

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

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

Студентка гр. 9383

Лапина А.А.

Преподаватель

Ефремов М. А.

Санкт-Петербург

2021

Цель работы.

Исследовать возможности построения загрузочного модуля динамической структуры и интерфейс между вызывающим и вызываемым модулями по управлению и по данным.

Постановка задачи.

Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет следующие функции:

1. подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором находится сам;
2. запускает вызываемый модуль с использованием загрузчика;
3. после запуска проверяет выполнение загрузчика, а затем результат выполнения вызываемой программы.

В качестве вызываемой программы необходимо взять программу второй лабораторной работы, которая распечатывает среду и командную строку.

Запустить отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры. Ввести символ из числа A-Z, пронаблюдать причину вывода и код.

Запустить отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры. Ввести комбинацию символов Ctrl-C, пронаблюдать причину вывода и код.

Запустить отлаженную программу, когда текущим каталогом является какой-либо другой каталог, отличный от того, в котором содержатся разработанные модули. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры. Повторить ввод комбинаций клавиш.

Запустить отлаженную программу, когда модули находятся в разных каталогах.

Выполнение работы.

Для выполнения данной работы были реализованы следующие функции:

- * `println` для вывода строки на экран;
- * `num2dec` для записи в строку числа, лежащего в регистре AL;
- * `free` для очистки неиспользуемой модулем памяти;
- * `setp` для создания блока параметров;
- * `getpath` для получения пути вызываемого модуля;
- * `callp` для вызова модуля;
- * `main` для выполнения поставленной в данной лабораторной работе задачи.

Для вывода информации на экран были созданы следующие строки:

- * `unknown`, хранящая в себе строку 'Unknown error\$'
- * `ferr7`, хранящая в себе строку 'Error: memory control block destroyed\$'
- * `ferr8`, хранящая в себе строку 'Error: not enough memory to execute the function\$'
- * `ferr9`, хранящая в себе строку 'Error: invalid memory block address\$'
- * `cerr1`, хранящая в себе строку 'Error: function number is incorrect\$'
- * `cerr2`, хранящая в себе строку 'Error: file could not be found\$'
- * `cerr5`, хранящая в себе строку 'Disk error\$'
- * `cerr8`, хранящая в себе строку 'Error: insufficient memory\$'
- * `cerrA`, хранящая в себе строку 'Error: wrong environment string\$'
- * `cerrB`, хранящая в себе строку 'Error: wrong format\$'
- * `lgood`, хранящая в себе строку 'The program has ended with the code: \$'
- * `lerr1`, хранящая в себе строку 'The program terminated by Ctrl-Break\$'
- * `lerr2`, хранящая в себе строку 'The program terminated by device error\$'
- * `lerr3`, хранящая в себе строку 'The program terminated by function 31h\$'.

Выполнение программы начинается с вызова функции `free` для освобождения неиспользуемой программой памяти. Для этого с помощью

функции 4Ah прерывания 21h выделяется необходимое программе количество памяти. В случае, если при этом возникла какая-либо ошибка, программа выводит соответствующее сообщение об ошибке.

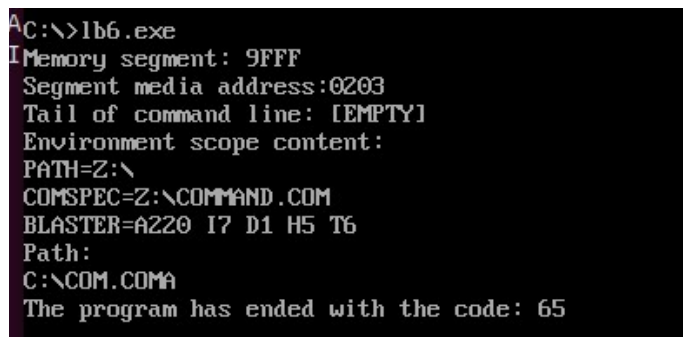
После этого вызывается функция setp, которая заполняет выделенный в сегменте данных блок параметров.

С помощью функции getpath считывается в строку путь до текущего файла, после чего имя файла меняется на имя вызываемого модуля.

