**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 .ЕХЕ, 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.

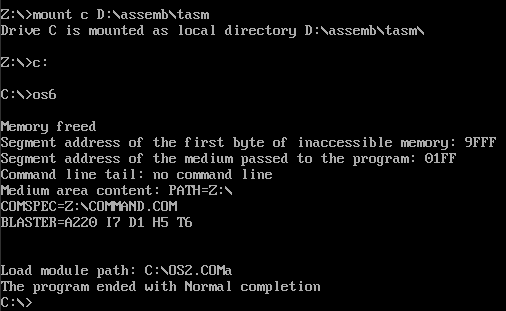


Рисунок 1 – Результат работы программы в 1-м случае

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.

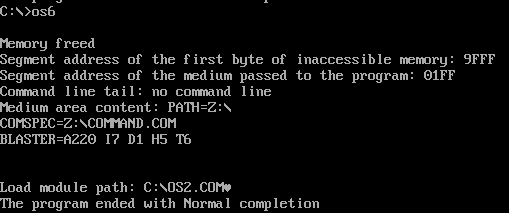


Рисунок 2 – Результат работы программы в 2-м случае

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

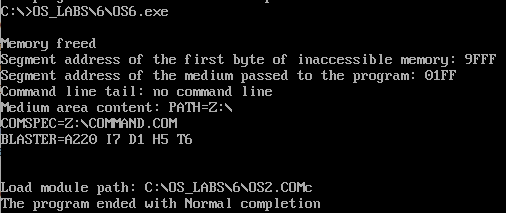


Рисунок 3 – Результат работы программы в 3-м случае

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

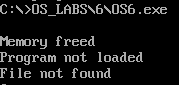


Рисунок 4 – Результат работы программы в 4-м случае

**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.

1. 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.

1. 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