

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

ЛАБОРАТОРНАЯ РАБОТА № 6
по дисциплине «Операционные системы»
ТЕМА: Построение модуля динамической структуры.

Студентка гр. 9383

Сергиенкова А.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры. В отличие от предыдущих лабораторных работ в этой работе рассматривается приложение, состоящее из нескольких модулей, а не из одного модуля простой структуры. В этом случае разумно предположить, что все модули приложения находятся в одном каталоге и полный путь в этот каталог можно взять из среды, как это делалось в работе 2. Понятно, что такое приложение должно запускаться в соответствии со стандартами ОС.

В работе исследуется интерфейс между вызывающим и вызываемым модулями по управлению и по данным. Для запуска вызываемого модуля используется функция 4B00h прерывания int 21h. Все загрузочные модули находятся в одном каталоге. Необходимо обеспечить возможность запуска модуля динамической структуры из любого каталога.

Ход работы.

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

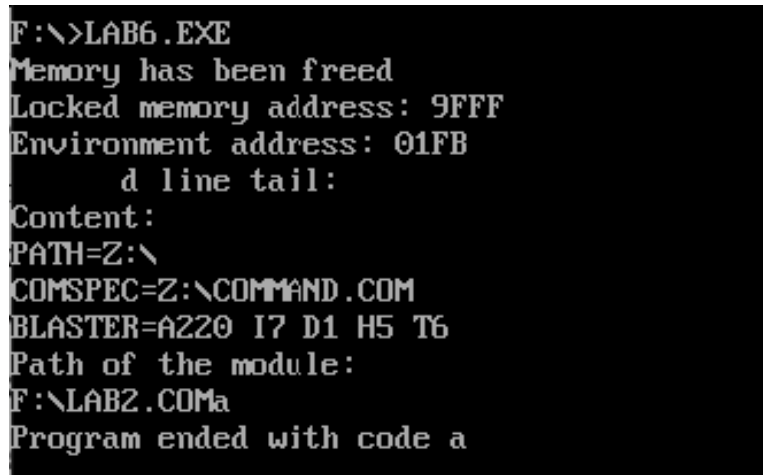
1. Подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором находится он сам. Вызываемому модулю передается новая среда, созданная вызывающим модулем и новая командная строка.

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.

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.



```
F:\>LAB6.EXE
Memory has been freed
Locked memory address: 9FFF
Environment address: 01FB
      d line tail:
Content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of the module:
F:\LAB2.COM
Program ended with code a
```

Рисунок 1 – Запуск отлаженной программы.

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.

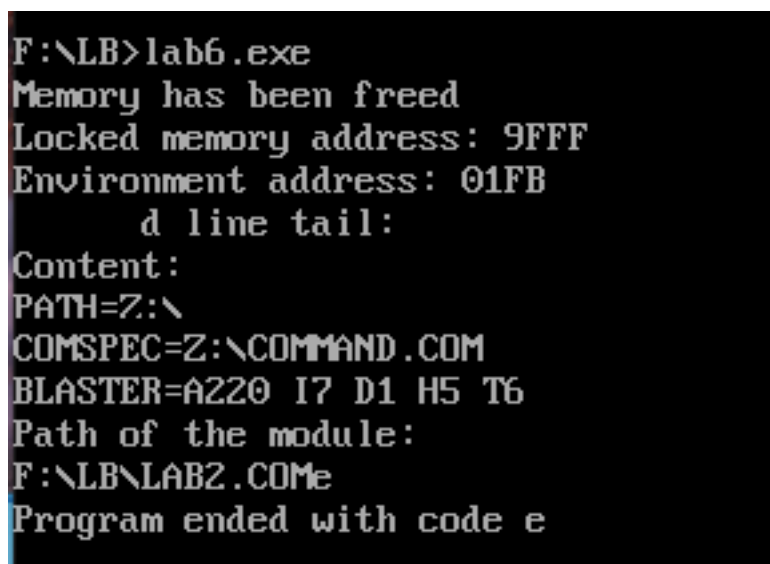


```
F:\>LAB6.EXE
Memory has been freed
Locked memory address: 9FFF
Environment address: 01FB
      d line tail:
Content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of the module:
F:\LAB2.COM
Program ended with code ♥
```

Рисунок 2 - Запуск отлаженной программы с введенной комбинацией.

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.

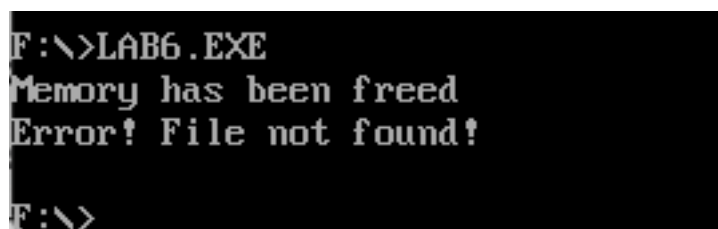


```
F:\LB>lab6.exe
Memory has been freed
Locked memory address: 9FFF
Environment address: 01FB
      d line tail:
Content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of the module:
F:\LB\LAB2.COMe
Program ended with code e
```

Рисунок 3 - Запуск отлаженной программы из другого каталога

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

Файл не был найден.



```
F:\>LAB6.EXE
Memory has been freed
Error! File not found!
F:\>
```

Рисунок 4 – Запуск отлаженной программы, все модули в разных каталогах.

Вывод:

Был построен и исследован загрузочный модуль динамической структуры. Также исследован интерфейс между вызывающим и вызываемым

модулями по управлению и по данным.

Ответы на контрольные вопросы:

1. Как реализовано прерывание Ctrl+C?

