

**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. 9383

\_\_\_\_\_

Арутюнян С.Н.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2021

## **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.

## Выполнение работы

**Шаг 1.** Был написан и отлажен программный модуль типа .EXE, который выполняет следующие 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. Запускает вызываемый модуль с помощью загрузчика.
3. Проверяет выполнение загрузчика, а затем результат выполнения вызываемой программы.

**Шаг 2.** Программа была запущена, когда текущим каталогом являлся каталог с разработанными модулями и в конце выполнения программы была введена буква g:

```
C:\>lab6.exe lab2.com
Unavailable memory: 9FFF
Environment address: 02CA
Command tail:
Content of the environment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path: C:\LAB2.COMg
The program executed okay!
The program returned: g
C:\>_
```

Рис. 1. Пример работы программы

**Шаг 3.** Программа была запущена, когда текущим каталогом являлся каталог с разработанными модулями и в конце выполнения программы была введена комбинация Ctrl + C:

```
C:\>lab6.exe lab2.com
Unavailable memory: 9FFF
Environment address: 02CA
Command tail:
Content of the environment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path: C:\LAB2.COM
The program executed okay!
The program returned: ♥
C:\>_
```

Рис. 2. Пример работы программы (2)

**Шаг 4.** Программа была запущена, когда текущим каталогом являлся другой каталог и в конце выполнения программы была введена комбинация Ctrl + C и g:

```
Z:\>c:\lab6.exe c:\lab2.com
Unavailable memory: 9FFF
Environment address: 02CA
Command tail:
Content of the environment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path: C:\LAB2.COMg
The program executed okay!
The program returned: g
Z:\>
```

Рис. 3. Пример работы программы (3)

```
Z:\>c:\lab6.exe c:\lab2.com
Unavailable memory: 9FFF
Environment address: 02CA
Command tail:
Content of the environment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path: C:\LAB2.COM♥
The program executed okay!
The program returned: ♥
Z:\>
```

Рис. 4. Пример работы программы (4)

**Шаг 5.** Программа была запущена, когда модули находились в разных каталогах и в конце выполнения программы была введена комбинация Ctrl + C и g:

```
C:\>c:\lab6.exe d:\lab2.com
Unavailable memory: 9FFF
Environment address: 02CA
Command tail:
Content of the environment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path: D:\LAB2.COMg
The program executed okay!
The program returned: g
C:\>
```

Рис. 5. Пример работы программы (5)

```
C:\>c:\lab6.exe d:\lab2.com
Unavailable memory: 9FFF
Environment address: 02CA
Command tail:
Content of the environment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path: D:\LAB2.COM♥
The program executed okay!
The program returned: ♥
C:\>_
```

Рис. 6. Пример работы программы (6)

## **Контрольные вопросы**

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

При нажатии на Ctrl + C вызывается прерывание int 23h, и управление передается коду по адресу 008C.

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

Программа заканчивается в точке вызова прерывания 21h с функцией выхода в DOS.

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

Программа по прерыванию Ctrl + C заканчивается в обработчике прерывания.

## **Заключение**

В процессе выполнения лабораторной работы был изучен механизм работы с динамической загрузкой внешних модулей.

## Приложение А

AStack SEGMENT STACK

dw 256 DUP(?) ; 1 килобайт

AStack ENDS

DATA SEGMENT

NEWLINE db 0dh, 0ah, '\$'

CONTROL\_BLOCK dw 0

db 0

db 0

db 0

COMMAND\_LINE db 1h, 0dh

DESTROYED\_ERROR db "The control memory block was destroyed!", 0dh, 0ah, '\$'

NOT\_ENOUGH\_MEMORY\_ERROR db "The memory is not enogh!", 0dh, 0ah, '\$'

INVALID\_ADDRESS\_ERROR db "The memory block's address is invalid!", 0dh, 0ah, '\$'

INVALID\_FUNCTION\_ERROR db "Invalid function!", 0dh, 0ah, '\$'

FILE\_NOT\_FOUND\_ERROR db "File not found!", 0dh, 0ah, '\$'

DISK\_ERROR db "Disk error!", 0dh, 0ah, '\$'

INVALID\_ENV\_STRING\_ERROR db "Invalid environment string!", 0dh, 0ah, '\$'

