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

# ОТЧЕТ

**по лабораторной работе №5**

**по дисциплине «Операционные системы»**

**ТЕМА: СОПРЯЖЕНИЕ СТАНДАРТНОГО И ПОЛЬЗОВАТЕЛЬСКОГО ОБРАБОТЧИКОВ ПРЕРЫВАНИЙ.**

Студентка гр. 0382 Чегодаева Е.А.

Преподаватель Ефремов М.А.

Санкт-Петербург

2022

**Цель работы.**

Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Пользовательский обработчик прерывания получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре. Он обрабатывает скан-код и осуществляет определенные действия, если скан-код совпадает с определенными кодами, которые он должен обрабатывать. Если скан-код не совпадает с этими кодами, то управление передается стандартному прерыванию.

**Задание.**

***Шаг 1***. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет такие же функции, как в программе ЛР 4, а именно:

1) Проверяет, установлено ли пользовательское прерывание с вектором 09h.

2) Если прерывание не установлено то, устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний. Адрес точки входа в стандартный обработчик прерывания находится в теле пользовательского обработчика. Осуществляется выход по функции 4Ch прерывания int 21h.

3) Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.

Выгрузка прерывания по соответствующему значению параметра в командной строке /un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождении памяти, занимаемой резидентом. Затем осуществляется выход по функции 4Ch прерывания int 21h.

Для того чтобы проверить установку прерывания, можно поступить следующим образом. Прочитать адрес, записанный в векторе прерывания. Предположим, что этот адрес указывает на точку входа в установленный резидент. На определенном, известном смещении в теле резидента располагается

сигнатура, некоторый код, который идентифицирует резидент. Сравнив известное значение сигнатуры с реальным кодом, находящимся в резиденте, можно определить, установлен ли резидент. Если значения совпадают, то резидент установлен. Длину кода сигнатуры должна быть достаточной, чтобы сделать случайное совпадение маловероятным.

Программа должна содержать код устанавливаемого прерывания в виде удаленной процедуры. Этот код будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

1) Сохранить значения регистров в стеке при входе и восстановить их при выходе.

2) При выполнении тела процедуры анализируется скан-код.

3) Если этот код совпадает с одним из заданных, то требуемый код записывается в буфер клавиатуры.

4) Если этот код не совпадает ни с одним из заданных, то осуществляется передача управления стандартному обработчику прерывания.

***Шаг 2***. Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания 09h установлен. Работа прерывания проверяется введением различных символов, обрабатываемых установленным обработчиком и стандартным обработчиком.

***Шаг 3***. Также необходимо проверить размещение прерывания в памяти. Для этого запустите программу ЛР 3, которая отображает карту памяти в виде списка блоков МСВ. Полученные результаты поместите в отчет.

***Шаг 4***. Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

***Шаг 5***. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

***Шаг 6***. Ответьте на контрольные вопросы.

**Выполнение работы.**

Шаг 1. На основе кода лабораторной работы №4 был написан и отлажен программный модуль типа .EXE, который выполняет следующие функции:

1) Проверяет, установлено ли пользовательское прерывание;

2) Устанавливает резидентную функцию для обработки прерывания;

3) Если резидентная функция уже установлена — выводит соответствующее сообщение;

4) Выгружает прерывание по значению параметра в командной строке: /un, восстанавливая стандартный вектор прерывания.

В данном случае обработчик пользовательского прерывания получает управление по прерыванию int 9h при нажатии клавиши на клавиатуре. Обрабатывается скан-код клавиши: если он совпадает с определенным заданным кодом, то символ будет заменён на другой, если не совпадает с заданным - управление вернётся к стандартному прерыванию.

Функции, реализованные в работе:

- ROUT — Обработчик прерывания (вместо символа «L» выводит сердечко «♥» );
- CHECK — Проверка, загружено ли пользовательское прерывание;
- SET_INT — Установка нового обработчика прерывания с запоминанием данных для восстановления предыдущего;
- DEL_ INT — Удаление пользовательского прерывания с восстановлением прерывания по умолчанию;
- PRINT — Осуществление вывода.

