

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. 8382

Мирончик П.Д.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры. В отличие от предыдущих лабораторных работ, в этой работе рассматривается приложение, состоящее из нескольких модулей, а не из одного модуля простой структуры. В этом случае разумно предположить, что все модули приложения находятся в одном каталоге и полный путь в этот каталог можно взять из среды, как это делалось в работе 2. Понятно, что такое приложение должно запускаться в соответствии со стандартами ОС.

В работе исследуется интерфейс между вызывающим и вызываемым модулями по управлению и по данным. Для запуска вызываемого модуля используется функция 4B00h прерывания int 21h. Все загрузочные модули находятся в одном каталоге. Необходимо обеспечить возможность запуска модуля динамической структуры из любого каталога.

Ход работы.

Была реализована программа, которая осуществляет вызов загрузочного модуля из того же каталога, в котором находится она сама. Перед запуском вызываемой программы осуществляется подготовка параметров для запуска. Так как изначально запущенной программе отводится вся доступная в данный момент память операционной системы, необходимо подготовить в памяти место для запуска новой программы, освободив память, не используемую вызывающей программой. Память освобождается при помощи функции 4Ah прерывания int 21h. Если память не может быть освобождена, выводится сообщение об ошибке в зависимости от кода ошибки.

В сегменте данных вызывающей программы расположен блок параметров, в который помещается следующая информация: сегментный адрес среды, сегмент и смещение командной строки, сегмент и смещение первого блока управления файлом (FCB), сегмент и смещение второго FCB. В качестве

сегментного адреса среды указывается 0, и в таком случае вызываемая программа наследует среду вызывающей программы. Другим параметрам 3

присваиваются значения соответствующих параметров вызывающей программы.

Осуществляется подготовка строки, содержащей путь до вызываемой программы. Для этого в строку записывается путь до вызываемой программы, который расположен после переменных среды, и дополняется именем вызываемого модуля.

Программа сохраняет содержимое регистров SS, SP и DS в переменных. В регистры DS:DX заносится адрес строки с маршрутом до вызываемой программы, в регистры ES:BX – адрес блока параметров. После этого вызывается загрузчик ОС с помощью функции 4Bh прерывания int 21h. Если программа не была загружена, выводится сообщение об ошибке в соответствии с кодом ошибки. Иначе обрабатывается завершение программы.

При помощи функции 4Dh прерывания int 21h определяется причина завершения программы. Функция возвращает код причины завершения в регистре AH. Программа выводит сообщение о причине завершения. Если программа завершилась нормально, выводится код завершения, возвращаемый функцией в регистре AL.

В качестве вызываемой программы использована программа, разработанная в лабораторной работе № 2, модифицированная так, чтобы перед выходом из программы функцией 01h прерывания int 21h запрашивался символ с клавиатуры и записывался в регистр AL в качестве кода завершения.

Программа была запущена из каталога с разработанными модулями. После вызова этой программой другой программы был введен символ «А».

Программа завершилась нормально с кодом завершения «А». Результат работы представлен на рисунке 1.

Программа была запущена повторно из того же каталога, при этом при запросе вызываемой программой символа с клавиатуры была введена комбинация Ctrl-C. Результат выполнения программы представлен на рисунке 2.

```
C:\>lab6
Unavailable memory address: 99FF
Enviroment address: 00FF
End of command line is empty
Enviroment data: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Loaded modlue path: C:\LAB2.COM
Program terminated normally
Termination code:A
```

Рисунок 1 - Запуск программы с вводом символа «А»

```
C:\>lab6
Unavailable memory address: 99FF
Enviroment address: 00FF
End of command line is empty
Enviroment data: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Loaded modlue path: C:\LAB2.COM
Program terminated normally
Termination code:♥
```

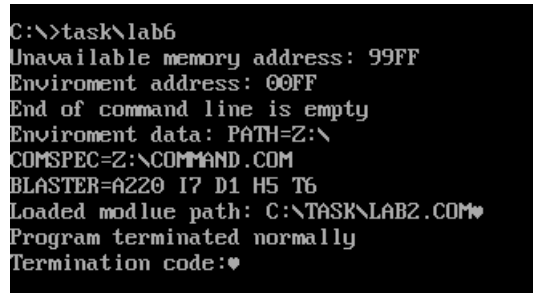
Рисунок 2 - Запуск программы с вводом символов Ctrl-C

В каталоге с разработанными модулями был создан новый каталог, куда были скопированы полученные модули. Программа была вызвана из нового каталога с вводом символа «W». Результат выполнения программы представлен на рисунке 3.

```
C:\>task\lab6
Unavailable memory address: 99FF
Enviroment address: 00FF
End of command line is empty
Enviroment data: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Loaded modlue path: C:\TASK\LAB2.COM
Program terminated normally
Termination code:W
```