При нажатии **Ctrl+C** срабатывает прерывание int 23h. Далее управление происходит по адресу 0000:008C. Этот адрес копируется в поле PSP функциями DOS 26h, которая создает PSP, и 4Ch. Исходное значение адреса обработчика Ctrl+C восстанавливается из PSP при завершении программы.

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

Программа завершается при выполнении функции 4ch прерывания int 21h.

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

Программа завершается в месте ожидания нажатия клавиши, сразу после обработки прерывания по Ctrl+C.

ПРИЛОЖЕНИЕ А

КОД ИСХОДНОЙ ПРОГРАММЫ

Lab6.asm

STACK segment stack

dw 128 dup(?)

STACK ends

DATA segment

program db 'lab2.com', 0

block_parameters dw 0

dd 0

dd 0

dd 0

mem_flag db 0

cmd db 1h, 0dh

pos db 128 dup(0)

keep_ss dw 0

keep_sp dw 0

keep_psp dw 0

ERROR_CRASH db 'Error! MCB crashed!', 0dh, 0ah, '\$'

ERROR_ADDRESS db 'Error! Invalid memory address!', 0dh, 0ah, '\$'

NO_MEMORY db 'Error! Not enough memory!', 0dh, 0ah, '\$'

NO_FILE db 'Error! File not found!', 0dh, 0ah, '\$'

ERROR_FUNCTION_NUMBER db 'Error! Invalid function number', 0dh, 0ah, '\$'

ERROR_DISK db 'Error! Disk error!', 0dh, 0ah, '\$'

ERROR_MEMORY db 'Error! Insufficient memory', 0dh, 0ah, '\$'

ERROR_ENV db 'Error! Wrong string of environment ', 0dh, 0ah, '\$'

WRONG_FORM db 'Error! Wrong format', 0dh, 0ah, '\$'

ERROR_DEFICE db 0dh, 0ah, 'Program ended by device error' , 0dh, 0ah, '\$'

FREE_MEMORY db 'Memory has been freed' , 0dh, 0ah, '\$'

END_CODE db 0dh, 0ah, 'Program ended with code ' , 0dh, 0ah, '\$'

END_CTRL db 0dh, 0ah, 'Program ended by CTRL-break' , 0dh, 0ah, '\$'

END_31 db 0dh, 0ah, 'Program was ended by int 31h' , 0dh, 0ah, '\$'

END_DATA db 0

DATA ends

CODE segment

ASSUME cs:CODE, ds:DATA, ss:STACK

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

```
MEM_FREE proc
    push ax
    push bx
    push cx
    push dx

    mov ax, offset END_DATA
    mov bx, offset FIN
    add bx, ax

    mov cl, 4
    shr bx, cl
    add bx, 2bh
    mov ah, 4ah
    int 21h

    jnc FINISH_MEM_FREE
    mov mem_flag, 1
```

```
CRASH_MCB:
    cmp ax, 7
    jne NOT_MEMORY
    mov dx, offset ERROR_CRASH
    call PRINT
    jmp RET_FUN
```

```
NOT_MEMORY:
    cmp ax, 8
    jne ADDRESS_ERROR
    mov dx, offset NO_MEMORY
    call PRINT
    jmp RET_FUN
```

```
ADDRESS_ERROR:
    cmp ax, 9
    mov dx, offset ERROR_ADDRESS
    call PRINT
    jmp RET_FUN
```

```
FINISH_MEM_FREE:
    mov mem_flag, 1
    mov dx, offset FREE_MEMORY
```

call PRINT

RET_FUN:

pop dx

pop cx

pop bx

pop ax

ret

MEM_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 block_parameters

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 ERROR_FUNCTION_NUMBER

call PRINT

jmp load_end

FILE_ERROR:

cmp ax, 2

jne DISK_ERROR

mov dx, offset NO_FILE


```

        call PRINT
        jmp load_end
DISK_ERROR:
        cmp ax, 5
        jne MEM_ERROR
        mov dx, offset ERROR_DISK
        call PRINT
        jmp load_end
MEM_ERROR:
        cmp ax, 8
        jne ENV_ERROR
        mov dx, offset ERROR_MEMORY
        call PRINT
        jmp load_end
ENV_ERROR:
        cmp ax, 10
        jne FORMAT_ERROR
        mov dx, offset ERROR_ENV
        call PRINT
        jmp load_end
FORMAT_ERROR:
        cmp ax, 11
        mov dx, offset WRONG_FORM
        call PRINT
        jmp load_end

```

loads:

```

        mov ah, 4dh
        mov al, 00h
        int 21h

        cmp ah, 0
        jne CTRL_FUNC
        push di
        mov di, offset END_CODE
        mov [di+26], al
        pop si
        mov dx, offset END_CODE
        call PRINT
        jmp load_end
CTRL_FUNC:
        cmp ah, 1
        jne DEVICE
        mov dx, offset END_CTRL
        call PRINT
        jmp load_end
DEVICE:

```

```

        cmp ah, 2
        jne INT_31
        mov dx, offset ERROR_DEFICE
        call PRINT
        jmp load_end
INT_31:
        cmp ah, 3
        mov dx, offset END_31
        call PRINT

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

```

```

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

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

        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 QUIT_LOOP
cmp dl, '\'
jne FIND_LOOP
mov cx, di
jmp FIND_LOOP
```

QUIT_LOOP:

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

END_FN:

```
mov dl, byte ptr [program+si]
mov byte ptr [pos+di], dl
inc di
inc si
cmp dl, 0
jne END_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 MEM_FREE
cmp mem_flag, 0
je QUIT
call PATH
call LOAD
```

QUIT:

```
xor al, al
mov ah, 4ch
int 21h
```

BEGIN endp

FIN:

CODE ends

end BEGIN