

**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. 9381

\_\_\_\_\_

Давыдов Д.С.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2021

### **Цель работы.**

Исследование возможности построения загрузочного модуля динамической структуры путем разработки приложения, состоящего из нескольких модулей вместо одного модуля простой 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.

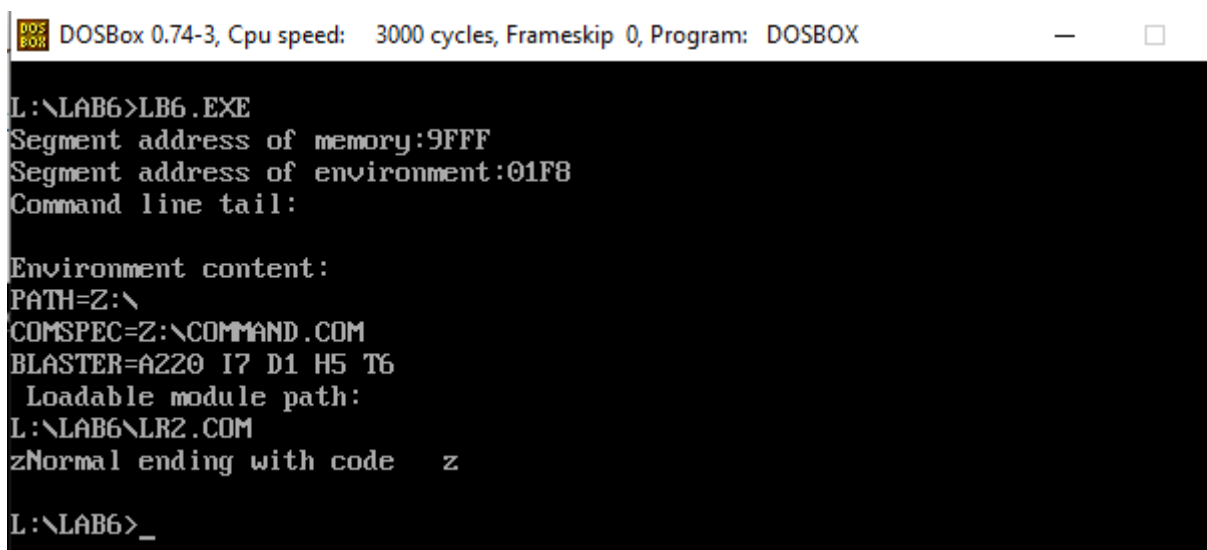
Шаг 4. Запустите отлаженную программу, когда текущим каталогом является какой-либо другой каталог, отличный от того, в котором содержатся разработанные программные модули.

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.

### Ход работы.

Была доработана программа второй лабораторной работы. После изменения программа перед закрытием будет ожидать символ с клавиатуры. Запустим программу и введем символ z:

A screenshot of a DOSBox window titled "DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX". The window shows a command prompt with the following text:

```
L:\LAB6>LB6.EXE
Segment address of memory:9FFF
Segment address of environment:01F8
Command line tail:

Environment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Loadable module path:
L:\LAB6\LR2.COM
zNormal ending with code z

L:\LAB6>_
```

Далее запустим программу, завершив ее нажатием Ctrl + C.

```
L:\LAB6>LB6.EXE
Segment address of memory:9FFF
Segment address of environment:01F8
Command line tail:

Environment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Loadable module path:
L:\LAB6\LR2.COM
♥Normal ending with code ♥

L:\LAB6>_
```

Поскольку DOSBox не поддерживает данную комбинацию — буден введен обычный символ и мы увидим его в результате работы программы.

