

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. 9383

Лихашва А.Д.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Постановка задачи

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры.

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) Вызываемой модуль запускается с использованием загрузчика.
- 3) После запуска проверяется выполнение загрузчика, а затем результат выполнения вызываемой программы. Необходимо проверить причину завершения и, в зависимости от значения, выводить соответствующее сообщение. Если причина завершения 0, то выводится код завершения.

В качестве вызываемой программы необходимо взять программа 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 полученный результат 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.

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.

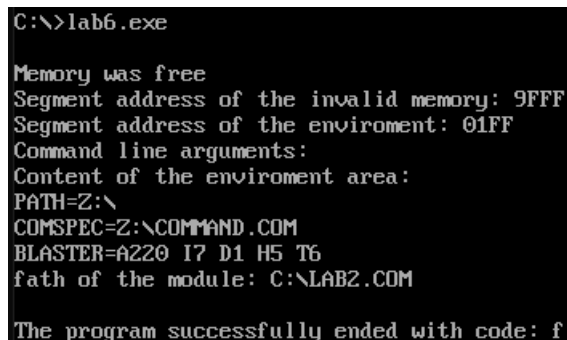
Выполнение шагов лабораторной работы:

Шаг 1.

Был разработан и отлажен программный модуль типа .exe, который выполняет функции заданные в задании.

Шаг 2.

Программа lab6.exe была запущена из директории с разработанными модулями. Был введен символ f.



```
C:\>lab6.exe

Memory was free
Segment address of the invalid memory: 9FFF
Segment address of the enviroment: 01FF
Command line arguments:
Content of the enviroment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
fath of the module: C:\LAB2.COM

The program successfully ended with code: f
```

Рисунок 1: Результат работы программы

Шаг 3.

Программа lab6.exe была запущена из директории с разработанными модулями. Была введена комбинация символов Ctrl + C. В виду того, что в DOSBOX не реализована обработка данной комбинации, Ctrl+C – это символ сердечка

```
C:\>lab6.exe

Memory was free
Segment address of the invalid memory: 9FFF
Segment address of the enviroment: 01FF
Command line arguments:
Content of the enviroment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
♥ath of the module: C:\LAB2.COM

The program successfully ended with code: ♥
```

Рисунок 2: Результат работы программы

Шаг 4.

Программа запущена из другой директории.

```
C:\>lab6\>lab6.exe

Memory was free
Segment address of the invalid memory: 9FFF
Segment address of the enviroment: 01FF
Command line arguments:
Content of the enviroment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
fath of the module: C:\LAB2.COM

The program successfully ended with code: f
```

Рисунок 3: Результат работы программы

Шаг 5.

Программа запущена, когда программный и загрузочный модули находятся в разных директориях.

```
C:\>LAB6.EXE

Memory was free
Error! File not found!
```

Рисунок 4: Результат работы программы

Ответы на контрольные вопросы

1. Как реализовано прерывания Ctrl-C?

При нажатии данной комбинации клавиш управление передается по адресу 0000:008C. Этот адрес копируется в PSP при помощи функций 26h и 4ch. При выходе из программы адрес восстанавливается.

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

При выполнении функции 4ch прерывания int 21h.

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

В том месте, где произошло прерывание, то есть там, где ожидается нажатие клавиш.

Заключение.

Была исследована возможность построения загрузочного модуля динамической структуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: lab.asm

```
AStack SEGMENT    STACK
                DW 128 DUP(?)
AStack ENDS
```

```
DATA SEGMENT
```

```
    P_BLOCK DW 0
                dd 0
                dd 0
                dd 0
```

```
    FILE_NAME DB 'LAB2.com', 0
    flag DB 0
    CMD DB 1h, 0dh
    POS db 128 DUP(0)
```

```
    KEEP_SS DW 0
    KEEP_SP DW 0
    KEEP_PSP DW 0
```

```
    STR_MEMORY_FREE DB 'Memory was free' , 0dh, 0ah, '$'
```

```
    STR_ERROR_CRASH DB 'Error! MCB crashed!', 0dh, 0ah, '$'
```

```
    STR_ERROR_NO_MEMORY DB 'Error! Not enough memory!',
0dh, 0ah, '$'
```

```
    STR_ERROR_ADDRESS DB 'Error! Invalid memory
addressess!', 0dh, 0ah, '$'
```

```
    STR_ERROR_NUMBER DB 'Error! Invalid function number!',
0dh, 0ah, '$'
```

```
    STR_ERROR_NO_FILE DB 'Error! File not found!', 0dh,
0ah, '$'
```

```
    STR_ERROR_DISK DB 'Error with disk!', 0dh, 0ah, '$'
```

```
STR_ERROR_MEMORY DB 'Error! Insufficient memory!', 0dh,  
0ah, '$'
```

```
STR_ERROR_ENVIROMENT DB 'Error! Wrong string of  
environment!', 0dh, 0ah, '$'
```

```
STR_ERROR_FORMAT DB 'Error! Wrong format!', 0dh, 0ah,  
'$'
```

```
STR_ERROR_DEVICE DB 0dh, 0ah, 'Error! Device error!' ,  
0dh, 0ah, '$'
```

```
STR_END_CODE DB 0dh, 0ah, 'The program successfully  
ended with code:      ' , 0dh, 0ah, '$'
```

```
STR_END_CTR DB 0dh, 0ah, 'The program was interrupted  
by ctrl-break' , 0dh, 0ah, '$'
```

```
STR_END_INTER DB 0dh, 0ah, 'The program was ended by  
interruption int 31h' , 0dh, 0ah, '$'
```

```
NEW_STRING DB 0DH,0AH,'$'
```

```
DATA_END DB 0  
DATA ENDS
```

```
CODE SEGMENT
```

