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 _____ Шишкин И.В.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры.

Ход работы.

Был написан 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.

В качестве вызываемой программы была взята программа ЛР 2, которая распечатывает среду и командную строку (программа была немного подкорректирована).

Была запущена отлаженная программа, когда текущим каталогом являлся каталог с разработанными модулями. Программа вызывает программу ЛР 2, которая останавливается, ожидая символ с клавиатуры. Был введен символ “a”. Результат выполнения программы приведен на рис. 1.

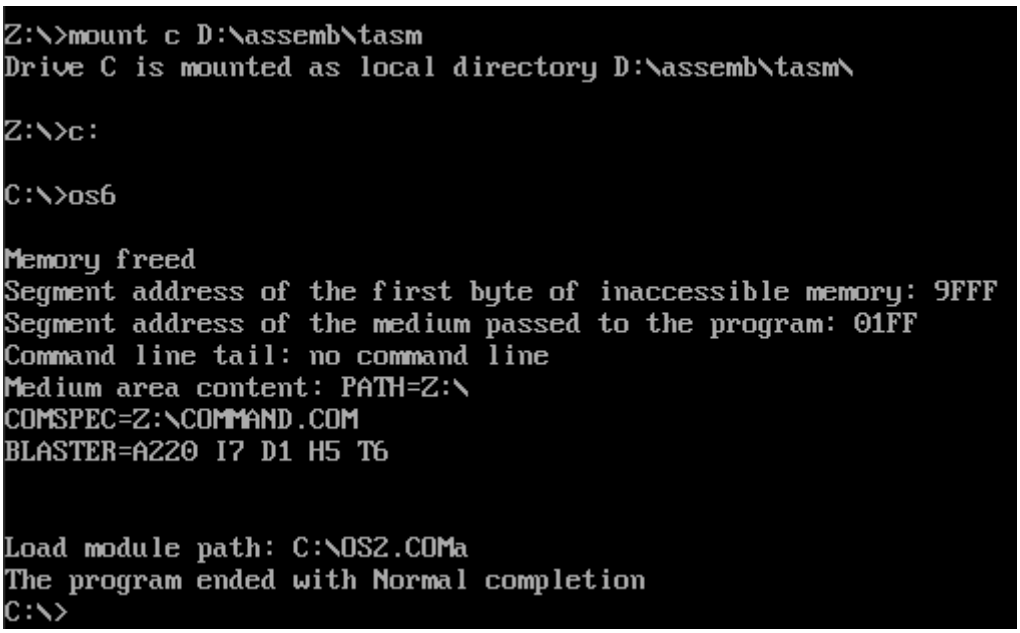

Рисунок 1 – Результат работы программы в 1-м случае

Затем, была запущена отлаженная программа, когда текущим каталогом являлся каталог с разработанными модулями. Программа вызывает программу ЛР 2, которая останавливается, ожидая символ с клавиатуры. Была введена комбинация символов Ctrl-C. Результат выполнения программы приведен на рис. 2.

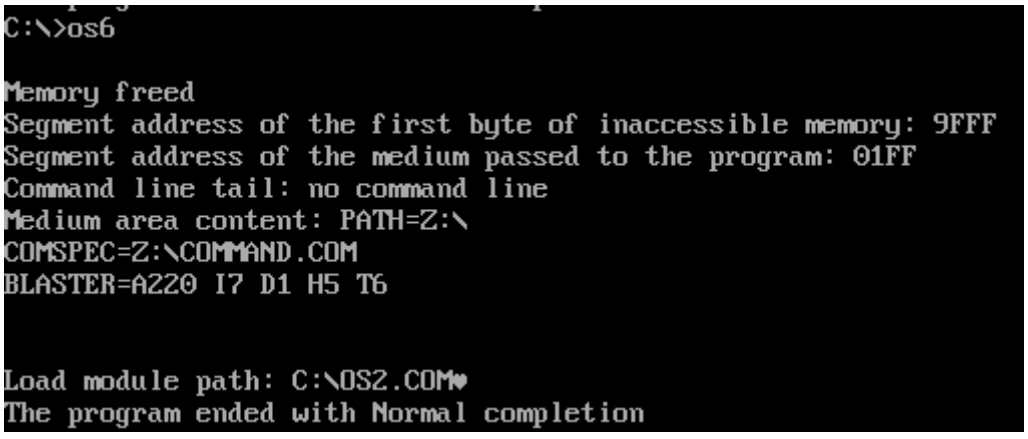

Рисунок 2 – Результат работы программы в 2-м случае

Была запущена отлаженная программа, когда текущим каталогом являлся каталог “OS_LABS\6\”. Результат выполнения программы приведен на рис. 3.

