

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

Бессуднов Г. И.

Преподаватель

Ефремов М. А.

Санкт-Петербург

2020

Цель работы

Исследование возможности построения загрузочного модуля динамической структуры. Исследование интерфейса между вызывающими и вызываемым модулями по управлению и по данным.

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.
- 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 модуль LR_2.COM и модифицирован в модуль LR_2_M.COM таким образом, что в конце выполнения он запрашивает пользователя ввести данные. Код написанного модуля приведен в приложении А, код модифицированного модуля приведен в приложении Б.

Результат запуска программы, когда в одном каталоге находятся оба модуля представлен на рис. 1.



```
C:\>LR_6.EXE
Free memory
Locked memory: F9FF
Environment: 10AF
Command line tail:
Environment content: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path: C:\LR_2_M.COM
Program ended
```

Рисунок 1 – Выполнение LR_6.EXE с вводом символа

Далее программа была запущена еще раз, но выход из нее был произведен по нажатию комбинации Ctrl + C. Результат выполнения представлен на рис. 2.

```

C:\>LR_6.EXE
Free memory
Locked memory: F9FF
Environment: 10AF
Command line tail:
Enviroment content: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path: C:\LR_2_M.COM
Program ended

C:\>_

```

Рисунок 2 – Выполнение LR_6.EXE с вводом Ctrl + C

Программы были помещены в новый каталог и запущены из него. Результат выполнения программы в новом каталоге представлен на рис. 3 и на рис. 4.

```

C:\>cd NEWFOL~1

C:\NEWFOL~1>LR_6.EXE
Free memory
Locked memory: F9FF
Environment: 10AF
Command line tail:
Enviroment content: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path: C:\NEWFOL~1\LR_2_M.COM
Program ended

C:\NEWFOL~1>

```

Рисунок 3 – Выполнение LR_6.EXE с вводом символа в новом каталоге

```

C:\>cd NEWFOL~1

C:\NEWFOL~1>LR_6.EXE
Free memory
Locked memory: F9FF
Environment: 10AF
Command line tail:
Enviroment content: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path: C:\NEWFOL~1\LR_2_M.COM
Program ended

C:\NEWFOL~1>

```

Рисунок 3 – Выполнение LR_6.EXE с вводом Ctrl + C в новом каталоге

Далее программа была запущена в каталоге без LR_2_M.COM. Результат выполнения представлен на рис. 5.



```
C:\NEWFOL~1>LR_6.EXE
Free memory
LOAD_MODULE ERROR 2: file not found
C:\NEWFOL~1>
```

Рисунок 5 – Выполнение LR_6.EXE без LR_2_M.COM

Контрольные вопросы

1. Как реализовано прерывание Ctrl + C?

Некоторые функции DOS перед выполнением проверяют наличие в буфере клавиатуры кода 03, который является кодом Ctrl + C. При обнаружении этого кода выполняется прерывание 23h. Обычно это прерывание завершает работу программы.

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

В точке вызова функции 4Ch прерывания int 21h.

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

В точке выполнения функции 01h прерывания int 21h, где программа требует пользователя совершить ввод с клавиатуры.

Выводы

В ходе выполнения лабораторной работы была исследована работа и организация загрузочных модулей динамической структуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД МОДУЛЯ LR_6.EXE

CODE SEGMENT

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

ASTACK SEGMENT STACK

DW 200 DUP(?)

ASTACK ENDS

DATA SEGMENT

FILE_NAME db "LR_2_M.COM", 0

PATH db 128 dup(0)

MES_ERR_7 db "ERROR 7: destriyed MCB",10,13,"\$"

MES_ERR_8 db "ERROR 8: low memory",10,13,"\$"

MES_ERR_9 db "ERROR 9: wrong address",10,13,"\$"

MES_GOOD db "Free memory",10,13,"\$"

MES_END_SUC db 10, 13,"Program ended",10, 13, "\$"

MES_END_CTRL C db "Program ended with CTRL+C command", 10, 13, "\$"

MES_END_DEV db "Program ended with device error",10,13,"\$"

MES_END_RES db "Program ended with 3lh",10,13,"\$"

MES_LOAD_ERR1 db "LOAD_MODULE ERROR 1: wrong number",10,13,"\$"

MES_LOAD_ERR2 db "LOAD_MODULE ERROR 2: file not found",10,13,"\$"

MES_LOAD_ERR5 db "LOAD_MODULE ERROR 5: disk error",10,13,"\$"

MES_LOAD_ERR8 db "LOAD_MODULE ERROR 8: low memory",10,13,"\$"

MES_LOAD_ERR10 db "LOAD_MODULE ERROR 10: wrong environment
string",10,13,"\$"

