МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по практической работе № 2

по дисциплине «Операционные системы»

Тема: Исследование интерфейсов программных модулей

Студент гр. 9382 _____ Иерусалимов Н.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

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.

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.

Сохраните результаты, полученные программой, и включите их в отчет.

Шаг 2.

Оформление отчета в соответствии с требованиями. В отчет включите скриншот с запуском программы и результатами.

Выполнение работы.

В ходе выполнения работы была реализована COM программа, которая выводит на экран:

- Сегментный адрес недоступной памяти, данные брались из PSP .
- Адрес среды в шестнадцатеричном виде, получили с использованием WRD_TO_HEX в первом пункте.
- Хвост команды (как по мне это можно назвать аргументом.)
- Путь до программы
- Адрес среды в текстовом виде.

Результат работы программы можно увидеть на рисунке 1\2.

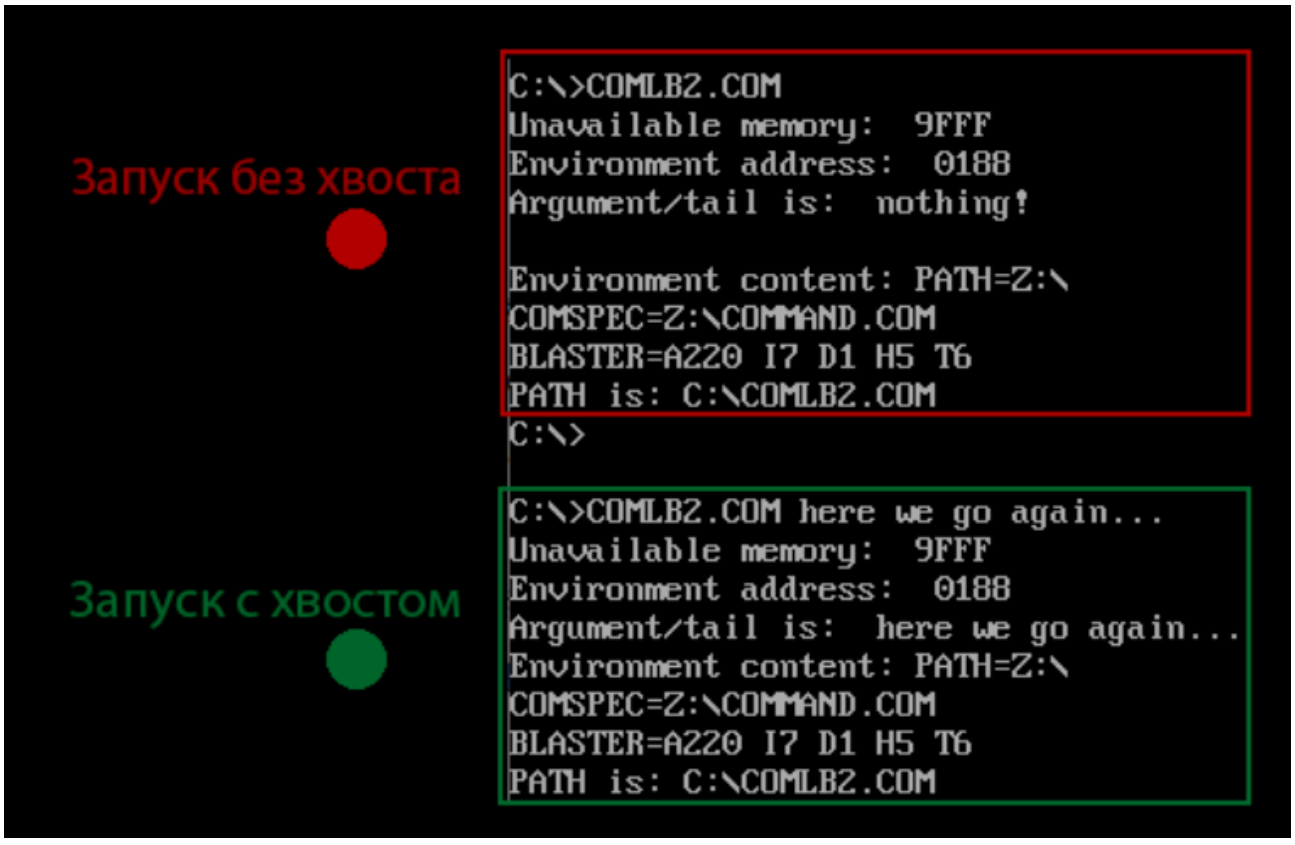

Рис.1 <Пример работы программы>

Ответы на вопросы.

