

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. 8383

Колмыков В.Д.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построение загрузочного модуля динамической структуры. Исследование интерфейса между вызывающим и вызываемым модулями по управлению и по данным.

Процедуры, используемые в работе.

Название процедуры	Описание
WRITE	Вывод строки из DX
FREE_MEMORY	Освобождение лишней памяти
SET_COMMAND_LINE	Запись местоположения вызываемого модуля
LOAD_PROGRAMM	Загрузка вызываемого модуля

Ход 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.
- 2) Вызываемый модуль запускается с использованием загрузчика.
- 3) После запуска проверяется выполнение загрузчика, а затем результат выполнения вызываемой программы.

В качестве вызываемого модуля был взят модифицированный модуль из ЛР2. Код написанного модуля приведен в приложении А.

Отлаженная программа была запущена, она вызвала другую программу и остановилась, ожидая введения символа. Был введен символ «v». Результат работы программы представлен на рис. 1.

```
C:\>lr6

Memory was freed successfully
Locked memory address is 9FFF
Enviroment address is 0863
Command line tail:
Enviroment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path is C:\LR2.COM
v
Button: v
The program ended with normal end
C:\>
```

Рисунок 1 – Результат работы программы при вводе символа «v»

Программ была рапущена еще раз, была введена комбинация Ctrl-C. Результат работы приведен на рис. 2.

```
C:\>lr6

Memory was freed successfully
Locked memory address is 9FFF
Enviroment address is 0863
Command line tail:
Enviroment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path is C:\LR2.COM
♥
Button: ♥
The program ended with normal end
C:\>_
```

Рисунок 2 – Результат работы программы при вводе Ctrl-C

Программа была дважды запущена из другого каталога. Ввод символов остался прежним. Результаты представлены на рис. 3-4.

```

Z:\>C:

C:\>FORDOS\lr6

Memory was freed successfully
Locked memory address is 9FFF
Environment address is 0863
Command line tail:
Environment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path is C:\FORDOS\LR2.COM
v
Button: v
The program ended with normal end
C:\>

```

Рисунок 3 – Результат работы программы при запуске из другого каталога
и вводе «v»

```

C:\>FORDOS\lr6

Memory was freed successfully
Locked memory address is 9FFF
Environment address is 0863
Command line tail:
Environment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path is C:\FORDOS\LR2.COM
^
Button: ^
The program ended with normal end
C:\>

```

Рисунок 4 – Результат работы программы при запуске из другого каталога
и вводе Ctrl-C

Программа была запущена при условии, что модули хранятся в разных директориях. Результат на рис. 5.

```
C:\>FORDOS\lr6  
Memory was freed successfully  
Program wasn't load  
File wasn't found  
C:\>
```

Рисунок 5 – Результат работы программы, модули хранятся в разных дерикториях

Ответы на контрольные вопросы.

1) Как реализовано прерывание Ctrl-C?

Часть функций DOS проверяют перед своим выполнением наличие в кольцевом буфере клавиатуры кода 03 (Ctrl+C) и при обнаружении выполняют команду int 23h. В векторе 23h обычно находится адрес программы DOS, завершающий текущий процесс. Завершается обработчик функцией 4Ch прерывания 21h.

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

В точке вызова функции 4Ch прерывания 21h.

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

В точке выполнения функции 01h (ввод символа с клавиатуры) прерывания 21h.

Выводы.