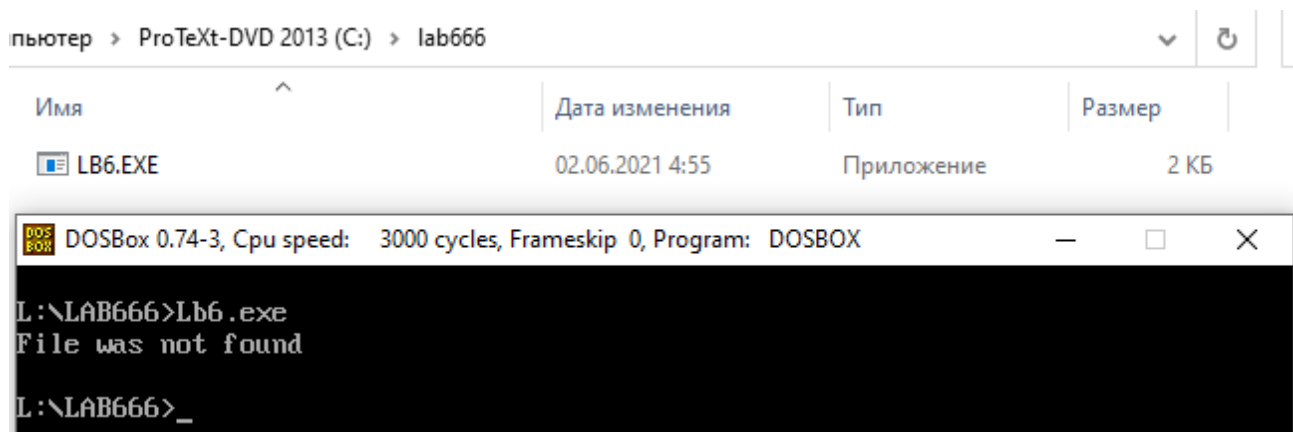
Далее изменим расположение обоих загрузочных модулей, поместив их в новый каталог, и снова запустим LB6.EXE. Результат останется прежним — программа успешно завершиться.

```
L:\LAB6>cd ..
L:\>cd lab666
L:\LAB666>lb6.exe
Segment address of memory:9FFF
Segment address of environment:01F8
Command line tail:

Environment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Loadable module path:
L:\LAB666\LR2.COM
oNormal ending with code o

L:\LAB666>_
```

Теперь, изменим расположением загрузочных файлов, поместив их в разные папки. Программа завершиться с сообщением о том, что файл LR2.com не был найден.



Разработанный программный смотреть в приложении А.

### **Ответы на контрольные вопросы.**

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

**Ответ:** DOS вызывает прерывание int 23h, когда пользователь вводит Ctrl+C и управление передается по адресу 0000:008Ch. Адрес по вектору int 23h копируется в PSP функциями DOS 26H и 4Ch и восстанавливается из PSP при завершении программы.

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

**Ответ:** Если код причины 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 (у нас в примере это происходит в месте ожидания нажатия клавиши: 01h вектора прерывания 21h).

**Выводы.**

В ходе выполнения лабораторной работы была написана программа реализующая возможность построения загрузочного модуля динамической структуры.

# ПРИЛОЖЕНИЕ А.

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb6.asm

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

```
LAB_STACK SEGMENT STACK
```

```
    DW 128 DUP(?)
```

```
LAB_STACK ENDS
```

```
DATA    SEGMENT
```

```
    paramsBlock    dw 0
```

```
                                dd 0
```

```
                                dd 0
```

```
                                dd 0
```

```
    programName db 'LR2.COM', 0
```

```
    cmd db 1h, 0dh
```

```
    pos db 128 dup(0)
```

```
    ssKeep dw 0
```

```
    spKeep dw 0
```

```
    pspKeep dw 0
```

```
    memoryDestroyed db 'Destroyed memory block',13,10,'$'
```

```
    memoryLowMemFunc    db    'Not    enough    memory    for    running
function',13,10,'$'
```

```
    memoryWrongMemAdr db 'Incorrect memory address',13,10,'$'
```

```
    errorWrongFuncNum  db 'Wrong function number',13,10,'$'
```

```
    errorMissFile  db 'File was not found',13,10,'$'
```

```
    errorDisk  db 'Disk error',13,10,'$'
```

```
    errorNotEnoughMem  db 'Not enough free disk memory space',13,10,'$'
```

```
    errorWrongStrFormat db 'Wrong string enviroment',13,10,'$'
```

```
    errorWrongFormat db 'Wrong format',13,10,'$'
```

```
    endSuccess db 'Normal ending with code    ',13,10,'$'
```

```
    endCtrlBreak db 'Program was interrupted by ctrl-break',13,10,'$'  
    endDeviceError db 'Program was ended with device error',13,10,'$'  
    endInterruption db 'Program was ended by int 31h  
interruption',13,10,'$'
```

```
    end_data db 0  
DATA ENDS
```

CODE SEGMENT

PRINT PROC

```
    push ax  
    mov ah, 09h  
    int 21h  
    pop ax  
    ret
```

PRINT ENDP

FREE\_MEMORY PROC

```
    push ax  
    push bx  
    push cx  
    push dx  
  
    mov ax, offset end_data  
    mov bx, offset END_APP  
    add bx, ax  
    shr bx, 1  
    shr bx, 1  
    shr bx, 1  
    shr bx, 1  
    add bx, 2bh  
    mov ah, 4ah  
    int 21h
```

jnc END\_FREE\_MEMORY

```
    lea dx, memoryDestroyed  
    cmp ax, 7
```



```

je WRITE_MEMORY_COMMENT
lea dx, memoryLowMemFunc
cmp ax, 8
je WRITE_MEMORY_COMMENT
lea dx, memoryWrongMemAdr
cmp ax, 9
je WRITE_MEMORY_COMMENT
jmp END_FREE_MEMORY