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

```
PRINT_STRING PROC
```

```
    push ax  
    mov ah, 09h  
    int 21h  
    pop ax  
    ret
```

```
PRINT_STRING ENDP
```

```
MEMORY_FREE PROC
```

```
    push ax
```

```
push bx
push cx
push dx

mov ax, offset DATA_END
mov bx, offset PR_END
add bx, ax
mov cl, 4
shr bx, cl
add bx, 2bh
mov ah, 4ah
int 21h
jnc end_memory_free
mov flag, 1
```

```
crash_MCB:
    cmp ax, 7
    jne not_memory
    mov dx, offset STR_ERROR_CRASH
    call PRINT_STRING
    jmp ret_f
```

```
not_memory:
    cmp ax, 8
    jne error_address
    mov dx, offset STR_ERROR_NO_MEMORY
    call PRINT_STRING
    jmp ret_f
```

```
error_address:
    cmp ax, 9
    mov dx, offset STR_ERROR_ADDRESS
    call PRINT_STRING
    jmp ret_f
```

```
end_memory_free:
```



```
    mov flag, 1
    mov dx, offset NEW_STRING
    call PRINT_STRING
    mov dx, offset STR_MEMORY_FREE
    call PRINT_STRING
```

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

```
MEMORY_FREE 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 P_BLOCK
    mov dx, offset CMD
    mov [bx+2], dx
    mov [bx+4], ds
    mov dx, offset POS
    mov ax, 4b00h
    int 21h
```

```
mov ss, KEEP_SS
mov sp, KEEP_SP
pop es
pop ds
jnc loads

cmp ax, 1
jne error_file

mov dx, offset STR_ERROR_NUMBER
call PRINT_STRING

jmp end_load
```

```
error_file:
    cmp ax, 2
    jne error_disk
    mov dx, offset STR_ERROR_NO_FILE
    call PRINT_STRING
    jmp end_load
```

```
error_disk:
    cmp ax, 5
    jne error_memory
    mov dx, offset STR_ERROR_DISK
    call PRINT_STRING
    jmp end_load
```

```
error_memory:
    cmp ax, 8
    jne error_enviroment
    mov dx, offset STR_ERROR_MEMORY
    call PRINT_STRING
    jmp end_load
```

```
error_enviroment:
```

```
    cmp ax, 10
    jne error_format
    mov dx, offset STR_ERROR_ENVIROMENT
    call PRINT_STRING
    jmp end_load
```

error_format:

```
    cmp ax, 11
    mov dx, offset STR_ERROR_FORMAT
    call PRINT_STRING
    jmp end_load
```

loads:

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

    cmp ah, 0
    jne contrl
    push di
    mov di, offset STR_END_CODE
    mov [di+44], al
    pop si
    mov dx, offset NEW_STRING
    call PRINT_STRING
    mov dx, offset STR_END_CODE
    call PRINT_STRING
    jmp end_load
```

contrl:

```
    cmp ah, 1
    jne device
    mov dx, offset STR_END_CTR
    call PRINT_STRING
    jmp end_load
```

```
device:
    cmp ah, 2
    jne interrupt
    mov dx, offset STR_ERROR_DEVICE
    call PRINT_STRING
    jmp end_load
```

```
interrupt:
    cmp ah, 3
    mov dx, offset STR_END_INTER
    call PRINT_STRING
```

```
end_load:
    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
```

```

find_path:
    inc bx
    cmp byte ptr es:[bx-1], 0
    jne find_path
    cmp byte ptr es:[bx+1], 0
    jne find_path
    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 end_find_loop
    cmp dl, '\'
    jne find_loop
    mov cx, di
    jmp find_loop

end_find_loop:
    mov di, cx
    mov si, 0

end_f:
    mov dl, byte ptr [FILE_NAME + si]
    mov byte ptr [POS + di], dl
    inc di
    inc si
    cmp dl, 0
    jne end_f

    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 MEMORY_FREE
        cmp flag, 0
        je end_begin
        call PATH
        call LOAD

end_begin:
        xor al, al
        mov ah, 4ch
        int 21h

BEGIN ENDP

PR_END:
CODE ENDS
END BEGIN

```