

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. 9383

Рыбников Р.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.

Постановка задачи.

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

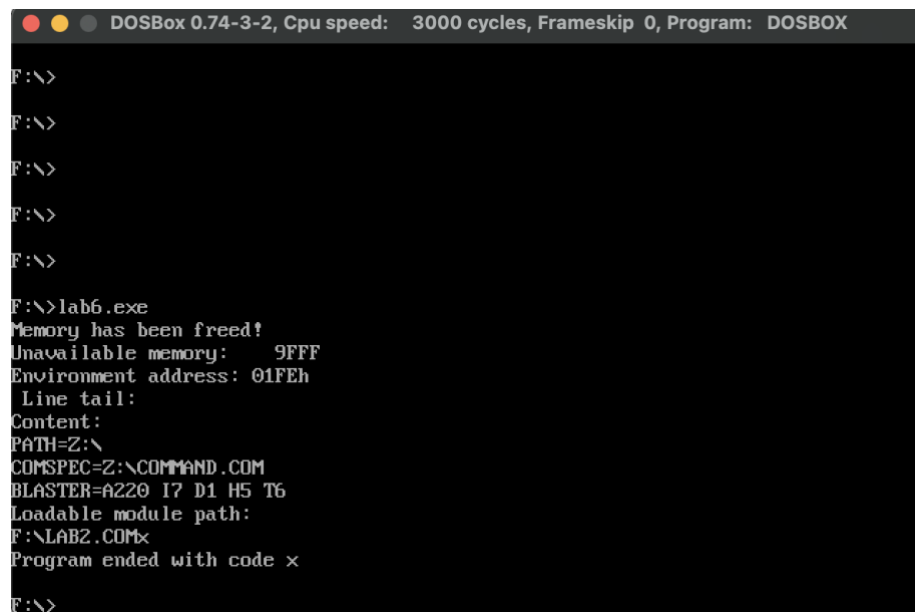
- 1) Подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором находится он сам. Вызываемому модулю передается новая среда, созданная вызывающим модулем и новая командная строка.
- 2) Вызываемый модуль запускается с использованием загрузчика.
- 3) После запуска проверяется 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.

Выполнение работы.

Сначала был написан код для .EXE модуля. На рисунке 1 показан запуск программы в каталоге с разработанными модулями.



```
DOSBox 0.74-3-2, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
F:\>
F:\>
F:\>
F:\>
F:\>
F:\>lab6.exe
Memory has been freed!
Unavailable memory: 9FFF
Environment address: 01FEh
Line tail:
Content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Loadable module path:
F:\LAB2.COMx
Program ended with code x
F:\>
```

Рисунок 1 – Запуск программы вместе с модулем из ЛР2.

На рисунке 2 показана обработка прерывания Ctrl-C. В терминал вывелся символ «сердечко», т. к. в эмуляторе DosBox не поддерживается данное прерывание и проверять обработку данного прерывания следует не в DosBox.

```
DOSBox 0.74-3-2, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
Line tail:
Content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Loadable module path:
F:\LAB2.COMx
Program ended with code x

F:\>

F:\>lab6.exe
Memory has been freed!
Unavailable memory: 9FFF
Environment address: 01FEh
Line tail:
Content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Loadable module path:
F:\LAB2.COMx
Program ended with code x

F:\>
```

Рисунок 2 – Запуск программы и прерывание по Ctrl-C.

На рисунке 3 представлен запуск программы в другом каталоге.

```
DOSBox 0.74-3-2, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Loadable module path:
F:\LAB2.COMx
Program ended with code x

F:\>

F:\>cd dir_2

F:\DIR_2>lab6.exe
Memory has been freed!
Unavailable memory: 9FFF
Environment address: 01FEh
Line tail:
Content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Loadable module path:
F:\DIR_2\LAB2.COMx
Program ended with code x

F:\DIR_2>
```

Рисунок 3 — Запуск программы в другом каталоге.

На рисунке 4 показан запуск программы, когда модуль из ЛР2 отсутствует.

```
F:\DIR_2>lab6.exe
Memory has been freed!
Error: file not found!
F:\DIR_2>
```

Рисунок 4 — Запуск программы без модуля из ЛР2.

Ответы на вопросы.

1. Как реализовано прерывание Ctrl+C?

При нажатии сочетания клавиш Ctrl+C управление передается по адресу 0000:008C после срабатывания прерывания 23h. Адрес копируется в PSP функциями 26h и 4Ch. Исходное значение адреса восстанавливается из PSP при выходе из программы.

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

В точке вызова функции 4Ch прерывания int 21h.

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

Программа завершится в точке, где была считана комбинация клавиш Ctrl+C

Выводы.

Было реализована программа, которая хранит в себе несколько модулей. Произведены тесты программы в разных ситуациях.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

lab6.asm:

ASTACK segment stack

dw 128 dup(?)

