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

# ОТЧЕТ

**по лабораторной работе №6**

**по дисциплине «Операционные системы»**

**Тема: Построение модуля динамической структуры**

Студент гр. 9383 _____ Хотяков Е.П.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

**Постановка задачи.**

***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

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.

**Исходный код.**

Исходный код представлен в приложении А.

**Результаты исследования проблем.**

***Шаг 1.*** Был написан и отлажен программный модуль типа .EXE, который выполняет, требуемый в задании функционал.

***Шаг 2.*** Программа была запущена, когда текущим каталогом является каталог с разработанными модулями. Был введен символ “x”.


Рисунок 1 – Демонстрация работы программы

***Шаг 3.*** Программа была запущена, когда текущим каталогом является каталог с разработанными модулями. Для прерывания была введена комбинация клавиш Ctrl+C, но так как в DOSBox не реализовано это прерывание, программа считает комбинацию клавиш за символ сердца.


Рисунок 2 – Демонстрация работы программы, при вводе в вызываемый модуль комбинации клавиш Ctrl+C.

***Шаг 4.*** Программа была запущена, когда текущим каталогом является другой каталог, отличный от того, в котором содержатся разработанные программные модули. Снова для завершения программы воспользуемся введением символа ‘z’, а затем комбинацией клавиш Ctrl+C.


Рисунок 3 – Демонстрация работы программы, запущенной из другого каталога, при вводе в вызываемый модуль символа ‘x’.

***Шаг 5.*** Программа была запущена, когда модули находятся в разных каталогах.

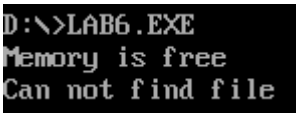


Рисунок 4 – Демонстрация корректной обработки ошибки при попытке вызова модуля, которого нет в каталоге.

***Ответ 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). Этот адрес копируется в PSP с помощью функций 26h и 4ch. При выходе из программы этот адрес восстанавливается.

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 том месте, где ожидался ввод сочетания клавиш Ctrl+C(в точке вызова функции 01h прерывания int 21h)

**Выводы.**

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

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

## lab6.asm:

```
MY_STACK SEGMENT STACK
     DW 64 DUP(?)
MY_STACK ENDS

DATA SEGMENT
     params_block dw 0
                  dd 0
                  dd 0
                  dd 0

    file_name db "LAB2.COM", 0
    SAVED_PSP dw 0
    SAVED_SP dw 0
    SAVED_SS dw 0

    new_command_line db 1h,0DH
    path_ db 128 dup(0)

    FREE_MEM_SUCCESS db "Memory is free ", 0DH, 0AH, '$'
    CONTROL_BLOCK_ERROR db "Control block was destroyed ", 0DH, 0AH, '$'
    FUNCTION_MEM_ERROR db "Not enough memory for function ", 0DH, 0AH,
'$'
    WRONG_ADDRESS db "Wrong address for block of memory", 0DH, 0AH, '$'

    WRONG_NUMBER_ERROR db "Wrong function number ", 0DH, 0AH, '$'
    CANT_FIND_ERROR db "Can not find file ", 0DH, 0AH, '$'
    DISK_ERROR db "Erorror on disk ", 0DH, 0AH, '$'
    MEMORY_ERROR db "Not enough memory ", 0DH, 0AH, '$'
    WRONG_STRING_ERROR db "Wrong environment string ", 0DH, 0AH, '$'
    WRONG_FORMAT_ERROR db "Wrong format ", 0DH, 0AH, '$'

    NORMAL_END db 0DH, 0AH, "End code is:  ", 0DH, 0AH, '$'
    BREAK_END db 0DH, 0AH, "End by ctrl+break ", 0DH, 0AH, '$'
    ERROR_END db 0DH, 0AH, "End by device error ", 0DH, 0AH, '$'
    FUNCTION_END db 0DH, 0AH, "End by function 31h", 0DH, 0AH, '$'

    ERR_FLAG db 0

    DATA_END db 0
DATA ENDS

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

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

