

**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. 0382

\_\_\_\_\_

Бочаров Г.С.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2022

## 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

**Шаг 1.** Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа **.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.

Сохранить результаты, полученные программой, и включить их в отчет.

**Шаг 2.** Оформить отчет в соответствии с требованиями. В отчет включить скриншот с запуском программы и результатами. Ответить на контрольные вопросы.

## Выполнение работы

1. При выполнении работы был использован шаблон и описанные в нем функции из методического пособия. Так же для получения и вывода требуемой информации были написаны следующие функции:

`unavailable_memory_print` — функция получает сегментный адрес недоступной памяти и выводит его на экран.

`environment_print` - функция получает сегментный адрес среды передаваемой программе и выводит его на экран.

`command_line_tail_print` — функция получает информацию о кол-ве символов в хвосте, далее в цикле их считывает и выводит по одному.

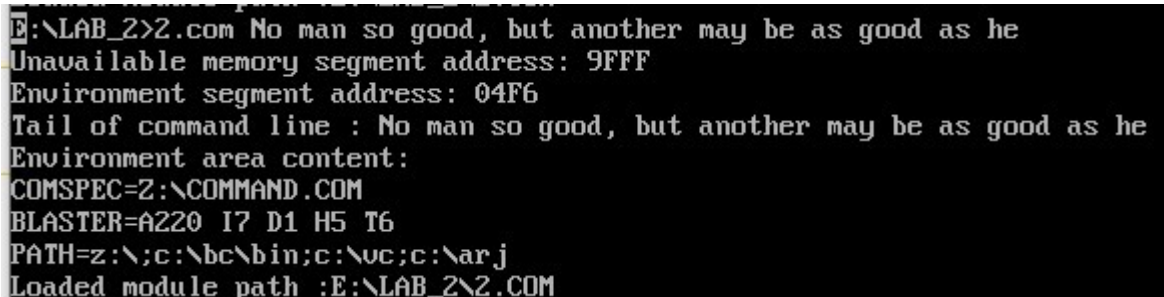
`print_env_path` — функция по символно считывает и выводит содержимое области среды до тех пор пока не встретит два нулевых байта. Далее начинается считывание и вывод пути загруженного модуля, пока не встретим нулевой байта.

Результаты работы программы представлены на Рисунке 1 и Рисунке 2



```
E:\LAB_2>2.com
Unavailable memory segment address: 9FFF
Environment segment address: 04F6
Tail of command line :empty
Environment area content:
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
PATH=z:\;c:\bc\bin;c:\uc;c:\arj
Loaded module path :E:\LAB_2\2.COM
```

Рисунок 1. Демонстрация работы программного модуля типа .COM (хвост командной строки — пустой)



```
E:\LAB_2>2.com No man so good, but another may be as good as he
Unavailable memory segment address: 9FFF
Environment segment address: 04F6
Tail of command line : No man so good, but another may be as good as he
Environment area content:
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
PATH=z:\;c:\bc\bin;c:\uc;c:\arj
Loaded module path :E:\LAB_2\2.COM
```

Рисунок 2. Демонстрация работы программного модуля типа .COM (хвост командной строки не пустой)

На основе написанной программе были даны ответы на контрольные вопросы.

### **Сегментный адрес недоступной памяти**

#### **1. На какую область памяти указывает адрес недоступной среды?**

На значение адреса первого байта сразу после памяти, выделенную под программу.

#### **2. Где расположен этот адрес по отношению области памяти, отведенной программе?**

Сразу после памяти, выделенной под программу.

#### **3. Можно ли в эту область памяти писать?**

Можно т.к. в DOS не предусмотрено механизмов защиты от перезаписи памяти.

### **Среда передаваемая программе**

#### **1. Что такое среда?**

Среда — специальная область памяти, которая хранит последовательность символьных строк вида: имя = параметр, для хранения настроек ОС.

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

Среда создается при запуске ОС. При запуске приложения создается копия среды, с параметрами для данного приложения

#### **3. Откуда берется информация, записываемая в среду?**

Из файла AUTOEXEC.BAT.

**Вывод.**

В ходе выполнения лабораторной работы было проведено исследование интерфейса управляющей программы и загрузочных модулей.. исследование префикса сегмента программы (PSP) и среды, передаваемой программе.

## Приложение А

### Файл 2.asm

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

unavailable_memory db      'Unavailable memory segment address:      ', 0DH,
0AH, '$'

environment db              'Environment segment address:            ', 0DH,
0AH, '$'

environment_area db         'Environment area content:      ', 0DH, 0AH, '$'

empty db                    'empty', 0DH, 0AH, '$'

endl db                     0DH, 0AH, '$'

command_line_tail db        'Tail of command line :$'

module_path db              'Loaded 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

```

```

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_1
        or al, 30h
        mov [si], al
end_1:   pop dx
        pop cx

```

```

        ret
byte_to_dec endp

;-----
print_word proc near
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
print_word endp

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

;-----

unavailable_memory_print proc near
    push ax
    push di
    push dx

    mov ax, ds:[02h]
    mov di, offset unavailable_memory
    add di, 39
    call wrd_to_hex
    mov dx, offset unavailable_memory

    call print_word

    pop dx
    pop di
    pop ax

    ret
unavailable_memory_print endp

```



```

environment_print proc near
    push ax
    push di
    push dx

    mov ax, ds:[02ch]
    mov di, offset environment
    add di, 32
    call wrd_to_hex
    mov dx, offset environment

    call print_word

    pop dx
    pop di
    pop ax

    ret
environment_print endp

command_line_tail_print proc near
    push di
    push cx
    push dx

    mov dx, offset command_line_tail
    call print_word
    xor cx, cx
    xor di, di
    mov cl, ds:[80h]
    cmp cl, 0h
    je empty_tail

    read_tail:
        mov dl, ds:[81h+di]
        call print_byte
        inc di
    loop read_tail

    mov dx, offset endl
    call print_word

```

```

    jmp final

empty_tail:
    mov dx, offset empty
    call print_word

final:
    pop dx
    pop cx
    pop di
    ret
command_line_tail_print endp


print_env_path proc near
    push ax
    push di
    push dx
    push es

    mov dx, offset environment_area
    call print_word
    xor di, di
    mov ax, ds:[2ch]
    mov es, ax

read_content:
    mov dl, es:[di]
    cmp dl, 0h
    je separator_word
    call print_byte
    inc di
    jmp read_content

separator_word:
    mov dl, 0ah
    call print_byte
    inc di
    mov dl, es:[di]
    cmp dl, 0h
    je read_path
    call print_byte

```

```

        inc di
        jmp read_content

read_path:
        add di, 3                                ;skip 2 bytes
        mov dx, offset module_path
        call print_word

        read_b:
            mov dl, es:[di]
            cmp dl, 0h
            je final_1
            call print_byte
            inc di
            jmp read_b

final_1:
        pop es
        pop dx
        pop di
        pop ax

        ret
print_env_path endp

BEGIN:
        call unavailable_memory_print
        call environment_print
        call command_line_tail_print
        call print_env_path

        xor al, al
        mov ah, 4ch
        int 21h

TESTPC ENDS
END START

```