

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

Студентка гр. 9382

Голубева В.П.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры.

Постановка задачи.

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.

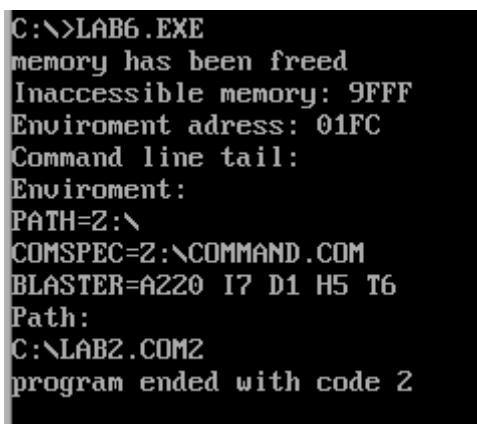
Шаг 4. Запустите отлаженную программу, когда текущим каталогом является какой-либо другой каталог, отличный от того, в котором содержатся разработанные программные модули.

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.

Запустим программу из директории с разработанными модулями и введём символ 2. Результат можно посмотреть в Рисунке 1.



```
C:\>LAB6.EXE
memory has been freed
Inaccessible memory: 9FFF
Enviroment address: 01FC
Command line tail:
Enviroment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path:
C:\LAB2.COM2
program ended with code 2
```

Рисунок 1. Запуск программы из директории с разработанными модулями

Теперь запустим программу и завершим с помощью Ctrl + C (программа завершается нормально, т.к. обработка данного сочетание клавиш в DOSBox не реализована, символ сердечко - это и есть Ctrl + C)



```
C:\LR6>LAB6.EXE
memory has been freed
Inaccessible memory: 9FFF
Enviroment address: 01FC
Command line tail:
Enviroment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path:
C:\LR6\LAB2.COM♥
Program ended with code ♥
```

Рисунок 2. Запуск программы из директории с разработанными модулями и завершением через Ctrl + C

Теперь запустим программу, находясь в другой директории.

```
C:\LR6>cd ..\

C:\>cd LR6\Lab6.exe
Unable to change to: LR6\Lab6.exe.

C:\>LR6\Lab6.exe
memory has been freed
Inaccessible memory: 9FFF
Enviroment address: 01FC
Command line tail:
Enviroment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path:
C:\LR6\LAB2.COMz
Program ended with code z
```

Рисунок 3. Запуск программы во время нахождения в другой директории

Теперь запустим программу при условии, что программный и загрузочный модуль находятся в разных директориях

```
C:\>LR6\Lab6.exe
memory has been freed
ERR: file not found
```

Рисунок 4. Запуск программы при условии, что программный и загрузочный модуль находятся в разных директориях

Ответы на контрольные вопросы.

1) Как реализовано прерывание Ctrl-C?

Ответ: При нажатии клавиш Ctrl-C управление передаётся по адресу 0000:008Ch. Этот адрес копируется в PSP функциями 26h и 4Ch и восстанавливается из PSP при выходе из программы.

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

Ответ: Если код завершения 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 было нажато Ctrl-C, то программа завершится непосредственно в том месте, в котором произошло нажатие сочетания клавиш (то есть в месте ожидания нажатия клавиши: 01h вектора прерывания 21h)

Вывод.

Был построен загрузочный модуль динамической структуры. Были получены навыки работы с памятью.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
stacks segment stack
    dw 128 dup(?)
stacks ends
```

```
data segment
```

```
    parameter_block dw 0
```

```
                                dd 0
```

```
                                dd 0
```

```
                                dd 0
```

```
    program db 'lab2.com', 0
```

```
    mem_flag db 0
```

```
    cmd_l db 1h, 0dh
```

```
    cl_pos db 128 dup(0)
```

```
    keep_ss dw 0
```

```
    keep_sp dw 0
```

```
    keep_psp dw 0
```

```
    str_mcb_crash_err db 'err: mcb crashed', 0dh, 0ah, '$'
```

```
    str_no_mem_err db 'err: there is not enough memory to execute  
this function', 0dh, 0ah, '$'
```