Функция callp сохраняет сегментные регистры DS и ES, а также регистры, отвечающие за стек. Устанавливаются регистры ES:BX так, чтобы они указывали на блок параметров, после чего вызывается программный модуль. После выполнения модуля регистры возвращаются, проверяется работа модуля.

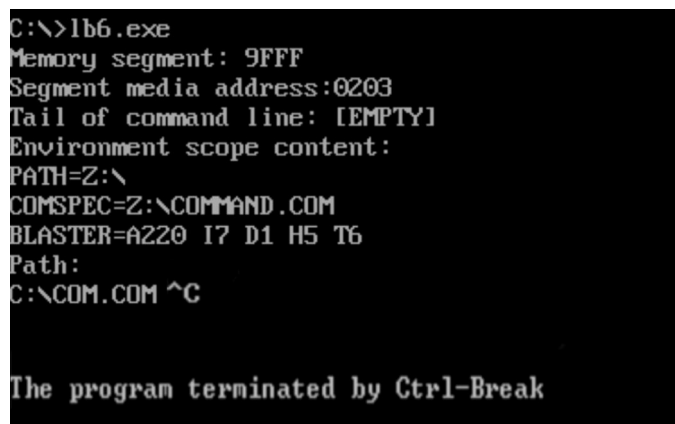
Разработанный программный код см. в приложении А.

Результаты работы программы представлены на рисунках 1-4.



```
A C:\>lb6.exe
Memory segment: 9FFF
Segment media address: 0203
Tail of command line: [EMPTY]
Environment scope content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path:
C:\COM.COM
The program has ended with the code: 65
```

Рисунок 1 – Ввод символа 'A'



```
C:\>lb6.exe
Memory segment: 9FFF
Segment media address: 0203
Tail of command line: [EMPTY]
Environment scope content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path:
C:\COM.COM ^C

The program terminated by Ctrl-Break
```

Рисунок 2 – Ввод комбинации символов Ctrl-C

```

C:\>LAB6\LB6.EXE
Memory segment: 9FFF
Segment media address:0203
Tail of command line: [EMPTY]
Environment scope content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path:
C:\LAB6\COM.COMs
The program has ended with the code: 115

```

Рисунок 3 – Запуск модуля из другой директории

```

C:\>lb6.exe

Error: file could not be found

```

Рисунок 4 – Запуск модуля из разных директорий

Выводы.

В ходе работы были исследованы возможности построения загрузочного модуля динамической структуры и интерфейс между вызывающим и вызываемым модулями по управлению и по данным.

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как реализовано прерывание Ctrl-C?

При нажатии комбинации клавиш Ctrl-C управление передаётся по адресу 0000:008Ch, который копируется в PSP функциями 26h и 4Ch и восстанавливается при выходе из программы.

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

В точке вызова функции 4Ch прерывания 21h.

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

В точке вызова функции 01h прерывания 21h.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb6.asm

```
code segment
assume CS:code, DS:data
```

```
println proc
    push    AX
    push    DX
    mov     AH, 09h
    int     21h
    mov     AH, 02h
    mov     DL, 0Ah
    int     21h
    mov     DL, 0Dh
    int     21h
    pop     DX
    pop     AX
    ret
println endp
```

```
num2dec proc
    push    AX
    push    BX
    push    CX
    push    DX
    push    SI
    mov     BX, 10
    xor     CX, CX
    xor     DX, DX
div10: div    BL
    mov     DL, AH
    push    DX
    xor     AH, AH
    inc     CX
    xor     DX, DX
    cmp     AL, 0
    jne     div10
loop10: pop    DX
    xor     DH, DH
    add     DL, 30h
```

```

    mov    byte ptr [SI], DL
    inc    SI
    loop   loop10
    pop    SI
    pop    DX
    pop    CX
    pop    BX
    pop    AX
    ret
num2dec endp

free  proc
    push   AX
    push   BX
    push   DX
    xor     DX, DX
    mov     AX, offset endprog
    add     AX, offset enddata
    add     AX, 300h
    mov     BX, 16
    div     BX
    mov     BX, AX
    inc     BX
    mov     AH, 4Ah
    int     21h
    jnc     endf
    cmp     AX, 7
    je      efree7
    cmp     AX, 8
    je      efree8
    cmp     AX, 9
    je      efree9
    jmp     efreeU
efree7: mov     DX, offset ferr7
        jmp     eprint
efree8: mov     DX, offset ferr8
        jmp     eprint
efree9: mov     DX, offset ferr9
        jmp     eprint
efreeU: mov     DX, offset unknown
eprint: call    println
endf:  pop     DX
        pop     BX

