

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

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

Студентка гр. 9383

\_\_\_\_\_

Чебесова И.Д.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2021

## ПОСТАНОВКА ЗАДАЧИ

### **Цель работы.**

Исследование возможности построения загрузочного модуля динамической структуры. Исследование интерфейса между вызывающим и вызываемым модулями по управлению и по данным.

### **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.

## РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ ПРОБЛЕМ

**Шаг 1.** Был написан и отлажен программный модуль типа .EXE, который выполняет, требуемый в задании функционал.

**Шаг 2.** Программа была запущена из текущего каталога. Введенным символом был символ «а».



```
C:\>lab6.exe
Memory was successfullle freed
Unavailable memory address:9FFF
Segment environment address:02D6
Command tail:
Segment environment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Module path:C:\LAB2.COM
a
Programm was finished: exit with code:  a
```

Рисунок 1 – Демонстрация работы программы, при вводе символа «а»

**Шаг 3.** Программа была запущена из текущего каталога. Для прерывания была введена комбинация клавиш Ctrl+C. На рисунке 2 видно, что комбинация клавиш отображается символом сердечка, это происходит т.к. DOSBox это эмулятор и в нем не реализовано это прерывание.

```

C:\>lab6.exe
Memory was successfule freed
Unavailable memory address:9FFF
Segment environment address:02D6
Command tail:
Segment environment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Module path:C:\LAB2.COM
♥
Programm was finished: exit with code: ♥

```

Рисунок 2 – Демонстрация работы программы, при вводе Ctrl+C

**Шаг 4.** Программа была запущена из другого каталога TMP. Введенным символом был символ «q», затем Ctrl+C.

```

C:\TMP>lab6.exe
Memory was successfule freed
Unavailable memory address:9FFF
Segment environment address:02D6
Command tail:
Segment environment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Module path:C:\TMP\LAB2.COM
q
Programm was finished: exit with code: q

```

Рисунок 3 – Демонстрация работы программы, запущенной из каталога TMP, при вводе «q»

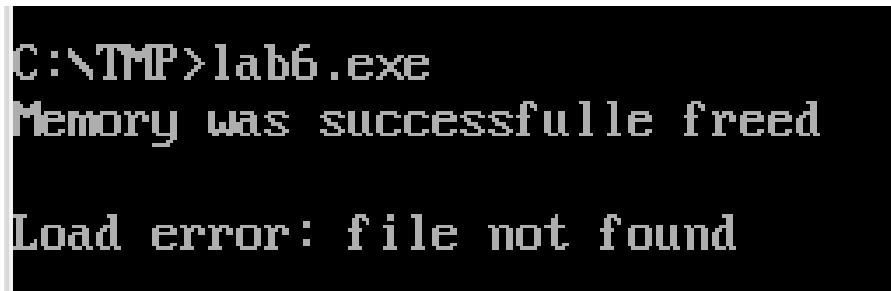
```

C:\TMP>lab6.exe
Memory was successfule freed
Unavailable memory address:9FFF
Segment environment address:02D6
Command tail:
Segment environment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Module path:C:\TMP\LAB2.COM
♥
Programm was finished: exit with code: ♥

```

Рисунок 4 – Демонстрация работы программы, запущенной из каталога TMP, при вводе Ctrl+C

**Шаг 5.** Программа была запущена, когда модули находятся в разных каталогах.



```
C:\TMP>lab6.exe
Memory was successfule freed
Load error: file not found
```

Рисунок 5 – Демонстрация работы программы с ошибкой

### Ответы на контрольные вопросы:

1. Как реализовано прерывание Ctrl-C?

**Ответ:** когда происходит нажатие сочетания клавиш Ctrl+C срабатывает прерывание - int 23h. Тогда управление передается по адресу - (0000:008C). С помощью функций 26h и 4ch этот адрес копируется в PSP. При выходе из программы исходное значение адреса восстанавливается.

2. В какой точке заканчивается вызываемая программа, если код причины завершения 0?

