**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. 9381 _____ Любимов В.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 строит префикс сегмента программы (PSP) и помещает его адрес в сегментный регистр. Исследование префикса сегмента программы (PSP) и среды, передаваемой программе.

**Описание функций и структуры данных**

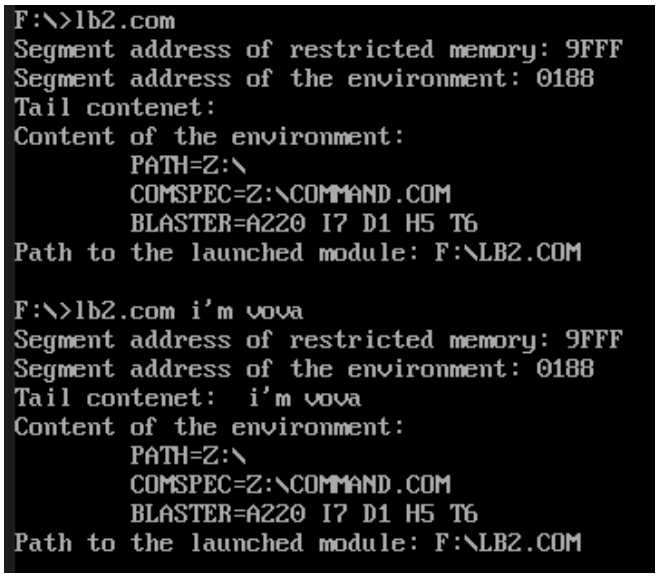
1. TETR_TO_HEX – переводит число, представляемое четырьмя младшими битами в регистре AL, в 16-ричную цифру-символ.
2. BYTE_TO_HEX - переводит число, содержащиеся в регистре AL, в 16-ричные цифры, записывающиеся в регистры AL и AH.
3. WRD_TO_HEX – переводит слово в регистре AX в четыре 16-ричные цифры, записывающие по адресу, находящемуся в DI.
4. BYTE_TO_DEC - переводит число, содержащиеся в регистре AL, в 10-ричные цифры, записывающиеся по адресу, находящемуся в SI.
5. PRINT_MES – при помощи функции 9h из прерывания 21h выводит строку на экран.
6. PRINT_INVALID_MEM – получает адрес первого байта недоступной памяти, находящийся в ds:[02h]. После этого переводит полученное значение в шестнадцатеричное число и выводит его на экран при помощи функций WRD_TO_HEX и PRINT_MES.
7. PRINT_ENVIR_ADRESS – получает сегментный адрес среды, находящийся в ds:[02ch]. После этого переводит полученное значение в шестнадцатеричное число и выводит его на экран при помощи функций WRD_TO_HEX и PRINT_MES.
8. PRINT_TAIL – сначала получает из ds:[080h] число символов в хвосте командной строки. Затем считывает и записывает в выходную строку

символы из хвоста строки в количестве равном полученному ранее. Выводит выходную строку на экран.

9. PRINT_ENVIR_AND_PATH – выводит на экран содержимое области среды в символьном виде и путь до загружаемого модуля.

**Ход выполнения работы**

Был написан код исходного модуля lb2.asm. Затем при помощи транслятора MASM, линковщика и утилиты EXE2BIN был получен загрузочный модуль lb2.com. Далее полученный модуль запущен как без символов в конце командной строки, так и с ними.




**Ответы на контрольные вопросы**

**Сегментный адрес недоступной памяти**

1. На какую область памяти указывает адрес недоступной памяти?

Данный адрес указывает на первый байт, идущий после выделенной под программу памяти.

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

Недоступная память располагается сразу после конца памяти, доступной программе.

3. Можно ли в эту область памяти писать?

Да, но такие действия могут привести к некорректной работе программы.

**Среда, передаваемая программе**

1. Что такое среда?

Среда – набор строк вида <имя переменной> = <значение переменной>00h, хранящие информацию о текущем состоянием системы.

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

Среда создаётся при запуске ОС. При запуске какой-либо программы её среда копируется с исходной среды, но в копируемую среду могут быть внесены изменения необходимые для корректной работы запускаемой программы.

3. Откуда берется информация, записываемая в среду?