Сегментный адрес недоступной памяти

1) На какую область памяти указывает адрес недоступной памяти?

   На первый байт после участка памяти, отведенного под программу

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

   Адрес недоступной памяти по отношению к области памяти, отведенной программе, расположен после. В сторону увеличения адресов.

3) Можно ли в эту область памяти писать?

   Можно, DOS не защищен от перезаписи памяти, для которых эта память не выделялась.

Среда передаваемая программе

1) Что такое среда?

   Набор переменных которые хранят некоторую информацию о состоянии системы.

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

   При загрузке DOS.

3) Откуда берется информация, записываемая в среду?

   Из файла AUTOEXEC.BAT, который расположен в корневом каталоге загрузочного устройства.

Выводы.

Был исследован интерфейс управляющей программы и загрузочных модулей. Изучили префикс сегмента программы (PSP) и среды, передаваемой программе.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: comLB2.ASM

```
testpc segment
    assume cs:testpc,ds:testpc,es:nothing,ss:nothing
    org 100h

start: jmp main

env_addr        db  'Environment address:', '$'
unavailable     db  'Unavailable memory:', '$'
ax_register     db  '      ', 0dh,0ah,'$'
ENV             db  'Environment content: ', '$'
NEW_LINE        db  0dh,0ah,'$'
PATH            db 'PATH is: ', '$'
ZERO_ARGUMENTS  db  ' nothing!', 0dh,0ah, 0dh,0ah,'$'
TAIL_ARGUMENTS  db  'Argument/tail is: ', '$'

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 si
    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
    pop si
    ret
Byte_to_dec endp
;////////////////////////////
```

```
;/////////_Code_///////////////
;//////////////////////////////
Writestring proc near
    push ax
    mov ah,09h
    int 21h
    pop ax
    ret
Writestring endp


NULLString proc near
    push si
    push cx

    xor si,si
    mov cx,5

Clear:
    mov [offset ax_register+si],0ff20h
    inc si
    loop Clear

    mov [offset ax_register+si],0ff20h
    pop cx
    pop si
    ret
NULLString endp


Display_info proc near
    call writestring
    mov di,offset ax_register
    add di,5
    call wrd_to_hex
    mov dx,offset ax_register
    call writestring
    call NULLString
    ret
Display_info endp


Display_UnMem proc near
    mov dx,offset unavailable
    mov ax,ds:[02h]
    call Display_info
    ret
Display_UnMem endp


Print_env_addr proc near
    mov dx,offset env_addr
    mov ax,ds:[2ch]
    call Display_info
    ret
Print_env_addr endp


Display_command_tail proc near
    push cx
    push ax
    xor cx,cx
```

```
    mov dx,offset TAIL_ARGUMENTS
    call writestring


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

    mov si,0

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

    mov dx,offset NEW_LINE
    call writestring

    jmp _exit_tail_print

_Empty_tail:
    mov dx,offset ZERO_ARGUMENTS
    call writestring


_Exit_tail_print:
    pop ax
    pop cx
    ret
Display_command_tail endp


Print_environment proc near
    push dx
    push ax
    push si
    push ds

    xor si,si

    mov dx,offset ENV
    call writestring

    mov ds,ds:[2ch]

_Read_env:
    mov dl,[si]
    cmp dl,0
    je _eof

    mov ah,02h
    int 21h

    inc si
    jmp _read_env

_Eof:
    inc si
    mov dl,[si]
```

```
    cmp dl,0
    je _end_reading_env

    pop ds
    mov dx,offset NEW_LINE
    call writestring
    push ds
    mov ds,ds:[2ch]

    jmp _read_env

_End_reading_env:
    pop ds
    mov dx,offset NEW_LINE
    call writestring

    mov dx,offset PATH
    call writestring
    push ds
    mov ds,ds:[2ch]

    add si,3

_Reading_path:
    mov dl,[si]
    cmp dl,0
    je _exit_print_env

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


_Exit_print_env:
    pop ds
    pop si
    pop ax
    pop dx
    ret
Print_environment endp


main:
    call Display_UnMem
    call print_env_addr
    call Display_command_tail
    call print_environment
    xor al,al
    mov ah,4ch
    int 21h


testpc ends
end start
```