**Ответ:** если код причины завершения 0, то вызываемая программа заканчивается в точке вызова функции 4ch прерывания int 21h.

3. В какой точке заканчивается вызываемая программа по прерыванию Ctrl-C?

**Ответ:** в данном случае, программа завершится в точке, где была введена и считана комбинация клавиш Ctrl+C.

## **ВЫВОДЫ**

Исследованы возможности построения загрузочного модуля динамической структуры. Исследован интерфейс между вызывающим и вызываемым модулями по управлению и по данным.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

#### lab6.asm:

```
ASTACK SEGMENT STACK
    DW 256 DUP(?)
ASTACK ENDS
```

#### DATA SEGMENT

```
    PARAM_BLOCK DW 0
    COMMAND_OFF DW 0
    COMMAND_SEG DW 0
    FCB1 DD 0
    FCB2 DD 0

    NEXT_COMMAND_LINE DB 1H, 0DH
    FILE_NAME DB 'LAB2.COM', 0H
    FILE_PATH DB 128 DUP(0)

    KEEP_SS DW 0
    KEEP_SP DW 0

    FREE_MEMORY DB 0
    STR_FREE_MEMORY_MCB_ERROR DB 'FREE MEMORY ERROR: MCB CRASHED',
0DH, 0AH, '$'
    STR_FREE_MEMORY_NOT_ENOUGH_ERROR DB 'FREE MEMORY ERROR: NOT ENOUGH
MEMORY', 0DH, 0AH, '$'
    STR_FREE_MEMORY_ADDRESS_ERROR DB 'FREE MEMORY ERROR: WRONG
ADDRESS', 0DH, 0AH, '$'
    STR_FREE_MEMORY_SUCCESSFULLY DB 'MEMORY WAS SUCCESSFULLE FREED',
0DH, 0AH, '$'

    STR_LOAD_FUNCTION_NUMBER_ERROR DB 'LOAD ERROR: FUNCTION NUMBER IS
WRONG', 0DH, 0AH, '$'
    STR_LOAD_FILE_NOT_FOUND_ERROR DB 'LOAD ERROR: FILE NOT FOUND',
0DH, 0AH, '$'
    STR_LOAD_DISK_ERROR DB 'LOAD ERROR: PROBLEM WITH DISK', 0DH, 0AH,
'$'
    STR_LOAD_MEMORY_ERROR DB 'LOAD ERROR: NOT ENOUGH MEMORY', 0DH,
0AH, '$'
    STR_LOAD_PATH_ERROR DB 'LOAD ERROR: WRONG PATH PARAM', 0DH, 0AH,
'$'
    STR_LOAD_FORMAT_ERROR DB 'LOAD ERROR: WRONG FORMAT', 0DH, 0AH, '$'

    STR_EXIT DB 'PROGRAMM WAS FINISHED: EXIT WITH CODE:      ', 0DH,
0AH, '$'
    STR_EXIT_CTRL_C DB 'EXIT WITH CTRL+BREAK', 0DH, 0AH, '$'
    STR_EXIT_ERROR DB 'EXIT WITH DEVICE ERROR', 0DH, 0AH, '$'
    STR_EXIT_INT31H DB 'EXIT WITH INT 31H', 0DH, 0AH, '$'

    DATA_END DB 0
DATA ENDS
```

#### CODE SEGMENT

```
    ASSUME CS:CODE, DS:DATA, SS:ASTACK
```

```

;-----
PRINT_MESSAGE PROC NEAR
    PUSH AX
    MOV AH, 9
    INT 21H
    POP AX
    RET
PRINT_MESSAGE ENDP

PRINT_EOF PROC NEAR
    PUSH AX
    PUSH DX
    MOV DL, 0DH
    PUSH AX
    MOV AH, 02H
    INT 21H
    POP AX
    MOV DL, 0AH
    PUSH AX
    MOV AH, 02H
    INT 21H
    POP AX
    POP DX
    POP AX
    RET
PRINT_EOF ENDP
;-----
;-----
FREE_MEMORY_PROC PROC FAR
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH ES
    XOR DX, DX
    MOV FREE_MEMORY, 0H
    MOV AX, OFFSET DATA_END
    MOV BX, OFFSET FINISH
    ADD AX, BX
    MOV BX, 10H
    DIV BX
    ADD AX, 100H
    MOV BX, AX
    XOR AX, AX
    MOV AH, 4AH
    INT 21H
    JNC FREE_MEMORY_SUCCESSFULLY
    MOV FREE_MEMORY, 1H
    CMP AX, 7
    JNE FREE_MEMORY_NOT_ENOUGH_ERROR
    MOV DX, OFFSET STR_FREE_MEMORY_MCB_ERROR
    CALL PRINT_MESSAGE
    JMP FREE_MEMORY_EXIT

FREE_MEMORY_NOT_ENOUGH_ERROR:

