

**МИНОБРНАУКИ РОССИИ**  
**Санкт-Петербургский государственный**  
**электротехнический университет**  
**«ЛЭТИ» им. В.И. Ульянова (Ленина)**  
**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №5**  
**по дисциплине «Операционные системы»**  
**Тема: Сопряжение стандартного и пользовательского обработчиков**  
**прерываний.**

Студент гр. 8382

Терехов А.Е.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

### 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 не совпадает с этими кодами, то управление передается стандартному прерыванию.

### Ход работы.

1. Был написан и отлажен EXE-модуль. Программа проверяет установлено ли пользовательское прерывание с вектором 09h. Если нет, то устанавливает резидентный обработчик прерывания и настраивает вектор прерываний. Суть обработки заключается в подмене реакции на нажатие Ctrl. При нажатии на Ctrl выводится символ 'Ы', а все остальные нажатия обрабатываются стандартным обработчиком. На рисунке 1 представлен результат выполнения программы.

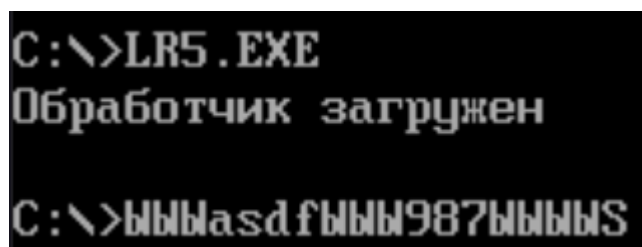


Рисунок 1. Результат выполнения программы

При попытке повторного запуска будет выведено соответствующее сообщение (рисунок 2).

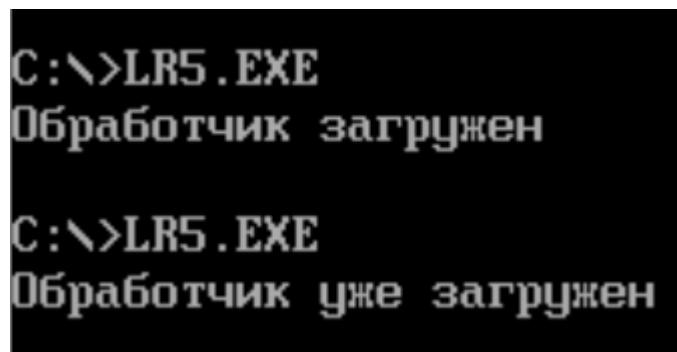
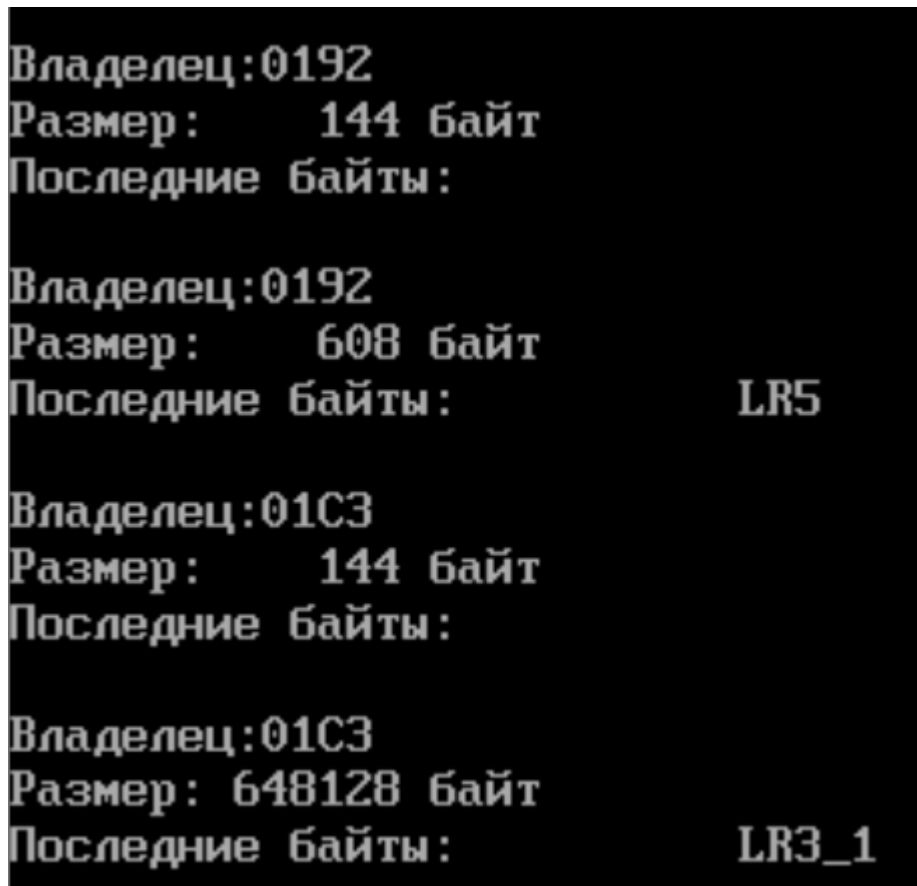


