

**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**

Студент гр. 9382

\_\_\_\_\_

Юрьев С.Ю.

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 проверить причину завершения и, в зависимости от значения, выводить соответствующее сообщение. Если причина завершения 0, то выводится код завершения.

В качестве вызываемой программы необходимо взять программа ЛР2, 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 полученный результат 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.

**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.

#### **Ход работы:**

**Шаг 1.** Был разработан и отлажен программный модуль типа .exe, который выполняет функции, заданные в задании.

**Шаг 2.** Программа lab6.exe была запущена из директории с разработанными модулями. Был введен символ f.

```
C:\>LINK LAB6.OBJ

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

Run File [LAB6.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:

C:\>LAB6..EXE

Memory was free
Segment address of the invalid memory: 9FFF
Segment address of the enviroment: 01FE
Command line arguments:
Content of the enviroment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of the module: C:\LAB2.COM

The program successfully ended with code: f

C:\>
```

Рис. 1: Результат работы программы при вводе буквы f.

**Шаг 3.** Программа lab6.exe была запущена из директории с разработанными модулями. Была введена комбинация символов Ctrl + C. В виду того, что в DOSBOX не реализована обработка данной комбинации, Ctrl+C – символ сердца.

```
Segment address of the enviroment: 01FE
Command line arguments:
Content of the enviroment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of the module: C:\LAB2.COM

The program successfully ended with code: f

C:\>LAB6.EXE

Memory was free
Segment address of the invalid memory: 9FFF
Segment address of the enviroment: 01FE
Command line arguments:
Content of the enviroment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
♥ath of the module: C:\LAB2.COM

The program successfully ended with code: ♥
```

Рис. 2: Результат работы программы при вводе комбинации клавиш Ctrl+C.

**Шаг 4.** Программа запущена из другой директории.

```

C:\NEW_DI~1>LAB6.EXE

Memory was free
Error! File not found!

C:\NEW_DI~1>LAB6.EXE

Memory was free
Segment address of the invalid memory: 9FFF
Segment address of the enviroment: 01FE
Command line arguments:
Content of the enviroment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of the module: C:\NEW_DI~1\LAB2.COM
The program successfully ended with code: f

```

Рис. 3:

Результат работы программы из другой директории (LAB2.COM и LAB6.EXE находятся в одной директории)

**Шаг 5.** Программа запущена, когда программный и загрузочный модули находятся в разных директориях.

```

C:\NEW_DI~1>LAB6.EXE

Memory was free
Error! File not found!

```

Рис.4 Результат работы программы из другой директории (LAB2.COM и LAB6.EXE находятся в разных директориях)

### Контрольные вопросы.

1) Как реализовано прерывание Ctrl-C?

Ответ: При нажатии этой комбинации управление передается по адресу 0000:008Ch. Этот адрес копируется в PSP функциями 26h и 4Ch и восстанавливается из PSP при выходе из программы.

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

Ответ: Если код завершения 0, то программа завершается при выполнении функции 4Ch прерывания int 21h

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

Ответ: Если во время выполнения программы была нажата эта комбинация, то программа завершится непосредственно в том месте, где было произведено нажатие (т. е. в месте ожидания нажатия клавиши: 01h вектора прерывания 21h)

### **Выводы.**

Была исследована возможность построения загрузочного модуля динамической структуры.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

#### Lab6.asm:

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

DATA SEGMENT

    P_BLOCK DW 0
                DD 0
                DD 0
                DD 0

    FILE_NAME DB 'LAB2.COM', 0
    FLAG DB 0
    CMD DB 1H, 0DH
    POS DB 128 DUP(0)

    KEEP_SS DW 0
    KEEP_SP DW 0
    KEEP_PSP DW 0

    STR_MEMORY_FREE DB 'MEMORY WAS FREE' , 0DH, 0AH, '$'

    STR_ERROR_CRASH DB 'ERROR! MCB CRASHED!', 0DH, 0AH, '$'
    STR_ERROR_NO_MEMORY DB 'ERROR! NOT ENOUGH MEMORY!', 0DH, 0AH, '$'
    STR_WRONG_ADDRESS DB 'ERROR! INVALID MEMORY ADDRESSES!', 0DH, 0AH, '$'
    STR_ERROR_NUMBER DB 'ERROR! INVALID FUNCTION NUMBER!', 0DH, 0AH, '$'
    STR_ERROR_NO_FILE DB 'ERROR! FILE NOT FOUND!', 0DH, 0AH, '$'
    STR_DISK_ERROR DB 'ERROR WITH DISK!', 0DH, 0AH, '$'
    STR_MEMORY_ERROR DB 'ERROR! INSUFFICIENT MEMORY!', 0DH, 0AH, '$'
    STR_ENVIRONMENT_ERROR DB 'ERROR! WRONG STRING OF ENVIRONMENT!', 0DH,
0AH, '$'
    STR_FORMAT_ERROR DB 'ERROR! WRONG FORMAT!', 0DH, 0AH, '$'
    STR_ERROR_DEVICE DB 0DH, 0AH, 'ERROR! DEVICE ERROR!' , 0DH, 0AH, '$'

    STR_END_CODE DB 0DH, 0AH, 'THE PROGRAM SUCCESSFULLY ENDED WITH CODE:
' , 0DH, 0AH, '$'
    STR_END_CTR DB 0DH, 0AH, 'THE PROGRAM WAS INTERED BY CTRL-BREAK' , 0DH,
0AH, '$'
```