Шаг 2. Загрузка программы.


Рисунок 1 — Результат загрузки code5.exe

Так как при вводе строки “LLL” – символы были заменены на заданный, а те, что не были указаны в обработчике прерывания остались без изменения, можно сделать вывод о том, что пользовательское прерывание было успешно установлено.

Шаг 3. Проверка размещения прерывания в памяти.

Для того, чтобы удостовериться в корректности проверки — фиксируем вывод информации о состоянии блоков MCB перед установкой прерывания (пользуясь программой из лабораторной работы №3).

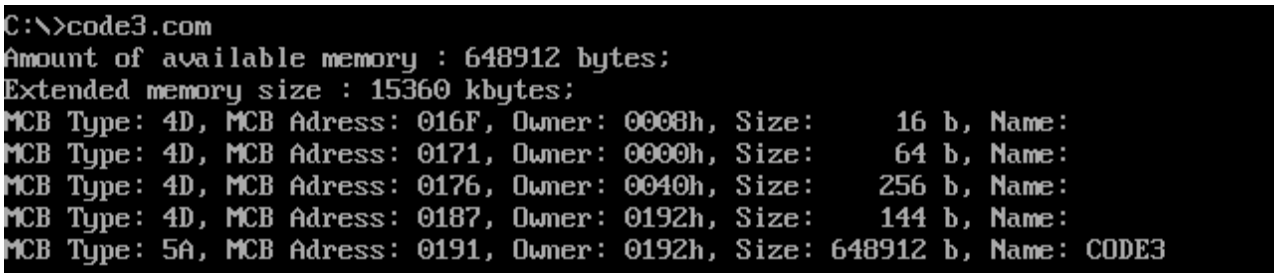


Рисунок 2 — Результат запуска code3.com перед установкой прерывания

Следом вновь запускаем code3.com после установки code5.exe:

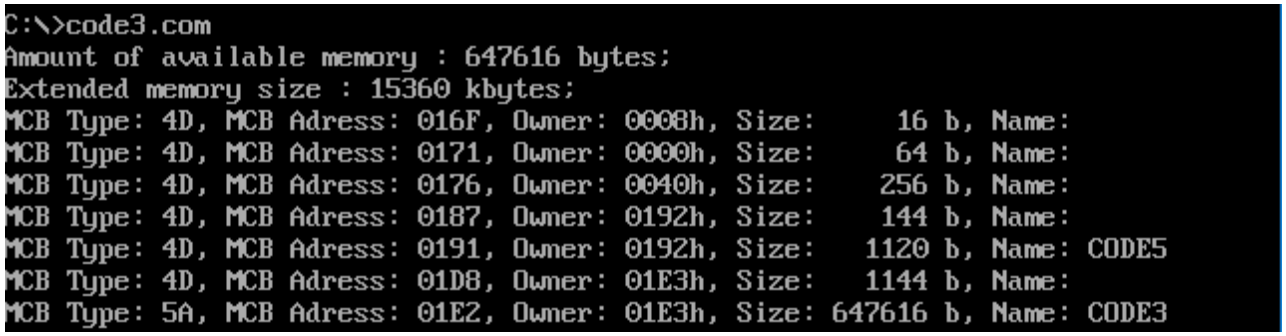


Рисунок 3 — Результат загрузки code3.com после установки прерывания

Исходя из вывода программы видно, что в памяти появились блоки MCB обработчика прерывания (code5.exe), что свидетельствует о том, что пользовательское прерывание успешно загружено в память.

Шаг 4. Повторный запуск программы.

```
C:\>code5.exe
User interrupt installed!

C:\>♥♥♥♥♥
Illegal command: ♥♥♥♥♥.

C:\>qwerty ♥♥♥
Illegal command: qwerty.
```

Рисунок 4 — Результат повторной загрузки lb.exe

В результате прерывание не было установлено повторно, о чем говорит соответствующее сообщение.