Рисунок 3 - Запуск программы из другого каталога с вводом символа «W»

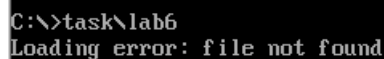
Программа была запущена еще раз из нового каталога, при этом при запросе ввода символа была введена комбинация Ctrl-C. Результат работы программы представлен на рисунке 4.



```
C:\>task\lab6
Unavailable memory address: 99FF
Enviroment address: 00FF
End of command line is empty
Enviroment data: PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Loaded modlue path: C:\TASK\LAB2.COM
Program terminated normally
Termination code:0
```

Рисунок 4 - Запуск программы из другого каталога с вводом символов Ctrl-C

Из нового каталога была удалена копия вызываемого модуля. Затем оттуда была запущена вызывающая программа. Результат ее работы представлен на рисунке 5.



```
C:\>task\lab6
Loading error: file not found
```

Рисунок 5 - Запуск программы, когда модули находятся в разных каталогах

Ответы на вопросы.

1. Как реализовано прерывание Ctrl-C?

При нажатии Ctrl-C процессу посылается сигнал SIGINT. Если процесс не установил собственный обработчик сигнала, система запускает обработчик по умолчанию, который завершает выполнение процесса.

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

Если код причины завершения 0, значит, была вызвана функция выхода из программы (4Ch прерывания int 21h) и программа завершилась нормально.

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

При нажатии Ctrl-C вызывается прерывание int 23h. Обработчик по умолчанию вызывает немедленное завершение программы.

Выводы

Было реализовано приложение, состоящее из двух загрузочных модулей, расположенных в одном каталоге: вызывающего и вызываемого. Вызывающий модуль осуществляет подготовку к запуску вызываемого модуля, а затем вызывает его при помощи функции 4Bh прерывания int 21h. В ходе выполнения работы был исследован интерфейс между вызывающим и вызываемым модулями по управлению и по данным.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
CODE    SEGMENT
        ASSUME DS:CODE, CS:CODE, SS:CODE, ES:CODE
        org 100h
START:
        jmp MAIN

        MemErr7      db      'Memory error: memory control block has
        been destroyed',13,10,'$'
        MemErr8      db      'Memory error: lack of memory',13,10,'$'
        MemErr9      db      'Memory error: incorrect block address'
        ,13,10,'$'
        ProgErr1     db      'Loading error: incorrect function numb
        er',13,10,'$'
        ProgErr2     db      'Loading error: file not found',13,10,'
        $'
        ProgErr5     db      'Loading error: disc error',13,10,'$'
        ProgErr8     db      'Loading error: lack of memory',13,10,'
        $'
        ProgErr10    db      'Loading error: incorrect environment s
        tring',13,10,'$'
        ProgErr11    db      'Loading error: incorrect format',13,10
        , '$'
        TermReas0    db      'Program terminated normally',13,10,'$'
        TermReas1    db      'Program terminated by Ctrl-
        Break',13,10,'$'
        TermReas2    db      'Device error termination',13,10,'$'
        TermReas3    db      'Function 31h termination',13,10,'$'
        TermCode     db      'Termination code:  ',13,10,'$'
        KEEP_SS      dw      0
        KEEP_SP      dw      0
        KEEP_DS      dw      0
        MStack       db      100h dup(0)
        ParameterBlock dw      0
                        dd      0      ;командная строка
                        dd      0      ;1-й FCB
                        dd      0      ;2-й FCB
        Path         db      128 DUP('$')
        Endl         db      13,10,'$'

;-----освобождение памяти-----
MEMFREE PROC NEAR
        push AX
        push BX
        push DX
        push CX

        mov BX,offset LAST_BYTE
        mov cl, 4
        shr BX,CL
        inc BX
        mov AH,4Ah
```

```

int 21h

jnc MEMFREE_success
cmp AX,7
je MEMFREE_err7
cmp AX,8
je MEMFREE_err8
cmp AX,9
je MEMFREE_err9

MEMFREE_err7:
    mov DX,offset MemErr7
    jmp MEMFREE_err_write
MEMFREE_err8:
    mov DX,offset MemErr8
    jmp MEMFREE_err_write
MEMFREE_err9:
    mov DX,offset MemErr9
    jmp MEMFREE_err_write
MEMFREE_err_write:
    mov AH,09h
    int 21h
    mov AH,4Ch
    int 21h

MEMFREE_success:
    pop CX
    pop DX
    pop BX
    pop AX
    ret
MEMFREE          ENDP

