

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

**ОТЧЕТ**  
**по лабораторной работе №6**  
**по дисциплине «Операционные системы»**  
**Тема: Построение модуля динамической структуры**

Студент гр. 9382

Русинов Д.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

## **Цель работы.**

Исследование возможности построения загрузочного модуля динамической структуры.

## **Задание.**

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет функции:

1) Подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором находится он сам. Вызываемому модулю передается новая среда, созданная вызывающим модулем и новая командная строка.

2) Вызываемый модуль запускается с использованием загрузчика.

3) После запуска проверяется выполнение загрузчика, а затем результат выполнения вызываемой программы. Необходимо проверять причину завершения и, в зависимости от значения, выводить соответствующее сообщение. Если причина завершения 0, то выводится код завершения.

В качестве вызываемой программы необходимо взять программу ЛР 2, которая распечатывает среду и командную строку. Эту программу следует немного модифицировать, вставив перед выходом из нее обращение к функции ввода символа с клавиатуры. Введенное значение записывается в регистр AL и затем происходит обращение к функции выхода 4Ch прерывания int 21h.

Шаг 2. Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры.

Введите произвольный символ из числа A-Z. Посмотрите причину завершения и код. Занесите полученные данные в отчет.

Шаг 3. Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры.

Введите комбинацию символов Ctrl-C. Посмотрите причину завершения и код. Занесите полученные данные в отчет.

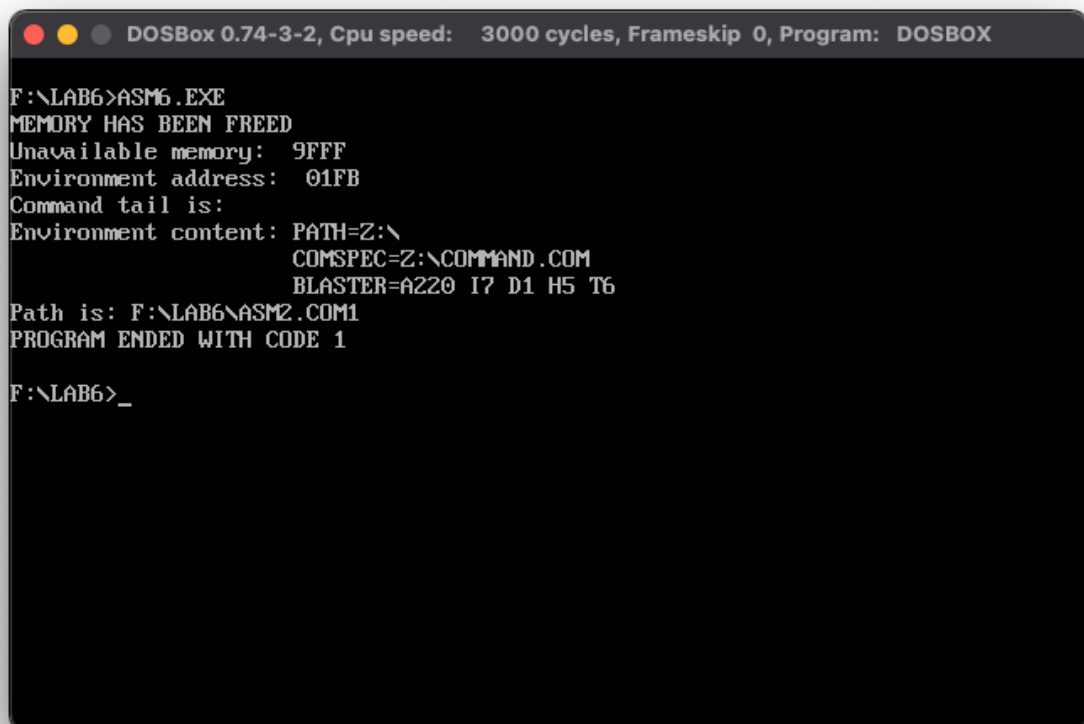
Шаг 4. Запустите отлаженную программу, когда текущим каталогом является какой-либо другой каталог, отличный от того, в котором содержатся разработанные программные модули.