```

```

        pop    AX
        ret
free    endp

```

```

setp    proc
        push   AX
        mov    AX, ES:[2Ch]
        mov    param, AX
        mov    param + 2, ES
        mov    param + 4, 80h
        pop    AX
        ret
setp    endp

```

```

getpath proc
        push   DX
        push   DI
        push   SI
        push   ES
        xor     DI, DI
        mov     ES, ES:[2Ch]
        mov     DL, ES:[DI]
        jmp     check
nextc:  inc     DI
        mov     DL, ES:[DI]
check:  cmp     DL, 00h
        jne     nextc
        inc     DI
        mov     DL, ES:[DI]
        cmp     DL, 00h
        jne     nextc
        xor     SI, SI
        add     DI, 3
getc:   mov     DL, ES:[DI]
        cmp     DL, 00h
        je      getf
        mov     byte ptr fpath[SI], DL
        inc     DI
        inc     SI
        jmp     getc
getf:   dec     SI
        mov     DL, fpath[SI]
        cmp     DL, '\'

```



```

    jne    getf
    inc    SI
    xor    DI, DI
addp:  mov    DL, fname[DI]
    cmp    DL, '$'
    je     endg
    mov    fpath[SI], DL
    inc    DI
    inc    SI
    jmp    addp
endg:  mov    fpath[SI], 00h
    pop    ES
    pop    SI
    pop    DI
    pop    DX
    ret
getpath endp

```

```

callp proc
    push    AX
    push    DX
    push    DS
    push    ES
    mov     keepSS, SS
    mov     keepSP, SP
    mov     AX, DS
    mov     ES, AX
    mov     BX, offset param
    mov     DX, offset fpath
    mov     AX, 4B00h
    int     21h
    mov     DX, keepSP
    mov     SP, DX
    mov     SS, keepSS
    pop     ES
    pop     DS
    jnc     ok
    cmp     AX, 1
    je      ecall1
    cmp     AX, 2
    je      ecall2
    cmp     AX, 5
    je      ecall5

```

```

    cmp    AX, 8
    je     ecall8
    cmp    AX, 10
    je     ecallA
    cmp    AX, 11
    je     ecallB
    jmp     ecallU
ecall1: mov    DX, offset cerr1
        jmp     print
ecall2: mov    DX, offset cerr2
        jmp     print
ecall5: mov    DX, offset cerr5
        jmp     print
ecall8: mov    DX, offset cerr8
        jmp     print
ecallA: mov    DX, offset cerrA
        jmp     print
ecallB: mov    DX, offset cerrB
        jmp     print
ecallU: mov    DX, offset unknown
        jmp     print
ok:     mov    AX, 4D00h
        int    21h
        cmp    AH, 0
        je     good
        cmp    AH, 1
        je     eprog1
        cmp    AH, 2
        je     eprog2
        cmp    AH, 3
        je     eprog3
        jmp     ecallU
eprog1: mov    DX, offset lerr1
        jmp     print
eprog2: mov    DX, offset lerr2
        jmp     print
eprog3: mov    DX, offset lerr3
        jmp     print
good:   mov    DX, offset lgood
        mov    SI, DX
        add    SI, 37
        call   num2dec
print:  push    AX
        push    DX

```

```

        mov     AH, 02h
        mov     DL, 0Dh
        int     21h
        mov     AH, 02h
        mov     DL, 0Ah
        int     21h
        pop     DX
        pop     AX
        call    println
        pop     DX
        pop     AX
        ret
callp    endp

```

```

main     proc far
        mov     AX, data
        mov     DS, AX
        call    free
        call    setp
        call    getpath
        call    callp
; --- End ---
        mov     AX, 4C00h
        int     21h
main     endp

