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. 9383 _____ Хотяков Е.П.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

Исследование организации управления памятью. Рассмотрение нестатической памяти и способов управления динамическими разделами. Исследование структур данных и работа функций управления памятью ядра операционной системы.

Задание.

Шаг 1. Написать и отладить программный модуль .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.

Выполнение работы.

Шаг 1: Была написана .COM программа на ассемблере, которая выводит требуемую информацию.

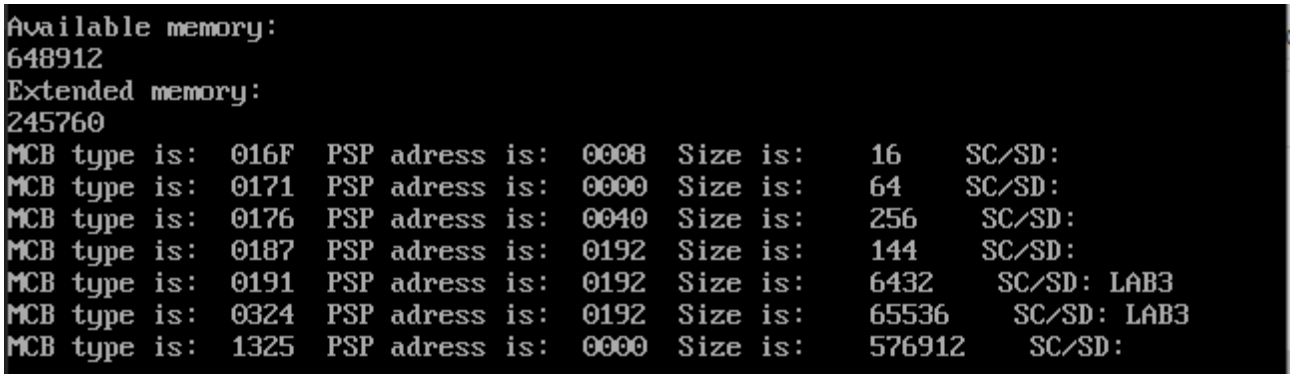
```
Available memory:
648912
Extended memory:
245760
MCB type is:  016F  PSP adress is:  0008  Size is:    16  SC/SD:
MCB type is:  0171  PSP adress is:  0000  Size is:    64  SC/SD:
MCB type is:  0176  PSP adress is:  0040  Size is:    256  SC/SD:
MCB type is:  0187  PSP adress is:  0192  Size is:    144  SC/SD:
MCB type is:  0191  PSP adress is:  0192  Size is:    648912  SC/SD: LAB3
```

Рисунок 1 - Пример работы программы

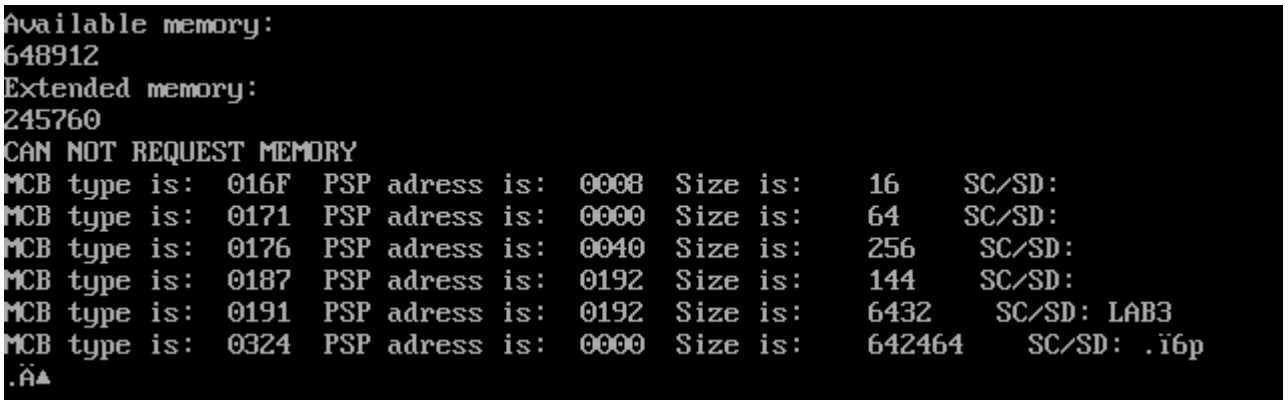
Шаг 2: Программа была изменена таким образом, чтобы неиспользуемая память освобождалась программой.

```
Available memory:
648912
Extended memory:
245760
MCB type is:  016F  PSP adress is:  0008  Size is:    16    SC/SD:
MCB type is:  0171  PSP adress is:  0000  Size is:    64    SC/SD:
MCB type is:  0176  PSP adress is:  0040  Size is:    256    SC/SD:
MCB type is:  0187  PSP adress is:  0192  Size is:    144    SC/SD:
MCB type is:  0191  PSP adress is:  0192  Size is:    6432    SC/SD: LAB3
MCB type is:  0324  PSP adress is:  0000  Size is:    642464    SC/SD: .ï6p
.Ä▲
```

