

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

Мололкин К.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

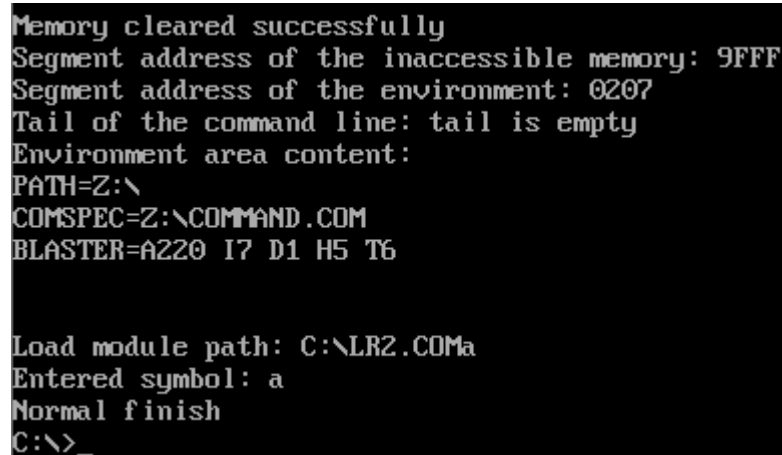
Цель работы

Исследовать возможность построения загрузочного модуля динамической структуры.

Ход работы

Первым шагом был написан и отлажен программный модуль типа .EXE, который подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором находится сам, передавая новую среду и командную строку. Вызываемый модуль запускается с использованием загрузчика. После запуска проверяется выполнение загрузчика, а затем результат выполнения вызываемой программы.

Следующим шагом была запущена отлаженная программа, когда текущим каталогом является каталог с разработанным модулем. Пример выполнения представлен на рис. 1. Во время выполнения программы был введен символ 'a'



```
Memory cleared successfully
Segment address of the inaccessible memory: 9FFF
Segment address of the environment: 0207
Tail of the command line: tail is empty
Environment area content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Load module path: C:\LR2.COMa
Entered symbol: a
Normal finish
C:\>_
```

Рисунок 1 – пример работы программы

Затем программы была запущена и введена комбинация Ctrl-C. Результат представлен на рис. 2.

```

C:\>LR6.EXE
Memory cleared successfully
Segment address of the inaccessible memory: 9FFF
Segment address of the environment: 0207
Tail of the command line: tail is empty
Environment area content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Load module path: C:\LR2.COM
Entered symbol: 
Normal finish

```

Рисунок 2 – результат выполнения с Ctrl-C

Следующим шагом данная программа была запущена когда текущим каталогом является какой-либо другой каталог, отличный от того в котором содержатся разработанные программные модули. Результат запуска с символом 'а' представлен на рис. 3, а с комбинацией Ctrl-C, на рис.4.

```

C:\>test\LR6.EXE
Memory cleared successfully
Segment address of the inaccessible memory: 9FFF
Segment address of the environment: 0207
Tail of the command line: tail is empty
Environment area content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Load module path: C:\TEST\LR2.COMa
Entered symbol: a
Normal finish

```

Рисунок 3 – запуск из другого каталога с символом 'а'

```

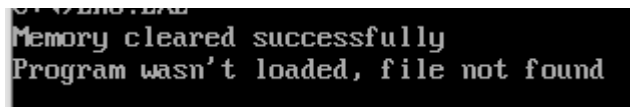
C:\>test\LR6.EXE
Memory cleared successfully
Segment address of the inaccessible memory: 9FFF
Segment address of the environment: 0207
Tail of the command line: tail is empty
Environment area content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Load module path: C:\TEST\LR2.COM
Entered symbol: 
Normal finish

```

Рисунок 4 – запуск из другого каталога с комбинацией Ctrl-C

На завершающем шаге. Программа была запущена, когда модули находятся в разных каталогах. Результат на рис. 5. Код программы LR6.ASM представлен в приложении А.



```
Memory cleared successfully
Program wasn't loaded, file not found
```

Рисунок 5 – работа программы с модулями в разных каталогах.

Контрольные вопросы

1. Как реализовано прерывание Ctrl-C?