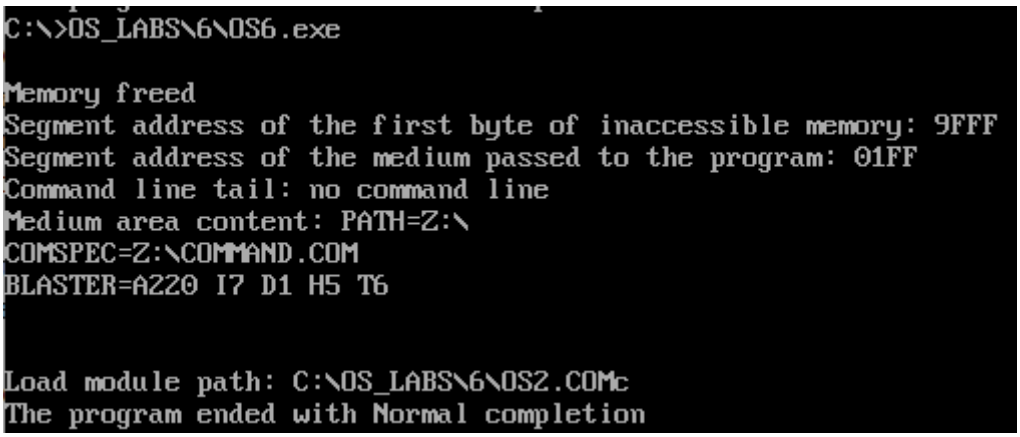

Рисунок 3 – Результат работы программы в 3-м случае

Была запущена отлаженная программа, когда модули находятся в разных каталогах. Результат выполнения программы приведен на рис. 4.

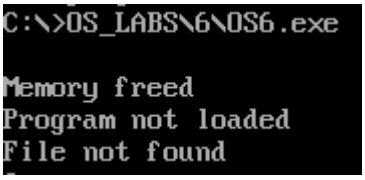

Рисунок 4 – Результат работы программы в 4-м случае

Контрольные вопросы.

1. Как реализовано прерывание Ctrl-C?

   Некоторые клавиатурные функции MS-DOS отслеживают комбинации клавиш Ctrl-C и Ctrl-Break. Если оператор ввел такую комбинацию клавиш, вызывается прерывание INT 23h, завершающее работу текущей программы.

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

   Код завершения формируется вызываемой программой в регистре AL перед выходом в OS с помощью функции 4Ch прерывания int 21h. То есть программа заканчивается в точке вызова функции 4Ch.

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

Тогда, когда производится ввод символа с клавиатуры. То есть программа заканчивается в точке выполнения функции 01h прерывания int 21h.

Выводы.

