

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. 9383

Рыбников Р.А.

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 состоит префикс сегмента программы (PSP) и помещает его адрес в сегментные регистр. Исследование префикса сегмента программы (PSP) и среды, передаваемой программе.

Порядок выполнения работы.

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа **.COM**, который выбирает и распечатывает следующую информацию:

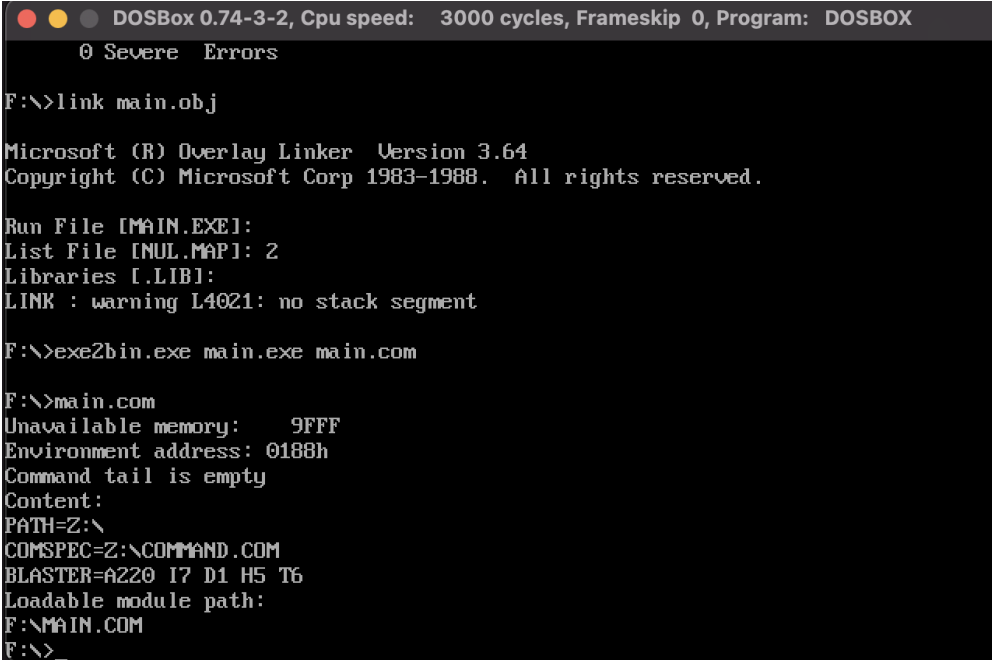
1. Сегментный адрес недоступной памяти, взятый из PSP, в шестнадцатеричном виде.
2. Сегментный адрес среды, передаваемой программе, в шестнадцатеричном виде.
3. Хвост командной строки в символьном виде.
4. Содержимое области среды в символьном виде.
5. Путь загружаемого модуля.

Шаг 2. Оформление отчета в соответствии с требованиями. В отчет включается скриншот с запуском программы и результатами.

Выполнение работы.

Были составлены функции для считывания данных из префикса и преобразования чисел.

В результате выполнения были получены следующие значения:



```
DOSBox 0.74-3-2, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
0 Severe Errors

F:\>link main.obj

Microsoft (R) Overlay Linker Version 3.64
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.

Run File [MAIN.EXE]:
List File [NUL.MAP]: 2
Libraries [LIB1]:
LINK : warning L4021: no stack segment

F:\>exe2bin.exe main.exe main.com

F:\>main.com
Unavailable memory: 9FFF
Environment address: 0188h
Command tail is empty
Content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Loadable module path:
F:\MAIN.COM
F:\>_
```

Рисунок 1 -- Пример работы программы.

Выводы.

В ходе лабораторной работы была реализована .COM программа, исходный код которой расположен в приложении А.

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

Сегментный адрес недоступной памяти:

1. На какую область памяти указывает адрес недоступной памяти?

На первый байт после памяти после программы.

2. Где расположен этот адрес по отношению области памяти, отведённой программе?

