

**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**

Студентка гр. 0382

\_\_\_\_\_

Деткова А.С.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2022

### **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.

### **Задание.**

**Шаг 1.** Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет функции:

1) Подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором находится он сам. Вызываемому модулю передается новая среда, созданная вызывающим модулем и новая командная строка.

2) Вызываемый модуль запускается с использованием загрузчика.

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

В качестве вызываемой программы необходимо взять программу ЛР 2, которая распечатывает среду и командную строку. Эту программу следует

немного модифицировать, вставив перед выходом из нее обращение к функции ввода символа с клавиатуры. Введенное значение записывается в регистр AL и затем происходит обращение к функции выхода 4Ch прерывания int 21h.

**Шаг 2.** Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры.

Введите произвольный символ из числа A-Z. Посмотрите причину завершения и код. Занесите полученные данные в отчет.

**Шаг 3.** Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры.

Введите комбинацию символов Ctrl-C. Посмотрите причину завершения и код. Занесите полученные данные в отчет.

**Шаг 4.** Запустите отлаженную программу, когда текущим каталогом является какой-либо другой каталог, отличный от того, в котором содержатся разработанные программные модули.

Повторите ввод комбинаций клавиш. Занесите полученные данные в отчет.

**Шаг 5.** Запустите отлаженную программу, когда модули находятся в разных каталогах. Занесите полученные данные в отчет.

### **Выполнение работы.**

Функции, используемые в программе:

1. *\_PRINT* — напечатать строку, адрес смещения до которой лежит в регистре DX.
2. *BYTE\_TO\_DEC* - переводит байт в число в дес. сс и записывает результат в строку по адресу SI.

3. *FREE\_UP\_MEMORY* — высвобождает лишнюю память.
4. *SET\_PARAMETERS* — создание блока параметров.
5. *GET\_PATH* — получение пути к вызываемому модулю.
6. *RUN\_LAB2* — запуск вызываемого модуля из основного модуля.
7. *MAIN* — вызывающая функция.

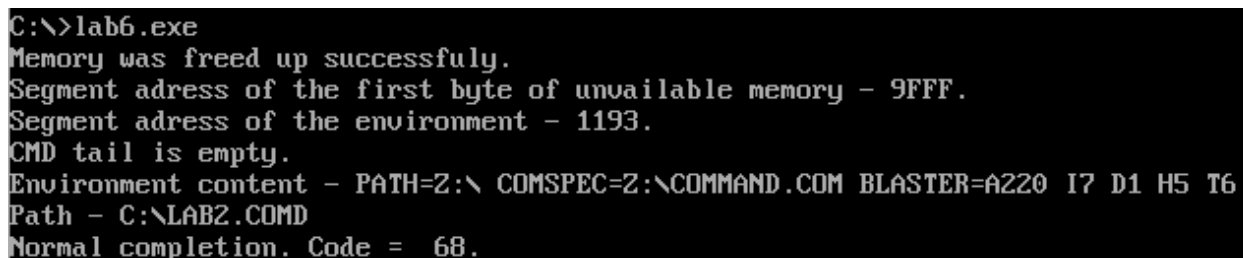
### **Шаг 1.**

На первом шаге был написан и отлажен .EXE модуль, который подготавливает параметры для запуска загрузочного модуля из той же директории, в которой находится сам вызываемый модуль, а также запускает его, выводя сообщения о результатах работы и ошибках. В качестве вызываемого модуля была взята программа lab2, модифицированная таким образом, чтобы в конце она запрашивала у пользователя ввести символ с клавиатуры.

### **Шаг 2.**

На втором шаге был запущен модуль .EXE и введен символ D. Результаты работы программ см. Рис. 1.

Как видно по рисунку 1, была запущен модуль lab6.exe, память успешно была освобождена. Далее был запущен модуль lab2.com, в конце был введен символ D, модуль был успешно завершен с кодом 68, что соответствует дес. представлению символа D в таблице ASCII.



```
C:\>lab6.exe
Memory was freed up successfully.
Segment adress of the first byte of unavailable memory - 9FFF.
Segment adress of the environment - 1193.
CMD tail is empty.
Environment content - PATH=Z:\ COMSPEC=Z:\COMMAND.COM BLASTER=A220 I7 D1 H5 T6
Path - C:\LAB2.COMD
Normal completion. Code = 68.
```

Рисунок 1: Результат второго шага

### Шаг 3.

На третьем шаге был запущен модуль .EXE и введена комбинация Ctrl+C. Результаты работы программ см. Рис. 2.

```
C:\>lab6.exe
Memory was freed up successfully.
Segment adress of the first byte of unavailable memory - 9FFF.
Segment adress of the environment - 1193.
CMD tail is empty.
Environment content - PATH=Z:\ COMSPEC=Z:\COMMAND.COM BLASTER=A220 I7 D1 H5 T6
Path - C:\LAB2.COM♥
Normal completion. Code = 3.
```

Рисунок 2: Результат третьего шага

Исходя из вывода программы видно, что результата аналогичен шагу 2, это связано с тем, что DosBox распознает комбинацию Ctrl+C как символ (сердечко, ASCII код = 3).