INVALID\_FORMAT\_ERROR db "Invalid format!", 0dh, 0ah, '\$'

OKAY\_EXITING\_MESSAGE db "The program executed okay!", 0dh, 0ah, '\$'

CTRL\_BREAK\_ERROR db "The program exited with Ctrl+Break!", 0dh, 0ah, '\$'

DEVICE\_ERROR db "Device error occured!", 0dh, 0ah, '\$'

RESIDENT\_31\_ERROR db "Resident 31h function exiting!", 0dh, 0ah, '\$'

RETURN\_VALUE db "The program returned: '\$"

PROGRAM\_NAME db 64 dup(0), '\$'



KEEP\_SS dw 0

KEEP\_SP dw 0

DATA\_SEGMENT\_END db 0

DATA ENDS

CODE SEGMENT

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

PRINT\_NEWLINE proc near

push ax

push dx

mov dx, offset NEWLINE

mov ah, 9h

int 21h

pop dx

pop ax

ret

PRINT\_NEWLINE endp

WRITE\_STRING proc near

cmp dx, 0

je write\_string\_exit

push ax

mov ah, 9h

int 21h

pop ax

write\_string\_exit:

ret

WRITE\_STRING endp

ALLOCATE\_MEMORY proc near

push bx

push dx

push ax

mov ax, offset main\_procedure\_end

mov bx, offset DATA\_SEGMENT\_END

add ax, bx

mov bx, 10h

xor dx, dx

div bx

mov bx, ax

add bx, dx

add bx, 100h

mov ah, 4ah

int 21h

xor dx, dx

; проверяем возможные ошибки

cmp ax, 7

je write\_destroyed\_error

cmp ax, 8

je not\_enough\_memory

cmp ax, 9

je invalid\_address

jmp allocate\_exit

write\_destroyed\_error:

mov dx, offset DESTROYED\_ERROR

jmp allocate\_exit

not\_enough\_memory:

mov dx, offset NOT\_ENOUGH\_MEMORY\_ERROR

jmp allocate\_exit

invalid\_address:

mov dx, offset INVALID\_ADDRESS\_ERROR

allocate\_exit:

call WRITE\_STRING

pop ax

pop dx

pop bx

ret

ALLOCATE\_MEMORY endp

PARSE\_PROGRAM\_NAME proc near

push ax

push cx

push es

push di

push bx

mov ah, 62h

int 21h

; в bx - адрес начала PSP

mov es, bx

xor cx, cx

mov cl, es:[80h]

dec cl

mov bx, 1

mov di, offset PROGRAM\_NAME

extracting\_program\_name:

mov al, es:[81h + bx]

mov [di], al

inc bx

inc di

loop extracting\_program\_name

pop bx

pop di

pop es

pop cx

pop ax

ret

PARSE\_PROGRAM\_NAME endp

FIIL\_CONTROL\_BLOCK proc near

push bx

push dx

; оставляем первое слово равным 0

mov bx, offset CONTROL\_BLOCK

mov dx, offset COMMAND\_LINE

mov [bx+2], dx

mov [bx+4], ds

pop dx

pop bx

ret

FIIL\_CONTROL\_BLOCK endp

LOAD\_MODULE proc near

push es

push bx

push ax

push dx

call FIIL\_CONTROL\_BLOCK

mov ax, seg CONTROL\_BLOCK

mov es, ax

mov bx, offset CONTROL\_BLOCK

mov ax, sp

mov KEEP\_SP, ax

mov ax, ss

mov KEEP\_SS, ax

mov dx, offset PROGRAM\_NAME

mov ax, 4b00h

int 21h

xor dx, dx

; если CF = 0

jnc restore\_everything

call PRINT\_NEWLINE

xor dx, dx

cmp ax, 1

```
je invalid_function
cmp ax, 2
je file_not_found
cmp ax, 5
je disk_error_label
cmp ax, 8
je load_not_enough_memory
cmp ax, 10
je invalid_env_string
cmp ax, 11
je invalid_format
```

invalid\_function:

```
mov dx, offset INVALID_FUNCTION_ERROR
jmp restore_everything
```