В ходе выполнения лабораторной работы была исследована возможность построения загрузочного модуля динамической структуры.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

```

DATA SEGMENT
    PSP_SEGMENT dw 0
    FLAG_IS_FREE_SUCCESS db 0
    PARAMETR_BLOCK dw 0 ;сегментный адрес среды
                        dd 0 ;сегмент и смещение командной строки
                        dd 0 ;сегмент и смещение первого FCB
                        dd 0 ;сегмент и смещение второго FCB
    NAME_OF_PROGRAMM db "LR2.COM", 0
    COMMAND_LINE db 1h, 0Dh
    STR_FILE_PATH db 50h dup(0)
    STR_SUCCES_OF_FREED db 13, 10, "Memory was freed successfully$"
    STR_ERROR_OF_FREED db 13, 10, "Memory wasn't freed$"
    STR_FREE_ERROR7 db 13, 10, "7: Memory block descriptor is
destroyed$"
    STR_FREE_ERROR8 db 13, 10, "8: Not enough memory for function$"
    STR_FREE_ERROR9 db 13, 10, "9: Invalid adress$"
    STR_ENDL db 13, 10, "$"
    STR_SUCCES_OF_LOAD db 13, 10, "Program was load successfully$"
    STR_ERROR_OF_LOAD db 13, 10, "Program wasn't load$"
    STR_LOAD_ERROR1 db 13, 10, "Incorrect function number$"
    STR_LOAD_ERROR2 db 13, 10, "File wasn't found$"
    STR_LOAD_ERROR5 db 13, 10, "Disc error$"
    STR_LOAD_ERROR8 db 13, 10, "Not enough memory$"
    STR_LOAD_ERROR10 db 13, 10, "Invalid environment$"
    STR_LOAD_ERROR11 db 13, 10, "Incorrect format$"
    STR_PROGRAM_END db 13, 10, "The program ended with $"
    STR_END_CODE0 db "normal end$"
    STR_END_CODE1 db "ctrl-break end$"
    STR_END_CODE2 db "device error$"
    STR_END_CODE3 db "31h end$"
    STR_BUTTON db 13, 10, "Button: $"
    COMMANDLINE_POS dw 0
DATA ENDS

STACKK SEGMENT STACK
    dw 100h dup(0)
STACKK ENDS

CODE SEGMENT
    SAVE_SS dw 0
    SAVE_SP dw 0
    ASSUME CS:CODE, DS:DATA, SS:STACKK
;
    WRITE PROC
        push AX
        mov AH, 9h
        int 21h
        pop AX
        ret
    WRITE ENDP
;
    FREE_MEMORY PROC NEAR
        push AX
        push BX
        push CX
        push DX

```

```

        mov BX, MEMORY_FREE      ;Попытка освобождения
        sub BX, PSP_SEGMENT
        mov CL, 4
        shl BX, CL
        mov AX, 4A00h
        int 21h

        jnc FREE_MEMORY_SUCCES   ;Проверка освобождения
        mov DX, offset STR_ERROR_OF_FREED
        call WRITE
        mov FLAG_IS_FREE_SUCCESS, 0
        cmp AX, 7
            je FREE_MEMORY_ERROR_7
        cmp AX, 8
            je FREE_MEMORY_ERROR_8
        cmp AX, 9
            je FREE_MEMORY_ERROR_9
FREE_MEMORY_ERROR_7:
        mov DX, offset STR_FREE_ERROR7
        call WRITE
        jmp FREE_MEMORY_RETURN
FREE_MEMORY_ERROR_8:
        mov DX, offset STR_FREE_ERROR8
        call WRITE
        jmp FREE_MEMORY_RETURN
FREE_MEMORY_ERROR_9:
        mov DX, offset STR_FREE_ERROR9
        call WRITE
        jmp FREE_MEMORY_RETURN
FREE_MEMORY_SUCCES:
        mov DX, offset STR_SUCCES_OF_FREED
        call WRITE
        mov FLAG_IS_FREE_SUCCESS, 1

FREE_MEMORY_RETURN:
        pop DX
        pop CX
        pop BX
        pop AX
        ret
FREE_MEMORY ENDP

```

;

```

SET_COMMAND_LINE PROC NEAR
    push AX
    push DI
    push SI
    push ES

    mov AX, PSP_SEGMENT
    mov ES, AX
    mov ES, ES:[2Ch]
    mov SI, 0
SCL_FIND0:
    mov AX, ES:[SI]
    inc SI
    cmp AX, 0
    jne SCL_FIND0
    add SI, 3
    mov DI, 0
SCL_WRITE:
    mov AL, ES:[SI]
    cmp AL, 0

```

```

        je SCL_WRITE_NAME
        cmp AL, '\'
        jne SCL_ADD_SYMB
        mov COMMANDLINE_POS, DI
SCL_ADD_SYMB:
        mov BYTE PTR [STR_FILE_PATH + DI], AL
        inc SI
        inc DI
        jmp SCL_WRITE
SCL_WRITE_NAME:
        cld
        mov DI, COMMANDLINE_POS
        inc DI
        add DI, offset STR_FILE_PATH
        mov SI, offset NAME_OF_PROGRAMM
        mov AX, DS
        mov ES, AX
SCL_REWRITE_NAME_SYMB:
        lodsb
        stosb
        cmp AL, 0
        jne SCL_REWRITE_NAME_SYMB

        pop ES
        pop SI
        pop DI
        pop AX
        ret
SET_COMMAND_LINE ENDP

