

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

Камзолов Н.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Постановка задачи.

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.

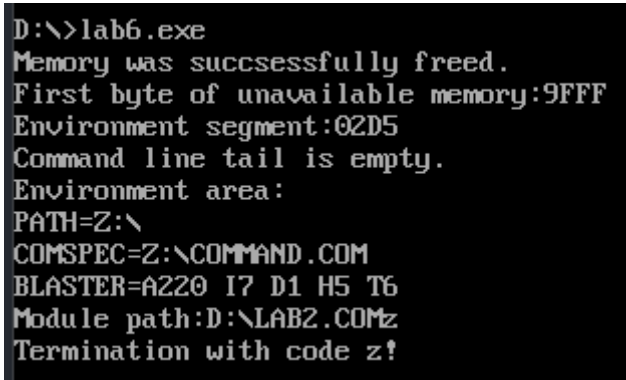
Шаг 4. Запустите отлаженную программу, когда текущим каталогом является какой-либо другой каталог, отличный от того, в котором содержатся разработанные программные модули. Повторите ввод комбинаций клавиш. Занесите полученные данные в отчет.

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

Результаты исследования проблем.

Шаг 1. Был написан и отлажен программный модуль типа .EXE, который выполняет, прописанные в задании функции.

Шаг 2. Программа была запущена, когда текущим каталогом является каталог с разработанными модулями. Введен символ 'z' для прерывания вызываемой программы.



```
D:\>lab6.exe
Memory was successfully freed.
First byte of unavailable memory:9FFF
Environment segment:02D5
Command line tail is empty.
Environment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Module path:D:\LAB2.COMz
Termination with code z!
```

Рисунок 1 – Демонстрация работы программы, при вводе в вызываемый модуль символа 'z'.

Шаг 3. Программа была запущена, когда текущим каталогом является каталог с разработанными модулями. Для прерывания была введена комбинация клавиш Ctrl+C, но так как в DOSBox не реализовано это прерывание, программа считает комбинацию клавиш за символ сердца.

```

D:\>lab6.exe
Memory was successsfully freed.
First byte of unavailable memory:9FFF
Environment segment:02D5
Command line tail is empty.
Environment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Module path:D:\LAB2.COM
Termination with code ♥!

```

Рисунок 2 – Демонстрация работы программы, при вводе в вызываемый модуль комбинации клавиш Ctrl+Z.

Шаг 4. Программа была запущена, когда текущим каталогом является другой каталог, отличный от того, в котором содержатся разработанные программные модули. Снова для завершения программы воспользуемся введением символа 'z', а затем комбинацией клавиш Ctrl+C.

```

D:\MASM>..\lab6.exe
Memory was successsfully freed.
First byte of unavailable memory:9FFF
Environment segment:02D5
Command line tail is empty.
Environment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Module path:D:\LAB2.COMz
Termination with code z!

```

Рисунок 3 – Демонстрация работы программы, запущенной из другого каталога, при вводе в вызываемый модуль символа 'z'.

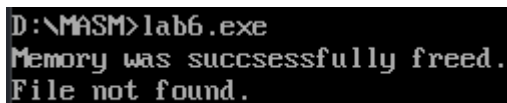
```

D:\MASM>..\lab6.exe
Memory was successsfully freed.
First byte of unavailable memory:9FFF
Environment segment:02D5
Command line tail is empty.
Environment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Module path:D:\LAB2.COM
Termination with code ♥!

```

Рисунок 4 – Демонстрация работы программы, запущенной из другого каталога, при вводе в вызываемый модуль комбинации клавиш Ctrl+Z.

Шаг 5. Программа была запущена, когда модули находятся в разных каталогах.



```
D:\MASM>lab6.exe
Memory was successfully freed.
File not found.
```

Рисунок 4 – Демонстрация корректной обработки ошибки при попытке вызова модуля, которого нет в каталоге.

По итогам выполнения работы можно ответить на контрольные вопросы:

1. Как реализовано прерывание Ctrl-C?

