

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ЛАБОРАТОРНАЯ РАБОТА №6
по дисциплине «Операционные системы»
ТЕМА: Построение модуля динамической
структуры.

Студент гр. 9381

Николаев А.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. 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.

Ход работы.

Был написан и отлажен программный модуль типа .EXE, который выполняет функции:

- 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.
- Вызываемый модуль запускается с использованием загрузчика.
- 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.

Запустили отлаженную программу, когда текущим каталогом являлся каталог с разработанными модулями. Тут программа вызвала другую программу, которая остановилась, ожидая символ с клавиатуры.

Ввели произвольный символ из A-Z.

```
C:\>lab6
Memory has been freed
Address of unavailable memory: 9FFF
Address of the environment: 01FC
Command line tail:
Content of environment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of the module: C:\LAB2.COM
Program ended with code 0
C:\>
```

Запустили отлаженную программу, когда текущим каталогом являлся каталог с разработанными модулями. Тут программа вызвала другую программу, которая остановилась, ожидая символ с клавиатуры.

Введем комбинацию символов Ctrl+C.

```
C:\>lab6
Memory has been freed
Address of unavailable memory: 9FFF
Address of the environment: 01FC
Command line tail:
Content of environment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of the module: C:\LAB2.COM
Program ended with code 0
```

(DOSBox не поддерживает комбинацию клавиш Ctrl + C, поэтому вывод считаем успешным)

Запустили отлаженную программу, когда текущим каталогом являлся какой-либо другой каталог, отличный от того, в котором содержатся разработанные программные модули.

Запуск программы из другого каталога:

```
C:\TASK>lab6
Memory has been freed
Address of unavailable memory: 9FFF
Address of the environment: 01FC
Command line tail:
Content of environment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of the module: C:\TASK\LAB2.COM
Program ended with code 0
```

```
C:\TASK>lab6
Memory has been freed
Address of unavailable memory: 9FFF
Address of the environment: 01FC
Command line tail:
Content of environment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of the module: C:\TASK\LAB2.COM
Program ended with code 0
```

Запустили отлаженную программу, когда модули находятся в разных каталогах.

Файл не был найден.

```
C:\TASK>lab6
Memory has been freed
Error! Can not find the file!
```

Ответы на контрольные вопросы:

1. Как реализовано прерывание Ctrl+C?

Если было нажато сочетание клавиш CTRL+C, то управление передаётся по адресу 0000:008Ch. Этот адрес копируется в PSP функциями 26h и 4Ch и восстанавливается из него при выходе из программы.

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

Код причины завершения 0 – это код успешного завершения программы.

Программа завершается при выполнении функции 4ch прерывания int 21h.

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

Программа завершается в любой ее точке, сразу после обработки прерывания по Ctrl+C. Конкретно в данной программе, точкой завершения была инструкция, ожидающая ввода символа от пользователя (именно на этом моменте произошло завершение программы, функции 01h вектора прерывания 21h).

Вывод:

Был построен загрузочный модуль динамической структуры. Исследован интерфейс между вызывающим и вызываемым модулями по управлению и по данным.

ПРИЛОЖЕНИЕ А

КОД lab6.asm

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

DATA segment
    program db 'lab2.com', 0
    block_parameters dw 0
                    dd 0
                    dd 0
                    dd 0

    memory db 0
    cmd db 1h, 0dh
    pos db 128 dup(0)
    ss_val dw 0
    sp_val dw 0
    psp_val dw 0

    McbCrashError db 'Error! MCB crashed!', 0dh, 0ah, '$'
    NoMemoryError db 'Error! Not enough memory!', 0dh, 0ah, '$'
    AddressError db 'Error! Invalid memory address!', 0dh, 0ah, '$'
    FunctionNumberError db 'Error! Invalid function number', 0dh,
0ah, '$'
    NoFileError db 'Error! Can not find the file!', 0dh, 0ah, '$'
    DiskError db 'Error! Disk error!', 0dh, 0ah, '$'
    MemoryError db 'Error! Insufficient memory', 0dh, 0ah, '$'
    EnvironmentError db 'Error! Wrong string of environment ', 0dh,
0ah, '$'
    WrongFormatError db 'Error! Wrong format', 0dh, 0ah, '$'
    DeviceError db 0dh, 0ah, 'Program ended by device error' , 0dh,
0ah, '$'

    FreeMemoryMessage db 'Memory has been freed' , 0dh, 0ah, '$'

    EndedCode db 0dh, 0ah, 'Program ended with code      ' , 0dh,
0ah, '$'
    EndedCtrl db 0dh, 0ah, 'Program ended by CTRL-break' , 0dh,
0ah, '$'
    Ended31 db 0dh, 0ah, 'Program ended by int 31h' , 0dh, 0ah, '$'
    end_data db 0
DATA ends

