

**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**

Студент гр. 8382

Мирончик П.Д.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

## Цель работы.

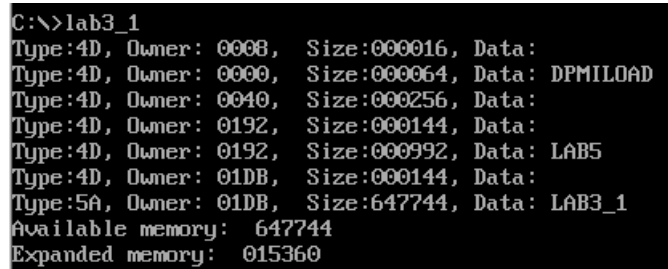
Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

## Ход работы.

Для реализации пользовательского обработчика прерываний от клавиатуры был частично использован подход, примененный в лабораторной работе 4. Пользовательский обработчик анализирует последнюю нажатую клавишу и, если была нажата клавиша q, заменяет ее на 1, иначе – передаёт управление стандартному обработчику. Полный исходный код программы находится в Приложении А.

Результатом работы программы при введённой строке “qeq” будет строка “1e1”.

На рисунке 1 рассмотрим МСВ блоки операционной системы после запуска программы.



```
C:\>lab3_1
Type:4D, Owner: 0008, Size:000016, Data:
Type:4D, Owner: 0000, Size:000064, Data: DPMILOAD
Type:4D, Owner: 0040, Size:000256, Data:
Type:4D, Owner: 0192, Size:000144, Data:
Type:4D, Owner: 0192, Size:000992, Data: LAB5
Type:4D, Owner: 01DB, Size:000144, Data:
Type:5A, Owner: 01DB, Size:647744, Data: LAB3_1
Available memory: 647744
Expanded memory: 015360
```

Рис.1. МСВ блоки после запуска программы

Теперь попытаемся повторно загрузить обработчик и ещё раз рассмотрим МСВ блоки (см. рис. 2).

```

C:\>lab5
Interraption is already installed

C:\>lab3_1
Type:4D, Owner: 0008, Size:000016, Data:
Type:4D, Owner: 0000, Size:000064, Data: DPMILOAD
Type:4D, Owner: 0040, Size:000256, Data:
Type:4D, Owner: 0192, Size:000144, Data:
Type:4D, Owner: 0192, Size:000992, Data: LAB5
Type:4D, Owner: 01DB, Size:000144, Data:
Type:5A, Owner: 01DB, Size:647744, Data: LAB3_1
Available memory: 647744
Expanded memory: 015360

```

Рис.2. MCB блоки после повторной установки прерывания

Можно заметить, что программа вывела сообщение о том, что обработчик уже загружен и не стала помещать в резидентную память ещё одну копию кода обработчика.

Выгрузим обработчик, попробуем ввести строку «qeq» и посмотрим на MCB блоки (см. рис. 3).

```

C:\>lab5 /un
Restoring default interruption...

C:\>qeq
Illegal command: qeq.

C:\>lab3_1
Type:4D, Owner: 0008, Size:000016, Data:
Type:4D, Owner: 0000, Size:000064, Data: DPMILOAD
Type:4D, Owner: 0040, Size:000256, Data:
Type:4D, Owner: 0192, Size:000144, Data:
Type:5A, Owner: 0192, Size:648912, Data: LAB3_1
Available memory: 648912
Expanded memory: 015360

```

Рис.3. Поведение и состояние системы после восстановления обработчика

## Ответы на вопросы.

1. Какого типа прерывания использовались в работе?

В работе использовалось прерывание DOS 21h, прерывание BIOS 16h для работы с клавиатурой, прерывание BIOS 09h, которое обрабатывалось

пользовательским кодом. Прерывание 09h генерируется при нажатии клавиши клавиатуры.

## 2. Чем отличается скан-код от кода ASCII?

Скан-код – это число, с помощью которого драйвер распознаёт какая клавиша была нажата. ASCII код – это числовое значение символа в таблице кодировки. Таким образом, символы «q» и «Q» имеют разные ASCII коды, но один скан-код, так как клавиша используется одна и та же.