Ответ: При нажатии сочетания клавиш Ctrl-C срабатывает прерывание INT 23H. Управление передается по адресу (0000:008C), а затем этот адрес копируется в поле PSP с помощью функций DOS: 26H и 4CH. Исходное значение адреса обработчика прерывания Ctrl-C восстанавливается из PSP при завершении программы.

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

Ответ: В точке вызова функции 4CH прерывания INT 21H.

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

Ответ: В том месте, где произошел ввод сочетания клавиш Ctrl+C. По-другому можно сказать: в точке вызова функции 01H прерывания INT 21H.

Выводы.

Исследованы возможности построения загрузочного модуля динамической структуры. Исследован интерфейс между вызывающим и вызываемым модулями по управлению и по данным.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

lab6.asm:

```
AStack    SEGMENT    STACK
           DW 64 DUP(?)
AStack    ENDS

DATA      SEGMENT
    parametr_block dw 0
                   db 0
                   db 0
                   db 0
    new_command_line db 1h,0DH
    path db 128 dup(0)
    file_name db "lab2.com", 0
    keep_ss dw 0
    keep_sp dw 0
    NORMAL_TERMINATION db 0DH, 0AH, "Normal termination with code ! ",
0DH, 0AH, '$'
    CTRL_TERMINATION db 0DH, 0AH, "Termination by Ctrl-Break. ", 0DH,
0AH, '$'
    DEVICE_TERMINATION db 0DH, 0AH, "Termination by device error. ", 0DH,
0AH, '$'
    FUNC_TERMINATION db 0DH, 0AH, "Termination by 31h function. ", 0DH,
0AH, '$'

    MEMORY_BLOCK_ERROR db "Memory control block destroyed. ", 0DH, 0AH,
'$'
    LOW_MEMORY db "Not enough memory to execute the function. ", 0DH,
0AH, '$'
    WRONG_PTR db "Invalid memory block address. ", 0DH, 0AH, '$'
    MEMORY_FREE_SUCCESS db "Memory was succsessfully freed. ", 0DH, 0AH,
'$'

    WRONG_FUNC_NUMBER db "Wrong function number.", 0DH, 0AH, '$'
    FILE_NOT_FOUND db "File not found.", 0DH, 0AH, '$'
    DISK_ERROR db "Disk error.", 0DH, 0AH, '$'
    NOT_ENOUGH_MEMORY db "Not enough memory.", 0DH, 0AH, '$'
    WRONG_ENVIRONMENT_STRING db "Wrong environment string.", 0DH, 0AH,
'$'
    WRONG_FORMAT db "Wrong format.", 0DH, 0AH, '$'

    carry_flag_value db 0
    last_data_byte db 0
DATA      ENDS

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:AStack

PRINT_BUF proc near
    push ax
    mov ah, 9h
    int 21h
```

```

        pop ax
        ret
PRINT_BUF endp

FREE_MEMORY proc near
    push ax
    push bx
    push dx

    mov ax, offset end_of_proc
    mov bx, offset last_data_byte
    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

    jnc memory_free_success_label
    mov carry_flag_value, 1

    cmp ax, 7
    je memory_block_destroyed_label
    cmp ax, 8
    je low_memory_label
    cmp ax, 9
    je invalid_address_label

memory_block_destroyed_label:
    mov dx, offset MEMORY_BLOCK_ERROR
    call PRINT_BUF
    jmp memory_free_end
low_memory_label:
    mov dx, offset LOW_MEMORY
    call PRINT_BUF
    jmp memory_free_end
invalid_address_label:
    mov dx, offset WRONG_PTR
    call PRINT_BUF
    jmp memory_free_end
memory_free_success_label:
    mov dx, offset MEMORY_FREE_SUCCESS
    call PRINT_BUF
memory_free_end:
    pop dx
    pop bx
    pop ax
    ret
FREE_MEMORY endp

PARAMETR_BLOCK_CREATE proc near
    push bx