```

```

endprog:
code     ends

```

```

data     segment
param    dw 7 dup (0)
fname    db 'COM.COM$'
fpath    db 64 dup (0), '$' ; or maybe more
keepSS   dw 0
keepSP   dw 0
unknown  db 'Unknown error$'
ferr7    db 'Error: memory control block destroyed$'
ferr8    db 'Error: not enough memory to execute the function$'
ferr9    db 'Error: invalid memory block address$'
cerr1    db 'Error: function number is incorrect$'
cerr2    db 'Error: file could not be found$'
cerr5    db 'Disk error$'
cerr8    db 'Error: insufficient memory$'

```

```

cerrA db 'Error: wrong environment string$'
cerrB db 'Error: wrong format$'
lgood db 'The program has ended with the code:  $'
lerr1 db 'The program terminated by Ctrl-Break$'
lerr2 db 'The program terminated by device error$'
lerr3 db 'The program terminated by function 31h$'
enddata db 0
data ends

stack segment stack
    db 256 dup (?)
stack ends

    end main

```

Название файла: com.asm

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

ORG 100H ;так как адресация начинается со смещением 100 в .com

START: JMP BEGIN ;точка входа (метка)

; Данные

MEMORY db 'Memory segment: ',0DH,0AH,'\$' ;сегментный адрес
недоступной памяти

MEDIA db 'Segment media address: ',0DH,0AH,'\$' ;сегментный адрес среды

TAIL db 'Tail of command line: ',0DH,0AH,'\$' ;хвост командной строки

EMPTY db 'Tail of command line: [EMPTY]',0DH,0AH,'\$';пустой хвост

CONTENT db 'Environment scope content:',0DH,0AH, '\$'

END_STRING db 0DH,0AH, '\$'

PATH db 'Path: ',0DH,0AH, '\$'

; Процедуры

;-----

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 шест. 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, 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
;-----

F1 PROC near
    mov ax, ds:[02h]

    mov di, offset MEMORY
    add di, 19
    call WRD_TO_HEX
    mov dx, offset MEMORY
    mov AH,09h

```

```

    int 21h
    ret
F1 ENDP

F2 PROC near
    mov ax, ds:[2Ch]

    mov di, offset MEDIA
    add di, 25
    call WRD_TO_HEX
    mov dx, offset MEDIA
    mov AH,09h
    int 21h
    ret
F2 ENDP

F3 PROC near
    mov cx, 0
    mov cl, ds:[80h]
    mov si, offset TAIL
    add si, 22
    cmp cl, 0      ;если пусто
    je empty_tail
    mov di, 0
    mov ax, 0
read_tail:
    mov al, ds:[81h+di]
    inc di
    mov [si], al
    inc si
    loop read_tail ;цикл считывания

    mov dx, offset TAIL
    jmp write_tail
empty_tail:
    mov dx, offset EMPTY
write_tail:
    mov AH,09h
    int 21h
    ret
F3 ENDP

```

```

F4 PROC near
    mov dx, offset CONTENT
    mov AH,09h
    int 21h
    mov di, 0
    mov ds, ds:[2Ch]
read_str:
    cmp byte ptr [di], 0
    je end_str
    mov dl, [di]
    mov ah, 02h
    int 21h
    jmp find_end
end_str:
    cmp byte ptr [di+1],00h
    je find_end
    push ds
    mov cx, cs
    mov ds, cx
    mov dx, offset END_STRING
    mov AH,09h
    int 21h
    pop ds
find_end:
    inc di
    cmp word ptr [di], 0001h
    je read_path
    jmp read_str
read_path:
    push ds
    mov ax, cs
    mov ds, ax
    mov dx, offset PATH
    mov AH,09h
    int 21h
    pop ds
    add di, 2
loop2:
    cmp byte ptr [di], 0
    je break
    mov dl, [di]
    mov ah, 02h
    int 21h
    inc di

```

```
    jmp loop2  
break:  
    ret  
F4 ENDP
```

; Код

BEGIN:

```
    call F1 ;определение адреса недоступной памяти  
    call F2 ;определение сегментного адреса среды  
    call F3 ;определение хвоста  
    call F4 ;получает содержимое области среды и путь
```

```
    mov AH, 01h  
    int 21h  
    mov AH, 4Ch  
    int 21h
```

TESTPC ENDS

END START; конец модуля, START - точка выхода