PSP по адресу 2ch.

3. Можно ли в эту область памяти писать?

Да, т.к. в DOS-е нет механизмов защиты.

Среда, передаваемая программе:

1. Что такое среда?

Среда -- это область памяти, в которой записаны переменные среды.

2. Когда создаётся среда? Перед запуском приложения или в другое время?

При загрузке ОС, но перед запуском приложения, она может быть изменена в соответствии с требованиями этого приложения.

3. Откуда берётся информация, записываемая в среду?

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

ПРИЛОЖЕНИЕ А.

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING,
SS:NOTHING

ORG 100H

START: jmp BEGIN

MEMORY_ADDRESS db 'Unavailable memory: h',13,10, 13, 10,
'\$'

ENV_ADDRESS db 'Environment address: h',13,10,'\$'

NOT_EMPTY_TAIL db 'Command line tail: ',13,10,'\$'

EMPTY_TAIL_STR db 'Command tail is empty',13,10,'\$'

CONTENT_STR db 'Content:',13,10, '\$'

END_OF_LINE db 13, 10, '\$'

PATH db 'Loadable module path: ',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

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_l
    or AL,30h
    mov [SI],AL
end_l:
    pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP

```

```

WRITE_STRING PROC near
    mov AH,09h
    int 21h
    ret
WRITE_STRING ENDP

```

```

UNAVAILABLE_MEMORY PROC near

```



```
    mov ax,ds:[02h]
    mov di, offset MEMORY_ADDRESS
    add di, 26
    call WRD_TO_HEX
    mov dx, offset MEMORY_ADDRESS
    call WRITE_STRING
    ret
UNAVAILABLE_MEMORY ENDP
```

```
ENVIROMENT_ADDRESS PROC near
    mov ax,ds:[2Ch]
    mov di, offset ENV_ADDRESS
    add di, 24
    call WRD_TO_HEX
    mov dx, offset ENV_ADDRESS
    call WRITE_STRING
    ret
ENVIROMENT_ADDRESS ENDP
```

```
COMMAND_LINE_TAIL PROC near
    xor cx, cx
        mov cl, ds:[80h]
        mov si, offset NOT_EMPTY_TAIL
```

```

        add si, 19
    cmp cl, 0h
    je empty_tail
        xor di, di
        xor ax, ax

next_tail:
        mov al, ds:[81h+di]
    inc di
    mov [si], al
    inc si
    loop next_tail

        mov dx, offset NOT_EMPTY_TAIL
    jmp TAIL_END

empty_tail:
        mov dx, offset EMPTY_TAIL_STR

TAIL_END:
    call WRITE_STRING
    ret

COMMAND_LINE_TAIL ENDP

```

CONTENT PROC near

mov dx, offset CONTENT_STR

call WRITE_STRING

xor di,di

mov ds, ds:[2Ch]

READ_LINE:

cmp byte ptr [di], 00h

jz END_LINE

mov dl, [di]

mov ah, 02h

int 21h

jmp find_end

END_LINE:

cmp byte ptr [di+1],00h

jz FIND_END

push ds

mov cx, cs

mov ds, cx

mov dx, offset END_OF_LINE

call WRITE_STRING

pop ds

FIND_END:

```
    inc di
    cmp word ptr [di], 0001h
    jz PATH_READING
    jmp READ_LINE
```

PATH_READING:

```
    push ds
    mov ax, cs
    mov ds, ax
    mov dx, offset PATH
    call WRITE_STRING
    pop ds
    add di, 2
```

LOOP_PATH:

```
    cmp byte ptr [di], 00h
    jz EXIT
    mov dl, [di]
    mov ah, 02h
    int 21h
    inc di
    jmp LOOP_PATH
```

EXIT:

ret

CONTENT ENDP

BEGIN:

call UNAVAILABLE_MEMORY

call ENVIROMENT_ADDRESS

call COMMAND_LINE_TAIL

call CONTENT

xor AL,AL

mov AH,4Ch

int 21H

TESTPC ENDS

END START