

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ
по лабораторной работе №6
по дисциплине «Операционные системы»
Тема: Построение модуля динамической структуры

Студент гр. 8383

Костарев К.В.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры.

Выполнение работы.

Для выполнения лабораторной работы был написан и отлажен программный модуль типа .EXE, в котором:

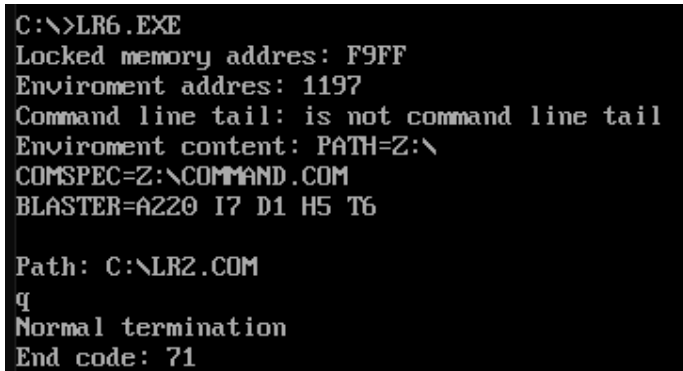
- 1) подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором он находится сам. Вызываемому модулю передаёт новую среду и новую командную строку;
- 2) Вызываемый модуль запускается с использованием загрузчика.

Исходный код программы представлен в приложении А.

После запуска проверяется выполнение загрузчика, а затем результат выполнения вызываемой программы. Проверяет причину завершения и, в зависимости от значения, выводит соответствующее сообщение. Если причина завершения 0, то выводится код завершения.

В качестве вызываемой программы используется реализованная программа из лабораторной работы № 2, которая печатает среду и командную строку. Данная программа была немного модифицирована, а точнее в конце ее работы перед выходом добавлено обращение к функции ввода символа с клавиатуры (01H прерывания 21H).

Для начала программа была запущена, когда оба модуля находятся в текущем каталоге. На рис. 1 представлен результат работы с вводом символа q.



```
C:\>LR6.EXE
Locked memory address: F9FF
Environment address: 1197
Command line tail: is not command line tail
Environment content: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path: C:\LR2.COM
q
Normal termination
End code: 71
```

Рисунок 1 – Случай, когда оба модуля в текущем каталоге, ввод символа q

При следующем запуске программы была введена комбинация “Ctrl+C”.
Результат работы показан на рис. 2.

```
C:\>LR6.EXE
Locked memory address: F9FF
Enviroment address: 1197
Command line tail: is not command line tail
Enviroment content: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path: C:\LR2.COM
♥
Normal termination
End code: 03
```

Рисунок 2 – Случай, когда оба модуля в текущем каталоге, ввод “Ctrl+C”

Далее программа была запущена, когда она находилась не в текущем каталоге. Результат работы программы с вводом символа @ представлен на рис. 3.

```
C:\DIR>LR6.EXE
Locked memory address: F9FF
Enviroment address: 1197
Command line tail: is not command line tail
Enviroment content: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path: C:\DIR\LR2.COM
@
Normal termination
End code: 40
```

Рисунок 3 – Случай, когда оба модуля не в текущем каталоге, ввод символа

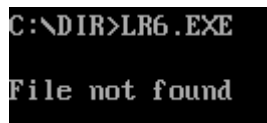
Аналогично программа была запущена с вводом комбинации клавиш “Ctrl+C”, результат работы можно видеть на рис. 4.

```
C:\DIR>LR6.EXE
Locked memory address: F9FF
Enviroment address: 1197
Command line tail: is not command line tail
Enviroment content: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path: C:\DIR\LR2.COM
♥
Normal termination
End code: 03
```

Рисунок 4 – Случай, когда оба модуля не в текущем каталоге, ввод “Ctrl+C”

Работа программы в том случае, когда модули находятся в разных каталогах, представлен на рис. 5.



```
C:\>DIR>LR6.EXE
File not found
```

Рисунок 5 – Случай, когда модули в разных каталогах

1) Как реализовано прерывание Ctrl-C?

Вызывается прерывание 23H, которое завершает текущий процесс и передает управление порождаемому процессу.

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

В точке вызова функции 4CH прерывания 21H.

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

В точке вызова функции 01H прерывания 21H.

Выводы.

