**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МОЭВМ**

# ОТЧЕТ

**по лабораторной работе №6**

**по дисциплине «Операционные системы»**

**Тема: Построение модуля динамической структуры**

Студент гр. 9382 _____ Субботин М.О.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

**Цель работы.**

Исследование возможности построения загрузочного модуля динамической структуры. В отличии 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.

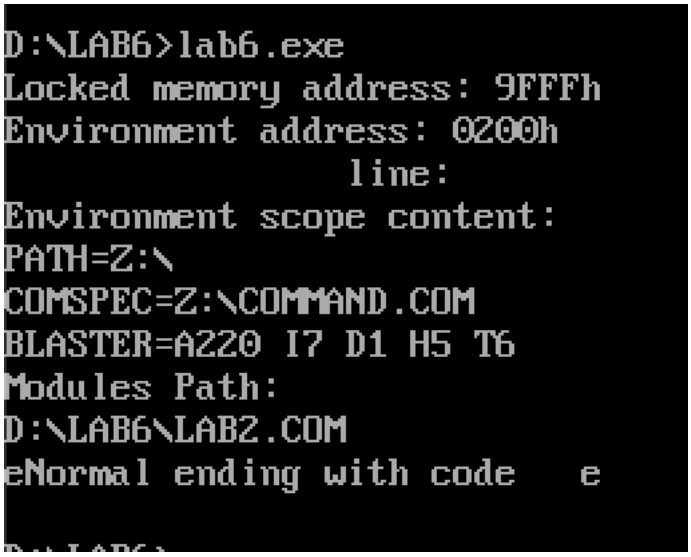
**Шаг 4.** Запустите отлаженную программу, когда текущим каталогом является какой- либо другой каталог, отличный от того, в котором содержатся разработанные программные модули.

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.

**Ход выполнения:**

Работа программы при запуске в директории с модулем и напечатанным символом ‘e’:



Работа программы при запуске в директории с модулем и нажатию ctrl+c:


Работа программы при запуске из другой директории:


Работа программы при запуске в директории, в который отсутствует вызываемый модуль:

**Ответы на контрольные вопросы.**

1. **Как реализовано прерывание Ctrl-C?**

   По нажатию Ctrl-C управление передается по адресу 0000:008Ch. Этот адрес копируется в PSP функциями 26h и 4Ch, затем восстанавливается из PSP при выходе из программы.

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

   Завершается при выполнении функции 4Ch прерывания int 21h.

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

   Программа завершится в месте ожидания нажатия клавиши(ctrl-c) 01h вектора прерывания int 21h.

**Выводы.**

Была исследована возможность построения загрузочного модуля динамической структуры.