Данная информация расположена в системном пакетном файле AUTOEXEC.BAT, расположенном в корневой директории загрузочного устройства. При загрузке ОС запускается командный интерпретатор COMMAND.COM, который и выполняет вышеназванный пакетный файл, устанавливая этим переменные среды.

**Вывод**

В результате выполнения работы исследован префикс сегмента программы (PSP) и среда, передаваемая программе. Также исследован интерфейс программных модулей.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

lb2.asm

```
MAIN SEGMENT
	ASSUME CS:MAIN, DS:MAIN, ES:NOTHING, SS:NOTHING
	ORG 100h

START:
	jmp BEGIN

restricted_memory_mes DB 'Segment address of restricted memory: $'
restricted_memory_adress DB '     ', 0Dh, 0Ah, '$'
envir_adress_mes DB 'Segment address of the environment: $'
envir_adress DB '     ', 0Dh, 0Ah, '$'
tail_mes DB 'Tail contenet: $'
envir_content_mes DB 'Content of the environment:', 0Dh, 0Ah, '$'
path_mes DB 'Path to the launched module: $'
some_content DB 256 DUP('$')

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

WORD_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
WORD_TO_HEX ENDP

PRINT_MES PROC near
	push ax
```

```
        mov ah, 09h
        int 21h
        pop ax
        ret
PRINT_MES ENDP

PRINT_INVALID_MEM PROC NEAR
        mov ax, DS:[0002h]
        mov di, offset restricted_memory_adress + 3
        call WORD_TO_HEX
        mov dx, offset restricted_memory_mes
        call PRINT_MES
        mov dx, offset restricted_memory_adress

        call PRINT_MES
        ret
PRINT_INVALID_MEM ENDP

PRINT_ENVIR_ADRESS PROC NEAR
        mov ax, DS:[002Ch]
        mov di, offset envir_adress + 3
        call WORD_TO_HEX
        mov dx, offset envir_adress_mes
        call PRINT_MES
        mov dx, offset envir_adress

        call PRINT_MES
        ret
PRINT_ENVIR_ADRESS ENDP

PRINT_TAIL PROC NEAR
        mov si, 0081h
        mov di, offset some_content
        mov cl, DS:[0080h]
        xor ch, ch
        rep movsb
        mov byte ptr [di], 0Dh
        mov byte ptr [di+1], 0Ah
        mov byte ptr [di+2], '$'
        mov dx, offset tail_mes
        call PRINT_MES
        mov dx, offset some_content

        call PRINT_MES
        ret
PRINT_TAIL ENDP

PRINT_ENVIR_AND_PATH PROC NEAR
        mov ax, DS:[002Ch]
        mov ds, ax
        xor si, si
        mov di, offset some_content

begin_1:
        mov al, 09h
        stosb
        lodsb
        cmp al, 0h
        je end_1
        stosb

begin_2:
        lodsb
        cmp al, 0h
        je end_2
```

```
        stosb
        jmp begin_2

end_2:
        mov al, 0Ah
        stosb
        jmp begin_1

end_1:
        mov byte ptr ES:[di], 0Dh
        mov byte ptr ES:[di+1], '$'
        mov bx, ds
        mov ax, es
        mov ds, ax
        mov dx, offset envir_content_mes
        call PRINT_MES
        mov dx, offset some_content
        call PRINT_MES
        mov di, offset some_content
        mov ds, bx
        lodsb
        lodsb

begin_3:
        lodsb
        cmp al, 0h
        je end_3
        stosb
        jmp begin_3

end_3:
        mov byte ptr ES:[di], 0Ah
        mov byte ptr ES:[di+1], 0Dh
        mov byte ptr ES:[di+2], '$'
        mov ax, es
        mov ds, ax
        mov dx, offset path_mes
        call PRINT_MES
        mov dx, offset some_content

        call PRINT_MES
        ret
PRINT_ENVIR_AND_PATH ENDP

BEGIN:
        call PRINT_INVALID_MEM
        call PRINT_ENVIR_ADRESS
        call PRINT_TAIL
        call PRINT_ENVIR_AND_PATH
        xor al, al
        mov AH, 4Ch
        int 21H

MAIN ENDS

         END START
```