### Шаг 4.

На четвертом шаге была создана директория TMP, куда были помещены загрузочные модули, а разработанные программы остались в основной папке. Был запущен модуль .EXE и введен символ G, а затем комбинация Ctrl+C.

```
C:\TMP>dir
Directory of C:\TMP\
.                <DIR>                30-04-2022 20:43
..               <DIR>                30-04-2022 20:43
LAB2             COM                  468 30-04-2022 18:12
LAB6             EXE                 1,680 30-04-2022 19:53
2 File(s)        2,148 Bytes.
2 Dir(s)         262,111,744 Bytes free.

C:\TMP>lab6.exe
Memory was freed up successfully.
Segment adress of the first byte of unavailable memory - 9FFF.
Segment adress of the environment - 1193.
CMD tail is empty.
Environment content - PATH=Z:\ COMSPEC=Z:\COMMAND.COM BLASTER=A220 I7 D1 H5 T6
Path - C:\TMP\LAB2.COMG
Normal completion. Code = 71.
```

Рисунок 3: Результаты четвертого шага

```
C:\TMP>lab6.exe
Memory was freed up successfully.
Segment address of the first byte of unavailable memory - 9FFF.
Segment address of the environment - 1193.
CMD tail is empty.
Environment content - PATH=2:\ COMSPEC=2:\COMMAND.COM BLASTER=A220 I7 D1 H5 T6
Path - C:\TMP\LAB2.COM
Normal completion. Code = 3.
```

Рисунок 4: Результат четвертого шага

Как видно по рис. 3-4, независимо от того, где находится написанная программа, важно, чтобы модули находились в одной папке.

### Шаг 5.

Затем модуль lab2.com был перемещен обратно в корневую директорию, а lab6.exe остался в директории TMP/.

```
C:\TMP>lab6.exe
Memory was freed up successfully.
File is not found.
```

Рисунок 5: Результат пятого шага

Как видно по рис. 5, если загрузочные модули находятся в разных директориях, то запустить второй модуль не получится.\

Исходный код программы см. в приложении А.

### Ответы на контрольные вопросы.

#### 1. Как реализовано прерывание Ctrl+C?

При нажатии сочетания клавиш Ctrl+C срабатывает прерывание int 23H. Управление передается по адресу 0000:008C, а затем этот адрес копируется в поле PSP с помощью функций 26H и 4CH, затем он восстанавливается из PSP при выходе из программы. Стандартный обработчик прерывания 23H завершает выполнение 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 месте вызова функции 01H прерывания int 21H, в месте, где считывается символ с клавиатуры.

### **Выводы.**

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

## ПРИЛОЖЕНИЕ А

### КОД МОДУЛЕЙ

Название файла: lab5.asm

```
AStack SEGMENT STACK
        DB 100H DUP('!')
AStack ENDS
```

DATA SEGMENT

```
ErrorMSG db 'Memory free up error: $'
Err7MSG db 'Memory control block (MCB) is destroyed.', 0DH, 0AH,
'$'
Err8MSG db 'Not enough memory to execute the function.', 0DH,
0AH, '$'
Err9MSG db 'Incorrect memory block address.', 0DH, 0AH, '$'
SuccesFreeMSG db 'Memory was freed up successfully.', 0DH, 0AH, '$'

ParametrBlock dw 0 ;сегментный адрес среды
               dd 0 ;сегмент и смещение командной строки
               dd 0 ;сегмент и смещение FCB
               dd 0 ;сегмент и смещение FCB

flag db 0

FileName db 'lab2.com$'
PathName db 50 dup (0)

KEEP_SS dw 0
KEEP_SP dw 0

LoadErr1MSG db 'Incorrect function number.', 0DH, 0AH, '$'
LoadErr2MSG db 'File is not found.', 0DH, 0AH, '$'
LoadErr5MSG db 'Disk error.', 0DH, 0AH, '$'
LoadErr8MSG db 'Insufficient memory.', 0DH, 0AH, '$'
LoadErr10MSG db 'Incorrect environment string.', 0DH, 0AH, '$'
LoadErr11MSG db 'Incorrect format.', 0DH, 0AH, '$'

End0 db 0DH, 0AH, 'Normal completion. Code = .', 0DH, 0AH, '$'
End1 db 0DH, 0AH, 'CTRL-Break completion.', 0DH, 0AH, '$'
End2 db 0DH, 0AH, 'Device error completion.', 0DH, 0AH, '$'
End3 db 0DH, 0AH, '31H completion.', 0DH, 0AH, '$'
```