```



```

    push dx

    mov bx, offset parametr_block
    mov dx, offset new_command_line
    mov [bx+2], dx
    mov [bx+4], ds

    pop dx
    pop bx
    ret
PARAMETR_BLOCK_CREATE endp

PATH_FIND proc near
    push ax
    push si
    push dx
    push es
    push bx
    push di

    mov ax, es:[2Ch]
    mov es, ax
    xor si, si
find_two_zeros:
    inc si
    mov dl, es:[si-1]
    cmp dl, 0
    jne find_two_zeros
    mov dl, es:[si]
    cmp dl, 0
    jne find_two_zeros

    add si, 3
    mov bx, offset path
while_not_point:
    mov dl, es:[si]
    mov [bx], dl
    cmp dl, '.'
    je loop_back

    inc bx
    inc si

    jmp while_not_point
loop_back:
    mov dl, [bx]
    cmp dl, '\'
    je break_loop
    mov dl, 0h
    mov [bx], dl
    dec bx
    jmp loop_back
break_loop:
    mov di, offset file_name
    inc bx
loop_new_file:
    mov dl, [di]

```

```

        cmp dl, 0
        je end_path_find
        mov [bx], dl
        inc di
        inc bx
        jmp loop_new_file
end_path_find:
        pop di
        pop bx
        pop es
        pop dx
        pop si
        pop ax
        ret
PATH_FIND endp

CALL_MODULE proc near
        push ax
        push bx
        push cx
        push dx
        push di
        push si
        push es
        push ds

        mov keep_sp, sp
        mov keep_ss, ss

        mov ax, data
        mov es, ax
        mov dx, offset path
        mov bx, offset parametr_block

        mov ax, 4b00h
        int 21h

        mov ss, keep_ss
        mov sp, keep_sp
        pop ds
        pop es

        jnc module_was_loaded
        cmp ax, 1
        je wrong_func_number_label
        cmp ax, 2
        je file_not_found_label
        cmp ax, 5
        je disk_error_label
        cmp ax, 8
        je not_enough_memory_label
        cmp ax, 10
        je wrong_environment_string_label
        cmp ax, 11
        je wrong_format_label
wrong_func_number_label:
        mov dx, offset WRONG_FUNC_NUMBER

```

```

        call PRINT_BUF
        jmp call_module_end
file_not_found_label:
        mov dx, offset FILE_NOT_FOUND
        call PRINT_BUF
        jmp call_module_end
disk_error_label:
        mov dx, offset DISK_ERROR
        call PRINT_BUF
        jmp call_module_end
not_enough_memory_label:
        mov dx, offset NOT_ENOUGH_MEMORY
        call PRINT_BUF
        jmp call_module_end
wrong_environment_string_label:
        mov dx, offset WRONG_ENVIRONMENT_STRING
        call PRINT_BUF
        jmp call_module_end
wrong_format_label:
        mov dx, offset WRONG_FORMAT
        call PRINT_BUF
        jmp call_module_end
module_was_loaded:
        mov ax, 4d00h
        int 21h

        cmp ah, 0
        jmp normal_termination_label
        cmp ah, 1
        jmp ctrl_termination_label
        cmp ah, 2
        jmp device_termination_label
        cmp ah, 3
        jmp func_termination_label
normal_termination_label:
        mov di, offset NORMAL_TERMINATION
        add di, 31
        mov [di], al
        mov dx, offset NORMAL_TERMINATION
        call PRINT_BUF
        jmp call_module_end
ctrl_termination_label:
        mov dx, offset CTRL_TERMINATION
        call PRINT_BUF
        jmp call_module_end
device_termination_label:
        mov dx, offset DEVICE_TERMINATION
        call PRINT_BUF
        jmp call_module_end
func_termination_label:
        mov dx, offset FUNC_TERMINATION
        call PRINT_BUF
        jmp call_module_end
call_module_end:
        pop si
        pop di
        pop dx

```

```

        pop cx
        pop bx
        pop ax
        ret
CALL_MODULE endp

MAIN proc far
    push ds
    push ax
    mov ax,data
    mov ds,ax

    call FREE_MEMORY
    call PATH_FIND
    call CALL_MODULE

    mov ah, 4ch
    int 21h
end_of_proc:
MAIN endp
CODE ends
END Main

```