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

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ OS6.EXE

```
AStack SEGMENT STACK
	dw 128 dup(0)
Astack ENDS

DATA SEGMENT
	PARAMETER_BLOCK dw 0 ;сегментный адрес среды
		dd 0 ;сегмент и смещение командной строки
		dd 0 ;сегмент и смещение FCB
		dd 0 ;сегмент и смещение FCB

	KEEP_SS dw 0
	KEEP_SP dw 0

	IS_MEMORY_FREED db 0
	STR_FUNCTION_COMPLETED db 13, 10, 'Memory freed$'
	STR_FUNC_NOT_COMPLETED db 13, 10, 'Memory is not freed$'
	ERROR_CODE_7 db 13, 10, 'Memory control block destroyed$'
	ERROR_CODE_8 db 13, 10, 'Not enough memory to execute function$'
	ERROR_CODE_9 db 13, 10, 'Invalid memory block address$'

	PROGRAM_NAME db 'OS2.COM$'
	PROGRAM_PATH db 50 dup (0)
	STR_COMMAND_LINE db 1h, 0Dh

	STR_PROG_NOT_LOADED db 13, 10, 'Program not loaded$'
	STR_LOADING_ERROR_CODE_1 db 13, 10, 'Function number is incorrect$'
	STR_LOADING_ERROR_CODE_2 db 13, 10, 'File not found$'
	STR_LOADING_ERROR_CODE_5 db 13, 10, 'Disc error$'
	STR_LOADING_ERROR_CODE_8 db 13, 10, 'Insufficient memory$'
	STR_LOADING_ERROR_CODE_10 db 13, 10, 'Wrong environment string$'
	STR_LOADING_ERROR_CODE_11 db 13, 10, 'Invalid format$'

	STR_PROGRAM_END db 13, 10, 'The program ended with $'
	STR_END_CODE_0 db 'Normal completion$'
	STR_END_CODE_1 db 'Completion by CTRL-Break$'
	STR_END_CODE_2 db 'Device error termination$'
	STR_END_CODE_3 db 'Termination by function 31h leaving the program
resident$'

	END_OF_DATAA db 0
```

```
DATA ENDS

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

;-----------------------------------------------
PRINT PROC near
   push AX
   mov AH, 09h
   int 21h
   pop AX
   ret
PRINT ENDP
;-----------------------------------------------
LOADING_MODULE_LR2 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 PARAMETER_BLOCK
        mov DX, offset COMMAND_LINE
        mov [BX + 2], DX
        mov [BX + 4], DS
        mov DX, offset PROGRAM_PATH

        mov AX, 4B00h
        int 21h
        mov SS, KEEP_SS
        mov SP, KEEP_SP

        pop ES
        pop DS

        jnc PROGRAM_UPLOADED
        mov DX, offset STR_PROG_NOT_LOADED
        call PRINT
```

```
cmp AX, 1
je LOADING_ERROR_CODE_1
cmp AX, 2
je LOADING_ERROR_CODE_2
cmp AX, 5
je LOADING_ERROR_CODE_5
cmp AX, 8
je LOADING_ERROR_CODE_8
cmp AX, 10
je LOADING_ERROR_CODE_10
cmp AX, 11
je LOADING_ERROR_CODE_11
LOADING_ERROR_CODE_1:
        mov DX, offset STR_LOADING_ERROR_CODE_1
        call PRINT
        jmp END_OF_LOADING_LR2
LOADING_ERROR_CODE_2:
        mov DX, offset STR_LOADING_ERROR_CODE_2
        call PRINT
        jmp END_OF_LOADING_LR2
LOADING_ERROR_CODE_5:
        mov DX, offset STR_LOADING_ERROR_CODE_5
        call PRINT
        jmp END_OF_LOADING_LR2
LOADING_ERROR_CODE_8:
        mov DX, offset STR_LOADING_ERROR_CODE_8
        call PRINT
        jmp END_OF_LOADING_LR2
LOADING_ERROR_CODE_10:
        mov DX, offset STR_LOADING_ERROR_CODE_10
        call PRINT
        jmp END_OF_LOADING_LR2
LOADING_ERROR_CODE_11:
        mov DX, offset STR_LOADING_ERROR_CODE_11
        call PRINT
        jmp END_OF_LOADING_LR2
PROGRAM_UPLOADED:
        mov AX, 4D00h
        int 21h
        mov DX, offset STR_PROGRAM_END
        call PRINT
        cmp AH, 0
        je END_CODE_0
        cmp AH, 1
```

```
        je END_CODE_1
        cmp AH, 2
        je END_CODE_2
