**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 _____ Соседков К.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

**Цель работы.**

Исследование возможности построения загрузочного модуля динамической структуры. В отличии 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.**

При выполнении работы был написан и отлажен программный модуль типа .EXE, который выполняет следующие 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.

Результаты работы программы когда текущим каталогом является каталог с разработанными модулями представлены на Рисунках 1 и 2.

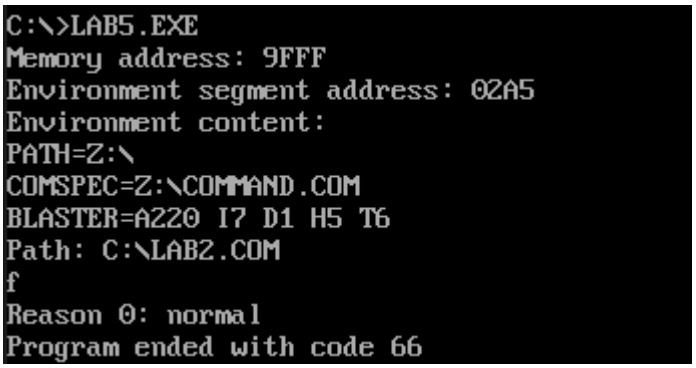


*Рисунок 1: Результат работы программы после ввода символа 'f'*

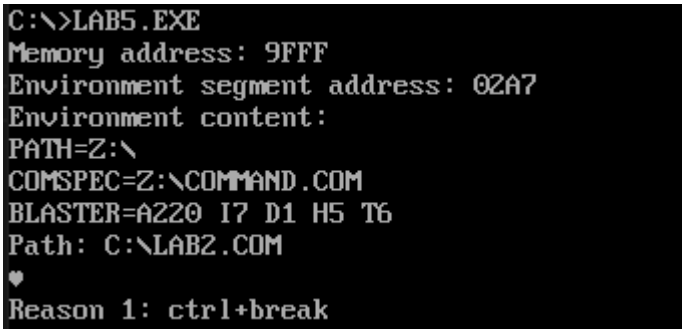


*Рисунок 2: Результат работы программы после ввода комбинации символов ctrl+break*

Результаты работы программы когда текущим каталогом является другой каталог(отличный от первого) представлены на Рисунках 3 и 4.


*Рисунок 3: Результат работы программы после ввода символа 'a'*


*Рисунок 4: Результат работы программы после ввода комбинации символов ctrl+break*

Результаты работы программы когда модули находятся в разных каталогах представлен на Рисунке 5.


*Рисунок 5: Результат работы программы когда модули находятся в разных каталогах*

**Контрольные вопросы.**

**1) Как реализовано прерывание Ctrl-C?**

При нажатии комбинации клавиш ctrl+c управление передается по адресу 0000:008Ch. Этот адрес копируется в PSP функциями 26h и 4Ch и восстанавливается при выходе.

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

Вызываемая программа заканчивается в месте вызова функции 4Ch прерывания 21h.

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

Вызываемая программа заканчивается в месте вызова функции 01h прерывания 21h.

**Выводы.**

При выполнении лабораторной работы были изучены возможности построения загрузочного модуля динамической структуры.

## ПРИЛОЖЕНИЕ А
## ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: lab6.asm

```
ASTACK segment stack
        dw 512 dup(?)
ASTACK ends




DATA segment
   EXEC_Parameter_Block dw 0
                                        db 0
                                        db 0
                                        db 0

   free_memory_error_msg db 'Free memory error',0dh,0ah,'$'
   file_path dw 50 dup(0)
   program db 'LAB2.COM', 0
   exit_code db 'Program ended with code $'
   endl db ' ',0dh,0ah,'$'

        KEEP_SS DW 0
        KEEP_SP DW 0
        KEEP_PSP DW 0

        error_wrong_function_number db 'Error 1: wrong function number',0dh,0ah,'$'
        error_file_not_found db 'Error 2: file not found',0dh,0ah,'$'
        error_disk db 'Error 5: disk error',0dh,0ah,'$'
        error_not_enough_memory db 'Error 8: not enough memory',0dh,0ah,'$'
        error_wrong_environment_string db 'Error 10: wrong environment string',0dh,0ah,'$'
```

```
        error_wrong_format db 'Error 11: wrong format',0dh,0ah,'$'

        reason_0 db 'Reason 0: normal',0dh,0ah,'$'
        reason_1 db 'Reason 1: ctrl+break',0dh,0ah,'$'
        reason_2 db 'Reason 2: device error',0dh,0ah,'$'
        reason_3 db 'Reason 3: 31h',0dh,0ah,'$'
DATA ends



CODE segment
   assume cs:CODE, ds:DATA, ss:ASTACK



TETR_TO_HEX PROC NEAR
                and   al,0Fh
                cmp   al,09
                jbe   NEXT
                add   al,07
NEXT:           add   al,30h
                ret
TETR_TO_HEX ENDP

BYTE_TO_HEX PROC NEAR
                push  cx
                mov   ah,al
                call  TETR_TO_HEX
```

```
        xchg  al,ah
        mov   cl,4
        shr   al,cl
        call  TETR_TO_HEX
        pop   cx
        ret
BYTE_TO_HEX ENDP



WRD_TO_HEX PROC NEAR
        push  BX
        mov         BH,AH
        call  BYTE_TO_HEX
        mov         [DI],AH
        dec         DI
        mov         [DI],AL
        dec         DI
        mov         AL,BH
        xor         AH,AH
        call  BYTE_TO_HEX
        mov         [DI],AH
        dec         DI
        mov         [DI],AL
        pop         BX
        ret
WRD_TO_HEX      ENDP
```