Рисунок 2. Попытка повторного запуска.

Вывод программы из лабораторной работы №3, после установки резидента представлен на рисунке 3.



```
Владелец:0192
Размер: 144 байт
Последние байты:

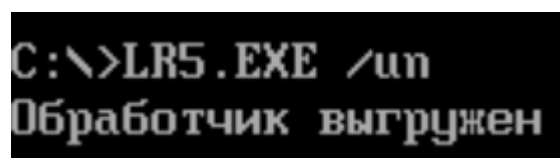
Владелец:0192
Размер: 608 байт
Последние байты: LR5

Владелец:01C3
Размер: 144 байт
Последние байты:

Владелец:01C3
Размер: 648128 байт
Последние байты: LR3_1
```

Рисунок 3. Расположение резидента в памяти.

Чтобы выгрузить резидент, необходимо передать ключ /un. С выводом программы при выгрузке резидента можно ознакомиться на рисунке 4.



```
C:\>LR5.EXE /un
Обработчик выгружен
```

Рисунок 4. Выгрузка резидента.

Вывод программы из лабораторной работы №3, после удаления резидента представлен на рисунке 5.

```
С:\>LR3_1.COM
Доступная память: 648912
Расширенная память: 58940 H

Владелец:MS DOS
Размер:      16 байт
Последние байты:

Владелец:свободный участок
Размер:      64 байт
Последние байты:

Владелец:0040
Размер:      256 байт
Последние байты:

Владелец:0192
Размер:      144 байт
Последние байты:

Владелец:0192
Размер: 648912 байт
Последние байты: LR3_1
```

Рисунок 5. Вывод блоков управления памятью после удаления обработчика.

### Ответы на вопросы.

1. Какого типа прерывания использовались в работе?

Аппаратное прерывание клавиатуры 09h, программные BIOS 16h и DOS 21h.

2. Чем отличается скан код от кода ASCII?

Скан код – это код каждой клавиши на клавиатуре. С его помощью драйвер клавиатуры определяет состояние каждой клавиши. ASCII код – это код символа в таблице ASCII. Так у клавиши Ctrl есть скан коды, но нет ASCII кода.

**Вывод.**

В ходе лабораторной работы была реализована программа встраивающая пользовательский обработчик прерываний в стандартный обработчик.