Шаг 3: Программа была изменена таким образом, чтобы неиспользуемая память освобождалась программой, после чего она запрашивала 64Кб памяти.



Шаг 4: Программа была изменена таким образом, чтобы запрос 64Кб памяти совершался до освобождения памяти. Также был написан обработчик завершения функции ядра, проверяющий флаг CF.



Ответы на контрольные вопросы:

1. Что означает “Доступный объем памяти”?

   Доступный объем памяти – это область памяти, которая выделяется для работы управляющей программы.

2. Где MCB блок Вашей программы в списке?

   Это те MCB-блоки(на скриншотах шагов 1-4), у которых значение SC/SD подписано названием исполняемой программы(т.е. LAB3).

3. Какой размер памяти занимает программа в каждом случае?

   В первом случае программу занимает всю доступную память.

   Во втором – необходимый ей объем памяти, т.е. 6432 байта.

В третьем случае – необходимый объем памяти + 64Кб, т.е. 6432 + 65536 байт памяти.

В четвертом случае программа занимает необходимый объём памяти, т.е. 6432 байт, потому что выделить еще 64Кб не удалось.

Вывод:

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

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab3_4.asm

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

AV_MEM_STRING db 'Available memory: ',0DH,0AH,'$'
EXTENDED_MEM_STRING db 'Extended memory: ',0DH,0AH,'$'
MCB_TYPE_STRING db 'MCB type is:      ', '$'
PSP_TYPE_STRING db 'PSP adress is:      ', '$'
SIZE_STRING db 'Size is:    ', '$'
SC_SD_STRING db 'SC/SD: ', '$'
FAIL_STRING db 'CAN NOT REQUEST MEMORY',0DH,0AH,'$'
NEWLINE db 0DH, 0AH, '$'
TAB db '  ', '$'
; Процедуры
;--------------------------------------------------------
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
; Байт в AL переводится в два символа шестн. числа AX
            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
            pop CX           ; В AH младшая цифра
            ret
BYTE_TO_HEX ENDP
;--------------------------------
WRD_TO_HEX PROC near
; Перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
        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
;--------------------------------------------------
WRITE_STRING PROC near; Вывод строки текста
        mov AH,09h
        int 21h
        ret
WRITE_STRING ENDP