В DOS функции отслеживают ввод комбинации символов Ctrl-C, и после ввода данной комбинации вызывают прерывание int 23h, которое завершает работу программы.

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

Текущая программа заканчивается в точке вызова функции 4Ch, прерывания int 21h.

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

Вызываемая программа заканчивается в точке выполнения функции 01h прерывания int 21h.

Вывод

Во время выполнения лабораторной работы была исследована возможность построения загрузочного модуля динамической структуры

Приложение А

```
ASTACK SEGMENT STACK
    dw 128 DUP(0)
ASTACK ENDS

DATA SEGMENT
    PARAMS dw 0
            dd 0
            dd 0
            dd 0
    ENTERED_SYMB db 13, 10, "Entered symbol:    $"
    IS_SUCCESS_CLEAR db 0
    SUCCESS_CLEAR_MEM db "Memory cleared successfully", 13,
10, "$"
    NOT_SUCCESS_CLEAR_MEM db "Memory wasn't cleared ", 13,
10, "$"
    CLEAR_ERROR_7 db "Memory clear error, control block
destroyed", 13, 10, "$"
    CLEAR_ERROR_8 db "Memory clear error, not enough memory
to execute func", 13, 10, "$"
    CLEAR_ERROR_9 db "Memory clear error, incorrect memory
block address", 13, 10, "$"
    CALLED_PROG db "LR2.COM$"
    CALLED_PROG_PATH db 50 dup (0)
    KEEP_SS dw 0
    KEEP_SP dw 0
    LOAD_ERROR_1 db "Program wasn't loaded, func number
incorrect", 13, 10, "$"
    LOAD_ERROR_2 db "Program wasn't loaded, file not found",
13, 10, "$"
    LOAD_ERROR_5 db "Program wasn't loaded, disk error", 13,
10, "$"
    LOAD_ERROR_8 db "Program wasn't loaded, insufficient
memory", 13, 10, "$"
    LOAD_ERROR_10 db "Program wasn't loaded, wrong
environment string", 13, 10, "$"
    LOAD_ERROR_11 db "Program wasn't loaded, wrong format",
13, 10, "$"
    PROG_FINISH_0 db 13, 10, "Normal finish$"
    PROG_FINISH_1 db 13, 10, "Finish by CTRL-Break$"
    PROG_FINISH_2 db 13, 10, "Finish by device error$"
    PROG_FINISH_3 db 13, 10, "Finish by function 31h$"
    DATA_ENDDDDD db 0
DATA ENDS

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

CLEAR_MEMORY PROC near
    push ax;
```

```

    push bx;
    push cx;
    push dx;

    mov bx, offset END_PART
    mov ax, offset DATA_ENDDDDD
    add bx, ax
    add bx, 30Fh
    mov cl, 4
    shr bx, cl
    mov ax, 4A00h
    int 21h
    jnc GOOD_END

    mov IS_SUCCESS_CLEAR, 0
    cmp ax, 7
    je C_ERROR_7
    cmp ax, 8
    je C_ERROR_8
    cmp ax, 9
    je C_ERROR_9

C_ERROR_7:
    mov dx, offset CLEAR_ERROR_7
    call PRINT_STRING
    jmp END_CLEAR
C_ERROR_8:
    mov dx, offset CLEAR_ERROR_8
    call PRINT_STRING
    jmp END_CLEAR
C_ERROR_9:
    mov dx, offset CLEAR_ERROR_9
    call PRINT_STRING
    jmp END_CLEAR

GOOD_END:
    mov dx, offset SUCCESS_CLEAR_MEM
    call PRINT_STRING
    mov IS_SUCCESS_CLEAR, 1

END_CLEAR:
    pop dx
    pop cx
    pop bx
    pop ax
    ret
CLEAR_MEMORY ENDP

LOAD_PROG PROC near
    push ax
    push bx
    push dx
    push ds