```



```

    CMP AX, 8
    JNE FREE_MEMORY_ADDRESS_ERROR
    MOV DX, OFFSET STR_FREE_MEMORY_NOT_ENOUGH_ERROR
    CALL PRINT_MESSAGE
    JMP FREE_MEMORY_EXIT

```

```

FREE_MEMORY_ADDRESS_ERROR:
    CMP AX, 9
    JNE FREE_MEMORY_EXIT
    MOV DX, OFFSET STR_FREE_MEMORY_ADDRESS_ERROR
    CALL PRINT_MESSAGE
    JMP FREE_MEMORY_EXIT

```

```

FREE_MEMORY_SUCCESSFULLY:
    MOV DX, OFFSET STR_FREE_MEMORY_SUCCESSFULLY
    CALL PRINT_MESSAGE
    JMP FREE_MEMORY_EXIT

```

```

FREE_MEMORY_EXIT:
    POP ES
    POP DX
    POP CX
    POP BX
    POP AX
    RET

```

```

FREE_MEMORY_PROC ENDP

```

```

;-----

```

```

;-----

```

```

LOAD PROC FAR
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH DS
    PUSH ES
    MOV KEEP_SP, SP
    MOV KEEP_SS, SS
    CALL PATH_BEGIN
    MOV AX, DATA
    MOV ES, AX
    MOV BX, OFFSET PARAM_BLOCK
    MOV DX, OFFSET NEXT_COMMAND_LINE
    MOV COMMAND_OFF, DX
    MOV COMMAND_SEG, DS
    MOV DX, OFFSET FILE_PATH
    MOV AX, 4B00H
    INT 21H
    MOV SS, KEEP_SS
    MOV SP, KEEP_SP
    POP ES
    POP DS
    CALL PRINT_EOF
    JNC LOAD_SUCCESSFULLY
    CMP AX, 1
    JE LOAD_FUNCTION_NUMBER_ERROR
    CMP AX, 2

```

```

        JE LOAD_FILE_NOT_FOUND_ERROR
        CMP AX, 5
        JE LOAD_DISK_ERROR
        CMP AX, 8
        JE LOAD_MEMORY_ERROR
        CMP AX, 10
        JE LOAD_PATH_ERROR
        CMP AX, 11
        JE LOAD_FORMAT_ERROR

LOAD_FUNCTION_NUMBER_ERROR:
        MOV DX, OFFSET STR_LOAD_FUNCTION_NUMBER_ERROR
        CALL PRINT_MESSAGE
        JMP LOAD_EXIT

LOAD_FILE_NOT_FOUND_ERROR:
        MOV DX, OFFSET STR_LOAD_FILE_NOT_FOUND_ERROR
        CALL PRINT_MESSAGE
        JMP LOAD_EXIT

LOAD_DISK_ERROR:
        MOV DX, OFFSET STR_LOAD_DISK_ERROR
        CALL PRINT_MESSAGE
        JMP LOAD_EXIT

LOAD_MEMORY_ERROR:
        MOV DX, OFFSET STR_LOAD_MEMORY_ERROR
        CALL PRINT_MESSAGE
        JMP LOAD_EXIT

LOAD_PATH_ERROR:
        MOV DX, OFFSET STR_LOAD_PATH_ERROR
        CALL PRINT_MESSAGE
        JMP LOAD_EXIT

LOAD_FORMAT_ERROR:
        MOV DX, OFFSET STR_LOAD_FORMAT_ERROR
        CALL PRINT_MESSAGE
        JMP LOAD_EXIT

LOAD_SUCCESSFULLY:
        MOV AX, 4D00H
        INT 21H
        CMP AH, 0
        JNE EXIT_CTRL_C
        MOV DI, OFFSET STR_EXIT
        ADD DI, 41
        MOV [DI], AL
        MOV DX, OFFSET STR_EXIT
        CALL PRINT_MESSAGE
        JMP LOAD_EXIT

EXIT_CTRL_C:
        CMP AH, 1
        JNE EXIT_ERROR
        MOV DX, OFFSET STR_EXIT_CTRL_C
        CALL PRINT_MESSAGE