```
    str_addr_err db 'err: invalid memory address', 0dh, 0ah, '$'
```

```
    str_free_mem db 'memory has been freed' , 0dh, 0ah, '$'
```

```
    str_fn_err db 'err: invalid function number', 0dh, 0ah, '$'
```

```
    str_file_error db 'err: file not found', 0dh, 0ah, '$'
```

```
    str_disk_err db 'err: disk error', 0dh, 0ah, '$'
```

```
    str_memory_error db 'err: insufficient memory', 0dh, 0ah, '$'
```

```
    str_envs_err db 'err: wrong string of environment ', 0dh, 0ah,  
'$'
```

```
    str_format_err db 'err: wrong format', 0dh, 0ah, '$'
```

```
    str_norm_fin db 0dh, 0ah, 'program ended with code      ' , 0dh,  
0ah, '$'
```

```
    str_ctrl_end db 0dh, 0ah, 'program ended by ctrl-break' , 0dh,  
0ah, '$'
```

```

        str_device_err db 0dh, 0ah, 'program ended by device error' ,
0dh, 0ah, '$'
        str_int_end db 0dh, 0ah, 'program ended by int 31h' , 0dh, 0ah,
'$'

```

```

        end_data db 0
data ends

```

```

code segment

```

```

assume cs:code, ds:data, ss:stacks

```

```

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

```

```

free_memory proc
    push ax
    push bx
    push cx
    push dx

    mov ax, offset end_data
    mov bx, offset eeend
    add bx, ax

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

    jnc _endf
    mov mem_flag, 1

```

```

mcb_crash:

```

```

        cmp ax, 7
        jne not_enought_memory
        mov dx, offset str_mcb_crash_err
        call print_str
        jmp freee
not_enought_memory:
        cmp ax, 8
        jne addr
        mov dx, offset str_no_mem_err
        call print_str
        jmp freee
addr:
        cmp ax, 9
        mov dx, offset str_addr_err
        call print_str
        jmp freee
_endf:
        mov mem_flag, 1
        mov dx, offset str_free_mem
        call print_str

freee:
        pop dx
        pop cx
        pop bx
        pop ax
        ret
free_memory 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 parameter_block
mov dx, offset cmd_l
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
```

```
fn_err:
```

```
    cmp ax, 1
    jne file_err
    mov dx, offset str_fn_err
    call print_str
    jmp load_end
```

```
file_err:
```

```
    cmp ax, 2
    jne disk_err
    mov dx, offset str_file_error
    call print_str
    jmp load_end
```

```
disk_err:
```

```
    cmp ax, 5
    jne mem_err
    mov dx, offset str_disk_err
    call print_str
    jmp load_end
```

```
mem_err:
```

```
    cmp ax, 8
    jne envs_err
    mov dx, offset str_memory_error
    call print_str
```

```

        jmp load_end
envs_err:
        cmp ax, 10
        jne format_err
        mov dx, offset str_envs_err
        call print_str
        jmp load_end
format_err:
        cmp ax, 11
        mov dx, offset str_format_err
        call print_str
        jmp load_end

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

_nend:
        cmp ah, 0
        jne ctrlc
        push di
        mov di, offset str_norm_fin
        mov [di+26], al
        pop si
        mov dx, offset str_norm_fin
        call print_str
        jmp load_end
ctrlc:
        cmp ah, 1
        jne device
        mov dx, offset str_ctrl_end
        call print_str
        jmp load_end
device:
        cmp ah, 2
        jne int_31h
        mov dx, offset str_device_err
        call print_str
        jmp load_end

```

```
int_31h:
    cmp ah, 3
    mov dx, offset str_int_end
    call print_str
```

```
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
```

```
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

eeend:
code ends
end begin
```