Повторите ввод комбинаций клавиш. Занесите полученные данные в отчет.

Шаг 5. Запустите отлаженную программу, когда модули находятся в разных каталогах. Занесите полученные данные в отчет.

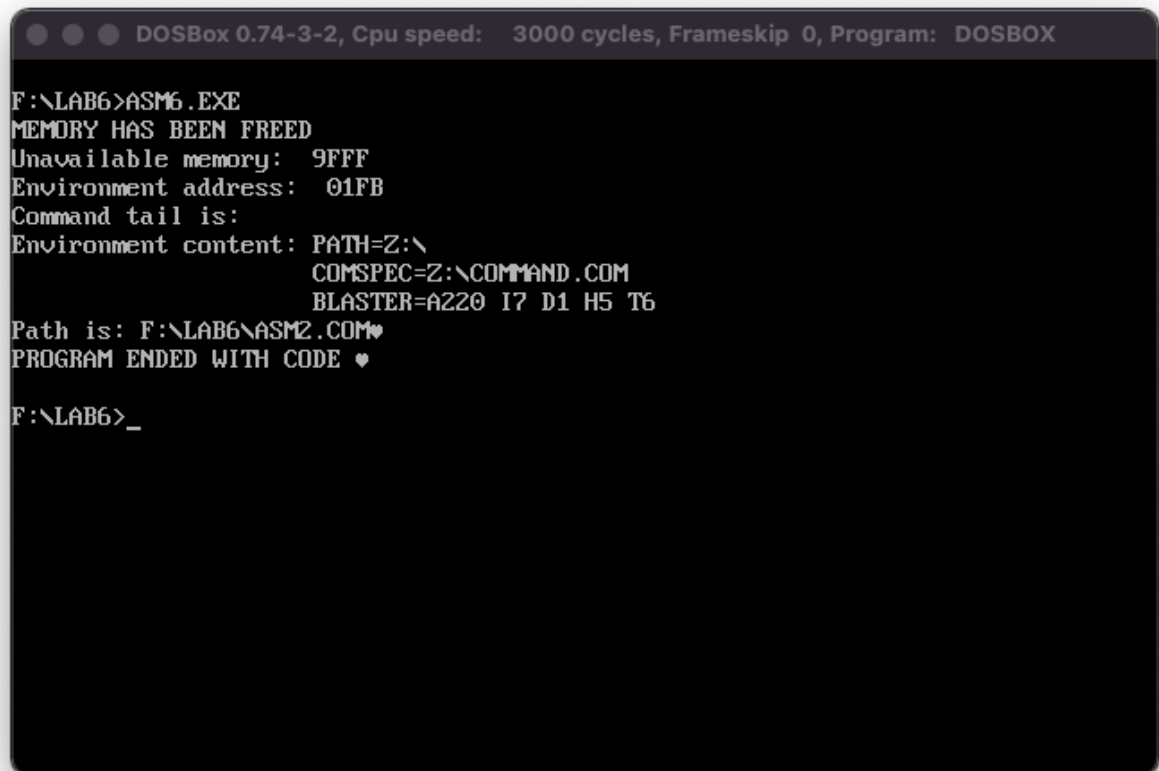
### **Выполнение работы.**

Была модифицирована программа из лабораторной работы номер два. После введенной модификации, программа перед завершением ожидает символ с клавиатуры. Запустим программу из директории с этими модулями и в конце введем символ 1.