CODE segment

assume cs:CODE, ds:DATA, ss:MY_STACK

PRINT proc
    push ax
    mov ah, 09h
    int 21h
    pop ax
```

```

        ret
PRINT endp

FREE_MEM proc
    push ax
    push bx
    push cx
    push dx

    mov ax, offset end_data
    mov bx, offset FINISH
    add bx, ax

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

    jnc FINISH_FREE
    mov memory, 1

MCB_DESTR:
    cmp ax, 7
    jne NOT_ENOUGH_MEMORY
    mov dx, offset McbCrashError
    call PRINT
    jmp RET_F
NOT_ENOUGH_MEMORY:
    cmp ax, 8
    jne ADD_TROUBLE
    mov dx, offset NoMemoryError
    call PRINT
    jmp RET_F
ADD_TROUBLE:
    cmp ax, 9
    mov dx, offset AddressError
    call PRINT
    jmp RET_F
FINISH_FREE:
    mov memory, 1
    mov dx, offset FreeMemoryMessage
    call PRINT

RET_F:
    pop dx
    pop cx
    pop bx
    pop ax
    ret
FREE_MEM endp

LOAD proc
    push ax
    push bx

```

```

push cx
push dx
push ds
push es
mov sp_val, sp
mov ss_val, ss

mov ax, DATA
mov es, ax
mov bx, offset block_parameters
mov dx, offset cmd
mov [bx+2], dx
mov [bx+4], ds
mov dx, offset pos

mov ax, 4b00h
int 21h

mov ss, ss_val
mov sp, sp_val
pop es
pop ds

jnc loads

```

F_N_ERROR:

```

    cmp ax, 1
    jne FILE_ERROR
    mov dx, offset FunctionNumberError
    call PRINT
    jmp load_end

```

FILE_ERROR:

```

    cmp ax, 2
    jne DISK_ERROR
    mov dx, offset NoFileError
    call PRINT
    jmp load_end

```

DISK_ERROR:

```

    cmp ax, 5
    jne MEMORY_ERROR
    mov dx, offset DiskError
    call PRINT
    jmp load_end

```

MEMORY_ERROR:

```

    cmp ax, 8
    jne ENV_ERROR
    mov dx, offset MemoryError
    call PRINT
    jmp load_end

```

ENV_ERROR:

```

    cmp ax, 10
    jne FORMAT_ERROR
    mov dx, offset EnvironmentError
    call PRINT
    jmp load_end

```



```
FORMAT_ERROR:
    cmp ax, 11
    mov dx, offset WrongFormatError
    call PRINT
    jmp load_end
```

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

    cmp ah, 0
    jne CTRL_FUNC
    push di
    mov di, offset EndedCode
    mov [di+26], al
    pop si
    mov dx, offset EndedCode
    call PRINT
    jmp load_end
```

```
CTRL_FUNC:
    cmp ah, 1
    jne DEVICE
    mov dx, offset EndedCtrl
    call PRINT
    jmp load_end
```

```
DEVICE:
    cmp ah, 2
    jne INT_31
    mov dx, offset DeviceError
    call PRINT
    jmp load_end
```

```
INT_31:
    cmp ah, 3
    mov dx, offset Ended31
    call PRINT
```

```
load_end:
    pop dx
    pop cx
    pop bx
    pop ax
    ret
```

```
load endp
```

```
WAY proc
    push ax
    push bx
    push cx
    push dx
    push di
    push si
    push es

    mov ax, psp_val
```

```

        mov es, ax
        mov es, es:[2ch]
        mov bx, 0

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

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

        add bx, 2
        mov di, 0

FIND_LOOP:
        mov dl, es:[bx]
        mov byte ptr [pos+di], dl
        inc di
        inc bx
        cmp dl, 0
        je QUIT_LOOP
        cmp dl, '\'
        jne FIND_LOOP
        mov cx, di
        jmp FIND_LOOP

QUIT_LOOP:
        mov di, cx
        mov si, 0

END_FN:
        mov dl, byte ptr [program+si]
        mov byte ptr [pos+di], dl
        inc di
        inc si
        cmp dl, 0
        jne END_FN

        pop es
        pop si
        pop di
        pop dx
        pop cx
        pop bx
        pop ax
        ret
WAY endp

BEGIN proc far
        push ds
        xor ax, ax
        push ax
        mov ax, DATA
        mov ds, ax

```

```
        mov psp_val, es
        call FREE_MEM
        cmp memory, 0
        je QUIT
        call WAY
        call LOAD
QUIT:
        xor al, al
        mov ah, 4ch
        int 21h

BEGIN   endp

FINISH:
CODE ends
end BEGIN
```