

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. 0382

Сергеев Д.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

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 2 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, требуемый по заданию.

Шаг 2. Программа была запущена из каталога с разработанными модулями. Была нажата клавиша “y”.



```
C:\>lab6.exe
Memory was freed
Inaccessable memory: 9FFFh
Segment Address: 02D3h
Tail:
Enviroment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path: C:\PROG.COM
y
Program was finished with exit code:  y
C:\>
```

Рисунок 1 – Результат работы написанной программы

Шаг 3. Программа была запущена из каталога с разработанными модулями, затем была нажата комбинация клавиш Ctrl+C.

```
C:\>lab6.exe
Memory was freed
Inaccessable memory: 9FFFh
Segment Address: 02D3h
Tail:
Enviroment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path: C:\PROG.COM
♥
Programm was finished with exit code: ♥
C:\>
```

Рисунок 2 – Результат работы программы при нажатии Ctrl+C

Как видно на рисунке, результат работы аналогичен предыдущему шагу, так как в DosBox не реализовано прерывание, которое происходит по нажатию Ctrl+C.

Шаг 4. Программа была дважды запущена из другого каталога, последовательно были нажаты “g” и Ctrl+C.

```
C:\TEST>lab6.exe
Memory was freed
Inaccessable memory: 9FFFh
Segment Address: 02D3h
Tail:
Enviroment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path: C:\TEST\PROG.COM
g
Programm was finished with exit code: g
C:\TEST>
```

Рисунок 3 – Результат работы программы из каталога TEST и нажатой клавише “g”

```

C:\TEST>lab6.exe
Memory was freed
Inaccessable memory: 9FFFh
Segment Address: 02D3h
Tail:
Enviroment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path: C:\TEST\PROG.COM
♥
Programm was finished with exit code: ♥
C:\TEST>_

```

Рисунок 4 – Результат работы программы из каталога TEST и нажатыми Ctrl+C

Шаг 5. Программа была запущена, при этом модули находились в разных каталогах.

```

C:\TEST>lab6.exe
Memory was freed

Error: file was not found

C:\TEST>

```

Рисунок 5 – Результат работы программы при выполнении шага 5.

Исходный программный код смотрите в приложении А.

Контрольные вопросы.

1. Как реализованы прерывание Ctrl+C?

Ответ: когда происходит нажатие сочетания клавиш Ctrl+C срабатывает прерывание - int 23h. Тогда управление передается по адресу - (0000:008C). С помощью функций 26h и 4ch этот адрес копируется в PSP. h и 4ch этот адрес копируется в PSP. При выходе из программы исходное значение адреса восстанавливается.

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

Ответ: вызываемая программа заканчивается в точке вызова функции 4ch прерывания int 21h.

3. В какой точке заканчивается вызываемая программа по прерыванию CtrlC?

Ответ: в данном случае, программа завершится в точке, где была введена и считана комбинация клавиш Ctrl+C.

Выводы.

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

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab6.asm

```
ASTACK SEGMENT STACK
    DW 256 DUP(?)
ASTACK ENDS

DATA SEGMENT
    param_block dw 0
    command_off dw 0
    command_seg dw 0
    dd 0
    dd 0

    next_command_line db 1h, 0dh
    file_name db 'prog.com', 0h
    file_path db 128 DUP(0)

    keep_ss dw 0
    keep_sp dw 0

    free_memory db 0
    str_free_memory_mcb_error db 'Free memory error: MCB crashed', 0DH, 0AH,
'$'
    str_free_memory_not_enough_error db 'Free memory error: not enough
memory', 0DH, 0AH, '$'
    str_free_memory_address_error db 'Free memory error: wrong address', 0DH,
0AH, '$'
    str_free_memory_successfully db 'Memory was freed', 0DH, 0AH, '$'

    str_load_function_number_error db 'Error: wrong function number', 0DH,
0AH, '$'
    str_load_file_not_found_error db 'Error: file was not found', 0DH, 0AH,
'$'
    str_load_disk_error db 'Error: disk problem', 0DH, 0AH, '$'
    str_load_memory_error db 'Error: not enough memory', 0DH, 0AH, '$'
    str_load_path_error db 'Error: wrong path param', 0DH, 0AH, '$'
    str_load_format_error db 'Error: wrong format', 0DH, 0AH, '$'

    str_exit db 'Programm was finished with exit code: ', 0DH, 0AH, '$'
    str_exit_ctrl_c db 'Exit with Ctrl+Break', 0DH, 0AH, '$'
    str_exit_error db 'Exit with device error', 0DH, 0AH, '$'
    str_exit_int31h db 'Exit with int 31h', 0DH, 0AH, '$'

    data_end db 0
DATA ENDS

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

;-----
    PRINT MESSAGE PROC NEAR
        push ax
        mov ah, 9
        int 21h
```

```

    pop ax
    ret
PRINT_MESSAGE ENDP

```

```

PRINT_EOF PROC NEAR

```

```

    push ax
    push dx
    mov dl, 0dh
    push ax
    mov ah, 02h
    int 21h
    pop ax
    mov dl, 0ah
    push ax
    mov ah, 02h
    int 21h
    pop ax
    pop dx
    pop ax
    ret

```

```

PRINT_EOF ENDP

```

```

;-----

```

```

;-----

```

```

FREE_MEMORY_PROC PROC FAR

```

```

    push ax
    push bx
    push cx
    push dx
    push es
    xor dx, dx
    mov free_memory, 0h
    mov ax, offset data_end
    mov bx, offset finish
    add ax, bx
    mov bx, 10h
    div bx
    add ax, 100h
    mov bx, ax
    xor ax, ax
    mov ah, 4ah
    int 21h
    jnc free_memory_successfully
    mov free_memory, 1h
    cmp ax, 7
    jne free_memory_not_enough_error
    mov dx, offset str_free_memory_mcb_error
    call PRINT_MESSAGE
    jmp free_memory_exit