## ПРИЛОЖЕНИЕ А

## КОД ПРОГРАММЫ

**Lab6.asm:**

```
ASTACK    SEGMENT  STACK
        DW 128 DUP(?)
ASTACK    ENDS

DATA  SEGMENT
  PARAMETERSS    DW 0
                 DD 0
                 DD 0
                 DD 0
  FILENAME DB 'LAB2.COM', 0
  CMD_L DB 1H, 0DH
  POS_CL DB 128 DUP(0)
  KEEP_SS DW 0
  KEEP_SP DW 0
  KEEP_PSP DW 0

  MEMORY_N7 DB 'DESTROYED MEMORY BLOCK',13,10,'$'
  MEMORY_N8 DB 'NOT ENOUGH MEMORY FOR RUNNING
FUNCTION',13,10,'$'
  MEMORY_N9 DB 'INCORRECT MEMORYS ADDRESS',13,10,'$'

  ERROR_N1  DB 'WRONG FUNCTIONS NUMBER',13,10,'$'
  ERROR_N2  DB 'FILE WAS NOT FOUND',13,10,'$'
  ERROR_N5  DB 'DISK ERROR',13,10,'$'
  ERROR_N8  DB 'DISK HAS NOT ENOUGH FREE MEMORY
SPACE',13,10,'$'
  ERROR_N10 DB 'WRONG STRING ENVIROMENT',13,10,'$'
  ERROR_N11 DB 'INCORRECT FORMAT',13,10,'$'

  END_N0 DB 'NORMAL ENDING WITH CODE     ',13,10,'$'
  END_N1 DB 'ENDING BY CTRL-BREAK',13,10,'$'
  END_N2 DB 'ENDING BY DEVICE ERROR',13,10,'$'
  END_N3 DB 'ENDING BY 31H FUNCTION',13,10,'$'

  END_DATA DB 0
DATA ENDS

CODE SEGMENT
```

```
  ASSUME CS:CODE,DS:DATA,SS:ASTACK

WRITE_STRING PROC
  PUSH AX
  MOV AH, 09H
  INT 21H
  POP AX
  RET
WRITE_STRING ENDP

FREE_MEMORY PROC
  PUSH AX
  PUSH BX
  PUSH CX
  PUSH DX

  MOV AX, OFFSET END_DATA
  MOV BX, OFFSET END_APP
  ADD BX, AX
  SHR BX, 1
  SHR BX, 1
  SHR BX, 1
  SHR BX, 1
  ADD BX, 2BH
  MOV AH, 4AH
  INT 21H

  JNC END_FREE_MEMORY

  LEA DX, MEMORY_N7
  CMP AX, 7
  JE WRITE_MEMORY_COMMENT
  LEA DX, MEMORY_N8
  CMP AX, 8
  JE WRITE_MEMORY_COMMENT
  LEA DX, MEMORY_N9
  CMP AX, 9
  JE WRITE_MEMORY_COMMENT
  JMP END_FREE_MEMORY

WRITE_MEMORY_COMMENT:
  MOV AH, 09H
  INT 21H
```

```
END_FREE_MEMORY:
  POP DX
  POP CX
  POP BX
  POP AX
  RET
FREE_MEMORY ENDP

SET_FULL_FILENAME PROC NEAR
  PUSH AX
  PUSH BX
  PUSH CX
  PUSH DX
  PUSH DI
  PUSH SI
  PUSH ES

  MOV AX, KEEP_PSP
  MOV ES, AX
  MOV ES, ES:[2CH]
  MOV BX, 0

FIND_SMTH:
  INC BX
  CMP BYTE PTR ES:[BX-1], 0
  JNE FIND_SMTH
  CMP BYTE PTR ES:[BX+1], 0
  JNE FIND_SMTH

  ADD BX, 2
  MOV DI, 0

FIND_LOOP:
  MOV DL, ES:[BX]
  MOV BYTE PTR [POS_CL + DI], DL
  INC DI
  INC BX
  CMP DL, 0
  JE END_LOOP
  CMP DL, '\'
  JNE FIND_LOOP
  MOV CX, DI
  JMP FIND_LOOP
END_LOOP:
```

```
    MOV DI, CX
    MOV SI, 0

LOOP_2:
    MOV DL, BYTE PTR[FILENAME + SI]
    MOV BYTE PTR [POS_CL + DI], DL
    INC DI
    INC SI
    CMP DL, 0
    JNE LOOP_2

    POP ES
    POP SI
    POP DI
    POP DX
    POP CX
    POP BX
    POP AX
    RET
SET_FULL_FILENAME ENDP

DEPLOY_ANOTHER_PROGRAM 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 DX, OFFSET CMD_L
    MOV BX, OFFSET PARAMETERSS
    MOV [BX+2], DX
    MOV [BX+4], DS
    MOV DX, OFFSET POS_CL

    MOV AX, 4B00H
    INT 21H

    MOV SS, KEEP_SS
    MOV SP, KEEP_SP
```

```
POP ES
POP DS

JNC COME_OVER

ERR_1:
CMP AX, 1
JNE ERR_2
MOV DX, OFFSET ERROR_N1
CALL WRITE_STRING
JMP DEPLOY_END

ERR_2:
CMP AX, 2
JNE ERR_5
MOV DX, OFFSET ERROR_N2
CALL WRITE_STRING
JMP DEPLOY_END

ERR_5:
CMP AX, 5
JNE ERR_8
MOV DX, OFFSET ERROR_N5
CALL WRITE_STRING
JMP DEPLOY_END

ERR_8:
CMP AX, 8
JNE ERR_10
MOV DX, OFFSET ERROR_N8
CALL WRITE_STRING
JMP DEPLOY_END

ERR_10:
CMP AX, 10
JNE ERR_11
MOV DX, OFFSET ERROR_N10
CALL WRITE_STRING
JMP DEPLOY_END

ERR_11:
CMP AX, 11
MOV DX, OFFSET ERROR_N11
CALL WRITE_STRING
```

```
  JMP DEPLOY_END

  COME_OVER:
  MOV AX, 4D00H
  INT 21H

  CMP AH, 0
  JNE END_1
  PUSH DI
  MOV DI, OFFSET END_N0
  MOV [DI+26], AL
  POP SI
  MOV DX, OFFSET END_N0
  CALL WRITE_STRING
  JMP DEPLOY_END

  END_1:
  CMP AH, 1
  JNE END_2
  MOV DX, OFFSET END_N1
  CALL WRITE_STRING
  JMP DEPLOY_END

  END_2:
  CMP AH, 2
  JNE END_3
  MOV DX, OFFSET END_N2
  CALL WRITE_STRING
  JMP DEPLOY_END

  END_3:
  CMP AH, 3
  MOV DX, OFFSET END_N3
  CALL WRITE_STRING

  DEPLOY_END:
  POP DX
  POP CX
  POP BX
  POP AX
  RET
DEPLOY_ANOTHER_PROGRAM ENDP

MAIN PROC FAR
```

```
  PUSH DS
  XOR   AX, AX
  PUSH  AX
  MOV   AX, DATA
  MOV   DS, AX
  MOV KEEP_PSP, ES
  CALL FREE_MEMORY
  CALL SET_FULL_FILENAME
  CALL DEPLOY_ANOTHER_PROGRAM

VERY_END:
  XOR AL,AL
  MOV AH,4CH
  INT 21H
MAIN ENDP
END_APP:
CODE ENDS
     END MAIN
```