

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. 9383

Нистратов Д.Г.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Постановка задачи.

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 4BO0B прерывания 21h. Все загрузочные модули находятся в одном каталоге. Необходимо обеспечить возможность запуска модуля динамической структуры из любого каталога.

Последовательность действий

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.

Шаг 6. Ответить на контрольные вопросы.

Выполнение работы.

Шаг 1. Был написан программный модуль типа .EXE, выполняющий функции из шага 1 Последовательность действий.

Шаг 2. Программа была запущена в каталоге являющимся каталогом с разработанными модулями. Программа была остановлена символом “s”, введенным с клавиатуры. См. Изображение 1

```

D:\LETI\OS\MASM>LAB6.EXE
Memory is free.
First byte of unavailable memory:9FFF
Environment segment:02DB
Command line is empty.
Environment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Module path:D:\LETI\OS\MASM\LAB2.COMs
Termination with msg s

```

Изображение 1 – Стандартный запуск программы, с прерыванием символом

Шаг 3. Программа была запущена и остановлена прерыванием “CTRL-C”, однако DosBox не поддерживает данное прерывание и ставится символ “сердца”, см. Изображение 2

```

D:\LETI\OS\MASM>LAB6.EXE
Memory is free.
First byte of unavailable memory:9FFF
Environment segment:02DB
Command line is empty.
Environment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Module path:D:\LETI\OS\MASM\LAB2.COM♥
Termination with msg ♥

```

Изображение 2 – Запуск программы с прерыванием CTRL-C

Шаг 4. Программа была запущена в каталоге, в котором не располагаются разработанные программные модули. См. Изображение 3

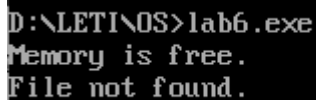
```

D:\LETI\OS>lab6.exe
Memory is free.
First byte of unavailable memory:9FFF
Environment segment:02DB
Command line is empty.
Environment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Module path:D:\LETI\OS\LAB2.COMs
Termination with msg s

```

Изображение 3 – Запуск программы в другом каталоге

Шаг 5. Программа была запущена в каталоге, в котором отсутствует модуль. См. Изображение 4



```
D:\LETI\OS>lab6.exe
Memory is free.
File not found.
```

Изображение 4 – Запуск программы с отсутствием модуля

Шаг 6. Ответы на вопросы:

1. Как реализовано прерывание Ctrl-C?

При нажатии Ctrl-C вызывается прерывание 23h. Адрес, по которому передается управление, устанавливается в 0000:008c

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

Заканчивает при выполнении функции 4Ch прерывания 21h.

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

В месте ожидания ввода функции 01h прерывания 21h.

Заключение.

В ходе лабораторной работы были исследованы возможности построения загрузочного модуля динамической структуры, а также был разработан интерфейс между вызывающим и вызываемым модулями по управлению и по данным.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab6.asm

```
AStack  SEGMENT STACK
```

```
        DW 64 DUP(?)
```

```
AStack  ENDS
```

```
DATA SEGMENT
```

```
    PARAMETR_BLOCK dw 0
```

```
        db 0
```

```
        db 0
```

```
        db 0
```

```
    CMD_LINE db 1h,0DH
```

```
    PATH_STR db 128 dup(0)
```

```
    FILE_NAME db "lab2.com", 0
```

```
    KEEP_SS dw 0
```

```
    KEEP_SP dw 0
```

```
    NORMAL_TERMINATION db 0DH, 0AH, "Termination with msg  ", 0DH, 0AH, '$'
```

```
    CTRL_TERMINATION db 0DH, 0AH, "Termination by Ctrl-C ", 0DH, 0AH, '$'
```

```
    DEVICE_TERMINATION db 0DH, 0AH, "Termination by device error ", 0DH, 0AH, '$'
```

```
    FUNC_TERMINATION db 0DH, 0AH, "Termination by function 31H ", 0DH, 0AH, '$'
```

```
    MEMORY_BLOCK_ERROR db "Memory control block error. ", 0DH, 0AH, '$'
```

```
    LOW_MEMORY db "Not enough memory. ", 0DH, 0AH, '$'
```

```
    WRONG_PTR db "Invalid memory address. ", 0DH, 0AH, '$'
```

```
    MEMORY_FREE_SUCCESS db "Memory is free. ", 0DH, 0AH, '$'
```

```
    WRONG_FUNC_NUMBER db "Wrong 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_ENVIRONMENT_STRING db "Wrong environment string.", 0DH, 0AH, '$'
```

```
    WRONG_FORMAT db "Wrong format.", 0DH, 0AH, '$'
```

```
    KEEP_FLAG db 0
```

```

    KEEP_DATA db 0
DATA    ENDS

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

WRITE proc near
    push ax
    mov ah, 9h
    int 21h
    pop ax
    ret
WRITE endp

FREE_MEMORY proc near
    push ax
    push bx
    push dx

    mov ax, offset MAIN_ENDS
    mov bx, offset KEEP_DATA
    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

    jnc MEM_S_P
    mov KEEP_FLAG, 1

    cmp ax, 7
    je M_B_D
    cmp ax, 8