```

;

```

LOAD_PROGRAMM PROC NEAR
        push AX
        push BX
        push DX
        push DS
        push ES
        mov SAVE_SP, SP
        mov SAVE_SS, SS

        mov AX, DATA
        mov ES, AX
        mov BX, offset PARAMETR_BLOCK
        mov DX, offset COMMAND_LINE
        mov [BX + 2], DX
        mov [BX + 4], DS
        mov DX, offset STR_FILE_PATH
        mov AX, 4B00h      ;Вызывается загрузчик
        int 21h
        mov SS, CS:SAVE_SS
        mov SP, CS:SAVE_SP
        pop ES
        pop DS

        jnc LOAD_PROGRAMM_SUCCESS      ;Проверка на выполнение
        mov DX, offset STR_ERROR_OF_LOAD
        call WRITE
        cmp AX, 1
        je LOAD_PROGRAMM_ERROR_1
        cmp AX, 2
        je LOAD_PROGRAMM_ERROR_2
        cmp AX, 5
        je LOAD_PROGRAMM_ERROR_5
        cmp AX, 8

```



```

        je LOAD_PROGRAMM_ERROR_8
        cmp AX, 10
        je LOAD_PROGRAMM_ERROR_10
        cmp AX, 11
        je LOAD_PROGRAMM_ERROR_11
LOAD_PROGRAMM_ERROR_1:
        mov DX, offset STR_LOAD_ERROR1
        call WRITE
        jmp LOAD_PROGRAMM_END
LOAD_PROGRAMM_ERROR_2:
        mov DX, offset STR_LOAD_ERROR2
        call WRITE
        jmp LOAD_PROGRAMM_END
LOAD_PROGRAMM_ERROR_5:
        mov DX, offset STR_LOAD_ERROR5
        call WRITE
        jmp LOAD_PROGRAMM_END
LOAD_PROGRAMM_ERROR_8:
        mov DX, offset STR_LOAD_ERROR8
        call WRITE
        jmp LOAD_PROGRAMM_END
LOAD_PROGRAMM_ERROR_10:
        mov DX, offset STR_LOAD_ERROR10
        call WRITE
        jmp LOAD_PROGRAMM_END
LOAD_PROGRAMM_ERROR_11:
        mov DX, offset STR_LOAD_ERROR11
        call WRITE
        jmp LOAD_PROGRAMM_END
LOAD_PROGRAMM_SUCCESS:
        mov AX, 4D00h
        int 21h
        mov DI, offset STR_BUTTON
        mov [DI + 10], AL
        mov DX, offset STR_BUTTON
        call WRITE
        mov DX, offset STR_PROGRAM_END
        call WRITE
        cmp AH, 0
        je LOAD_PROGRAMM_END0
        cmp AH, 1
        je LOAD_PROGRAMM_END1
        cmp AH, 2
        je LOAD_PROGRAMM_END2
        cmp AH, 3
        je LOAD_PROGRAMM_END3
LOAD_PROGRAMM_END0:
        mov DX, offset STR_END_CODE0
        call WRITE
        jmp LOAD_PROGRAMM_END
LOAD_PROGRAMM_END1:
        mov DX, offset STR_END_CODE1
        call WRITE
        jmp LOAD_PROGRAMM_END
LOAD_PROGRAMM_END2:
        mov DX, offset STR_END_CODE2
        call WRITE
        jmp LOAD_PROGRAMM_END
LOAD_PROGRAMM_END3:
        mov DX, offset STR_END_CODE3
        call WRITE

LOAD_PROGRAMM_END:

```

```

        pop DX
        pop BX
        pop AX
        ret
LOAD_PROGRAMM ENDP
;
MAIN:
    mov BX, DS
    mov AX, DATA
    mov DS, AX
    mov PSP_SEGMENT, BX
    call FREE_MEMORY
    cmp FLAG_IS_FREE_SUCCESS, 1
    jne MAIN_END
    call SET_COMMAND_LINE
    call LOAD_PROGRAMM

MAIN_END:
    mov AX, 4C00h
    int 21h
CODE ENDS
MEMORY_FREE SEGMENT
MEMORY_FREE ENDS
END MAIN

```