PARAGRAPH_TO_BYTE PROC
                mov bx, 0ah
                xor cx, cx

loop_pb:
                div bx
                push dx
                inc cx
                sub dx, dx
                cmp ax, 0h
                jne loop_pb
write_symbol:
                pop dx
                add dl,30h
                mov ah,02h
                int 21h

                loop write_symbol

                ret
PARAGRAPH_TO_BYTE ENDP

AVAILABLE_MEM PROC near
   MOV dx, offset AV_MEM_STRING
   CALL WRITE_STRING
```

```
    MOV AH,4AH
    MOV BX,0FFFFH ; заведомо большой блока памяти
    INT 21H

    MOV AX, BX
    MOV BX, 16
    MUL BX
    CALL PARAGRAPH_TO_BYTE
    MOV DX, offset NEWLINE
    CALL WRITE_STRING

    ret

AVAILABLE_MEM ENDP

EXTENDED_MEM PROC near
    MOV dx, offset EXTENDED_MEM_STRING
    CALL WRITE_STRING

    mov AL, 30h
    out 70h, AL
    in AL, 71h
    MOV BL,AL
            mov AL, 31h
    out 70h, AL
    in AL, 71h
    MOV BH, AL
    MOV AX, BX

    MOV BX, 16
    MUL BX
    CALL PARAGRAPH_TO_BYTE
    MOV DX, offset NEWLINE
    CALL WRITE_STRING
    ret

EXTENDED_MEM ENDP

MCB PROC near
    MOV ah, 52h
    int 21H
    mov AX, ES:[BX-2]
    MOV ES, AX

MCB_loop:
    MOV AX, ES
    MOV DI, offset MCB_TYPE_STRING
```

```
    add DI, 17
    CALL WRD_TO_HEX
    MOV DX, offset MCB_TYPE_STRING
    CALL WRITE_STRING
    MOV DX, offset TAB
    CALL WRITE_STRING

    MOV AX, ES:[1]
    MOV DI, offset PSP_TYPE_STRING
    add DI, 19
    CALL WRD_TO_HEX
    MOV DX, offset PSP_TYPE_STRING
    CALL WRITE_STRING
    MOV DX, offset TAB
    CALL WRITE_STRING

    MOV DX, offset SIZE_STRING
    CALL WRITE_STRING
    MOV AX, ES:[3]
    MOV DI, offset SIZE_STRING
    ADD DI, 10
    MOV BX, 16
    MUL BX
    CALL PARAGRAPH_TO_BYTE
    MOV DX, offset TAB
    CALL WRITE_STRING

    MOV DX, offset TAB
    CALL WRITE_STRING

    MOV DX, offset SC_SD_STRING
    call WRITE_STRING

    MOV BX, 8
    MOV CX, 08h

SC_SD_loop:
    MOV DL, ES:[BX]
    MOV AH, 02H
    INT 21H
    INC BX
    LOOP SC_SD_LOOP

    MOV DX, offset NEWLINE
    CALL WRITE_STRING

    MOV BX, ES:[3H]
```

```
   MOV AL, ES:[0H]
   CMP AL, 5AH
   JE MCB_END

   MOV AX, ES
   INC AX
   ADD AX, BX
   MOV ES, AX
   JMP MCB_loop

MCB_END:
            ret

MCB ENDP

CLEAR_MEMORY PROC near
   MOV AX, CS
   MOV ES, AX
   MOV BX, offset TESTPC_END
   MOV AX, ES
   MOV BX, AX
   MOV AH, 4AH
   INT 21H
   RET
CLEAR_MEMORY ENDP

MEMORY_REQUEST PROC near
   MOV BX, 1000h ;64kb
   MOV AH, 48h
   int 21h

   JC FAIL
   JMP MEMORY_REQUEST_END
FAIL:
   MOV DX, offset FAIL_STRING
   CALL WRITE_STRING
MEMORY_REQUEST_END:
   ret
MEMORY_REQUEST ENDP


BEGIN:
      CALL AVAILABLE_MEM
      CALL EXTENDED_MEM
      CALL MEMORY_REQUEST
      CALL CLEAR_MEMORY
      CALL MCB
```

```
;. . . . . .
; Выход в DOS
        xor AL,AL
        mov AH,4Ch
        int 21H

TESTPC_END:
TESTPC  ENDS
        END START ; Конец модуля, START - точка входа
```