## **Выводы**

В результате выполнения работы был реализован пользовательский обработчик прерываний от клавиатуры, который размещается в резидентной памяти. Обработчик заменяет введённую с помощью клавиатуры букву, либо передаёт управление стандартному обработчику.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

```
CODE    SEGMENT
        ASSUME cs:CODE, ds:CODE, ss:NOTHING
        org 100h

START:  jmp BEGIN

INSTALLING_STR      DB      'Installing...',10,13,'$'
ALREADY_INSTALLED_STR DB      'Interraption is already installed',10,13
, '$'
NOT_INSTALLED_STR   DB      'Interruption is not installed',10,13,'$'
RESTORING_STR        DB      'Restoring default interruption...',10,13
, '$'

INT_KEY PROC FAR
        jmp INT_KEY_begin

INT_KEY_data:
        ; data
        INT_KEY_sign      DW      1234h
        INT_KEY_stack     DB      100h DUP(0)
        INT_KEY_keep_int_ip DW      0
        INT_KEY_keep_int_cs DW      0
        INT_KEY_keep_ss   DW      0
        INT_KEY_keep_sp   DW      0

        ; code
INT_KEY_begin:
        push ds
        push ax
        mov ax, cs
        mov ds, ax

        mov INT_KEY_keep_ss, ss
        mov INT_KEY_keep_sp, sp

        mov ax, cs
        mov ss, ax
        mov sp, offset INT_KEY_stack
        add sp, 100h

        push es
        push bp
        push cx
        push bx
        push dx
        push di
        push si

        in al, 60h
        cmp al, 10h
        je INT_KEY_q
        jmp INT_KEY_default
```

```

INT_KEY_default:
    pushf
    call dword ptr INT_KEY_keep_int_ip
    jmp INT_KEY_finish

INT_KEY_q:
    mov cl, '1'
    jmp INT_KEY_handler

INT_KEY_handler:
    in al, 61h
    mov ah, al
    or al, 80h
    out 61h, al
    xchg ah, al
    out 61h, al
    mov al, 20h
    out 20h, al

    mov ah, 05h
    mov ch, 00h
    int 16h
    jmp INT_KEY_finish

INT_KEY_finish:
    pop si
    pop di
    pop dx
    pop bx
    pop cx
    pop bp
    pop es

    mov ss, INT_KEY_keep_ss
    mov sp, INT_KEY_keep_sp
    pop ax
    pop ds

    mov al, 20h
    out 20h, al
    iret
INT_KEY ENDP

WRD_TO_DEC PROC near
; input ax - value
;         di - lower num address
;         si - address of highest available num position (DI-max), or 0 if
;             prefix isn't need
;
; converts AX to DEC and writes to di address (to DI, DI-1, DI-2, ...)
    push bx
    push dx
    push di
    push si
    push ax

    mov bx, 10
WRD_TO_DEC_loop:

```

```

        div bx
        add dl, '0'
        mov [di], dl
        xor dx, dx
        dec di
        cmp ax, 0
        jne WRD_TO_DEC_loop

    cmp si, 0
    je WRD_TO_DEC_no_prefix
    cmp si, di
    jge WRD_TO_DEC_no_prefix
WRD_TO_DEC_prefix_loop:
        mov dl, '0'
        mov [di], dl
        dec di
        cmp di, si
        jl WRD_TO_DEC_prefix_loop

WRD_TO_DEC_no_prefix:
    pop ax
    pop si
    pop di
    pop dx
    pop bx
    ret
WRD_TO_DEC ENDP

LOAD_INT PROC NEAR
    mov ah, 35h
    mov al, 09h
    int 21h
    mov INT_KEY_keep_int_ip, bx
    mov INT_KEY_keep_int_cs, es

    mov dx, offset INT_KEY
    mov ah, 25h
    mov al, 09h
    int 21h

    mov dx, offset PROGRAM_END_BYTE
    mov cl, 4
    shr dx, cl
    inc dx
    mov ah, 31h
    int 21h
LOAD_INT ENDP

RELOAD_INT PROC NEAR
    push dx
    push ds
    push es
    push bx

    mov ah, 35h
    mov al, 09h
    int 21h

```

```

        mov ax, es
        mov ds, ax
        mov dx, INT_KEY_keep_int_ip
        mov ax, INT_KEY_keep_int_cs
        mov ds, ax
        mov ah, 25h
        mov al, 09h
        int 21h

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

        pop bx
        pop es
        pop ds
        pop dx
        ret
RELOAD_INT ENDP

CHECK_INT PROC NEAR
        push ax
        push bx
        push es

        mov ah, 35h
        mov al, 09h
        int 21h

        push ds
        mov ax, es
        mov ds, ax
        mov ax, INT_KEY_sign
        cmp ax, 1234h
        pop ds

        pop es
        pop bx
        pop ax
        ret
CHECK_INT ENDP

BEGIN:
        cmp byte ptr es:[81h+1], '/'
        jne LOAD_IF_NEEDED
        cmp byte ptr es:[81h+2], 'u'
        jne LOAD_IF_NEEDED
        cmp byte ptr es:[81h+3], 'n'
        jne LOAD_IF_NEEDED

        call CHECK_INT
        jne NOT_INSTALLED
        call RELOAD_INT
        mov dx, offset RESTORING_STR

```



```
    mov ah, 09h
    int 21h
    jmp EXIT
```

```
NOT_INSTALLED:
    mov dx, offset NOT_INSTALLED_STR
    mov ah, 09h
    int 21h
    jmp EXIT
```

```
LOAD_IF_NEED:
    call CHECK_INT
    je INSTALLED
    mov dx, offset INSTALLING_STR
    mov ah, 09h
    int 21h
    call LOAD_INT
    jmp EXIT
```

```
INSTALLED:
    mov dx, offset ALREADY_INSTALLED_STR
    mov ah, 09h
    int 21h
    jmp EXIT
```

```
EXIT:
    xor     al,al
    mov     ah,4Ch
    int     21h
```

```
PROGRAM_END_BYTE:
CODE    ENDS
END START
```