```

```

free_memory_not_enough_error:

```

```

    cmp ax, 8
    jne free_memory_address_error
    mov dx, offset str_free_memory_not_enough_error
    call PRINT_MESSAGE
    jmp free_memory_exit

```

```

free_memory_address_error:

```



```

    cmp ax, 9
    jne free_memory_exit
    mov dx, offset str_free_memory_address_error
    call PRINT_MESSAGE
    jmp free_memory_exit

```

```

free_memory_successfully:
    mov dx, offset str_free_memory_successfully
    call PRINT_MESSAGE
    jmp free_memory_exit

```

```

free_memory_exit:
    pop es
    pop dx
    pop cx
    pop bx
    pop ax
    ret

```

```

FREE_MEMORY_PROC ENDP

```

```

;-----

```

```

;-----

```

```

LOAD PROC FAR
    push ax
    push bx
    push cx
    push dx
    push ds
    push es
    mov keep_sp, sp
    mov keep_ss, ss
    call PATH_1
    mov ax, data
    mov es, ax
    mov bx, offset param_block
    mov dx, offset next_command_line
    mov command_off, dx
    mov command_seg, ds
    mov dx, offset file_path
    mov ax, 4b00h
    int 21h
    mov ss, keep_ss
    mov sp, keep_sp
    pop es
    pop ds
    call PRINT_EOF
    jnc load_successfully
    cmp ax, 1
    je load_function_number_error
    cmp ax, 2
    je load_file_not_found_error
    cmp ax, 5
    je load_disk_error
    cmp ax, 8
    je load_memory_error
    cmp ax, 10
    je load_path_error
    cmp ax, 11

```

```

        je load_format_error

load_function_number_error:
    mov dx, offset str_load_function_number_error
    call PRINT_MESSAGE
    jmp load_exit

load_file_not_found_error:
    mov dx, offset str_load_file_not_found_error
    call PRINT_MESSAGE
    jmp load_exit

load_disk_error:
    mov dx, offset str_load_disk_error
    call PRINT_MESSAGE
    jmp load_exit

load_memory_error:
    mov dx, offset str_load_memory_error
    call PRINT_MESSAGE
    jmp load_exit

load_path_error:
    mov dx, offset str_load_path_error
    call PRINT_MESSAGE
    jmp load_exit

load_format_error:
    mov dx, offset str_load_format_error
    call PRINT_MESSAGE
    jmp load_exit

load_successfully:
    mov ax, 4d00h
    int 21h
    cmp ah, 0
    jne exit_ctrl_c
    mov di, offset str_exit
    add di, 39
    mov [di], al
    mov dx, offset str_exit
    call PRINT_MESSAGE
    jmp load_exit

exit_ctrl_c:
    cmp ah, 1
    jne exit_error
    mov dx, offset str_exit_ctrl_c
    call PRINT_MESSAGE
    jmp load_exit

exit_error:
    cmp ah, 2
    jne exit_int31h
    mov dx, offset str_exit_error
    call PRINT_MESSAGE
    jmp load_exit

```

```

exit_int31h:
    cmp ah, 3
    jne load_exit
    mov dx, offset str_exit_int31h
    call PRINT_MESSAGE
    jmp load_exit

```

```

load_exit:
    pop dx
    pop cx
    pop bx
    pop ax
    ret

```

```

LOAD ENDP

```

```

;-----

```

```

;-----

```

```

PATH_1 PROC NEAR
    push ax
    push dx
    push es
    push di
    xor di, di
    mov ax, es:[2ch]
    mov es, ax

```

```

loop_for_PATH_1:
    mov dl, es:[di]
    cmp dl, 0
    je go_to_path
    inc di
    jmp loop_for_PATH_1

```

```

go_to_path:
    inc di
    mov dl, es:[di]
    cmp dl, 0
    jne loop_for_PATH_1
    call PATH_2
    pop di
    pop es
    pop dx
    pop ax
    ret

```

```

PATH_1 ENDP

```

```

PATH_2 PROC NEAR
    push ax
    push bx
    push bp
    push dx
    push es
    push di
    mov bx, offset file_path
    add di, 3

```

```

loop_for_symbol_boot:
    mov dl, es:[di]

```

```

        mov [bx], dl
        cmp dl, '.'
        je loop_for_symbol_slash
        inc di
        inc bx
        jmp loop_for_symbol_boot

loop_for_symbol_slash:
        mov dl, [bx]
        cmp dl, '\'
        je get_file_name
        mov dl, 0h
        mov [bx], dl
        dec bx
        jmp loop_for_symbol_slash

get_file_name:
        mov di, offset file_name
        inc bx

add_file_name:
        mov dl, [di]
        cmp dl, 0h
        je path_exit
        mov [bx], dl
        inc bx
        inc di
        jmp add_file_name

path_exit:
        mov [bx], dl
        pop di
        pop es
        pop dx
        pop bp
        pop bx
        pop ax
        ret
PATH_2 ENDP
;-----

;-----
MAIN PROC FAR
        mov ax, data
        mov ds, ax
        call FREE_MEMORY_PROC
        cmp free_memory, 0h
        jne main_exit
        call PATH_1
        call LOAD

main_exit:
        xor al, al
        mov ah, 4ch
        int 21h
MAIN ENDP

```

finish:

```
CODE ENDS  
END MAIN
```