```
FREE_MEM PROC near
    push ax
    push bx
    push dx
    push cx

    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 FMS
    mov ERR_FLAG, 1

    cmp ax, 7
    je CBE
    cmp ax, 8
    je FME
    cmp ax, 9
    je WA

CBE:
    mov dx, offset CONTROL_BLOCK_ERROR
    call WRITE_STRING
    jmp FREE_MEM_END
FME:
    mov dx, offset FUNCTION_MEM_ERROR
    call WRITE_STRING
    jmp FREE_MEM_END
WA:
    mov dx, offset WRONG_ADDRESS
    call WRITE_STRING
    jmp FREE_MEM_END
FMS:
    mov dx, offset FREE_MEM_SUCCESS
    call WRITE_STRING

FREE_MEM_END:
    pop dx
    pop bx
    pop cx
    pop ax
    ret
FREE_MEM ENDP

PATH PROC
     push ax
     push bx
     push cx
     push dx
     push di
     push si
```

```
      push es

      mov ax, SAVED_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 [path_ + 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 [path_ + 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

LOAD PROC
    push ax
      push bx
      push cx
      push dx
      push ds
```

```
      push es

      mov SAVED_SP, sp
      mov SAVED_SS, ss
      mov ax, DATA
      mov es, ax
     mov bx, offset params_block
      mov dx, offset new_command_line
      mov [bx+2], dx
      mov [bx+4], ds
      mov dx, offset path_
      mov ax, 4b00h
      int 21h

      mov ss, SAVED_SS
      mov sp, SAVED_SP
      pop es
      pop ds

     jnc LOAD_SUCCESS
     cmp ax, 1
     je E_1
     cmp ax, 2
     je E_2
     cmp ax, 5
     je E_5
     cmp ax, 8
     je E_8
     cmp ax, 10
     je E_10
     cmp ax, 11
     je E_11
E_1:
     mov dx, offset WRONG_NUMBER_ERROR
     call WRITE_STRING
     jmp LOAD_END
E_2:
     mov dx, offset CANT_FIND_ERROR
     call WRITE_STRING
     jmp LOAD_END
E_5:
     mov dx, offset DISK_ERROR
     call WRITE_STRING
     jmp LOAD_END
E_8:
     mov dx, offset MEMORY_ERROR
     call WRITE_STRING
     jmp LOAD_END
E_10:
     mov dx, offset WRONG_STRING_ERROR
     call WRITE_STRING
     jmp LOAD_END
E_11:
     mov dx, offset WRONG_FORMAT_ERROR
     call WRITE_STRING
     jmp LOAD_END
```

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

    cmp ah, 0
    jmp NEND
    cmp ah, 1
    jmp BEND
    cmp ah, 2
    jmp EEND
    cmp ah, 3
    jmp FEND
NEND:
    mov di, offset NORMAL_END
    add di, 15
    mov [di], al
    mov dx, offset NORMAL_END
    call WRITE_STRING
    jmp LOAD_END

BEND:
    mov dx, offset BREAK_END
    call WRITE_STRING
    jmp LOAD_END
EEND:
    mov dx, offset ERROR_END
    call WRITE_STRING
    jmp LOAD_END
FEND:
    mov dx, offset FUNCTION_END
    call WRITE_STRING

LOAD_END:
    pop si
     pop di
     pop dx
     pop cx
     pop bx
     pop ax
    ret
LOAD ENDP

MAIN PROC far
    push dx
    push ax
    mov ax, DATA
    mov ds, ax

    call FREE_MEM
    cmp ERR_FLAG, 1
    je MAIN_END
    call PATH
    call LOAD

MAIN_END:
    xor al, al
    mov ah, 4ch
```

```
    int 21h

PROC_END:
MAIN endp
CODE ends
END Main
```