MES_LOAD_ERR11 db "LOAD_MODULE ERROR 11: wrong format",10,13,"\$"

COM_LINE db 1h, 0Dh

PAR_BLOCK dw 0

dd 0

dd 0

dd 0

KEEP_SS dw 0

KEEP_SP dw 0

KEEP_PSP dw 0

DATA_END db 0

DATA ENDS

WRITE_STR PROC NEAR

push AX

mov AH, 09h

int 21h

pop AX

ret

WRITE_STR ENDP

FREE_MEMORY PROC NEAR

```

        push BX
        push DX
        mov BX, offset PROGRAMM_END
        mov AX, offset DATA_END
        add BX, AX
        push CX
        mov CL, 4
        shr BX, CL
        add BX, 2Bh
        pop CX
        mov AH, 4Ah
        int 21h

        jnc MEM_GOOD

        cmp AX, 7
        je ERR_7
        cmp AX, 8
        je ERR_8
        cmp AX, 9
        je ERR_9

ERR_7:
        mov DX, offset MES_ERR_7
        jmp FAIL

ERR_8:
        mov DX, offset MES_ERR_8
        jmp FAIL

ERR_9:
        mov DX, offset MES_ERR_9
        jmp FAIL

MEM_GOOD:
        mov AX, 1
        mov DX, offset MES_GOOD
        call WRITE_STR
        jmp FREE_MEMORY_END

FAIL:
        mov AX, 0
        call WRITE_STR

FREE_MEMORY_END:
        pop DX
        pop BX
        ret
FREE_MEMORY ENDP

LOAD_MODULE PROC NEAR
        push AX
        push BX
        push CX
        push DX
        push DS
        push ES

        mov KEEP_SP, SP

```

```

        mov KEEP_SS, SS

        mov AX, DATA
        mov ES, AX
        mov BX, offset PAR_BLOCK
        mov DX, offset COM_LINE
        mov [BX+2], DX
        mov [BX+4], DS
        mov DX, offset PATH

        mov AX, 4B00h
        int 21h

        mov SS, KEEP_SS
        mov SP, KEEP_SP
        pop ES
        pop DS

        jnc LOAD_GOOD

        cmp AX, 1
        je LOAD_ERR_1
        cmp AX, 2
        je LOAD_ERR_2
        cmp AX, 5
        je LOAD_ERR_5
        cmp AX, 8
        je LOAD_ERR_8
        cmp AX, 10
        je LOAD_ERR_10
        cmp AX, 11
        je LOAD_ERR_11

LOAD_ERR_1:
        mov DX, offset MES_LOAD_ERR1
        jmp LOAD_FAIL
LOAD_ERR_2:
        mov DX, offset MES_LOAD_ERR2
        jmp LOAD_FAIL
LOAD_ERR_5:
        mov DX, offset MES_LOAD_ERR5
        jmp LOAD_FAIL
LOAD_ERR_8:
        mov DX, offset MES_LOAD_ERR8
        jmp LOAD_FAIL
LOAD_ERR_10:
        mov DX, offset MES_LOAD_ERR10
        jmp LOAD_FAIL
LOAD_ERR_11:
        mov DX, offset MES_LOAD_ERR11
        jmp LOAD_FAIL
LOAD_GOOD:
        mov AH, 4Dh
        mov AL, 00h
        int 21h
        cmp AH, 0
        je GOOD

```

```

        cmp AH, 1
        je CTRLC
        cmp AH, 2
        je DEV
        cmp AH, 3
        je RES

GOOD:
        mov DX, offset MES_END_SUC
        jmp LOAD_GOOD_END

CTRLC:
        mov DX, offset MES_END_CTRLC
        jmp LOAD_GOOD_END

DEV:
        mov DX, offset MES_END_DEV
        jmp LOAD_GOOD_END

RES:
        mov DX, offset MES_END_RES
        jmp LOAD_GOOD_END

LOAD_GOOD_END:
        call WRITE_STR
        jmp LOAD_END

LOAD_FAIL:
        call WRITE_STR

LOAD_END:
        pop DX
        pop CX
        pop BX
        pop AX
        ret

LOAD_MODULE ENDP

MAIN PROC
        PUSH DS
        SUB AX, AX
        PUSH AX
        MOV AX, DATA
        MOV DS, AX
        mov KEEP_PSP, ES

        call FREE_MEMORY

        cmp AX, 2
        je MAIN_END

        ;-----
        mov AX, KEEP_PSP
        mov ES, AX
        mov ES, ES:[2Ch]
        mov BX, 0

PRINT_ENV_VAR:
        cmp BYTE PTR ES:[BX], 0
        je ENV_VAR_END
        inc BX