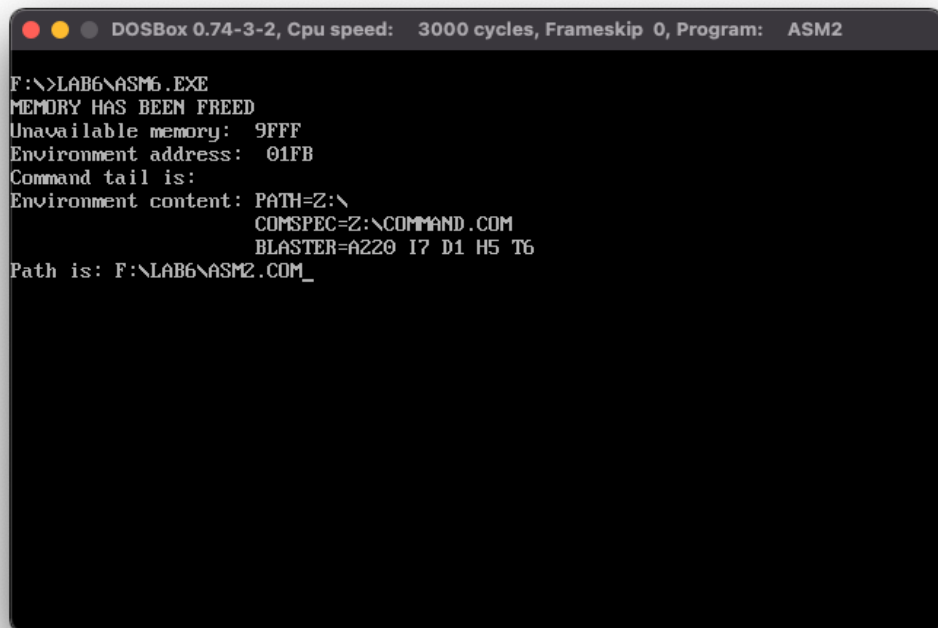
```
DOSBox 0.74-3-2, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
F:\LAB6>ASM6.EXE
MEMORY HAS BEEN FREED
Unavailable memory: 9FFF
Environment address: 01FB
Command tail is:
Environment content: PATH=Z:\
                     COMSPEC=Z:\COMMAND.COM
                     BLASTER=A220 I7 D1 H5 T6
Path is: F:\LAB6\ASM2.COM1
PROGRAM ENDED WITH CODE 1
F:\LAB6>_
```

После этого запустим программу и завершим нажатием клавиш Ctrl + C. Программа будет завершена успешно, так как сочетание данных клавиш не реализовано в DOSBox, будет символ “♥”.



```
DOSBox 0.74-3-2, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
F:\LAB6>ASM6.EXE
MEMORY HAS BEEN FREED
Unavailable memory: 9FFF
Environment address: 01FB
Command tail is:
Environment content: PATH=Z:\
                     COMSPEC=Z:\COMMAND.COM
                     BLASTER=A220 I7 D1 H5 T6
Path is: F:\LAB6\ASM2.COM♥
PROGRAM ENDED WITH CODE ♥
F:\LAB6>_
```

Запустим программу при условии, что мы находимся вне директории с этими модулями.



```
DOSBox 0.74-3-2, Cpu speed: 3000 cycles, Frameskip 0, Program: ASM2
F:\>LAB6\ASM6.EXE
MEMORY HAS BEEN FREED
Unavailable memory: 9FFF
Environment address: 01FB
Command tail is:
Environment content: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path is: F:\LAB6\ASM2.COM_
```

После этого запустим модуль при условии, что модули находятся в разных директориях.



```
DOSBox 0.74-3-2, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
F:\LAB6>ASM6.EXE
MEMORY HAS BEEN FREED
ERR: FILE NOT FOUND
F:\LAB6>_
```

## **Вывод.**

Был построен загрузочный модуль динамической структуры. Были получены навыки работы с памятью.

## **Ответы на контрольные вопросы.**

### **1) Как реализовано прерывание Ctrl-C?**

Если было нажато сочетание клавиш Ctrl + C и флаг Break имеет значение ON, то управление будет передано по адресу 0000:008Ch. Затем этот адрес копируется в PSP функциями 26h и 4Ch и восстанавливается при выходе из программы.

### **2) В какой точке заканчивается вызываемая программа, если код причины завершения 0?**

В месте вызова функции 4Ch прерывания 21h.

### **3) В какой точке заканчивается вызываемая программа по прерыванию Ctrl + C?**

В месте вызова функции 01h прерывания 21h.

# ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

```
STACKS SEGMENT STACK
    DW 128 DUP(?)
STACKS ENDS
```

```
DATA SEGMENT
    PARAMETER_BLOCK          DW
0
                                DD
0
                                DD
0
                                DD
0
    PROGRAM                  DB 'ASM2.COM',
0
    MEM_POINTER              DB
0
    SOME_CMD                 DB
1H, 0DH
    POS_CL                   DB
128 DUP(0)
    KEEP_SS                  DW
0
    KEEP_SP                  DW
0
    KEEP_PSP                 DW
0

    STR_CRASH_MCB_ERR        DB 'ERR: MCB CRASHED',
0DH, 0AH, '$'
    STR_NO_MEM_ERR           DB 'ERR: THERE IS NOT ENOUGH MEMORY TO EXECUTE
THIS FUNCTION', 0DH, 0AH, '$'
    STR_SOM_ADDR_ERR         DB 'ERR: INVALID MEMORY SOM_ADDRESS',
0DH, 0AH, '$'
    STR_FREE_MEM             DB 'MEMORY HAS BEEN FREED',
0DH, 0AH, '$'

    STR_FN_ERR               DB 'ERR: INVALID FUNCTION NUMBER',
0DH, 0AH, '$'
    STR_FILE_ERROR           DB 'ERR: FILE NOT FOUND',
0DH, 0AH, '$'
    STR_DISK_ERR             DB 'ERR: DISK ERROR',
0DH, 0AH, '$'
    STR_MEMORY_ERROR         DB 'ERR: INSUFFICIENT MEMORY',
0DH, 0AH, '$'
    STR_ENVS_ERR             DB 'ERR: WRONG STRING OF ENVIRONMENT ',
0DH, 0AH, '$'
    STR_FORMAT_ERR           DB 'ERR: WRONG FORMAT',
0DH, 0AH, '$'

    STR_NORM_FIN             DB 0DH, 0AH, 'PROGRAM ENDED WITH CODE      ' ,
0DH, 0AH, '$'
    STR_CTRL_END             DB 0DH, 0AH, 'PR ENDED BY CTRL-BREAK' ,
0DH, 0AH, '$'
    STR_DEVICE_ERR           DB 0DH, 0AH, 'PR ENDED BY DEVICE ERROR' ,
```

```

0DH, 0AH, '$'
    STR_INT_END          DB 0DH, 0AH, 'PR ENDED BY INT 31H' ,
0DH, 0AH, '$'

    END_DATA            DB
0
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA, SS:STACKS

PRINT_STRING PROC
    PUSH AX
    MOV AH, 09H
    INT 21H
    POP AX
    RET
PRINT_STRING ENDP

FREE_MEMORY PROC
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX

    MOV AX, OFFSET END_DATA
    MOV BX, OFFSET EEEND
    ADD BX, AX

    MOV CL, 4
    SHR BX, CL
    ADD BX, 2BH
    MOV AH, 4AH
    INT 21H

    JNC _END_SUP_FREEEE
    MOV MEM_POINTER, 1

CRASH_MCB:
    CMP AX, 7
    JNE NOT_ENOUGH_MEMORY
    MOV DX, OFFSET STR_CRASH_MCB_ERR
    CALL PRINT_STRING
    JMP SUP_FREE
NOT_ENOUGH_MEMORY:
    CMP AX, 8
    JNE SOM_ADDR
    MOV DX, OFFSET STR_NO_MEM_ERR
    CALL PRINT_STRING
    JMP SUP_FREE
SOM_ADDR:
    CMP AX, 9
    MOV DX, OFFSET STR_SOM_ADDR_ERR
    CALL PRINT_STRING
    JMP SUP_FREE
_END_SUP_FREEEE:
    MOV MEM_POINTER, 1
    MOV DX, OFFSET STR_FREE_MEM

