

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

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

Студент гр. 8383

Шишкин И.В.

Преподаватель

Ефремов М.А.

Санкт-Петербург

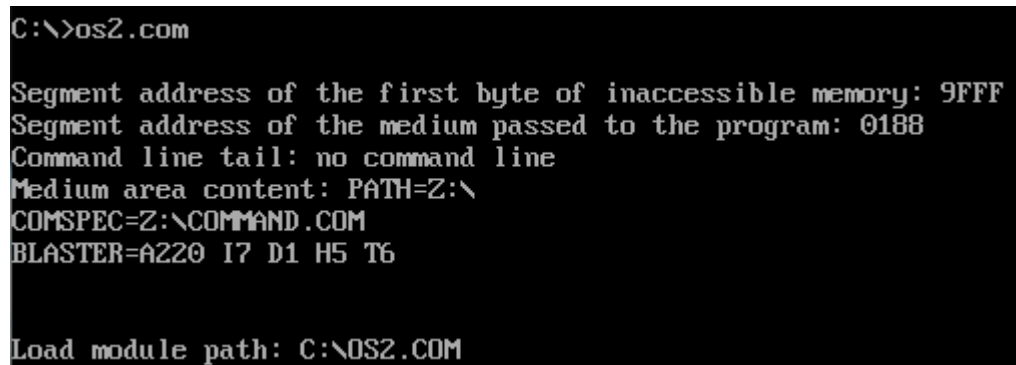
2020

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 PSP и среды, передаваемой программе.

Ход работы.

Был написан .COM модуль, который выбирает и распечатывает следующую информацию: Сегментный адрес недоступной памяти, взятый из PSP, в шестнадцатеричном виде; Сегментный адрес среды, передаваемой программе, в шестнадцатеричном виде; Хвост командной строки в символьном виде; Содержимое области среды в символьном виде; Путь загружаемого модуля. Результат выполнения программы представлен на рис. 1.



```
C:\>os2.com

Segment address of the first byte of inaccessible memory: 9FFF
Segment address of the medium passed to the program: 0188
Command line tail: no command line
Medium area content: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Load module path: C:\OS2.COM
```

Рисунок 1 – Результат выполнения программы

Контрольные вопросы.

Сегментный адрес недоступной памяти

1) На какую область памяти указывает адрес недоступной памяти?

Первые 640 Кбайт адресного пространства с сегментными адресами от 0000h до 9FFFh отводятся под основную оперативную память. Конкретно адрес 9FFFh, который выводит программа (см. рис. 1), указывает на конец памяти, свободной для загрузки любых системных или прикладных программ.

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

В конце программной памяти. Для ясности, распределение адресного пространства см. на рис. 2.

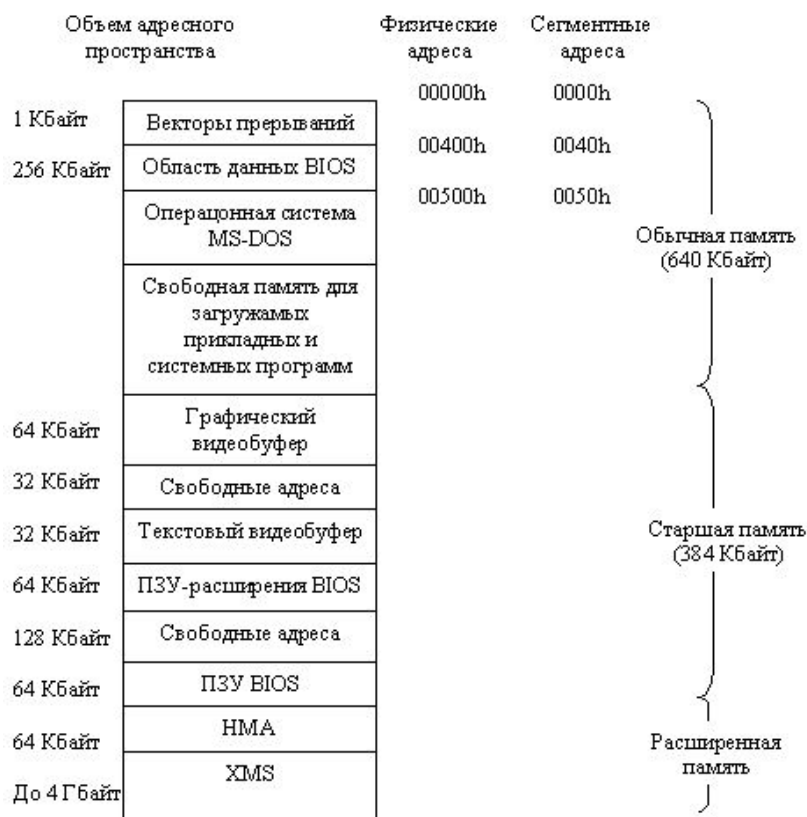


Рисунок 2 – Распределение адресного пространства

3) Можно ли в эту область памяти писать?

Да, так как DOS не контролирует обращение программ к памяти.

Среда передаваемая программе

1) Что такое среда?

Среда представляет собой последовательность строк вида параметр=значение. Общая длина строк среды не более 32 Кбайт; среда начинается с границы параграфа. После последней строки следует нулевой байт. Среда, передаваемая задаче от COMMAND, содержит, как минимум, параметр COMSPEC=(значение этого параметра - полное имя файла,

содержащего используемый COMMAND.COM). Она также содержит значения, установленные командами PATH, PROMPT и SET.