file\_not\_found:

```
mov dx, offset FILE_NOT_FOUND_ERROR
jmp restore_everything
```

disk\_error\_label:

```
mov dx, offset DISK_ERROR
jmp restore_everything
```

load\_not\_enough\_memory:

```
mov dx, offset NOT_ENOUGH_MEMORY_ERROR
jmp restore_everything
```

invalid\_env\_string:

```
mov dx, offset INVALID_ENV_STRING_ERROR
jmp restore_everything
```

invalid\_format:

mov dx, offset INVALID\_FORMAT\_ERROR

restore\_everything:

call WRITE\_STRING

mov ax, KEEP\_SP

mov sp, ax

mov ax, KEEP\_SS

mov ss, ax

mov ah, 4dh

int 21h

mov cl, al

xor dx, dx

cmp ah, 0

je okay\_exiting\_code

cmp ah, 1

je ctrl\_break\_code

cmp ah, 2

je device\_error\_code

cmp ah, 3

je resident\_31\_code

okay\_exiting\_code:

mov dx, offset OKAY\_EXITING\_MESSAGE

jmp load\_exit

ctrl\_break\_code:

mov dx, offset CTRL\_BREAK\_ERROR



```
jmp load_exit
```

```
device_error_code:
```

```
    mov dx, offset DEVICE_ERROR
```

```
    jmp load_exit
```

```
resident_31_code:
```

```
    mov dx, offset RESIDENT_31_ERROR
```

```
load_exit:
```

```
    call PRINT_NEWLINE
```

```
    call WRITE_STRING
```

```
    mov dx, offset RETURN_VALUE
```

```
    call WRITE_STRING
```

```
    mov dl, cl
```

```
    mov ah, 02h
```

```
    int 21h
```

```
    call PRINT_NEWLINE
```

```
    pop dx
```

```
    pop ax
```

```
    pop bx
```

```
    pop es
```

```
    ret
```

```
LOAD_MODULE endp
```

Main proc far

mov ax, DATA

mov ds, ax

call ALLOCATE\_MEMORY

call PARSE\_PROGRAM\_NAME

call LOAD\_MODULE

; ВЫХОД В DOS

xor al, al

mov ah, 4ch

int 21h

main\_procedure\_end:

Main endp

CODE ENDS

END Main

## Приложение Б (lab2)

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

org 100h

start: jmp begin

; DATA

UNAVAILABLE\_MEM\_GREET db 'Unavailable memory: \$'

ENV\_ADDRESS\_GREET db 'Environment address: \$'

EMPTY\_COMMAND\_TAIL db 'The command tail is empty\$'

COMMAND\_TAIL\_STR db 'Command tail:\$'

CONTENT\_GREET db 'Content of the environment:', 0dh, 0ah, '\$'

PATH\_GREET db 'Path: \$'

HEXED\_AX db ' ', '\$' ; ax - 2 байта, 2\*\*16 содержит 5 символов => резервируем 5 байт + символ конца строки

NEWLINE db 0dh, 0ah, '\$'

; Перевод ax в строку HEXED\_AX

AX\_TO\_STRING proc near

push bx

push di

xor bx, bx

mov di, offset HEXED\_AX

; high byte

mov bl, ah

```
and bl, 0f0h
call BL_TO_ASCII_HEX
mov [di], bl
inc di
```

```
mov bl, ah
and bl, 00fh
call BL_TO_ASCII_HEX
mov [di], bl
inc di
```

```
; low byte
mov bl, al
and bl, 0f0h
call BL_TO_ASCII_HEX
mov [di], bl
inc di
```

```
mov bl, al
and bl, 00fh
call BL_TO_ASCII_HEX
mov [di], bl
inc di
```

```
mov byte ptr [di], '$'
```

```
pop di
pop bx
ret
```

```
AX_TO_STRING endp
```