```

WRITE\_MEMORY\_COMMENT:

```

mov ah, 09h
int 21h

```

END\_FREE\_MEMORY:

```

pop dx
pop cx
pop bx
pop ax
ret

```

FREE\_MEMORY ENDP

SET\_FULL\_PROG\_NAME PROC NEAR

```

push ax
push bx
push cx
push dx
push di
push si
push es

```

```

mov ax, pspKeep
mov es, ax
mov es, es:[2ch]
mov bx, 0

```

FIND\_SMTH:

```

inc bx
cmp byte ptr es:[bx-1], 0
jne FIND_SMTH

```

```
    cmp byte ptr es:[bx+1], 0
    jne FIND_SMTH
```

```
    add bx, 2
    mov di, 0
```

FIND\_LOOP:

```
    mov dl, es:[bx]
    mov byte ptr [pos + di], dl
    inc di
    inc bx
    cmp dl, 0
    je END_LOOP
    cmp dl, '\'
    jne FIND_LOOP
    mov cx, di
    jmp FIND_LOOP
```

END\_LOOP:

```
    mov di, cx
    mov si, 0
```

LOOP\_2:

```
    mov dl, byte ptr[programName + si]
    mov byte ptr [pos + di], dl
    inc di
    inc si
    cmp dl, 0
    jne LOOP_2
```

```
    pop es
    pop si
    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    ret
```

SET\_FULL\_PROG\_NAME ENDP

DEPLOY\_ANOTHER\_PROGRAM PROC NEAR

```
    push ax
    push bx
    push cx
    push dx
    push ds
    push es
    mov spKeep, sp
    mov ssKeep, ss
    mov ax, DATA
    mov es, ax
    mov dx, offset cmd
    mov bx, offset paramsBlock
    mov [bx+2], dx
    mov [bx+4], ds
    mov dx, offset pos

    mov ax, 4B00h
    int 21h

    mov ss, ssKeep
    mov sp, spKeep

    pop es
    pop ds

    jnc END_SUCCESS
```

ERROR\_1:

```
    cmp ax, 1
    jne ERROR_2
    mov dx, offset errorWrongFuncNum
    call PRINT
    jmp DEPLOY_END
```

ERROR\_2:

```
    cmp ax, 2
    jne ERROR_3
    mov dx, offset errorMissFile
```

```
    call PRINT
    jmp DEPLOY_END
```

```
ERROR_3: ;5
    cmp ax, 5
    jne ERROR_4
    mov dx, offset errorDisk
    call PRINT
    jmp DEPLOY_END
```

```
ERROR_4: ;8
    cmp ax, 8
    jne ERROR_5
    mov dx, offset errorNotEnoughMem
    call PRINT
    jmp DEPLOY_END
```

```
ERROR_5: ;10
    cmp ax, 10
    jne ERROR_6
    mov dx, offset errorWrongStrFormat
    call PRINT
    jmp DEPLOY_END
```

```
ERROR_6: ;11
    cmp ax, 11
    mov dx, offset errorWrongFormat
    call PRINT
    jmp DEPLOY_END
```

```
END_SUCCESS:
    mov ax, 4D00h
    int 21h

    cmp ah, 0
    jne END_1
    push di
    mov di, offset endSuccess
    mov [di+26], al
```

```
    pop si
    mov dx, offset endSuccess
    call PRINT
    jmp DEPLOY_END
```

```
END_1:
    cmp ah, 1
    jne END_2
    mov dx, offset endCtrlBreak
    call PRINT
    jmp DEPLOY_END
```

```
END_2:
    cmp ah, 2
    jne END_3
    mov dx, offset endDeviceError
    call PRINT
    jmp DEPLOY_END
```

```
END_3:
    cmp ah, 3
    mov dx, offset endInterruption
    call PRINT
```

```
DEPLOY_END:
    pop dx
    pop cx
    pop bx
    pop ax
    ret
```

```
DEPLOY_ANOTHER_PROGRAM ENDP
```

```
MAIN PROC FAR
    push ds
    xor  ax, ax
    push ax
    mov  ax, DATA
    mov  ds, ax
    mov  pspKeep, es
```

```
call FREE_MEMORY  
call SET_FULL_PROG_NAME  
call DEPLOY_ANOTHER_PROGRAM
```

```
VERY_END:  
    xor al,al  
    mov ah,4ch  
    int 21h  
MAIN ENDP
```

```
END_APP:
```

```
CODE ENDS  
    END MAIN
```