**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 |  | Рыжих Р.В. |
| 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) Вызываемой 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.

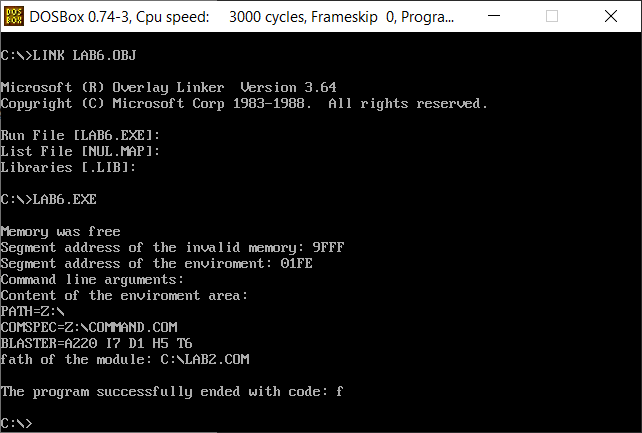


Рис. 1: Результат работы программы при вводе буквы f.

**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.

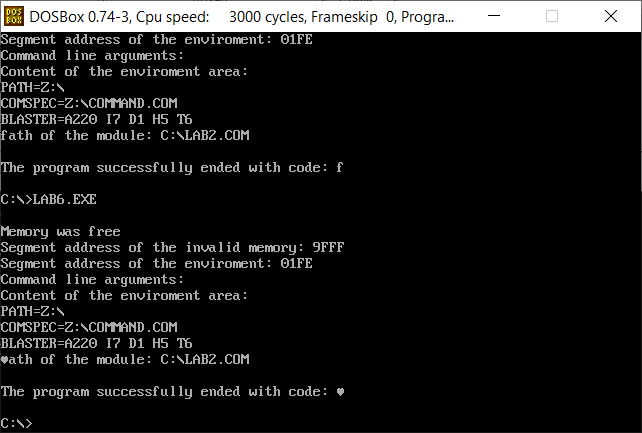


Рис. 2: Результат работы программы при вводе комбинации клавиш Ctrl+C.

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

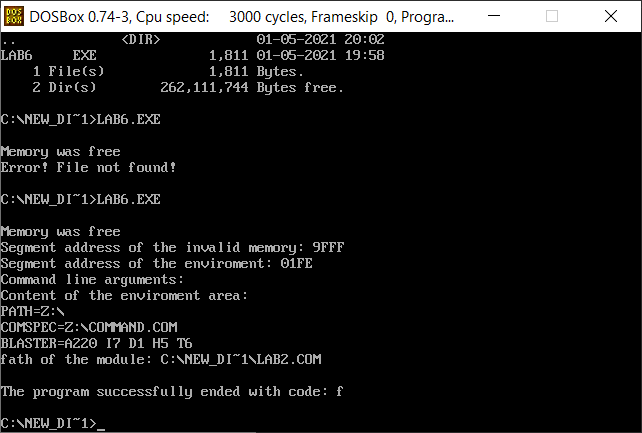


Рис. 3: Результат работы программы из другой директории (LAB2.COM и LAB6.EXE находятся в одной директории)

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

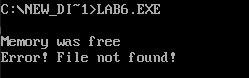


Рис.4 Результат работы программы из другой директории (LAB2.COM и LAB6.EXE находятся в разных директориях)

**Ответы на контрольные вопросы**

1. Как реализовано прерывания Ctrl-C?

При нажатии данной комбинации клавиш управление передается по адресу 0000:008C. Этот адрес копируется в PSP при помощи функций 26h и 4ch. При выходе из программы адрес восстанавливается.

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

При выполнении функции 4ch прерывания int 21h.

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

В том месте, где произошло прерывание, то есть там, где ожидается нажатие клавиш.

**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 addressess!', 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