Шаг 5. Восстановление прерывания по умолчанию.

```
C:\>code5.exe /un
User interrupt unloaded!

C:\>LLL lll
Illegal command: LLL.
```

Рисунок 5 — Результат запуска code5.exe с ключом выгрузки

Символы введённой строки не были изменены, следовательно, пользовательское прерывание было выгружено и стандартный вектор рассматриваемого прерывания восстановлен.

```
C:\>code3.com
Amount of available memory : 648912 bytes;
Extended memory size : 15360 kbytes;
MCB Type: 4D, MCB Adress: 016F, Owner: 0008h, Size:     16 b, Name:
MCB Type: 4D, MCB Adress: 0171, Owner: 0000h, Size:     64 b, Name:
MCB Type: 4D, MCB Adress: 0176, Owner: 0040h, Size:    256 b, Name:
MCB Type: 4D, MCB Adress: 0187, Owner: 0192h, Size:    144 b, Name:
MCB Type: 5A, MCB Adress: 0191, Owner: 0192h, Size: 648912 b, Name: CODE3
```

Рисунок 6 — Результат запуска code3.com после выгрузки прерывания

Вывод программы code3.com демонстрирует то, что блоки памяти, в которых хранилось пользовательское прерывание — удалены, значит память, занимаемая резидентом — освобождена, что так же указывает на успешную выгрузку.

Исходный код программ см. в приложении А

**Контрольные вопросы.**

1) Какого типа прерывания использовались в работе?
   - Ответ: В работе использовались программные прерывания (int 21h) и аппаратные (int 16h, int 09h).
2) Чем отличается скан код от кода ASCII?
   - Ответ: Скан код – код клавиши на клавиатуре; Код ASCII – код символа согласно таблице ASCII.