## ПРИЛОЖЕНИЕ А

```
CODE SEGMENT
ASSUME cs:CODE, ds:DATA, ss:AStack
```

```
ROUT PROC far
    jmp handler_start
    HANDLER_SIGN DW 4200h
    KEEP_PSP DW 0
    KEEP_IP DW 0
    KEEP_CS DW 0
    KEEP_SS DW 0
    KEEP_SP DW 0
    KEEP_AX DW 0
    REQ_KEY DB 1dh
    HANDLER_STACK DW 100 DUP(0)
    stack_top:
handler_start:
    mov KEEP_SS, ss
    mov KEEP_SP, sp
    mov KEEP_AX, ax
    mov ax, seg HANDLER_STACK
    mov ss, ax
    lea sp, stack_top
    push ax
    push bx
    push cx
    push dx
    push si
    push ds
    push bp
    push es

    in al, 60h
    cmp al, REQ_KEY
    je do_req
    pushf
    call dword ptr cs:KEEP_IP
    jmp end_handler
do_req:
    push ax
    in al, 61h
    mov ah, al
    or al, 80h
    out 61h, al
    xchg ah, al
    out 61h, al
    mov al, 20h
    out 20h, al
    pop ax
push_to_buffer:
    mov ah, 05h
    mov cl, 'H'
    xor ch, ch
    int 16h
    or al, al
    jnz skip
```

```

        jmp end_handler

skip:
    mov al, 40h
    mov es, ax
    mov ax, es:[lah]
    mov es:[lch], ax
    jmp push_to_buffer

end_handler:
    pop es
    pop bp
    pop ds
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    mov sp, KEEP_SP
    mov ax, KEEP_SS
    mov ss, ax
    mov ax, KEEP_AX
    mov al, 20h
    out 20h, al
    iret
LAST_BYTE:
ROUT ENDP

CHECK_IS_LOAD proc near
    push ax
    push bx
    push si
    push dx
    push es
    mov ah, 35h
    mov al, 09h
    int 21h
    lea si, HANDLER_SIGN
    sub si, offset ROUT
    mov ax, es:[bx+si]
    cmp ax, 4200h
    jne END_CHECK
    mov IS_LOAD, 1
END_CHECK:
    pop es
    pop dx
    pop si
    pop bx
    pop ax
    ret
CHECK_IS_LOAD ENDP

LOAD proc near
    push ax
    push dx
    push ds

```

```

        mov ah,35h
        mov al,09h
        int 21h
        mov KEEP_IP, bx
        mov KEEP_CS, es
        lea dx, ROUT
        mov ax,seg ROUT
        mov ds,ax
        mov ah,25h
        mov al,09h
        int 21h
        pop ds
        pop dx
        pop ax
        ret
LOAD ENDP

UNLOAD proc near
        cli
        push ax
        push bx
        push dx
        push es
        push si
        push ds

        mov ah, 35h
        mov al, 09h
        int 21h
        lea si, KEEP_IP
        sub si, offset ROUT

        mov dx, es:[bx+si]
        mov ax, es:[bx+si+2]
        mov ds, ax
        mov ah, 25h
        mov al, 09h
        int 21h
        pop ds

        mov ax, es:[bx+si-2]
        mov es, ax

        push es
        mov ax, es:[2ch]
        mov es, ax
        mov ah, 49h
        int 21h
        pop es

        mov ah, 49h
        int 21h

end_unload:
        pop si
        pop es
        pop dx

```



```

        pop bx
        pop ax
        sti
        ret
UNLOAD ENDP

WRITE proc near
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
WRITE ENDP

MAIN proc far
    mov ax, DATA
    mov ds, ax
    mov KEEP_PSP, es
    call CHECK_IS_LOAD
    mov al, IS_LOAD
    cmp al, 1
    je loaded
    jmp n_loaded
loaded:
    cmp byte ptr es:[81h+1], '/'
    jne end_main_loaded
    cmp byte ptr es:[81h+2], 'u'
    jne end_main_loaded
    cmp byte ptr es:[81h+3], 'n'
    jne end_main_loaded
    call UNLOAD
    jmp end_main_not_loaded
n_loaded:
    cmp byte ptr es:[81h+1], '/'
    jne load_handler
    cmp byte ptr es:[81h+2], 'u'
    jne load_handler
    cmp byte ptr es:[81h+3], 'n'
    jne load_handler
    jmp end_main_not_loaded
load_handler:
    call LOAD
    lea dx, M_LOADED
    call WRITE
    lea dx, LAST_BYTE
    mov cl, 4
    shr dx, cl
    inc dx
    add dx, CODE
    sub dx, KEEP_PSP
    xor al, al
    mov ah, 31h
    int 21h
end_main_loaded:
    lea dx, M_ALREADY_LOADED
    call WRITE
    jmp exit

```

```

end_main_not_loaded:
    lea dx, M_NOT_LOADED
    call WRITE
exit:
    xor al, al
    mov ah, 4ch
    int 21h
MAIN ENDP
CODE ENDS

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

DATA SEGMENT
    IS_LOAD DB 0
    M_LOADED DB 'Обработчик загружен',0Dh,0Ah,'$'
    M_ALREADY_LOADED DB 'Обработчик уже загружен',0Dh,0Ah,'$'
    M_NOT_LOADED DB 'Обработчик выгружен',0Dh,0Ah,'$'
    M_ISL DB ' ',0Dh,0Ah,'$'

DATA ENDS
END MAIN

```