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, создранная вызывающий млодулем 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:

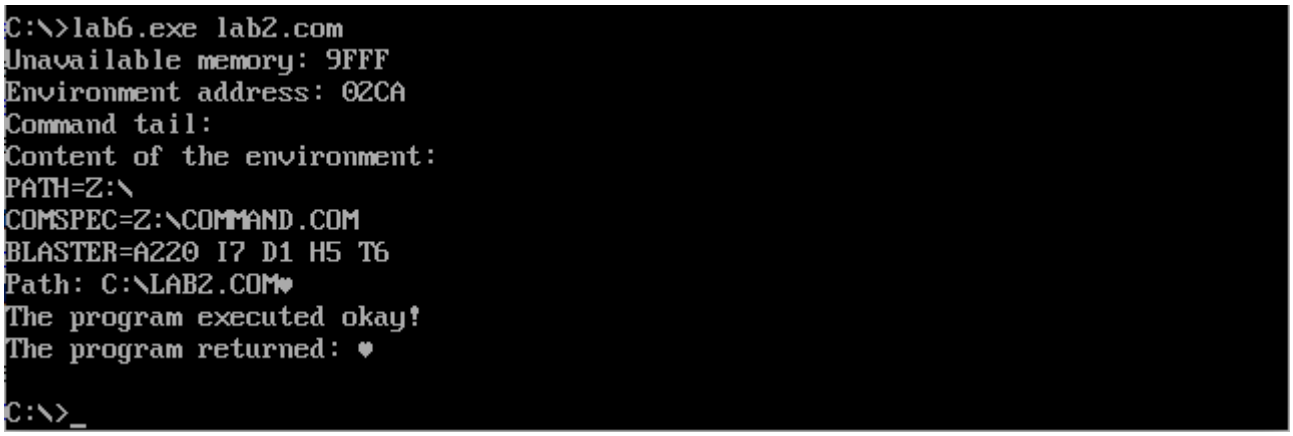

Рис. 2. Пример работы программы (2)

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

Рис. 3. Пример работы программы (3)

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:

Рис. 5. Пример работы программы (5)

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. Код по адресу 008C ведет к остановке текущей программы.

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 при ожидании нажатия клавиши (а точнее, в программе lab2.asm при вызове функции 1 прерывания 21h).

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
```