**Выводы.**

Были изучены возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

Разработан пользовательский обработчик прерывания при нажатии клавиши на клавиатуре.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: code5.asm

```
AStack SEGMENT STACK
        DW 100 DUP(?)
AStack ENDS

DATA SEGMENT
LOADING  db 'User interrupt loaded...' , 0DH, 0AH, '$'
LOADED   db 'User interrupt installed!', 0DH, 0AH, '$'
UNLOADED db 'User interrupt unloaded!' , 0DH, 0AH, '$'
DATA ENDS

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

PRINT PROC NEAR
        push AX
        mov AH, 09h
        int 21h
        pop AX
        ret
PRINT ENDP

ROUT PROC far
        jmp _ROUT

        _STACK dw 100 dup (0)
        SIGN db '0000'
        KEEP_IP dw 0
        KEEP_CS dw 0
        KEEP_PSP dw 0
        KEEP_SS dw 0
        KEEP_AX         dw 0
        KEEP_SP dw 0
        OLD db 26h
        NEW db 03h

_ROUT:
        mov KEEP_SS, SS
        mov KEEP_AX, AX
        mov KEEP_SP, SP
        mov AX, seg _STACK
        mov SS, AX
        mov SP, 0
```

```
        mov AX, KEEP_AX

        in AL, 60h  ;ч и т а т ь  к л ю ч
        cmp AL, OLD ;э т о  т р е б у е м ы й  к о д?
        je DO_REQ     ;д а , а к т и в и з и р о в а т ь  о б р а б о т к у
REQ_KEY
                ;н е т , у й т и  н а  и с х о д н ы й  о б р а б о т ч и к
        pushf
        call dword ptr KEEP_IP ;п е р е х о д  н а  п е р в о н а ч а л ь н ы й  о
б р а б о т ч и к
        jmp FINAL

    DO_REQ:
        push AX
        in al, 61h ;в з я т ь  з н а ч е н и е  п о р т а  у п р а в л е н и я  к
л а в и а т у р о й
        mov AH, AL ;с о х р а н и т ь  е г о
        or AL, 80h ;у с т а н о в и т ь  б и т  р а з р е ш е н и я  д л я  к л
а в и а т у р ы
        out 61h, AL ;и  в ы в е с т и  е г о  в  у п р а в л я ю щ и й  п о р т
        xchg AH, AL ;и з в л е ч ь  и с х о д н о е  з н а ч е н и е  п о р т а
        out 61h, AL ;и  з а п и с а т ь  е г о  о б р а т н о
        mov AL, 20h ;п о с л а т ь  с и г н а л "к о н е ц  п р е р ы в а н и
я"
        out 20h, AL ;к о н т р о л л е р у  п р е р ы в а н и й 8259
        pop AX

    UPDATE:
        mov AL, 0
        mov AH, 05h ;К о д  ф у н к ц и и
        mov CL, NEW ;П и ш е м  с и м в о л  в  б у ф е р  к л а в и а т у р ы
        mov CH, 00h
        int 16h
        or AL, AL   ;п р о в е р к а  п е р е п о л н е н и я  б у ф е р а
        jz FINAL
        jmp UPDATE

    FINAL:
        pop ES
        pop DS
        pop DX
        pop AX
        mov AX,KEEP_SS
        mov SS, AX
        mov SP, KEEP_SP
        mov AX, KEEP_AX

        mov AL, 20h
```

```
        out 20h, AL

        iret
ROUT ENDP


CHECK PROC
        mov AH, 35h     ; функция получения вектора
        mov AL, 09h     ; номер вектора
        int 21h
        mov SI, offset SIGN
        sub SI, offset ROUT
        mov AX,'00'
        cmp AX, ES:[BX+SI]
        jne UNLOAD
        cmp AX, ES:[BX+SI+2]
        je LOAD

UNLOAD:
        call SET_INT
        mov DX, offset LAST_BYTE ; размер в байтах от начала сегмента
        mov CL, 4           ; перевод в параграфы
        shr DX, CL
        inc DX              ; размер в параграфах
        add DX, CODE
        sub DX, KEEP_PSP
        xor AL, AL
        mov AH, 31h
        int 21h

LOAD:
        push ES
        push AX
        mov AX, KEEP_PSP
        mov ES, AX
        cmp byte ptr ES:[82h],'/'
        jne stop
        cmp byte ptr ES:[83h],'u'
        jne stop
        cmp byte ptr ES:[84h],'n'
        je _UNLOAD

stop:
        pop AX
        pop ES
        mov DX, offset LOADED
        call PRINT
        ret
```

```
_UNLOAD:
        pop AX
        pop ES
        call DEL_INT
        mov DX, offset UNLOADED
        call PRINT
        ret
CHECK ENDP

SET_INT PROC
        push DX
        push DS
        mov AH, 35h      ; функция получения вектора
        mov AL, 09h      ; номер вектора
        int 21h
        mov KEEP_IP, BX  ; запоминание смещения
        mov KEEP_CS, ES  ; и сегмента

        mov DX, offset ROUT ; смещение для процедуры в DX
        mov AX, seg ROUT  ; сегмент процедуры
        mov DS, AX       ; помещаем в DS
        mov AH, 25h      ; функция установки вектора
        mov AL, 09h      ; номер вектора
        int 21h          ; меняем прерывание
        pop DS
        mov DX, offset LOADING
        call PRINT
        pop DX
        ret
SET_INT ENDP

DEL_INT PROC
        CLI
        push DS
        mov DX, ES:[BX+SI+4]
        mov AX, ES:[BX+SI+6]
        mov DS, AX
        mov AX, 2509h
        int 21h
        push ES
        mov AX, ES:[BX+SI+8]
        mov ES, AX
        mov ES, ES:[2Ch]
        mov AH, 49h
        int 21h
        pop ES
        mov ES, ES:[BX+SI+8]
```

```
        mov AH, 49h
        int 21h
        pop DS
        STI
        ret
DEL_INT ENDP

Main PROC FAR
        mov AX, DATA
        mov DS, AX
        mov KEEP_PSP, ES
        call CHECK
        xor AL, AL
        mov AH, 4Ch
        int 21h
Main ENDP

        LAST_BYTE:
CODE ENDS
  END Main
```

