

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. 0382

Михайлова О.Д.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

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 2. 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.

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.

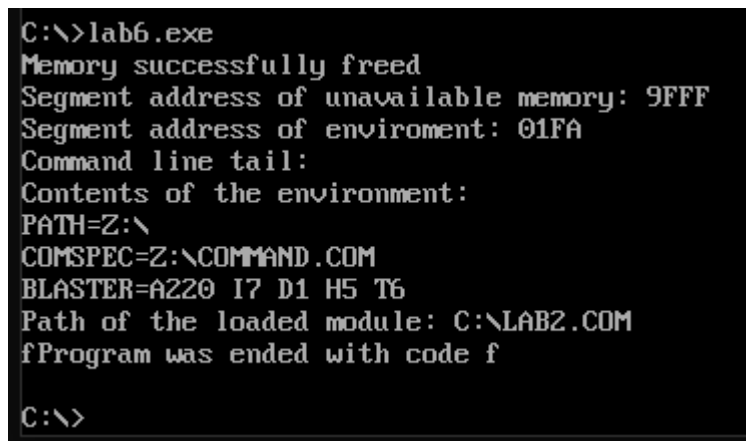
Для выполнения задания были использованы шаблоны из методических указаний, а также были добавлены следующие процедуры:

- PRINT_STRING – процедура вывода строки на экран;
- FREE_MEMORY – освобождение памяти и обработка ошибок;
- LOAD – загрузка модуля и обработка ошибок;
- PATH_READ – получение пути к файлу;

Шаг 1. Был написан и отлажен программный модуль .EXE, который выполняет все заданные в условии функции. В качестве вызываемой программы

была использована модифицированная программа из ЛР2: перед выходом из нее была добавлена функция ввода символа с клавиатуры.

Шаг 2. Была запущена отлаженная программа, когда текущим каталогом является каталог с разработанными модулями. После запуска был введен символ “f”.



```
C:\>lab6.exe
Memory successfully freed
Segment address of unavailable memory: 9FFF
Segment address of enviroment: 01FA
Command line tail:
Contents of the environment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of the loaded module: C:\LAB2.COM
fProgram was ended with code f
C:\>
```

Рисунок 1 - Результат запуска модуля lab6.exe, когда текущим каталогом является каталог с разработанными модулями, с введенным символом f.

Шаг 3. Была запущена отлаженная программа, когда текущим каталогом является каталог с разработанными модулями. После запуска была введена комбинация символов Ctrl-C.

```

C:\>lab6.exe
Memory successfully freed
Segment address of unavailable memory: 9FFF
Segment address of enviroment: 01FA
Command line tail:
Contents of the environment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of the loaded module: C:\LAB2.COM
♥Program was ended with code ♥
C:\>_

```

Рисунок 2 - Результат запуска модуля lab6.exe, когда текущим каталогом является каталог с разработанными модулями, с введенной комбинацией символов Ctrl-C.

Шаг 4. Была запущена отлаженная программа, когда текущим каталогом является другой каталог, отличный от того, в котором содержатся разработанные программные модули.

```

C:\TEST>lab6.exe
Memory successfully freed
Segment address of unavailable memory: 9FFF
Segment address of enviroment: 01FA
Command line tail:
Contents of the environment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of the loaded module: C:\TEST\LAB2.COM
fProgram was ended with code f

```

Рисунок 3 - Результат запуска модуля lab6.exe, когда текущим каталогом является каталог, отличный от того, в котором содержатся разработанные программные модули, с введенным символом f.

```
C:\TEST>lab6.exe
Memory successfully freed
Segment address of unavailable memory: 9FFF
Segment address of enviroment: 01FA
Command line tail:
Contents of the environment:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of the loaded module: C:\TEST\LAB2.COM
♥Program was ended with code ♥
```

Рисунок 4 - Результат запуска модуля lab6.exe, когда текущим каталогом является каталог, отличный от того, в котором содержатся разработанные программные модули, с введенной комбинацией символов Ctrl-C.

Шаг 5. Была запущена отлаженная программа, когда модули находятся в разных каталогах.

```
C:\TEST>lab6.exe
Memory successfully freed
File not found
```

Рисунок 5 - Результат запуска модуля lab6.exe, когда модули находятся в разных каталогах.

Исходный код программы см. в приложении А.

Ответы на контрольные вопросы.

1. Как реализовано прерывание Ctrl-C?

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 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?

В точке вызова функции 4Ch прерывания int 21h.

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

В точке, где произошли ввод и считывание сочетания клавиш Ctrl-C.

Выводы.

В ходе работы были исследованы возможности построение загрузочного модуля динамической структуры. Также был исследован интерфейс между вызывающим и вызываемым модулями по управлению и по данным.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab6.asm

