

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

**ОТЧЕТ**  
**по лабораторной работе №2**  
**по дисциплине «Операционные системы»**  
**Тема: Исследование интерфейсов программных модулей**

Студент гр. 9383

\_\_\_\_\_

Камзолов Н.А.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2021

### **Постановка задачи.**

### ***Цель работы.***

Исследование интерфейса управляющей программы и загрузочных модулей. Этот интерфейс состоит в передаче запускаемой программе управляющего блока, содержащего адреса и системные данные. Так загрузчик строит префикс сегмента программы (PSP) и помещает его адрес в сегментный регистр. Исследование префикса сегмента программы (PSP) и среды, передаваемой программе.

### ***Задание.***

Написать и отладить программный модуль .COM, который выбирает и распечатывает следующую информацию:

1. Сегментный адрес недоступной памяти, взятый из PSP, в шестнадцатеричном виде.
2. Сегментный адрес среды, передаваемой программе, в шестнадцатеричном виде.
3. Хвост командной строки в символьном виде.
4. Содержимое области среды в символьном виде.
5. Путь загружаемого модуля.

### ***Функции.***

**TETR\_TO\_HEX** – переводит 4 последних бита регистра AL в символ шестнадцатеричного числа. Ответ помещается в регистр AL.

**BYTE\_TO\_HEX** – переводит значение регистра AL(1 байта) в два символа шестнадцатеричного числа. Ответ помещается в регистр AX.

**WRD\_TO\_HEX** – переводит значение регистра AX(2 байта) в символы шестнадцатеричной системы счисления. Ответ кладется по адресу, указанному в DI.

**PRINT\_NEW\_LINE** – печатает в командную строку символ переноса строки и символ перевода каретки в начало строки.

**PRINT\_BUF** – печатает в консоль текущее содержимое регистра DX.

**GET\_UNAVAILABLE\_MEMORY** – печатает в командную строку сегментный адрес недоступной памяти, взятый из PSP, в шестнадцатеричном виде.

**GET\_SEGMENT\_ADRESS** - печатает в командную строку сегментный адрес среды, передаваемой программе, в шестнадцатеричном виде.

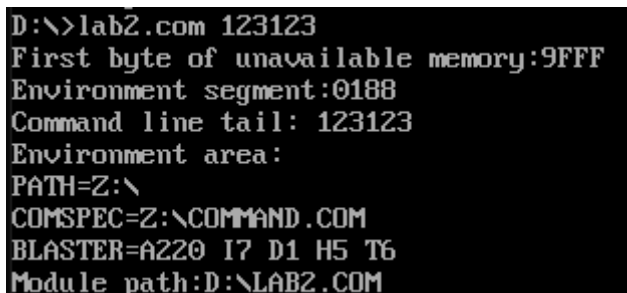
**GET\_TAIL** – печатает в командную строку хвост командной строки в символьном виде.

**GET\_ENVIRONMENT\_AREA** – печатает в командную строку содержимое области среды в символьном виде и путь загружаемого модуля

### ***Последовательность действий.***

1. Сразу попадаем на метку begin(для модуля .COM) и в функцию MAIN(для модуля .EXE).
2. Здесь поочерёдно вызываются функции GET\_UNAVAILABLE\_MEMORY, GET\_SEGMENT\_ADRESS, GET\_TAIL, GET\_ENVIRONMENT\_AREA.
3. Завершаем программу с помощью прерывания 4CH.

### **Результаты исследования проблем.**



```
D:\>lab2.com 123123
First byte of unavailable memory:9FFF
Environment segment:0188
Command line tail: 123123
Environment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Module path:D:\LAB2.COM
```

Рисунок 1 – Демонстрация корректной работы программы.

### ***1. По итогам написания программы можно ответить на контрольные вопросы «Сегментный адрес недоступной памяти»:***

1) На какую область памяти указывает адрес недоступной памяти?

Ответ: на первый байт недоступной памяти, который располагается после области, выделенной под программу.

2) Где расположен этот адрес по отношению области памяти, отведенной программе?

Ответ: в сторону увеличения адресов.

3) *Можно ли в эту область памяти писать?*

Ответ: можно, в DOS нет защиты памяти от перезаписи.

**2. По итогам написания программы можно ответить на контрольные вопросы «Среда передаваемая программе»:**

1) *Что такое среда?*

Ответ: Область памяти, в которой хранятся переменные среды, значения путей и других ресурсов операционной системы.