Название файла: code3.asm

```
TESTPC SEGMENT
        ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
        ORG     100H
START: jmp BEGIN
; ДАННЫЕ

AM db 'Amount of available memory :       bytes;', 0DH, 0AH, '$'
EM db 'Extended memory size :      kbytes;', 0DH, 0AH, '$'
MCB db 'MCB Type:   , MCB Adress:     , Owner:     h, Size:       b, Name: $'
END_STR db 0DH, 0AH,'$'

; 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 16-го 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, в 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
;--------------------------------------
BYTE_TO_DEC PROC near
; перевод в 10 с/с, в SI- адрес поля младшей цифры
        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_1
        or AL, 30h
        mov [SI], AL
end_1:  pop DX
        pop CX
        ret
```

```
BYTE_TO_DEC ENDP
;----------------------
; КОД
WRD_TO_DEC PROC near
        push CX
        push DX
        mov CX, 10
loop_wd:
        div CX
        or DL, 30h
        mov [SI], DL
        dec SI
        xor DX, DX
        cmp AX, 10
        jae loop_wd
        cmp AL, 00h
        je end_2
        or AL, 30h
        mov [SI], AL
end_2:
        pop DX
        pop CX
        ret
WRD_TO_DEC ENDP

PRINT PROC near
        mov AH, 09h
        int 21h
        ret
PRINT ENDP

_AM PROC near
        mov SI, offset AM
        add SI, 34
        mov AH, 4Ah
        mov BX, 0FFFFh
        int 21h
        mov AX, 16
        mul BX
        call WRD_TO_DEC
        mov DX, offset AM
        call PRINT
        ret
_AM ENDP

_EM PROC near
        mov SI, offset EM
        add SI, 27
        sub DX, DX
```

```
        mov AL, 30h  ; запись адреса ячейки CMOS
        out 70h, AL
        in AL, 71h      ; чтение младшего байта
        mov BL, AL    ; размер расширенной памяти
        mov AL, 31h  ; запись адреса ячейки CMOS
        out 70h, AL
        in AL, 71h      ; чтение старшего байта
                        ; размера расширенной памяти
        mov AH, AL
        mov Al, BL
        call WRD_TO_DEC
        mov DX, offset EM
        call PRINT
        ret
_EM ENDP

_MCB PROC  near
        mov AH, 52h
        int 21h
        sub BX, 2
        mov AX, ES:[BX]
        mov ES, AX
Chain:
;Type
        mov DI, offset MCB
        add DI, 10
        mov AX, ES:[00h]
        call BYTE_TO_HEX
         mov [DI], AL
         add DI, 1
        mov [DI], AH
;Adress
        mov DI, offset MCB
        add DI, 29
        mov AX, ES
        call WRD_TO_HEX
;Owner
        mov DI, offset MCB
        add DI, 42
         mov AX, ES:[01h]
        call WRD_TO_HEX
;Size
        mov SI, offset MCB
         add SI, 57
        mov AX, ES:[03h]
        mov BX, 16
        mul BX
         call WRD_TO_DEC
```

```
;Print
        mov DX, offset MCB
        call PRINT
;Name
         mov DI, offset MCB
        add DI, 58
        mov BX, 8
        mov CX, 7
        cycle:
        mov DL, ES:[BX]
                mov AH, 02h
                int 21h
                add BX, 1
                loop cycle

        mov AL, ES:[0h]
        cmp AL, 5ah
        je final

        mov BX, ES
        mov AX, ES:[03h]
        add AX, BX
        add AX, 1
        mov ES, AX
        mov DX, offset END_STR
        call PRINT
        jmp Chain
final:
      ret
_MCB ENDP

BEGIN:
        call _AM
        call _EM
        call _MCB
; Выход в DOS
        xor AL, AL
        mov AH, 4Ch
        int 21h
ENDING:
TESTPC  ENDS
        END START
```