ASTACK ends

DATA SEGMENT

param_block dw 0

dd 0

dd 0

dd 0

program db 'lab2.com', 0

mem_flag db 0

cmd_1 db 1h, 0dh

cl_pos db 128 dup(0)

KEEP_SS dw 0

KEEP_SP dw 0

KEEP_PSP dw 0

mcb_crash db 'Error: memory block crashed!', 0dh, 0ah, '\$'

no_mem_err db 'Error: there is not enough memory to execute this function!', 0dh, 0ah, '\$'

address_err db 'Error: invalid memory address!', 0dh, 0ah, '\$'

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

func_err db 'Error: invalid function number!', 0dh, 0ah, '\$'

file_err db 'Error: file not found!', 0dh, 0ah, '\$'

disk_err db 'Error: disk error!', 0dh, 0ah, '\$'

memory_err db 'Error: insufficient memory!', 0dh, 0ah, '\$'

envs_err db 'Error: wrong string of environment!', 0dh, 0ah, '\$'

format_err db 'Error: wrong format!', 0dh, 0ah, '\$'

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

```
ctrl_end db 0dh, 0ah, 'Program ended by ctrl-break.' , 0dh, 0ah, '$'  
device_err db 0dh, 0ah, 'Program ended by device error.' , 0dh, 0ah, '$'  
int31_end db 0dh, 0ah, 'Program ended by int 31h.' , 0dh, 0ah, '$'
```

```
end_data db 0
```

```
DATA ENDS
```

```
CODE SEGMENT
```

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

```
PRINT_STRING PROC near
```

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

```
PRINT_STRING ENDP
```

```
FREE_MEMORY PROC near
```

```
    push ax  
    push bx  
    push cx  
    push dx  
  
    mov ax, offset end_data  
    mov bx, offset exit  
    add bx, ax  
  
    mov cl, 4  
    shr bx, cl  
    add bx, 2Bh  
    mov ah, 4Ah  
    int 21h  
  
    jnc end_f
```

```

        mov mem_flag, 1

m_mcb_crash:
        cmp ax, 7
        jne not_enought_memory
        mov dx, offset mcb_crash
        call PRINT_STRING
        jmp m_free_mem
not_enought_memory:
        cmp ax, 8
        jne addr
        mov dx, offset no_mem_err
        call PRINT_STRING
        jmp m_free_mem
addr:
        cmp ax, 9
        mov dx, offset address_err
        call PRINT_STRING
        jmp m_free_mem
end_f:
        mov mem_flag, 1
        mov dx, offset free_mem
        call PRINT_STRING

m_free_mem:
        pop dx
        pop cx
        pop bx
        pop ax
        ret
FREE_MEMORY ENDP

LOAD PROC near
        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 param_block
mov dx, offset cmd_1
mov [bx+2], dx
mov [bx+4], ds
mov dx, offset cl_pos

mov ax, 4B00h
int 21h

mov ss, KEEP_SS
mov sp, KEEP_SP
pop es
pop ds

jnc loads

```

```

m_func_err:
    cmp ax, 1
    jne m_file_err
    mov dx, offset func_err
    call PRINT_STRING
    jmp load_end
m_file_err:
    cmp ax, 2
    jne m_disk_err
    mov dx, offset file_err

```

```

        call PRINT_STRING
        jmp load_end
m_disk_err:
        cmp ax, 5
        jne mem_err
        mov dx, offset disk_err
        call PRINT_STRING
        jmp load_end
mem_err:
        cmp ax, 8
        jne m_envs_err
        mov dx, offset memory_err
        call PRINT_STRING
        jmp load_end

m_envs_err:
        cmp ax, 10
        jne m_format_err
        mov dx, offset envs_err
        call PRINT_STRING
        jmp load_end

m_format_err:
        cmp ax, 11
        mov dx, offset format_err
        call PRINT_STRING
        jmp load_end

loads:
        mov ah, 4Dh
        mov al, 00h
        int 21h

_nend:
        cmp ah, 0

```

```

    jne ctrlc
    push di
    mov di, offset good_end
    mov [di+26], al
    pop si
    mov dx, offset good_end
    call PRINT_STRING
    jmp load_end
ctrlc:
    cmp ah, 1
    jne device
    mov dx, offset ctrl_end
    call PRINT_STRING
    jmp load_end
device:
    cmp ah, 2
    jne int_31h
    mov dx, offset device_err
    call PRINT_STRING
    jmp load_end
int_31h:
    cmp ah, 3
    mov dx, offset int31_end
    call PRINT_STRING

load_end:
    pop dx
    pop cx
    pop bx
    pop ax
    ret
LOAD ENDP

PATH PROC near
    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

findz:

inc bx

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

jne findz

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

jne findz

add bx, 2

mov di, 0

_loop:

mov dl, es:[bx]

mov byte ptr [cl_pos+di], dl

inc di

inc bx

cmp dl, 0

je _end_loop

cmp dl, '\'

jne _loop

mov cx, di

jmp _loop

_end_loop:

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

_fn:

```
mov dl, byte ptr [program+si]
mov byte ptr [cl_pos+di], dl
inc di
inc si
cmp dl, 0
jne _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 free_memory
cmp mem_flag, 0
je _end
call PATH
call LOAD
```

_end:

```
xor al, al
```

```
mov ah, 4Ch
int 21h
```

Begin ENDP

exit:

CODE ENDS

END Begin

lab2.asm:

TESTPC SEGMENT

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

ORG 100H

START: jmp BEGIN

MEMORY_ADDRESS db 'Unavailable memory: h',13,10, 13, 10, '\$'

ENV_ADDRESS db 'Environment address: h',13,10,'\$'

NOT_EMPTY_TAIL db ' Line tail: ',13,10,'\$'

EMPTY_TAIL_STR db 'Command tail is empty',13,10,'\$'

CONTENT_STR db 'Content:',13,10, '\$'

END_OF_LINE db 13, 10, '\$'

PATH db 'Loadable module path: ',13,10,'\$'

TETR_TO_HEX PROC near

and AL,0Fh

cmp AL,09

jbe next

add AL,07

next:

add AL,30h

ret

TETR_TO_HEX ENDP

BYTE_TO_HEX PROC near

```
push CX
mov AH,AL
call TETR_TO_HEX
xchg AL,AH
mov CL,4
shr AL,CL
call TETR_TO_HEX
pop CX
ret
```

BYTE_TO_HEX ENDP

WRD_TO_HEX PROC near

```
push BX
mov BH,AH
call BYTE_TO_HEX
mov [DI],AH
dec DI
mov [DI],AL
dec DI
mov AL,BH
call BYTE_TO_HEX
mov [DI],AH
dec DI
mov [DI],AL
pop BX
ret
```

WRD_TO_HEX ENDP

BYTE_TO_DEC PROC near

```
    push CX
    push DX
    xor AH,AH
    xor DX,DX
    mov CX,10
loop_bd:
    div CX
    or DL,30h
    mov [SI],DL
    dec SI
    xor DX,DX
    cmp AX,10
    jae loop_bd
    cmp AL,00h
    je end_l
    or AL,30h
    mov [SI],AL
end_l:
    pop DX
    pop CX
    ret
```

BYTE_TO_DEC ENDP

WRITE_STRING PROC near

```
    mov AH,09h
    int 21h
    ret
```

WRITE_STRING ENDP


```

UNAVAILABLE_MEMORY PROC near
    mov ax,ds:[02h]
    mov di, offset MEMORY_ADDRESS
    add di, 26
    call WRD_TO_HEX
    mov dx, offset MEMORY_ADDRESS
    call WRITE_STRING
    ret
UNAVAILABLE_MEMORY ENDP

```

```

ENVIROMENT_ADDRESS PROC near
    mov ax,ds:[2Ch]
    mov di, offset ENV_ADDRESS
    add di, 24
    call WRD_TO_HEX
    mov dx, offset ENV_ADDRESS
    call WRITE_STRING
    ret
ENVIROMENT_ADDRESS ENDP

```

```

COMMAND_LINE_TAIL PROC near
    xor cx, cx
        mov cl, ds:[80h]
        mov si, offset NOT_EMPTY_TAIL
        add si, 19
    cmp cl, 0h
    je empty_tail
        xor di, di
        xor ax, ax

next_tail:
    mov al, ds:[81h+di]

```

```

inc di
mov [si], al
    inc si
    loop next_tail

    mov dx, offset NOT_EMPTY_TAIL
    jmp TAIL_END

empty_tail:
    mov dx, offset EMPTY_TAIL_STR

TAIL_END:
    call WRITE_STRING
    ret

COMMAND_LINE_TAIL ENDP

CONTENT PROC near
    mov dx, offset CONTENT_STR
    call WRITE_STRING
    xor di, di
    mov ds, ds:[2Ch]

READ_LINE:
    cmp byte ptr [di], 00h
    jz END_LINE
    mov dl, [di]
    mov ah, 02h
    int 21h
    jmp find_end

END_LINE:
    cmp byte ptr [di+1], 00h

```

```
jz FIND_END
push ds
mov cx, cs
    mov ds, cx
    mov dx, offset END_OF_LINE
    call WRITE_STRING
pop ds
```

```
FIND_END:
    inc di
    cmp word ptr [di], 0001h
    jz PATH_READING
    jmp READ_LINE
```

```
PATH_READING:
    push ds
    mov ax, cs
    mov ds, ax
    mov dx, offset PATH
    call WRITE_STRING
pop ds
add di, 2
```

```
LOOP_PATH:
    cmp byte ptr [di], 00h
    jz EXIT
    mov dl, [di]
    mov ah, 02h
    int 21h
    inc di
    jmp LOOP_PATH
```

```
EXIT:
    ret
```

```
CONTENT ENDP
```

BEGIN:

call UNAVAILABLE_MEMORY

call ENVIROMENT_ADDRESS

call COMMAND_LINE_TAIL

call CONTENT

xor al, al

mov AH,01h

int 21H

mov ah, 4Ch

int 21h

TESTPC ENDS

END START