```
STR_END_INTER DB 0DH, 0AH, 'THE PROGRAM WAS ENDED BY INTERION INT
31H' , 0DH, 0AH, '$'
```

```
NEW_STRING DB 0DH,0AH,'$'
```

```
DATA_END DB 0
DATA ENDS
```

```
CODE SEGMENT
```

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

```
WRITESTRING PROC
```

```
PUSH AX
MOV AH, 09H
INT 21H
POP AX
RET
```

```
WRITESTRING ENDP
```

```
MEMORY_FREE PROC
```

```
PUSH AX
PUSH BX
PUSH CX
PUSH DX
```

```
MOV AX, OFFSET DATA_END
MOV BX, OFFSET PR_END
ADD BX, AX
MOV CL, 4
SHR BX, CL
ADD BX, 2BH
MOV AH, 4AH
INT 21H
JNC END_FREE_MEMORY
MOV FLAG, 1
```

```
CRASH_MCB:
```

```
CMP AX, 7
JNE NOT_MEMORY
MOV DX, OFFSET STR_ERROR_CRASH
```



```

        CALL WRITESTRING
        JMP RET_F

NOT_MEMORY:
        CMP AX, 8
        JNE WRONG_ADDRESS
        MOV DX, OFFSET STR_ERROR_NO_MEMORY
        CALL WRITESTRING
        JMP RET_F

WRONG_ADDRESS:
        CMP AX, 9
        MOV DX, OFFSET STR_WRONG_ADDRESS
        CALL WRITESTRING
        JMP RET_F

END_FREE_MEMORY:
        MOV FLAG, 1
        MOV DX, OFFSET NEW_STRING
        CALL WRITESTRING
        MOV DX, OFFSET STR_MEMORY_FREE
        CALL WRITESTRING

RET_F:
        POP DX
        POP CX
        POP BX
        POP AX
        RET

MEMORY_FREE 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 P_BLOCK
MOV DX, OFFSET CMD
MOV [BX+2], DX
MOV [BX+4], DS
MOV DX, OFFSET POS
MOV AX, 4B00H
INT 21H

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

CMP AX, 1
JNE FILE_ERROR

MOV DX, OFFSET STR_ERROR_NUMBER
CALL WRITESTRING

JMP END_LOAD

FILE_ERROR:
CMP AX, 2
JNE DISK_ERROR
MOV DX, OFFSET STR_ERROR_NO_FILE
CALL WRITESTRING
JMP END_LOAD

DISK_ERROR:
CMP AX, 5
JNE MEMORY_ERROR
MOV DX, OFFSET STR_DISK_ERROR
CALL WRITESTRING
JMP END_LOAD

MEMORY_ERROR:
CMP AX, 8
JNE ENVIRONMENT_ERROR

```

```

    MOV DX, OFFSET STR_MEMORY_ERROR
    CALL WRITESTRING
    JMP END_LOAD

ENVIRONMENT_ERROR:
    CMP AX, 10
    JNE FORMAT_ERROR
    MOV DX, OFFSET STR_ENVIRONMENT_ERROR
    CALL WRITESTRING
    JMP END_LOAD

FORMAT_ERROR:
    CMP AX, 11
    MOV DX, OFFSET STR_FORMAT_ERROR
    CALL WRITESTRING
    JMP END_LOAD

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

    CMP AH, 0
    JNE CTRL
    PUSH DI
    MOV DI, OFFSET STR_END_CODE
    MOV [DI+44], AL
    POP SI
    MOV DX, OFFSET NEW_STRING
    CALL WRITESTRING
    MOV DX, OFFSET STR_END_CODE
    CALL WRITESTRING
    JMP END_LOAD

CTRL:
    CMP AH, 1
    JNE DEVICE
    MOV DX, OFFSET STR_END_CTR
    CALL WRITESTRING
    JMP END_LOAD

DEVICE:

```

```

        CMP AH, 2
        JNE INTER
        MOV DX, OFFSET STR_ERROR_DEVICE
        CALL WRITESTRING
        JMP END_LOAD

INTER:
        CMP AH, 3
        MOV DX, OFFSET STR_END_INTER
        CALL WRITESTRING

END_LOAD:
        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

FIND_PATH:
        INC BX
        CMP BYTE PTR ES:[BX-1], 0
        JNE FIND_PATH
        CMP BYTE PTR ES:[BX+1], 0
        JNE FIND_PATH
        ADD BX, 2

```

```

        MOV DI, 0

LOOP_P:
        MOV DL, ES:[BX]
        MOV BYTE PTR [POS + DI], DL
        INC DI
        INC BX
        CMP DL, 0
        JE END_LOOP_P
        CMP DL, '\'
        JNE LOOP_P
        MOV CX, DI
        JMP LOOP_P

END_LOOP_P:
        MOV DI, CX
        MOV SI, 0

END_F:
        MOV DL, BYTE PTR [FILE_NAME + SI]
        MOV BYTE PTR [POS + DI], DL
        INC DI
        INC SI
        CMP DL, 0
        JNE END_F

        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 MEMORY_FREE
        CMP FLAG, 0
        JE FINISH
        CALL PATH
        CALL LOAD

FINISH:
        XOR AL, AL
        MOV AH, 4CH
        INT 21H

BEGIN ENDP

PR_END:
CODE ENDS
END BEGIN

```