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

```
FREE_MEMORY_ERROR proc near
   push dx
   mov dx, offset free_memory_error_msg
   call PRINT
   pop dx
   ret
FREE_MEMORY_ERROR endp
```

```
FREE_MEMORY proc far
        push bx
        push ax
        push cx
        push dx

        mov bx, offset _end
        mov ax, es
        sub bx, ax
        add bx, 100h
        mov ah, 4Ah
        int 21h
        jc error
        jmp fm_exit
        error:
                call FREE_MEMORY_ERROR
        fm_exit:
                pop dx
                pop cx
                pop ax
                pop bx
                ret
FREE_MEMORY endp
```

```
READ_PATH PROC
```

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

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

find_path_loop:
        cmp byte ptr es:[bx],00h
        jne continue
    cmp byte ptr es:[bx+1],00h
    jne continue
    jmp find_path_loop_end
    continue:
       inc bx
       jmp find_path_loop

find_path_loop_end:
        add bx, 4
        xor si, si
    lea di, file_path

read_path_loop:
    mov dl, es:[bx+si]
        cmp   byte ptr es:[bx+si],0
        je              copy_program_name
        mov   [di], dl
        inc     di
        inc     si
        jmp   read_path_loop
```

```
copy_program_name:
    sub di, 8
    mov cx, 8
    lea si, program
    copy_loop:
        mov al, [si]
        mov [di], al
        inc si
        inc di
    loop copy_loop
    mov [di], byte ptr '$'

          pop es
          pop si
          pop di
          pop dx
          pop cx
          pop bx
          pop ax
          ret
READ_PATH ENDP



MAIN proc far
          mov ax, DATA
          mov ds, ax

    call FREE_MEMORY
    call READ_PATH

    push        ds
```

```
        pop    es
        mov   bx,offset EXEC_Parameter_Block
    mov     dx, offset file_path
        mov   keep_sp, SP
        mov   keep_ss, SS

    mov ax, 4b00h
        int 21h
    mov cx, ax

        mov ss, KEEP_SP
        mov sp, KEEP_SP




    jnc run_prog

        cmp ax, 1
        je error_1
        cmp ax, 2
        je error_2
        cmp ax, 5
        je error_5
        cmp ax, 8
        je error_8
        cmp ax, 10
        je error_10
        cmp ax, 11
        je error_11
        jmp exit

error_1:
```

```
        mov dx, offset error_wrong_function_number
        call PRINT
        jmp exit
error_2:
        mov dx, offset error_file_not_found
        call PRINT
        jmp exit
error_5:
        mov dx, offset error_disk
        call PRINT
        jmp exit
error_8:
        mov dx, offset error_not_enough_memory
        call PRINT
        jmp exit
error_10:
        mov dx, offset error_wrong_environment_string
        call PRINT
        jmp exit
error_11:
        mov dx, offset error_wrong_format
        call PRINT
        jmp exit


run_prog:
        mov dx, offset endl
        call PRINT

        mov ax,4d00h
        int 21h

        cmp ah, 0
        je reason_0_label
```

```
        cmp ah, 1
        je reason_1_label
        cmp ah, 2
        je reason_2_label
        cmp ah, 3
        je reason_3_label


reason_0_label:
        mov dx, offset reason_0
        call PRINT
        jmp print_exit_code

reason_1_label:
        mov dx, offset reason_1
        call PRINT
        jmp exit

reason_2_label:
        mov dx, offset reason_2
        call PRINT
        jmp exit

reason_3_label:
        mov dx, offset reason_3
        call PRINT
        jmp exit


print_exit_code:
        mov dx, offset exit_code
        call PRINT

        call BYTE_TO_HEX
```

```
        push ax
        mov ah,02h
        mov dl,al
        int 21h
        pop ax
        xchg ah,al
        mov ah,02h
        mov dl,al
        int 21h

exit:
        xor al,al
        mov ah,4ch
        int 21h
MAIN endp
_end:
CODE ends
end main
```