FILL_PARAM_BLOCK PROC NEAR
    push AX
    push BX
    push CX

    mov BX,offset ParameterBlock
    mov AX,ES
                                ;загружаем сегментный адрес среды
    mov CX,0                    ;если адрес 0, то вызываемая программа
    mov [BX],CX                ;наследует среду вызывающей

    mov CX,80h
    mov [BX+2],AX               ;сегмент
    mov [BX+4],CX               ;и смещение командной строки

    mov CX,5Ch
    mov [BX+6],AX               ;сегмент
    mov [BX+8],CX               ;и смещение первого FCB

    mov CX,6Ch
    mov [BX+10],AX              ;сегмент
    mov [BX+12],CX              ;и смещение второго FCB

    pop CX
    pop BX

```



```

        pop AX
        ret
FILL_PARAM_BLOCK ENDP

PREPARE_PATH PROC NEAR
    push ES
    push SI
    push DI
    push DX

    mov ES,DS:[2Ch]          ;извлекаем сегментный адрес среды
    mov SI,0                 ;инициализируем счетчик
PREPARE_PATH_env_loop:
    mov DL,ES:[SI]
    inc SI
    cmp DL,0h
    jne PREPARE_PATH_env_loop

    mov DL,ES:[SI]
    inc SI
    cmp DL,0h
    jne PREPARE_PATH_env_loop

    add SI,2

    mov DI,offset Path
PREPARE_PATH_path_loop:
    mov DL,ES:[SI]
    cmp DL,00h
    je PREPARE_PATH_path_end
    mov [DI],DL
    inc DI
    inc SI
    jmp PREPARE_PATH_path_loop

PREPARE_PATH_path_end:
    sub DI,8
    mov [DI], byte ptr 'l'
    mov [DI+1], byte ptr 'a'
    mov [DI+2], byte ptr 'b'
    mov [DI+3], byte ptr '2'
    mov [DI+4], byte ptr '.'
    mov [DI+5], byte ptr 'c'
    mov [DI+6], byte ptr 'o'
    mov [DI+7], byte ptr 'm'
    mov [DI+8], byte ptr 0

    pop DX
    pop DI
    pop SI
    pop ES
    ret
PREPARE_PATH ENDP

LOAD PROC NEAR
    mov KEEP_SS,SS
    mov KEEP_SP,SP

```

```

mov AX,DS
mov ES,AX
mov BX,offset ParameterBlock
mov DX,offset Path
mov AX,4B00h
mov al, 0h
int 21h

mov cx, cs
mov DS, cx
mov ES, cx
mov SP,KEEP_SP
mov SS,KEEP_SS

jnc LOAD_loaded
cmp AX,1
je LOAD_err1
cmp AX,2
je LOAD_err2
cmp AX,5
je LOAD_err5
cmp AX,8
je LOAD_err8
cmp AX,10
je LOAD_err10
cmp AX,11
je LOAD_err11

LOAD_err1:
    mov DX,offset ProgErr1
    jmp LOAD_err_write
LOAD_err2:
    mov DX,offset ProgErr2
    jmp LOAD_err_write
LOAD_err5:
    mov DX,offset ProgErr5
    jmp LOAD_err_write
LOAD_err8:
    mov DX,offset ProgErr8
    jmp LOAD_err_write
LOAD_err10:
    mov DX,offset ProgErr10
    jmp LOAD_err_write
LOAD_err11:
    mov DX,offset ProgErr11
    jmp LOAD_err_write
LOAD_err_write:
    mov AH,09h
    int 21h
    mov AH,4Ch
    int 21h

LOAD_loaded:
    mov DX,offset Endl
    mov ah, 09h
    int 21h

    mov AH,4Dh

```

```

        int 21h

        cmp AH,0
        je LOAD_res0
        cmp AH,1
        je LOAD_res1
        cmp AH,2
        je LOAD_res2
        cmp AH,3
        je LOAD_res3

LOAD_res0:
        mov DX,offset TermReas0
        mov AH,09h
        int 21h
        mov DI,offset TermCode
        add DI,17
        mov [DI],AL
        mov [DI+1],AH
        mov DX,offset TermCode
        mov AH,09h
        int 21h
        jmp LOAD_end

LOAD_res1:
        mov DX,offset TermReas1
        jmp LOAD_res_write

LOAD_res2:
        mov DX,offset TermReas2
        jmp LOAD_res_write

LOAD_res3:
        mov DX,offset TermReas3
        jmp LOAD_res_write

LOAD_res_write:
        mov ah, 09h
        int 21h

LOAD_end:
        ret
LOAD ENDP

;-----
MAIN:
        mov SP,offset MStack
        add SP,100h

        call MEMFREE
        call FILL_PARAM_BLOCK
        call PREPARE_PATH
        call LOAD

        mov al, 0
        mov AH, 4Ch
        int 21h

LAST_BYTE:
CODE      ENDS
END START

```