2) *Когда создается среда? Перед запуском приложения или в другое время?*

Ответ: Изначально среда создается при запуске операционной системы и копируется в адресное пространство запущенной программы при ее(программы) запуске. Однако среда может меняться в соответствии с параметрами программы.

3) *Откуда берется информация, записываемая в среду?*

Ответ: Информация берется из файла AUTOEXEC.BAT(automatic execution)

### **Выводы.**

Исследованы интерфейсы управляющей программы и загрузочных модулей. Также были исследованы префикс сегмента программы (PSP) и среда, передаваемой программе.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

#### lab2.asm:

```
TESTPC  SEGMENT
        ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
org 100h

start:  jmp begin

UNAVAILABLE_MEMORY db 'First byte of unavailable memory:      ',
0DH, 0AH, '$'
SEGMENT_ADRESS db 'Environment segment:      ', 0DH, 0AH, '$'
EMPTY_TAIL db 'Command line tail is empty.', 0DH, 0AH, '$'
ENVIRONMENT_AREA db 'Environment area:', 0DH, 0AH, '$'
TAIL db 'Command line tail:$'
MODULE_PATH db 'Module path:$'

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

```

```

PRINT_NEW_LINE proc near
    push ax
    push dx

    mov dl, 0Dh
    mov ah, 02h
    int 21h

    mov dl, 0Ah
    mov ah, 02h
    int 21h

    pop dx
    pop ax

```

```

        ret
PRINT_NEW_LINE endp

PRINT_BUF proc near
    push ax
    mov ah, 9h
    int 21h
    pop ax
    ret
PRINT_BUF endp

GET_UNAVAILABLE_MEMORY proc near
    push ax
    push di

    mov ax, ds:[02h]
    mov di, offset UNAVAILABLE_MEMORY
    add di, 36
    call WRD_TO_HEX
    mov dx, offset UNAVAILABLE_MEMORY
    call PRINT_BUF

    pop di
    pop ax

    ret
GET_UNAVAILABLE_MEMORY endp

GET_SEGMENT_ADRESS proc near
    push ax
    push di

    mov ax, ds:[2Ch]
    mov di, offset SEGMENT_ADRESS
    add di, 23
    call WRD_TO_HEX
    mov dx, offset SEGMENT_ADRESS
    call PRINT_BUF

```



```

        pop di
        pop ax

        ret
GET_SEGMENT_ADRESS endp

GET_TAIL proc near
    push cx
    push ax
    push si
    push dx

    mov cl, ds:[80h]
    cmp cl, 0
    je empty_tail_print

    mov dx, offset TAIL
    call PRINT_BUF

    xor dl, dl
    xor si, si
print_loop:
    mov dl, ds:[81h+si]
    mov ah, 02h
    int 21h
    inc si
    loop print_loop

    call PRINT_NEW_LINE
    jmp end_12

empty_tail_print:
    mov dx, offset EMPTY_TAIL
    call PRINT_BUF

end_12:

```

```

        pop dx
        pop si
        pop ax
        pop cx
        ret
GET_TAIL endp

GET_ENVIRONMENT_AREA proc near
    push ax
    push si
    push dx
    push es

    mov dx, offset ENVIRONMENT_AREA
    call PRINT_BUF

    mov ax, ds:[2Ch]
    mov es, ax
    xor si, si
loop_1:
    mov dl, es:[si]
    cmp dl, 0
    je next_line
print_symbol:
    mov ah, 02h
    int 21h
    inc si
    jmp loop_1
next_line:
    call PRINT_NEW_LINE
    inc si
    mov dl, es:[si]
    cmp dl, 0
    jne print_symbol

    add si, 3

```

```

        mov dx, offset MODULE_PATH
        call PRINT_BUF
loop_path:
        mov dl, es:[si]
        cmp dl, 0
        je end_l3

        mov ah, 02h
        int 21h
        inc si
        jmp loop_path

end_l3:
        pop es
        pop dx
        pop si
        pop ax
        ret
GET_ENVIRONMENT_AREA endp

begin:

        call GET_UNAVAILABLE_MEMORY
        call GET_SEGMENT_ADRESS
        call GET_TAIL
        call GET_ENVIRONMENT_AREA
        xor al, al
        mov ah, 4ch
        int 21h

TESTPC  ENDS
        END start

```