

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

Дейнега В.Е.

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 модифицированный модуль из ЛР2. Код написанного модуля приведен в приложении А.

Модуль LR2.COM был доработан таким образом, чтобы в конце его выполнения считывался символ с клавиатуры. Результат запуска отлаженной программы, когда текущим каталогом является каталог с модулем LR2.COM, представлен на рис. 1.



```
C:\>cd lr6

C:\LR6>lr6.exe
Succeed freeing up memory

Address of locked memory: 9FFF
Address of enviroment: 01FC
Tail comand_line:
Enviroment: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path: C:\LR6\LR2.COM
u
Program ended with code u

C:\LR6>
```

Рисунок 1 – Выполнение lr6.exe

Для того чтобы выйти из программы была нажата клавиша “u”. На рисунке видно, что программа выводит нажатую клавишу.

Далее программа была запущена в той же директории, но выход из нее произошел по нажатию ctrl-c. Результат выполнения представлен на рис. 2.

```
C:\LR6>lr6.exe
Succeed freeing up memory

Address of locked memory: 9FFF
Address of enviroment: 01FC
Tail comand_line:
Enviroment: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path: C:\LR6\LR2.COM
♥
Program ended with code ♥
C:\LR6>
```

Рисунок 2 – Выход из программы по ctrl-c

Программы были помещены в новый каталог. Результат выполнения программы в новом каталоге представлен на рис. 3.

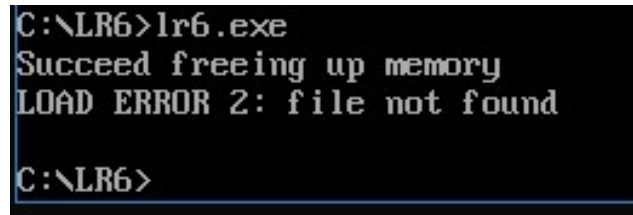
```
C:\LR6>cd ..
C:\>cd new
C:\NEW>lr6.exe
Succeed freeing up memory

Address of locked memory: 9FFF
Address of enviroment: 01FC
Tail comand_line:
Enviroment: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path: C:\NEW\LR2.COM
k
Program ended with code k
C:\NEW>_
```

Рисунок 3 – Выполнение lr6.exe в новом каталоге

Далее программа была запущена в другом каталоге, в котором нет модуля LR2.COM. Результат выполнения представлен на рис. 4.



```
C:\LR6>lr6.exe
Succeed freeing up memory
LOAD ERROR 2: file not found
C:\LR6>
```

Рисунок 4 – Выполнение lr6.exe без lr2.com

Контрольные вопросы.

1. Как реализовано прерывание Ctrl-C?

Прерывание 23h вызывается, если была нажата комбинация клавиш Ctrl-C или Ctrl-Break. Адрес, по которому передается управление (0000:008c). Управление передаётся тогда, когда DOS распознает, что пользователь нажал Ctrl-Break или Ctrl-C. Адрес по вектору INT 23h копируется в поле PSP Ctrl-Break Address функциями DOS 26h (создать PSP) и 4Ch (EXEC).

Исходное значение адреса обработчика Ctrl-Break восстанавливается из PSP при завершении программы. Таким образом, по завершении порожденного процесса будет восстановлен адрес обработчика Ctrl-Break из родительского процесса.

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 вызова функции 01h прерывания int 21h, где программа ожидала ввод с клавиатуры.

Выводы.

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

ПРИЛОЖЕНИЕ

КОД ПРОГРАММЫ

```

DATA SEGMENT
    FILENAME db "LR2.COM", 0
    FULL_PATH db 128 dup(0)
    STR_MEM_ERR7 db "ERROR 7 - MCB destroyed!",10,13,"$"
    STR_MEM_ERR8 db "ERROR 8 - not enough memory!",10,13,"$"
    STR_MEM_ERR9 db "ERROR 9 - wrong memory block address!",10,13,"$"
    STR_MEM_OK db "Succeed freeing up memory",10,13,"$"

    STR_SUCCSES db 10, 13,"Program ended with code ",10,
13, "$"
    STR_CTRLC db "Program ended by CTRL+break command", 10, 13,
"$"
    STR_DEVICE db "Program ended by device error",10,13,"$"
    STR_RES db "by int 31h (resident)",10,13,"$"

    STR_LOAD_ERROR1 db "LOAD ERROR 1: wrong function number",10,13,"$"
    STR_LOAD_ERROR2 db "LOAD ERROR 2: file not found",10,13,"$"
    STR_LOAD_ERROR5 db "LOAD ERROR 5: disk error",10,13,"$"
    STR_LOAD_ERROR8 db "LOAD ERROR 8: not enough memory",10,13,"$"
    STR_LOAD_ERROR10 db "LOAD ERROR 10: wrong env string",10,13,"$"
    STR_LOAD_ERROR11 db "LOAD ERROR 11: ",10,13,"$"

    COMMAND_LINE db 1h, 0Dh
    PARAMETERS_BLOCK dw 0 ; seg address of env
                        dd 0 ; seg and offset of command line
                        dd 0 ; seg and offset first FCB
                        dd 0 ; seg and offset second FCB

    KEEP_SS dw 0
    KEEP_SP dw 0
    KEEP_PSP dw 0

    END_OF_DATA db 0
DATA ENDS

AStack SEGMENT STACK
    DW 200 DUP(?)
AStack ENDS

CODE SEGMENT

    ASSUME CS:CODE, DS:DATA, ES:NOTHING, SS:AStack

    WRITE_STR PROC NEAR
        push DX
        push AX

        mov AH, 09h
        int 21h

        pop AX
        pop DX
        ret