```



```

        jmp PRINT_ENV_VAR
ENV_VAR_END:
        inc BX
        cmp BYTE PTR ES:[BX+1], 0
        jne PRINT_ENV_VAR

        add BX, 2
        mov DI, 0
MARK:
        mov DL, ES:[BX]
        mov BYTE PTR [PATH+DI], DL
        inc BX
        inc DI
        cmp DL, 0
        je LOOP_END
        cmp DL, '\'
        jne MARK
        mov CX, DI
        jmp MARK
LOOP_END:
        mov DI, CX
        mov SI, 0
FILE_NAME_LOOP:
        mov DL, BYTE PTR [FILE_NAME+SI]
        mov BYTE PTR [PATH+DI], DL
        inc DI
        inc SI
        cmp DL, 0
        jne FILE_NAME_LOOP
        ;-----

        call LOAD_MODULE

MAIN_END:
        xor AL, AL
        mov AH, 4Ch
        int 21h
MAIN ENDP
PROGRAMM_END:

CODE ENDS

END MAIN

```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД МОДУЛЯ LR_2_M.COM

```
TESTPC SEGMENT
ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
NEW_LINE db 13, 10, '$'
LOCK_MEM_MES db 'Locked memory: $'
ENV_MES db 'Environment: $'
TAIL_MES db 'Command line tail: $'
NO_TAIL_MES db 'empty as always $'
ENV_CON_MES db 'Enviroment content: $'
PATH_MES db 'Path: $'
```

```
;-----
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
```

```
;-----
PRINT_SYMBOL PROC near
    push AX
    mov AH, 02H
    int 21H
    pop AX
    ret
PRINT_SYMBOL ENDP
```

```
;-----
PRINT_MESSAGE PROC near
    push AX
    mov AH, 09H
    int 21H
    pop AX
    ret
PRINT_MESSAGE ENDP
;-----
```

```

PRINT_HEX PROC near
    push AX
    mov AL, AH
    call BYTE_TO_HEX
    mov DL, AH
    call PRINT_SYMBOL
    mov DL, AL
    call PRINT_SYMBOL
    pop AX
    call BYTE_TO_HEX
    mov DL, AH
    call PRINT_SYMBOL
    mov DL, AL
    call PRINT_SYMBOL
    ret
PRINT_HEX ENDP
;-----

BEGIN:
;-----LOCK_MEM-----
    mov DX, offset LOCK_MEM_MES
    call PRINT_MESSAGE
    mov AX, DS:[02H]
    call PRINT_HEX
    mov DX, offset NEW_LINE
    call PRINT_MESSAGE
;-----ENV-----
    mov DX, offset ENV_MES
    call PRINT_MESSAGE
    mov AX, DS:[2CH]
    call PRINT_HEX
    mov DX, offset NEW_LINE
    call PRINT_MESSAGE
;-----TAIL-----
    mov DX, offset TAIL_MES
    call PRINT_MESSAGE
    mov CX, 0
    mov CL, DS:[80H]
    cmp CL, 0
    je TAIL_EMPTY
    mov DI, 0
    xor DX, DX
TAIL_LOOP:
    mov DL, DS:[81H + DI]
    call PRINT_SYMBOL
    inc DI
    loop TAIL_LOOP
    jmp TAIL_END
TAIL_EMPTY:
    mov DX, OFFSET NO_TAIL_MES
    call PRINT_MESSAGE
TAIL_END:
    mov DX, offset NEW_LINE
    call PRINT_MESSAGE
;-----ENV_PATH_MES-----
    mov DX, offset ENV_CON_MES
    call PRINT_MESSAGE

```

```

        xor SI, SI
        mov BX, 2CH
        mov ES, [BX]
PRINT_CONTENT:
        cmp BYTE PTR ES:[SI], 0H
        je NEXT_CONTENT
        mov AL, ES:[SI]
        mov DL, AL
        call PRINT_SYMBOL
        jmp CHECK
NEXT_CONTENT:
        mov DX, offset NEW_LINE
        call PRINT_MESSAGE
CHECK:
        inc SI
        cmp WORD PTR ES:[SI], 0001H
        je PRINT_PATH
        jmp PRINT_CONTENT
PRINT_PATH:
        mov DX, offset PATH_MES
        call PRINT_MESSAGE
        add SI, 2
PATH_LOOP:
        cmp BYTE PTR ES:[SI], 00H
        je CONTENT_END
        mov AL, ES:[SI]
        mov DL, AL
        call PRINT_SYMBOL
        inc SI
        jmp PATH_LOOP
CONTENT_END:

        xor AL, AL

        mov AH, 01h
        int 21h

        mov AH, 4CH
        int 21H
TESTPC ENDS
END START

```