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

Среда создается при запуске программы и копируется из родительской среды.

3) Откуда берется информация, записываемая в среду?

Можно рассматривать переменные среды как своего рода параметры, передаваемые программе при ее запуске, аналогично тому, как подпрограмма получает параметры при вызове. Интерпретатор команд COMMAND.COM также имеет свою среду, которую называют корневой средой. Для создания переменных корневой среды, их удаления и изменения значений может использоваться системная команда SET. При этом среды создаются после запуска командного файла AUTOEXEC.BAT.

Выводы.

В ходе выполнения лабораторной работы был исследован интерфейс управляющей программы и загрузочных модулей, а также исследован префикс PSP и среды, передаваемой программе.

ПРИЛОЖЕНИЕ

КОД ПРОГРАММЫ

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

        INACCESSIBLE_MEMORY db 13, 10, "Segment address of the first
byte of inaccessible memory:      $"
        ADDRESS_TO_PROGRAMM db 13, 10, "Segment address of the
medium passed to the program:      $"
        TAIL db 13, 10, "Command line tail: $"
        IF_LEN_OF_TAIL_0 db "no command line$"
        MEDIUM_CONTENT db 13, 10, "Medium area content: $"
        LOAD_MODULE_PATH db 13, 10, "Load module path: $"
        SINGLE_SYMBOL db 13, 10, '$'

;-----
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 в 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
        ret
BYTE_TO_HEX ENDP
;-----
WRD_TO_HEX PROC near
;перевод в 16 с.с. 16-ти разрядного числа
; в AX - число, в 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
;-----
BYTE_TO_DEC PROC near
; перевод в 10 с.с., SI - адрес поля младшей цифры
        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 PROC near
        push ax
        mov ah, 09h
        int 21h
        pop ax
        ret
PRINT ENDP
;-----
PRINT_TAIL PROC near
        push ax
        push cx
        push dx
        push bx

        mov bl, es:[0080h]
        mov dx, offset TAIL
        call PRINT
        xor cx, cx
        mov cl, bl
        cmp cl, 0
        jne if_len_not0

```

```

mov dx, offset IF_LEN_OF_TAIL_0
call PRINT
jmp tail_end

```

```

if_len_not0:
    xor si, si
    xor ax, ax

```

```

cycle:
    mov al, es:[0081h+si]
    call PRINT_BYTE
    inc si
    loop cycle

```

```

tail_end:
    pop bx
    pop dx
    pop cx
    pop ax
ret

```

PRINT_TAIL ENDP

;-----

PRINT_BYTE PROC near

```

push ax
push dx

```

```

xor dx, dx
mov dl, al
mov ah, 02h
int 21h

```

```

pop dx
pop ax

```



```

        ret
PRINT_BYTE ENDP
;-----
PRINT_MEDIUM_CONTENT PROC near
    push ax
    push bx
    push dx
    push si
    push es

    mov dx, offset MEDIUM_CONTENT
    call PRINT
    xor si, si
    mov bx, 2Ch
    mov es, [bx]

reading_content:
    cmp BYTE PTR es:[si], 0h
    je next_line
    mov al, es:[si]
    call PRINT_BYTE
    jmp check

next_line:
    mov dx, offset SINGLE_SYMBOL
    call PRINT

check:
    inc si
    cmp WORD PTR es:[si], 0001h
    je end_of_med_content
    jmp reading_content

end_of_med_content:

```

```

        pop es
        pop si
        pop dx
        pop bx
        pop ax

    ret
PRINT_MEDIUM_CONTENT ENDP
;-----
PRINT_LOAD_MODULE_PATH PROC near
    push ax
    push bx
    push dx
    push es
    push si

    xor si, si
    mov bx, 2Ch
    mov es, [bx]

reading:
    inc si
    cmp WORD PTR es:[si], 0001h
    je path
    jmp reading
path:
    mov dx, offset LOAD_MODULE_PATH
    call PRINT
    add si, 2
cyclee:
    cmp BYTE PTR es:[si], 00h
    je ending
    mov al, es:[si]
    call PRINT_BYTE
    inc si

```

```

        jmp cyclee

ending:
        pop si
        pop es
        pop dx
        pop bx
        pop ax
        ret
PRINT_LOAD_MODULE_PATH ENDP
;-----

BEGIN:

        mov bx, es:[0002h]
        mov al, bh
        mov di, offset INACCESSIBLE_MEMORY
        call BYTE_TO_HEX
        mov [di+60], ax
        mov al, bl
        mov di, offset INACCESSIBLE_MEMORY
        call BYTE_TO_HEX
        mov [di+62], ax
        mov dx, offset INACCESSIBLE_MEMORY
        call PRINT

        xor di, di
        xor dx, dx

        mov bx, es:[002Ch]
        mov al, bh
        mov di, offset ADDRESS_TO_PROGRAMM
        call BYTE_TO_HEX

```

```

    mov [di+55], ax
    mov al, bl
    mov di, offset ADDRESS_TO_PROGRAMM
    call BYTE_TO_HEX
    mov [di+57], ax
    mov dx, offset ADDRESS_TO_PROGRAMM
    call PRINT

    xor di, di
    xor dx, dx

    call PRINT_TAIL
    call PRINT_MEDIUM_CONTENT
    call PRINT_LOAD_MODULE_PATH

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

```