```



```

        CALL PRINT_STRING

SUP_FREE:
        POP DX
        POP CX
        POP BX
        POP AX
        RET
FREE_MEMORY ENDP

LOAD PROC
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX
        PUSH DS
        PUSH ES
        MOV KEEP_SP, SP
        MOV KEEP_SS, SS

        MOV AX, DATA
        MOV ES, AX
        MOV BX, OFFSET PARAMETER_BLOCK
        MOV DX, OFFSET SOME_CMD
        MOV [BX+2], DX
        MOV [BX+4], DS
        MOV DX, OFFSET POS_CL

        MOV AX, 4B00H
        INT 21H

        MOV SS, KEEP_SS
        MOV SP, KEEP_SP
        POP ES
        POP DS

        JNC LOADS

FN_ERR:
        CMP AX, 1
        JNE FILE_ERR
        MOV DX, OFFSET STR_FN_ERR
        CALL PRINT_STRING
        JMP LOAD_END
FILE_ERR:
        CMP AX, 2
        JNE DISK_ERR
        MOV DX, OFFSET STR_FILE_ERROR
        CALL PRINT_STRING
        JMP LOAD_END
DISK_ERR:
        CMP AX, 5
        JNE MEM_ERR
        MOV DX, OFFSET STR_DISK_ERR
        CALL PRINT_STRING
        JMP LOAD_END
MEM_ERR:
        CMP AX, 8
        JNE ENVS_ERR

```

```

        MOV DX, OFFSET STR_MEMORY_ERROR
        CALL PRINT_STRING
        JMP LOAD_END
ENVS_ERR:
        CMP AX, 10
        JNE FORMAT_ERR
        MOV DX, OFFSET STR_ENVS_ERR
        CALL PRINT_STRING
        JMP LOAD_END
FORMAT_ERR:
        CMP AX, 11
        MOV DX, OFFSET STR_FORMAT_ERR
        CALL PRINT_STRING
        JMP LOAD_END

LOADS:
        MOV AH, 4DH
        MOV AL, 00H
        INT 21H

_NEND:
        CMP AH, 0
        JNE CTRLC
        PUSH DI
        MOV DI, OFFSET STR_NORM_FIN
        MOV [DI+26], AL
        POP SI
        MOV DX, OFFSET STR_NORM_FIN
        CALL PRINT_STRING
        JMP LOAD_END
CTRLC:
        CMP AH, 1
        JNE DEVICE
        MOV DX, OFFSET STR_CTRL_END
        CALL PRINT_STRING
        JMP LOAD_END
DEVICE:
        CMP AH, 2
        JNE INT_31H
        MOV DX, OFFSET STR_DEVICE_ERR
        CALL PRINT_STRING
        JMP LOAD_END
INT_31H:
        CMP AH, 3
        MOV DX, OFFSET STR_INT_END
        CALL PRINT_STRING

LOAD_END:
        POP DX
        POP CX
        POP BX
        POP AX
        RET
LOAD ENDP

PATH PROC
        PUSH AX
        PUSH BX
        PUSH CX

```

```

    PUSH DX
    PUSH DI
    PUSH SI
    PUSH ES

    MOV AX, KEEP_PSP
    MOV ES, AX
    MOV ES, ES:[2CH]
    MOV BX, 0

FINDZ:
    INC BX
    CMP BYTE PTR ES:[BX-1], 0
    JNE FINDZ

    CMP BYTE PTR ES:[BX+1], 0
    JNE FINDZ

    ADD BX, 2
    MOV DI, 0

_LOOP:
    MOV DL, ES:[BX]
    MOV BYTE PTR [POS_CL+DI], DL
    INC DI
    INC BX
    CMP DL, 0
    JE _END_LOOP
    CMP DL, '\'
    JNE _LOOP
    MOV CX, DI
    JMP _LOOP
_END_LOOP:
    MOV DI, CX
    MOV SI, 0

_FN:
    MOV DL, BYTE PTR [PROGRAM+SI]
    MOV BYTE PTR [POS_CL+DI], DL
    INC DI
    INC SI
    CMP DL, 0
    JNE _FN

    POP ES
    POP SI
    POP DI
    POP DX
    POP CX
    POP BX
    POP AX
    RET
PATH ENDP

BEGIN PROC FAR
    PUSH DS
    XOR AX, AX
    PUSH AX

```

```

        MOV AX, DATA
        MOV DS, AX
        MOV KEEP_PSP, ES
        CALL FREE_MEMORY
        CMP MEM_POINTER, 0
        JE _END
        CALL PATH
        CALL LOAD
_END:
        XOR AL, AL
        MOV AH, 4CH
        INT 21H

BEGIN          ENDP

EEEND:
CODE ENDS
END BEGIN

```