```

```

        JMP LOAD_EXIT

EXIT_ERROR:
    CMP AH, 2
    JNE EXIT_INT31H
    MOV DX, OFFSET STR_EXIT_ERROR
    CALL PRINT_MESSAGE
    JMP LOAD_EXIT

EXIT_INT31H:
    CMP AH, 3
    JNE LOAD_EXIT
    MOV DX, OFFSET STR_EXIT_INT31H
    CALL PRINT_MESSAGE
    JMP LOAD_EXIT

LOAD_EXIT:
    POP DX
    POP CX
    POP BX
    POP AX
    RET
LOAD ENDP
;-----
;-----

PATH_BEGIN PROC NEAR
    PUSH AX
    PUSH DX
    PUSH ES
    PUSH DI
    XOR DI, DI
    MOV AX, ES:[2CH]
    MOV ES, AX

LOOP_FOR_PATH_BEGIN:
    MOV DL, ES:[DI]
    CMP DL, 0
    JE GO_TO_PATH
    INC DI
    JMP LOOP_FOR_PATH_BEGIN

GO_TO_PATH:
    INC DI
    MOV DL, ES:[DI]
    CMP DL, 0
    JNE LOOP_FOR_PATH_BEGIN
    CALL PATH
    POP DI
    POP ES
    POP DX
    POP AX
    RET
PATH_BEGIN ENDP

PATH PROC NEAR
    PUSH AX

```

```

        PUSH BX
        PUSH BP
        PUSH DX
        PUSH ES
        PUSH DI
        MOV BX, OFFSET FILE_PATH
        ADD DI, 3

LOOP_FOR_SYMBOL_BOOT:
        MOV DL, ES:[DI]
        MOV [BX], DL
        CMP DL, '.'
        JE LOOP_FOR_SYMBOL_SLASH
        INC DI
        INC BX
        JMP LOOP_FOR_SYMBOL_BOOT

LOOP_FOR_SYMBOL_SLASH:
        MOV DL, [BX]
        CMP DL, '\'
        JE GET_FILE_NAME
        MOV DL, 0H
        MOV [BX], DL
        DEC BX
        JMP LOOP_FOR_SYMBOL_SLASH

GET_FILE_NAME:
        MOV DI, OFFSET FILE_NAME
        INC BX

ADD_FILE_NAME:
        MOV DL, [DI]
        CMP DL, 0H
        JE PATH_EXIT
        MOV [BX], DL
        INC BX
        INC DI
        JMP ADD_FILE_NAME

PATH_EXIT:
        MOV [BX], DL
        POP DI
        POP ES
        POP DX
        POP BP
        POP BX
        POP AX
        RET
PATH ENDP
;-----

;-----
MAIN PROC FAR
        MOV AX, DATA
        MOV DS, AX
        CALL FREE_MEMORY_PROC
        CMP FREE_MEMORY, 0H

```

```
JNE MAIN_EXIT  
CALL PATH_BEGIN  
CALL LOAD
```

```
MAIN_EXIT:  
    XOR AL, AL  
    MOV AH, 4CH  
    INT 21H  
MAIN ENDP
```

```
FINISH:  
CODE ENDS  
END MAIN
```