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

```
DATA SEGMENT
```

```
    P_BLOCK dw 0
            dd 0
            dd 0
            dd 0
```

```
    KEEP_SS dw 0
    KEEP_SP dw 0
    KEEP_PSP dw 0
```

```
    FILE_NAME db 'lab2.com', 0
    NEW_CMD_LINE db 1h, 0Dh
    PATH db 128 dup(0)
```

```
    MEM_CMB db 'Memory error: control block destroyed', 0Dh, 0Ah,
'$'
```

```
    MEM_NOT_ENOUGH db 'Memory error: not enough memory to execute
the function', 0Dh, 0Ah, '$'
```

```
    MEM_WRONG_ADDR db 'Memory error: invalid block address', 0Dh,
0Ah, '$'
```

```
    MEM_FREE_SUCCESS db 'Memory successfully freed', 0Dh, 0Ah, '$'
```

```
    WRONG_FUNC_NUM db 'Invalid 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_ENV_STR db 'Wrong environment string', 0Dh, 0Ah, '$'
```

```
    WRONG_FORMAT db 'Wrong format', 0Dh, 0Ah, '$'
```

```
    END_NORMAL db 'Program was ended with code ', 0Dh, 0Ah, '$'
```

```
    END_CTRL db 'Termination by Ctrl-Break', 0Dh, 0Ah, '$'
```

```
    END_DEVICE db 'Termination by device error', 0Dh, 0Ah, '$'
```

```
    END_31h db 'Termination by int 31h', 0Dh, 0Ah, '$'
```

```
    data_end db 0
    flag db 0
```

```
DATA ENDS
```

```
CODE SEGMENT
```

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

```
PRINT_STRING PROC near
```

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



```

        pop ax
        ret
PRINT_STRING ENDP

FREE_MEMORY PROC near
    push ax
    push bx
    push cx
    push dx

    mov ax, offset data_end
    mov bx, offset proc_end
    add bx, ax
    mov cl, 4
    shr bx, cl
    add bx, 2bh
    mov ah, 4ah
    int 21h
    jnc free_memory_success
    mov flag, 1

mem_error_7:
    cmp ax, 7
    jne mem_error_8
    mov dx, offset MEM_CMB
    call PRINT_STRING
    jmp end_free_memory

mem_error_8:
    cmp ax, 8
    jne mem_error_9
    mov dx, offset MEM_NOT_ENOUGH
    call PRINT_STRING
    jmp end_free_memory

mem_error_9:
    cmp ax, 9
    mov dx, offset MEM_WRONG_ADDR
    call PRINT_STRING
    jmp end_free_memory

free_memory_success:
    mov dx, offset MEM_FREE_SUCCESS
    call PRINT_STRING

end_free_memory:
    pop dx
    pop cx
    pop bx
    pop ax
    ret

FREE_MEMORY ENDP

LOAD PROC near

```

```

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 NEW_CMD_LINE
mov [bx+2], dx
mov [bx+4], ds
mov dx, offset PATH
mov ax, 4b00h
int 21h

mov ss, KEEP_SS
mov sp, KEEP_SP
pop es
pop ds

jnc load_success

load_error_1:
    cmp ax, 1
    jne load_error_2
    mov dx, offset WRONG_FUNC_NUM
    call PRINT_STRING
    jmp end_load

load_error_2:
    cmp ax, 2
    jne load_error_5
    mov dx, offset FILE_NOT_FOUND
    call PRINT_STRING
    jmp end_load

load_error_5:
    cmp ax, 5
    jne load_error_8
    mov dx, offset DISK_ERROR
    call PRINT_STRING
    jmp end_load

load_error_8:
    cmp ax, 8
    jne load_error_10
    mov dx, offset NOT_ENOUGH_MEMORY
    call PRINT_STRING
    jmp end_load

load_error_10:

```

```

        cmp ax, 10
        jne load_error_11
        mov dx, offset WRONG_ENV_STR
        call PRINT_STRING
        jmp end_load

load_error_11:
        cmp ax, 11
        mov dx, offset WRONG_ENV_STR
        call PRINT_STRING
        jmp end_load

load_success:
        mov ah, 4dh
        mov al, 00h
        int 21h

        cmp ah, 0
        jne end_error_1
        push di
        mov di, offset END_NORMAL
        mov [di+28], al
        pop si
        mov dx, offset END_NORMAL
        call PRINT_STRING
        jmp end_load

end_error_1:
        cmp ah, 1
        jne end_error_2
        mov dx, offset END_CTRL
        call PRINT_STRING
        jmp end_load

end_error_2:
        cmp ah, 2
        jne end_error_3
        mov dx, offset END_DEVICE
        call PRINT_STRING
        jmp end_load

end_error_3:
        cmp ah, 3
        mov dx, offset END_31h
        call PRINT_STRING

end_load:
        pop dx
        pop cx
        pop bx
        pop ax
        ret

LOAD ENDP

```

```

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

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

find_two_zeros:
    inc bx
    mov dl, es:[bx-1]
    cmp dl, 0
    jne find_two_zeros
    mov dl, es:[bx]
    cmp dl, 0
    jne find_two_zeros

    add bx, 3
    xor si, si
    mov si, offset PATH

while_not_point:
    mov dl, es:[bx]
    mov [si], dl
    cmp dl, '.'
    je loop_back

    inc bx
    inc si
    jmp while_not_point

loop_back:
    mov dl, [si]
    cmp dl, '\'
    je break_loop
    mov dl, 0h
    mov [si], dl
    dec si
    jmp loop_back

break_loop:
    mov di, offset FILE_NAME
    inc si

loop_new_file:
    mov dl, [di]
    cmp dl, 0
    je end_path_read
    mov [si], dl
    inc di
    inc si
    jmp loop_new_file

```

```

end_path_read:
    pop di
    pop bx
    pop es
    pop dx
    pop si
    pop ax
    ret

PATH_READ endp

Main PROC FAR
    mov ax, DATA
    mov ds, ax

    call FREE_MEMORY
    cmp flag, 0
    jne final
    call PATH_READ
    call LOAD

final:
    mov ah, 4Ch
    int 21h
Main ENDP
proc_end:
CODE ENDS
END Main

```