        cmp AH, 3
        je END_CODE_3
    END_CODE_0:
        mov DX, offset STR_END_CODE_0
        call PRINT
        jmp END_OF_LOADING_LR2
    END_CODE_1:
        mov DX, offset STR_END_CODE_1
        call PRINT
        jmp END_OF_LOADING_LR2
    END_CODE_2:
        mov DX, offset STR_END_CODE_2
        call PRINT
        jmp END_OF_LOADING_LR2
    END_CODE_3:
        mov DX, offset STR_END_CODE_3
        call PRINT

    END_OF_LOADING_LR2:
        pop DX
        pop BX
        pop AX
        ret
LOADING_MODULE_LR2 ENDP
;-----------------------------------------
COMMAND_LINE PROC near
        push AX
        push DI
        push SI
        push ES

        mov ES, ES:[2Ch] ;смещение до сегмента окружения (environment)
        xor DI, DI

    NEXT:    ;ищем 2 нуля - т.к. строка запуска программы за ними
        mov AL, ES:[DI]
        ;inc DI
        cmp AL, 0
        je AFTER_FIRST_0
        inc DI
        jmp NEXT
```

```
AFTER_FIRST_0:
        inc DI
        mov AL, ES:[DI]
        cmp AL, 0
        jne NEXT
        add DI, 3h    ;нашли 2 нуля, пропускаем 3 цифры
        mov SI, 0

WRITE_NUM:
        mov AL, ES:[DI]
        cmp AL, 0
        je DELETE_FILE_NAME
        ;mov BYTE PTR [PROGRAM_PATH + SI], AL
        mov PROGRAM_PATH[SI], AL
        inc DI
        inc SI
        jmp WRITE_NUM

DELETE_FILE_NAME:
        dec si
        cmp PROGRAM_PATH[SI], '\'
        je READY
        jmp DELETE_FILE_NAME

READY:
        mov DI, -1

ADD_FILE_NAME:
        inc SI
        inc DI
        ;mov AL, BYTE PTR [PROGRAM_NAME + DI]
        mov AL, PROGRAM_NAME[DI]
        cmp AL, '$'
        je END_OF_COMMAND_LINE
        ;mov BYTE PTR [PROGRAM_PATH + SI], AL
        mov PROGRAM_PATH[SI], AL
        jmp ADD_FILE_NAME

END_OF_COMMAND_LINE:
        pop ES
        pop SI
        pop DI
```

```
		pop AX
		ret
COMMAND_LINE ENDP
;-----------------------------------------
FREEING_UP_MEMORY PROC near
	push AX
	push BX
	push CX
	push DX

	mov BX, offset END_OF_PROGRAM
	mov AX, offset END_OF_DATAA
	add BX, AX
	add BX, 30Fh
	mov CL, 4
	shr BX, CL
	mov AX, 4A00h    ;сжать или расширить блок памяти
	int 21h
	jnc FUNCTION_COMPLETED

	mov DX, offset STR_FUNC_NOT_COMPLETED
	call PRINT
	mov IS_MEMORY_FREED, 0
	cmp AX, 7
	je IF_ERROR_CODE_7
	cmp AX, 8
	je IF_ERROR_CODE_8
	cmp AX, 9
	je IF_ERROR_CODE_9

	IF_ERROR_CODE_7:
		mov DX, offset ERROR_CODE_7
		call PRINT
		jmp END_OF_FREEING
	IF_ERROR_CODE_8:
		mov DX, offset ERROR_CODE_8
		call PRINT
		jmp END_OF_FREEING
	IF_ERROR_CODE_9:
		mov DX, offset ERROR_CODE_9
		call PRINT
		jmp END_OF_FREEING

	FUNCTION_COMPLETED:
```

```
                mov DX, offset STR_FUNCTION_COMPLETED
                call PRINT
                mov IS_MEMORY_FREED, 1

        END_OF_FREEING:
                pop DX
                pop CX
                pop BX
                pop AX
                ret
FREEING_UP_MEMORY ENDP
;-----------------------------------------
BEGIN PROC FAR
        xor AX, AX
        push AX
        mov AX, DATA
        mov DS, AX
        mov BX, DS

        call FREEING_UP_MEMORY
        cmp IS_MEMORY_FREED, 1
        jne ENDD
        call COMMAND_LINE
        call LOADING_MODULE_LR2

        ENDD:
                xor AL, AL
                mov AH, 4Ch
                int 21h
BEGIN ENDP
END_OF_PROGRAM:
CODE ENDS
        END BEGIN
```