BL\_TO\_ASCII\_HEX proc near

push cx

cmp bx, 16 ; если bl >= 16, то нужно сдвинуть все вправо на 4 бита

jl start\_converting

mov cl, 4

shr bx, cl

start\_converting:

cmp bl, 0ah

jge bl\_is\_letter

; bl is digit

add bl, 48

jmp exit

bl\_is\_letter:

add bl, 55

exit:

pop cx

ret

BL\_TO\_ASCII\_HEX endp

PRINT\_NEWLINE proc near

push ax

push dx

mov dx, offset NEWLINE

```
mov ah, 9h
```

```
int 21h
```

```
pop dx
```

```
pop ax
```

```
ret
```

```
PRINT_NEWLINE endp
```

```
; Печатает строку в dx
```

```
WRITE_STRING proc near
```

```
push ax
```

```
mov ah, 9h
```

```
int 21h
```

```
pop ax
```

```
ret
```

```
WRITE_STRING endp
```

```
PSP_UNAVAILABLE proc near
```

```
push ax
```

```
push dx
```

```
mov dx, offset UNAVAILABLE_MEM_GREET
```

```
call WRITE_STRING
```

```
mov ax, ds:[02h]
```

```
call AX_TO_STRING
mov dx, offset HEXED_AX
call WRITE_STRING
```

```
pop dx
pop ax
ret
```

```
PSP_UNAVAILABLE endp
```

```
SEGMENT_ENV_ADDRESS proc near
```

```
push ax
push dx
```

```
mov dx, offset ENV_ADDRESS_GREET
call WRITE_STRING
```

```
mov ax, ds:[2ch]
call AX_TO_STRING
mov dx, offset HEXED_AX
call WRITE_STRING
```

```
pop dx
pop ax
ret
```

```
SEGMENT_ENV_ADDRESS endp
```

```
COMMAND_TAIL proc near
```

push cx

push dx

push si

xor cx, cx

; Сначала вытаскиваем количество параметров

mov cl, ds:[80h]

cmp cl, 0

je empty\_tail

mov dx, offset COMMAND\_TAIL\_STR

call WRITE\_STRING

; Для чтения

mov si, 0

; Для печати

read\_tail:

mov dl, ds:[81h+si]

mov ah, 02h

int 21h

inc si

loop read\_tail

jmp command\_tail\_exit

empty\_tail:

mov dx, offset EMPTY\_COMMAND\_TAIL

call WRITE\_STRING

command\_tail\_exit:



```
pop si
pop dx
pop cx
ret
```

```
COMMAND_TAIL endp
```

```
ENV_CONTENT proc near
```

```
push ax
push dx
push si
push ds
```

```
mov dx, offset CONTENT_GREET
call WRITE_STRING
```

```
xor si, si
mov ds, ds:[2ch]
```

```
read_line_loop:
    mov dl, [si]
    cmp dl, 0
    je end_of_line
```

```
mov ah, 02h
int 21h
```

```
inc si
jmp read_line_loop
```

```
end_of_line:
```

```
inc si
mov dl, [si]
cmp dl, 0          ; если конец всей области среды
je content_end
```

```
pop ds
call PRINT_NEWLINE
push ds
mov ds, ds:[2ch]
```

```
jmp read_line_loop    ; иначе возвращаемся и читаем заново
```

content\_end:

```
pop ds
call PRINT_NEWLINE
mov dx, offset PATH_GREET
call WRITE_STRING
push ds
mov ds, ds:[2ch]
```

; Теперь нужно вывести путь программы, si указывает на последний ноль среды  
; после него идут 00h и 01h, а потом уже путь.

```
add si, 3
```

read\_path\_loop:

```
mov dl, [si]
cmp dl, 0
je env_printing_exit
```

```
mov ah, 02h
```

```
int 21h
```

```
inc si
```

```
jmp read_path_loop
```

env\_printing\_exit:

pop ds

pop si

pop dx

pop ax

ret

ENV\_CONTENT endp

begin:

; 1. Вывести адрес, содержащийся в байтах 2 и 3 от начала PSP

call PSP\_UNAVAILABLE

call PRINT\_NEWLINE

; 2. Вывести сегментный адрес среды

call SEGMENT\_ENV\_ADDRESS

call PRINT\_NEWLINE

; 3. Вывести хвост командной строки

call COMMAND\_TAIL

call PRINT\_NEWLINE

; 4, 5. Вывести содержимое области среды в символьном виде

call ENV\_CONTENT

mov ah, 01h

int 21h

; выход в DOS

mov ah, 4ch

int 21h

TESTPC ENDS

END start