В ходе выполнения данной лабораторной работы была изучена возможность построения загрузочного модуля динамической структуры и реализован интерфейс вызывающего и вызываемого модуля, где в качестве последнего была использована программа из лабораторной работы № 2.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ВЫЗЫВАЮЩЕГО МОДУЛЯ

```
AStack      SEGMENT      STACK
             db          512  dup(0)
AStack ENDS
```

```
DATA SEGMENT
    PARAMETER dw ?
    dd ?
    dd ?
    dd ?
    ERROR_MEM_7 db 13, 10, 'MCB destroyed',13,10,'$'
    ERROR_MEM_8 db 13, 10, 'Not enough memory',13,10,'$'
    ERROR_MEM_9 db 13, 10, 'Wrong address',13,10,'$'
    ERROR_LOAD_1 db 13, 10, 'Number of function is wrong',13,10,'$'
    ERROR_LOAD_2 db 13, 10, 'File not found',13,10,'$'
    ERROR_LOAD_5 db 13, 10, 'Disk error',13,10,'$'
    ERROR_LOAD_8 db 13, 10, 'Insufficient value of memory',13,10,'$'
    ERROR_LOAD_10 db 13, 10, 'Incorrect environment string',13,10,'$'
    ERROR_LOAD_11 db 13, 10, 'Wrong format',13,10,'$'
    NORMAL db 13, 10, 'Normal termination',13,10,'$'
    CTRL db 13, 10, 'Ended by Ctrl-Break',13,10,'$'
    DEVICE_ERROR db 13, 10, 'Ended with device error',13,10,'$'
    END_31H db 13, 10, 'Ended by 31h',13,10,'$'
    PATH db '
',13,10,'$',0
    KEEP_SS dw 0
    KEEP_SP dw 0
    END_CODE db 'End code: ',13,10,'$'
DATA ENDS
DUM_SEGMENT SEGMENT
DUM_SEGMENT ENDS
```

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

```
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

WRITE_STRING PROC near
        push AX
        mov AH, 09H
        int 21H
        pop AX
        ret
WRITE_STRING ENDP

FREE PROC
        mov BX,offset DUM_SEGMENT
        mov AX, ES
        sub BX, AX
        mov CL, 4H
        shr BX, CL
        mov AH, 4AH
        int 21H
        jnc GOOD
        cmp AX, 7
        mov DX, offset ERROR_MEM_7
        je HAVE_ERROR
        cmp AX, 8
        mov DX, offset ERROR_MEM_8
        je HAVE_ERROR
        cmp AX, 9
        mov DX, offset ERROR_MEM_9
HAVE_ERROR:
        call WRITE_STRING
        xor AL,AL
        mov AH,4CH
        int 21H
GOOD:
        mov AX, ES
        mov PARAMETER, 0
        mov PARAMETER+2, AX
        mov PARAMETER+4, 80H
        mov PARAMETER+6, AX
        mov PARAMETER+8, 5CH
        mov PARAMETER+10, AX
        mov PARAMETER+12, 6CH
        ret
FREE ENDP

```

```

RUN_P PROC      NEAR
    mov ES, ES:[2Ch]
    mov SI, 0
ENV:
    mov DL, ES:[SI]
    cmp DL, 00H
    je EOL
    inc SI
    jmp ENV
EOL:
    inc SI
    mov DL, ES:[SI]
    cmp DL, 00H
    jne ENV
    add SI, 03H
    push DI
    lea DI, PATH
G_PATH:
    mov DL, ES:[SI]
    cmp DL, 00H
    je EOL_2
    mov [DI], DL
    inc DI
    inc SI
    jmp G_PATH
EOL_2:
    sub DI, 7
    mov [DI], byte ptr 'L'
    mov [DI+1], byte ptr 'R'
    mov [DI+2], byte ptr '2'
    mov [DI+3], byte ptr '.'
    mov [DI+4], byte ptr 'C'
    mov [DI+5], byte ptr 'O'
    mov [DI+6], byte ptr 'M'
    mov [DI+7], byte ptr 0H
    pop DI
    mov KEEP_SP, SP
    mov KEEP_SS, SS
    push DS
    pop ES
    mov BX, offset PARAMETER
    mov DX, offset PATH
    mov     AX, 4B00H
    int 21H
    jnc IS_LOADED
    push AX
    mov AX, DATA
    mov DS, AX
    pop AX

```

```

    mov SS, KEEP_SS
    mov SP, KEEP_SP
    cmp AX, 1
    mov DX, offset ERROR_LOAD_1
    je END_ERROR
    cmp AX, 2
    mov DX, offset ERROR_LOAD_2
    je END_ERROR
    cmp AX, 5
    mov DX, offset ERROR_LOAD_5
    je END_ERROR
    cmp AX, 8
    mov DX, offset ERROR_LOAD_8
    je END_ERROR
    cmp AX, 10
    mov DX, offset ERROR_LOAD_10
    je END_ERROR
    cmp AX, 11
    mov DX, offset ERROR_LOAD_11
END_ERROR:
    call WRITE_STRING
    xor AL, AL
    mov AH, 4CH
    int 21H
IS_LOADED:
    mov AX, 4D00H
    int 21H
    cmp AH, 0
    mov DX, offset NORMAL
    je END_RUN
    cmp AH, 1
    mov DX, offset CTRL
    je END_RUN
    cmp AH, 2
    mov DX, offset DEVICE_ERROR
    je END_RUN
    cmp AH, 3
    mov DX, offset END_31H
END_RUN:
    call WRITE_STRING
    mov DI, offset END_CODE
    call BYTE_TO_HEX
    add DI, 0AH
    mov [DI], AL
    add DI, 1
    xchg AH, AL
    mov [DI], AL
    mov DX, offset END_CODE
    call WRITE_STRING
    ret

```



```
RUN_P ENDP

MAIN PROC NEAR
    mov AX, DATA
    mov DS, AX
    call FREE
    call RUN_P
    mov AX, 4C00H
    int 21H
    ret
LAST_BYTE:
MAIN ENDP
CODE ENDS
    END MAIN
```