```

```
je L_M
cmp ax, 9
je I_A
```

M_B_D:

```
mov dx, offset MEMORY_BLOCK_ERROR
call WRITE
jmp memory_free_end
```

L_M:

```
mov dx, offset LOW_MEMORY
call WRITE
jmp memory_free_end
```

I_A:

```
mov dx, offset WRONG_PTR
call WRITE
jmp memory_free_end
```

MEM_S_P:

```
mov dx, offset MEMORY_FREE_SUCCESS
call WRITE
```

memory_free_end:

```
pop dx
pop bx
pop ax
ret
```

FREE_MEMORY endp

PARAMETR_BLOCK_CREATE proc near

```
push bx
push dx
```

```
mov bx, offset PARAMETR_BLOCK
mov dx, offset CMD_LINE
mov [bx+2], dx
mov [bx+4], ds
```

```
pop dx
pop bx
ret
```


PARAMETR_BLOCK_CREATE endp

PATH_FIND proc near

push ax

push si

push dx

push es

push bx

push di

mov ax, es:[2Ch]

mov es, ax

xor si, si

FOUND_ZERO:

inc si

mov dl, es:[si-1]

cmp dl, 0

jne FOUND_ZERO

mov dl, es:[si]

cmp dl, 0

jne FOUND_ZERO

add si, 3

mov bx, offset PATH_STR

LOOP_FINDER:

mov dl, es:[si]

mov [bx], dl

cmp dl, '.'

je LOOP_BREAK

inc bx

inc si

jmp LOOP_FINDER

LOOP_BREAK:

mov dl, [bx]

cmp dl, '\'

je END_LOOP

```

    mov dl, 0h
    mov [bx], dl
    dec bx
    jmp LOOP_BREAK
END_LOOP:
    mov di, offset FILE_NAME
    inc bx
NEW_LOOP:
    mov dl, [di]
    cmp dl, 0
    je END_PATH_FIND
    mov [bx], dl
    inc di
    inc bx
    jmp NEW_LOOP
END_PATH_FIND:
    pop di
    pop bx
    pop es
    pop dx
    pop si
    pop ax
    ret
PATH_FIND endp

```

```

MAIN_HANDLER proc near

```

```

    push ax
    push bx
    push cx
    push dx
    push di
    push si
    push es
    push ds

```

```

    mov KEEP_SP, sp
    mov KEEP_SS, ss

```

```

mov ax, data
mov es, ax
mov dx, offset PATH_STR
mov bx, offset PARAMETR_BLOCK

mov ax, 4b00h
int 21h

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

jnc LOADED_SUCCESS
cmp ax, 1
je WRONG_FUNC_NUM
cmp ax, 2
je FILE_NOT_FOUND_ERR
cmp ax, 5
je DISK_ERR_FOUND
cmp ax, 8
je NOT_EN_MEM
cmp ax, 10
je STRING_ERR
cmp ax, 11
je FORMAT_ERR
WRONG_FUNC_NUM:
    mov dx, offset WRONG_FUNC_NUMBER
    call WRITE
    jmp END_HANDLER
FILE_NOT_FOUND_ERR:
    mov dx, offset FILE_NOT_FOUND
    call WRITE
    jmp END_HANDLER
DISK_ERR_FOUND:
    mov dx, offset DISK_ERROR
    call WRITE
    jmp END_HANDLER

```

NOT_EN_MEM:

```
    mov dx, offset NOT_ENOUGH_MEMORY
    call WRITE
    jmp END_HANDLER
```

STRING_ERR:

```
    mov dx, offset WRONG_ENVIRONMENT_STRING
    call WRITE
    jmp END_HANDLER
```

FORMAT_ERR:

```
    mov dx, offset WRONG_FORMAT
    call WRITE
    jmp END_HANDLER
```

LOADED_SUCCESS:

```
    mov ax, 4d00h
    int 21h
```

```
    cmp ah, 0
    jmp NORMAL
    cmp ah, 1
    jmp CTRL_C
    cmp ah, 2
    jmp DEVICE_TER
    cmp ah, 3
    jmp FUNC_TER
```

NORMAL:

```
    mov di, offset NORMAL_TERMINATION
    add di, 24
    mov [di], al
    mov dx, offset NORMAL_TERMINATION
    call WRITE
    jmp END_HANDLER
```

CTRL_C:

```
    mov dx, offset CTRL_TERMINATION
    call WRITE
    jmp END_HANDLER
```

DEVICE_TER:

```
    mov dx, offset DEVICE_TERMINATION
    call WRITE
```

```

    jmp END_HANDLER
FUNC_TER:
    mov dx, offset FUNC_TERMINATION
    call WRITE
    jmp END_HANDLER
END_HANDLER:
    pop si
    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    ret
MAIN_HANDLER endp

MAIN proc far
    push ds
    push ax
    mov ax,data
    mov ds,ax

    call FREE_MEMORY
    call PATH_FIND
    call MAIN_HANDLER

    mov ah, 4ch
    int 21h
MAIN_ENDS:
MAIN endp
CODE ends
END Main

```