```

```

push es
mov KEEP_SP, sp
mov KEEP_SS, ss
mov ax, DATA
mov es, ax

mov bx, offset PARAMS
mov dx, offset NEW_COMMAND_LINE
mov [bx + 2], dx
mov [bx + 4], ds
mov dx, offset CALLED_PROG_PATH
mov ax, 4B00h
int 21h
mov SS, KEEP_SS
mov SP, KEEP_SP
pop es
pop ds

jnc UPLOAD
cmp ax, 1
je ERROR_1
cmp ax, 2
je ERROR_2
cmp ax, 5
je ERROR_5
cmp ax, 8
je ERROR_88
cmp ax, 10
je ERROR_10
cmp ax, 11
je ERROR_11
ERROR_1:
mov dx, offset LOAD_ERROR_1
call PRINT_STRING
jmp END_LOAD_PROG
ERROR_2:
mov dx, offset LOAD_ERROR_2
call PRINT_STRING
jmp END_LOAD_PROG
ERROR_5:
mov dx, offset LOAD_ERROR_5
call PRINT_STRING
jmp END_LOAD_PROG
ERROR_88:
mov dx, offset LOAD_ERROR_8
call PRINT_STRING
jmp END_LOAD_PROG
ERROR_10:
mov dx, offset LOAD_ERROR_10
call PRINT_STRING
jmp END_LOAD_PROG
ERROR_11:
mov dx, offset LOAD_ERROR_11

```

```

        call PRINT_STRING
        jmp END_LOAD_PROG
UPLOAD:
        mov ax, 4D00h
        int 21h
        push si
        mov si, offset ENTERED_SYMB
        mov [si + 18], al
        pop si
        mov dx, offset ENTERED_SYMB
        call PRINT_STRING
        cmp ah, 0
        je FINISH_0
        cmp ah, 1
        je FINISH_1
        cmp ah, 2
        je FINISH_2
        cmp ah, 3
        je FINISH_3

FINISH_0:
        mov dx, offset PROG_FINISH_0
        call PRINT_STRING
        jmp END_LOAD_PROG
FINISH_1:
        mov dx, offset PROG_FINISH_1
        call PRINT_STRING
        jmp END_LOAD_PROG
FINISH_2:
        mov dx, offset PROG_FINISH_2
        call PRINT_STRING
        jmp END_LOAD_PROG
FINISH_3:
        mov dx, offset PROG_FINISH_3
        call PRINT_STRING

END_LOAD_PROG:
        pop dx
        pop bx
        pop ax
        ret
LOAD_PROG ENDP

NEW_COMMAND_LINE PROC near
        push ax
        push di
        push si
        push es
        mov es, es:[2Ch]
        xor di, di

NEXT_0:
        mov al, es:[di]

```



```

        cmp al, 0
        je FIRST_0
        inc di
        jmp NEXT_0
FIRST_0:
        inc di
        mov al, es:[di]
        cmp al, 0
        jne NEXT_0
        add di, 3h
        mov si, 0
PRINT_NUMBER:
        mov al, es:[di]
        cmp al, 0
        je WRITE_PATH
        mov CALLED_PROG_PATH[si], al
        inc di
        inc si
        jmp PRINT_NUMBER
WRITE_PATH:
        dec si
        cmp CALLED_PROG_PATH[si], '\\'
        je COMPLETE_PATH
        jmp WRITE_PATH

COMPLETE_PATH:
        mov di, -1

COMPLETE_NAME:
        inc si
        inc di
        mov al, CALLED_PROG[di]
        cmp al, "$"
        je END_NEW_COMMAND_LINE
        mov CALLED_PROG_PATH[si], al
        jmp COMPLETE_NAME

END_NEW_COMMAND_LINE:
        pop es
        pop si
        pop di
        pop ax
        ret
NEW_COMMAND_LINE ENDP

PRINT_STRING PROC near
        push AX
        mov ah, 09h
        int 21h
        pop AX
        ret
PRINT_STRING ENDP

```

```

MAIN PROC
    mov ax, DATA
    mov ds, ax
    mov bx, ds
    call CLEAR_MEMORY
    cmp IS_SUCCESS_CLEAR, 1
    jne END_MAIN
    call NEW_COMMAND_LINE
    call LOAD_PROG

END_MAIN:
    xor al, al
    mov ah, 4Ch
    int 21h
MAIN ENDP
END_PART:
CODE ENDS
END MAIN

```