```

```

WRITE_STR ENDP

FREE_MEMORY PROC NEAR
    push BX
    push DX
    mov BX, offset END_OF_PROG
    mov AX, offset END_OF_DATA
    add BX, AX
    push CX
    mov CL, 4
    shr BX, CL
    add BX, 2Bh
    pop CX
    mov AH, 4Ah
    int 21h

    jnc MEM_OK

    cmp AX, 7
    je MEM_ERR7
    cmp AX, 8
    je MEM_ERR8
    cmp AX, 9
    je MEM_ERR9

MEM_ERR7:
    mov DX, offset STR_MEM_ERR7
    jmp MEM_FAIL
MEM_ERR8:
    mov DX, offset STR_MEM_ERR8
    jmp MEM_FAIL
MEM_ERR9:
    mov DX, offset STR_MEM_ERR9
    jmp MEM_FAIL

MEM_OK:
    mov AX, 1
    mov DX, offset STR_MEM_OK
    call WRITE_STR
    jmp FREE_MEMORY_END

MEM_FAIL:
    mov AX, 0
    call WRITE_STR

FREE_MEMORY_END:
    pop DX
    pop BX
    ret
FREE_MEMORY ENDP

PATH_MAKE PROC NEAR
    push AX
    push CX
    push BX
    push DI
    push SI

```

```

push ES

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

print_env_variable:
    cmp BYTE PTR ES:[BX], 0
    je variable_end
    inc BX
    jmp print_env_variable
variable_end:
    inc BX
    cmp BYTE PTR ES:[BX+1], 0
    jne print_env_variable

add BX, 2
mov DI, 0

MARK:
    mov DL, ES:[BX]
    mov BYTE PTR [FULL_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
filename_loop:
    mov DL, BYTE PTR [FILENAME+SI]
    mov BYTE PTR [FULL_PATH+DI], DL
    inc DI
    inc SI
    cmp DL, 0
    jne filename_loop

pop ES
pop SI
pop DI
pop BX
pop CX
pop AX

ret
PATH_MAKE ENDP

LOAD 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 PARAMETERS_BLOCK
mov DX, offset COMMAND_LINE
mov [BX+2], DX
mov [BX+4], DS
mov DX, offset FULL_PATH

mov AX, 4B00h
int 21h

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

jnc LOAD_SUCCSES

cmp AX, 1
je LOAD_ERROR1
cmp AX, 2
je LOAD_ERROR2
cmp AX, 5
je LOAD_ERROR5
cmp AX, 8
je LOAD_ERROR8
cmp AX, 10
je LOAD_ERROR10
cmp AX, 11
je LOAD_ERROR11

LOAD_ERROR1:
    mov DX, offset STR_LOAD_ERROR1
    jmp LOAD_FAIL
LOAD_ERROR2:
    mov DX, offset STR_LOAD_ERROR2
    jmp LOAD_FAIL
LOAD_ERROR5:
    mov DX, offset STR_LOAD_ERROR5
    jmp LOAD_FAIL
LOAD_ERROR8:
    mov DX, offset STR_LOAD_ERROR8
    jmp LOAD_FAIL
LOAD_ERROR10:
    mov DX, offset STR_LOAD_ERROR10
    jmp LOAD_FAIL
LOAD_ERROR11:
    mov DX, offset STR_LOAD_ERROR11
    jmp LOAD_FAIL

```

```

LOAD_SUCCSES:
    mov AH, 4Dh
    mov AL, 00h
    int 21h

    cmp AH, 0
    je SUCCSES
    cmp AH, 1
    je CTRLC
    cmp AH, 2
    je DEVICE
    cmp AH, 3
    je RES

SUCCSES:
    push DI
    mov DI, offset STR_SUCCSES
    mov [DI+26], AL
    pop DI
    mov DX, offset STR_SUCCSES
    jmp LOAD_SUCCSES_END

CTRLC:
    mov DX, offset STR_CTRLC
    jmp LOAD_SUCCSES_END

DEVICE:
    mov DX, offset STR_DEVICE
    jmp LOAD_SUCCSES_END

RES:
    mov DX, offset STR_RES
    jmp LOAD_SUCCSES_END

LOAD_SUCCSES_END:
    call WRITE_STR

    jmp LOAD_END

LOAD_FAIL:
    call WRITE_STR
LOAD_END:
    pop DX
    pop CX
    pop BX
    pop AX
    ret
LOAD 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

call PATH_MAKE

call LOAD
MAIN_END:
    xor AL, AL
    mov AH, 4Ch
    int 21h
MAIN ENDP
END_OF_PROG:

CODE ENDS

END MAIN
```