ПРИЛОЖЕНИЕ Б

КОД ПРОГРАММЫ OS2.COM

```
TESTPC SEGMENT
ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
        INACCESSIBLE_MEMORY db 13, 10, "Segment address of the first byte of
inaccessible memory:     $"
        ADDRESS_TO_PROGRAMM db 13, 10, "Segment address of the medium
passed to the program:     $"
        TAIL db 13, 10, "Command line tail: $"
        IF_LEN_OF_TAIL_0 db "no command line$"
        MEDIUM_CONTENT db 13, 10, "Medium area content: $"
        LOAD_MODULE_PATH db 13, 10, "Load module path: $"
        SINGLE_SYMBOL db 13, 10, '$'

;-------------------------------------------------
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
; байт в AL переводится в два символа шестн. числа в AX
        push CX
        mov AH,AL
        call TETR_TO_HEX
        xchg AL,AH
        mov CL,4
        shr AL,CL
        call TETR_TO_HEX ; в AL старшая цифра
        pop CX ;в AH - младшая
        ret
BYTE_TO_HEX ENDP
;------------------------------
WRD_TO_HEX PROC near
;перевод в 16 с.с. 16-ти разрядного числа
; в AX - число, в DI - адрес последнего символа
```

```
        push BX
        mov BH,AH
        call BYTE_TO_HEX
        mov [DI],AH
        dec DI
        mov [DI],AL
        dec DI
        mov AL,BH
        call BYTE_TO_HEX
        mov [DI],AH
        dec DI
        mov [DI],AL
        pop BX
        ret
WRD_TO_HEX ENDP
;--------------------------------------------------
BYTE_TO_DEC PROC near
; перевод в 10 с.с., SI - адрес поля младшей цифры
        push CX
        push DX
        xor AH,AH
        xor DX,DX
        mov CX,10
loop_bd: div CX
        or DL,30h
        mov [SI],DL
        dec SI
        xor DX,DX
        cmp AX,10
        jae loop_bd
        cmp AL,00h
        je end_l
        or AL,30h
        mov [SI],AL
end_l: pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP
;-----------------------------------------------
PRINT PROC near
        push ax
        mov ah, 09h
        int 21h
        pop ax
```

```
        ret
PRINT ENDP
;-----------------------------------------
PRINT_TAIL PROC near
        push ax
        push cx
        push dx
        push bx

        mov bl, es:[0080h]
        mov dx, offset TAIL
        call PRINT
        xor cx, cx
        mov cl, bl
        cmp cl, 0
        jne if_len_not0
        mov dx, offset IF_LEN_OF_TAIL_0
        call PRINT
        jmp tail_end

        if_len_not0:
                xor si, si
                xor ax, ax

        cycle:
                mov al, es:[0081h+si]
                call PRINT_BYTE
                inc si
                loop cycle

        tail_end:
                pop bx
                pop dx
                pop cx
                pop ax
        ret

PRINT_TAIL ENDP
;-----------------------------------------
PRINT_BYTE PROC near
        push ax
        push dx

        xor dx, dx
```

```
        mov dl, al
        mov ah, 02h
        int 21h

        pop dx
        pop ax
        ret
PRINT_BYTE ENDP
;-----------------------------------------
PRINT_MEDIUM_CONTENT PROC near
        push ax
        push bx
        push dx
        push si
        push es

        mov dx, offset MEDIUM_CONTENT
        call PRINT
        xor si, si
        mov bx, 2Ch
        mov es, [bx]

        reading_content:
                cmp BYTE PTR es:[si], 0h
                je next_line
                mov al, es:[si]
                call PRINT_BYTE
                jmp check

        next_line:
                mov dx, offset SINGLE_SYMBOL
                call PRINT

        check:
                inc si
                cmp WORD PTR es:[si], 0001h
                je end_of_med_content
                jmp reading_content

        end_of_med_content:
                pop es
                pop si
                pop dx
                pop bx
```

```
		pop ax
	ret
PRINT_MEDIUM_CONTENT ENDP
;-----------------------------------------
PRINT_LOAD_MODULE_PATH PROC near
	push ax
	push bx
	push dx
	push es
	push si

	xor si, si
	mov bx, 2Ch
	mov es, [bx]

	reading:
		inc si
		cmp WORD PTR es:[si], 0001h
		je path
		jmp reading
	path:
		mov dx, offset LOAD_MODULE_PATH
		call PRINT
		add si, 2
	cyclee:
		cmp BYTE PTR es:[si], 00h
		je ending
		mov al, es:[si]
		call PRINT_BYTE
		inc si
		jmp cyclee

	ending:
		pop si
		pop es
		pop dx
		pop bx
		pop ax
	ret
PRINT_LOAD_MODULE_PATH ENDP
;-----------------------------------------

BEGIN:
```

```
        mov bx, es:[0002h]
        mov al, bh
        mov di, offset INACCESSIBLE_MEMORY
        call BYTE_TO_HEX
        mov [di+60], ax
        mov al, bl
        mov di, offset INACCESSIBLE_MEMORY
        call BYTE_TO_HEX
        mov [di+62], ax
        mov dx, offset INACCESSIBLE_MEMORY
        call PRINT

        xor di, di
        xor dx, dx

        mov bx, es:[002Ch]
        mov al, bh
        mov di, offset ADDRESS_TO_PROGRAMM
        call BYTE_TO_HEX
        mov [di+55], ax
        mov al, bl
        mov di, offset ADDRESS_TO_PROGRAMM
        call BYTE_TO_HEX
        mov [di+57], ax
        mov dx, offset ADDRESS_TO_PROGRAMM
        call PRINT

        xor di, di
        xor dx, dx

        call PRINT_TAIL
        call PRINT_MEDIUM_CONTENT
        call PRINT_LOAD_MODULE_PATH

        xor AL, AL
        mov AH, 01h
    int 21h
        mov AH, 4Ch
        int 21h
TESTPC ENDS
        END START
```