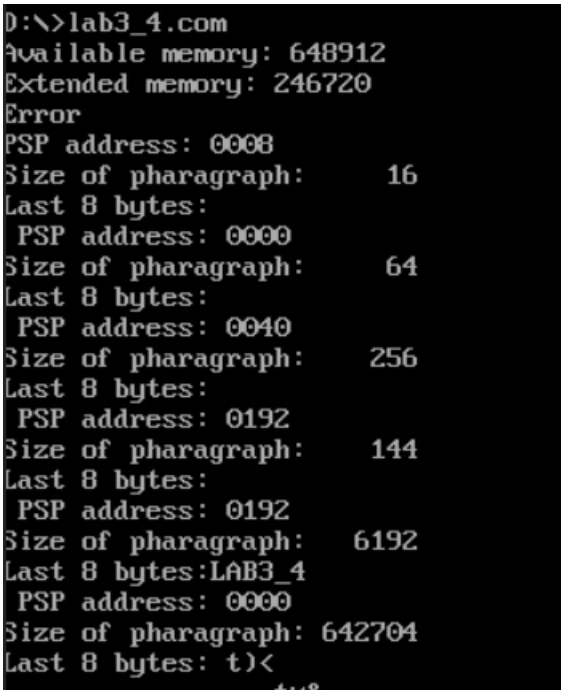

Рис.4 Результат lab3_4.com

Вывод.

В ходе выполнения данной лабораторной работы была написана программа, которая работает с памятью, а также выводит некоторую информация о ней(количество расширенной и доступной памяти, информацию о MCB блоках).

Ответы на контрольные вопросы

Контрольные вопросы по лабораторной работе №3

1) Что означает "доступный объем памяти"?

Это объем памяти, который может быть использован в программе.

2) Где MCB блок Вашей программы в списке?

По адресу 0192(см. Рис.1)

3) Какой размер памяти занимает программа в каждом случае?

На первом шаге: 648912(вся память)

На втором шаге: 140(количество байт, которое нужно для программы)

На третьем шаге: 65536+6064(64Кб + необжодимый объем памяти для программы)

На четвертом шаге: 6192.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
TESTPC SEGMENT
 ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
 ORG 100H
START: JMP BEGIN

SIZE_OF_PARAGRAPH db "Size of pharagraph:        ", 0dh, 0ah, '$'
PSP_ADDRESS db "PSP address:      ", 0dh, 0ah, '$'
EXTENDED_MEMORY db "Extended memory:       ", 0dh, 0ah, '$'
AVAILIBLE_MEMORY db "Available memory:       ", 0dh, 0ah, '$'
LAST db "Last 8 bytes:",'$'
NEXT_LINE db 0dh, 0ah

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
 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
;-------------------------------
; КОД
print proc near
    mov ah, 09h
    int 21h
    ret
print endp

HEX_TO_BYTE proc
    push ax
    push bx
    push cx
    push dx
```

```
    push si

      mov bx, 16
      mul bx
      mov bx, 0ah
      mov cx, 0

div_loop:
      div bx
      push dx
      inc cx
      sub dx, dx
      cmp ax, 0
      jnz div_loop

print_sym:
      pop dx
      add dl, 30h
      mov [si], dl
      inc si
loop print_sym

    pop si
    pop dx
    pop cx
    pop bx
    pop ax
ret
HEX_TO_BYTE endp


AVAILIBLE_MEMORY_PROC proc near
      mov bx, 0ffffh
    mov ah, 4ah
    int 21h

    mov ax, bx
      mov dx, offset AVAILIBLE_MEMORY
    mov si, offset AVAILIBLE_MEMORY
    add si, 18
    call HEX_TO_BYTE
    call print
ret
```

```
AVAILIBLE_MEMORY_PROC ENDP

EXTENDED_MEMORY_PROC PROC near
   mov AL,30h
   out 70h,AL
   in AL,71h
   mov BL,AL;
   mov AL,31h
   out 70h,AL
   in AL,71h

   mov ah, al
   mov dx, offset EXTENDED_MEMORY
   mov si, offset EXTENDED_MEMORY
   add si, 17

   call HEX_TO_BYTE

   call print
   ret
EXTENDED_MEMORY_PROC ENDP

PSP_ADDRESS_PROC proc near
   push ax
   push di

   mov di, offset PSP_ADDRESS
   add di, 16

   mov ax, es:[01h]
   call WRD_TO_HEX
   mov dx, offset PSP_ADDRESS
   call print

   pop di
   pop ax
   ret
PSP_ADDRESS_PROC ENDP

SIZE_OF_PARAGRAPH_PROC proc near
      push ax
      push bx
      push si
```

```
      mov si, offset SIZE_OF_PARAGRAPH
      add si, 25
      mov ax, es:[03h]
      mov bx, 10h
      mul bx
      call BYTE_TO_DEC
      mov dx, offset SIZE_OF_PARAGRAPH
      call print

      pop si
      pop bx
      pop ax
      ret
SIZE_OF_PARAGRAPH_PROC ENDP

LAST_EIGHT_PROC proc
      push bx
      push cx
      push ax

      mov dx, offset LAST
      call print

      sub bx, bx
      mov cx, 8

      last_eight:
            mov al, es:[08h+bx]
            int 29h
            inc bx
            loop last_eight

      pop ax
      pop cx
      pop bx
      ret
LAST_EIGHT_PROC ENDP

MCP_PROC proc near
      mov ah, 52h
      int 21h
      mov ax, es:[bx-2]
```

```
      mov es, ax

      looping:
            call PSP_ADDRESS_PROC
            call SIZE_OF_PARAGRAPH_PROC
            call LAST_EIGHT_PROC

            mov al, es:[0]
            cmp al, 5ah
            je ending
            mov ax, es:[03h]
            mov bx, es
            add bx, ax
            inc bx
            mov es, bx
            mov dx, offset NEXT_LINE
            call print
            call print
            jmp looping

    ending:
            ret


MCP_PROC ENDP

BEGIN:

    call AVAILIBLE_MEMORY_PROC
      call EXTENDED_MEMORY_PROC
      call MCP_PROC

 xor AL,AL
 mov AH,4Ch
 int 21H
TESTPC ENDS
 END START
```