DATA ENDS

CODE SEGMENT

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



BYTE\_TO\_DEC PROC NEAR

```
    push CX
    push DX
    xor AH,AH
    xor DX,DX
    mov CX,10
loop_bd:
    div CX
    or DL,30h
    mov [SI],DL
    dec SI
    xor DX,DX
    cmp AX,10
    jae loop_bd
    cmp AL,00h
    je end_l
    or AL,30h
    mov [SI],AL

end_l:
    pop DX
    pop CX
    ret
```

BYTE\_TO\_DEC ENDP

\_PRINT PROC NEAR

```
    push AX

    mov AH,09H
    int 21H

    pop AX
    ret
```

\_PRINT ENDP

FREE\_UP\_MEMORY PROC NEAR

```
    push AX
    push BX
    push CX
    push DX

    lea BX,end_program
    mov AX,ES
    sub BX,AX
    shr BX,4
    inc BX
    mov AH,4AH
```

```

    int 21H
    jnc success_free_up
    mov flag,01H

    mov DX,offset ErrorMessage
    call _PRINT

    cmp AX,07H
    mov DX,offset Err7MSG
    je end_free_up

    cmp AX,08H
    mov DX,offset Err8MSG
    je end_free_up

    cmp AX,09H
    mov DX,offset Err9MSG
    je end_free_up

success_free_up:
    mov DX,offset SuccesFreeMSG

end_free_up:
    call _PRINT

    pop DX
    pop CX
    pop BX
    pop AX
    ret

FREE_UP_MEMORY ENDP

SET_PARAMETERS PROC NEAR

    push AX

    mov AX,ES:[2CH]
    mov ParametrBlock,AX
    mov ParametrBlock+2,ES
    mov ParametrBlock+4,80H

    pop AX
    ret

SET_PARAMETERS ENDP

GET_PATH PROC NEAR

    push AX
    push BX
    push DX
    push SI

```

```

    push DI
    push ES

    xor DI,DI
    mov ES,ES:[2CH]

skip_content:
    mov DL,ES:[DI]
    cmp DL,0H
    je last_content
    inc DI
    jmp skip_content

last_content:
    inc DI
    mov DL,ES:[DI]
    cmp DL,0H
    jne skip_content

    add DI,3H
    mov SI,0H

write_path:
    mov DL,ES:[DI]
    cmp DL,0H
    je delete_file_name
    mov PathName[SI],DL
    inc DI
    inc SI
    jmp write_path

delete_file_name:
    dec SI
    cmp PathName[SI],'\ '
    je ready_add_file_name
    jmp delete_file_name

ready_add_file_name:
    mov DI,-1

add_file_name:
    inc SI
    inc DI
    mov DL,FileName[DI]
    cmp DL,'$'
    je path_end
    mov PathName[SI],DL
    jmp add_file_name

path_end:

    pop ES
    pop DI
    pop SI
    pop DX

```

```
pop BX
pop AX
ret
```

GET\_PATH ENDP

RUN\_LAB2 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 ParametrBlock
mov DX, offset PathName
mov AX, 4B00H
int 21H
```

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

```
jnc loading_successful
```

```
cmp AX, 01H
mov DX, offset LoadErr1MSG
je print_err
cmp AX, 02H
mov DX, offset LoadErr2MSG
je print_err
cmp AX, 05H
mov DX, offset LoadErr5MSG
je print_err
cmp AX, 08H
mov DX, offset LoadErr8MSG
je print_err
cmp AX, 10H
mov DX, offset LoadErr10MSG
je print_err
cmp AX, 11H
mov DX, offset LoadErr11MSG
je print_err
```

loading\_successful:

```

    mov AX,4D00H
    int 21H

    cmp AH,01H
    mov DX,offset End1
    je print_err
    cmp AH,02H
    mov DX,offset End2
    je print_err
    cmp AH,03H
    mov DX,offset End3
    je print_err
    cmp AH,00H
    jne end_run

    mov SI,offset End0+30
    call BYTE_TO_DEC
    mov DX,offset End0

print_err:
    call _PRINT

end_run:
    pop DX
    pop CX
    pop BX
    pop AX

    ret

RUN_LAB2 ENDP


MAIN    PROC    FAR

    sub AX,AX
    push AX
    mov AX,DATA
    mov DS,AX

    call FREE_UP_MEMORY
    cmp flag,0
    jne final
    call SET_PARAMETERS
    call GET_PATH
    call RUN_LAB2

final:
    mov AH,4CH
    xor AL,AL
    int 21H

Main    ENDP

```

```
end_program:  
CODE      ENDS  
          END MAIN
```