

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. 8383

Федоров И.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

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

Ход работы.

Был написан и отлажен программный модуль типа **.EXE**, который выполняет следующие функции:

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.
3. Если прерывание установлено, то выводится соответствующее сообщение.
4. Выгрузка прерывания по соответствующему значению параметра в командной строке /un.

Исходный код **.EXE** модуля приведен в приложении А.

При запуске программы устанавливается новый обработчик прерывания 09h, если он не был установлен. Если при вводе попадают определенные символы (в данном случае это строчные 'x' и 'z'), то в буфер клавиатуры вместо них помещаются другие символы. Заглавные 'X' и 'Z' обрабатываются стандартным обработчиком. Результат работы программы представлен на рис. 1. Карта памяти до запуска программы представлена на рис. 2, после установления обработчика - на рис. 3. Были введены символы: "XZfdxz". Заменяются лишь прописные символы 'x' и 'z'.

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
0 Warning Errors
0 Severe Errors

C:\MASM>link LB_5.OBJ

Microsoft (R) Overlay Linker Version 3.60
Copyright (C) Microsoft Corp 1983-1987. All rights reserved.

Run File [LB_5.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:

C:\MASM>LB_5.EXE

C:\MASM>XZfd |||
```

Рисунок 1 – Работа нового обработчика

```
Size available memory (b): 648912
Size extend memory (kb): 15360
-----
Num: 1
PSP address: 0008
Size (b): 16
Name:
-----
Num: 2
PSP address: 0000
Size (b): 64
Name:
-----
Num: 3
PSP address: 0040
Size (b): 256
Name:
-----
Num: 4
PSP address: 0192
Size (b): 144
Name:
-----
Num: 5
PSP address: 0192
Size (b): 648912
Name: LR3COM
```

Рисунок 2 – Карта памяти до размещения обработчика в памяти

```

Size available memory (b): 648128
Size extend memory (kb): 15360
-----
Num: 1
PSP address: 0008
Size (b): 16
Name:
-----
Num: 2
PSP address: 0000
Size (b): 64
Name:
-----
Num: 3
PSP address: 0040
Size (b): 256
Name:
-----
Num: 4
PSP address: 0192
Size (b): 144
Name:
-----
Num: 5
PSP address: 0192
Size (b): 608
Name: LB_5
-----
Num: 6
PSP address: 01C3
Size (b): 144
Name:
-----
Num: 7
PSP address: 01C3
Size (b): 648128
Name: LR3COM

```

Рисунок 3 – Карта памяти после размещения нового прерывания

Программа при повторном запуске определяет установлен ли новый обработчик прерываний, проверяя сигнатуру (просто значение), расположенную в резиденте и идентифицирующую его. При совпадении выводится соответствующее сообщение. Результат приведен на рис. 4.

```

C:\MASM>LB_5.EXE
C:\MASM>LB_5.EXE
INTERRUPT ALREADY LOAD!
C:\MASM>

```

Рисунок 4 – Определение установки обработчика

При запуске программы с соответствующим ключом выгрузки /un программа выгружает резидентный обработчик, выводит соответствующее

сообщение и освобождает память. Результат работы приведен на рис. 5, состояние памяти после выгрузки приведено на рис. 6. Выгрузка не срабатывает, если обработчик не был размещен в памяти (см. рис. 7).

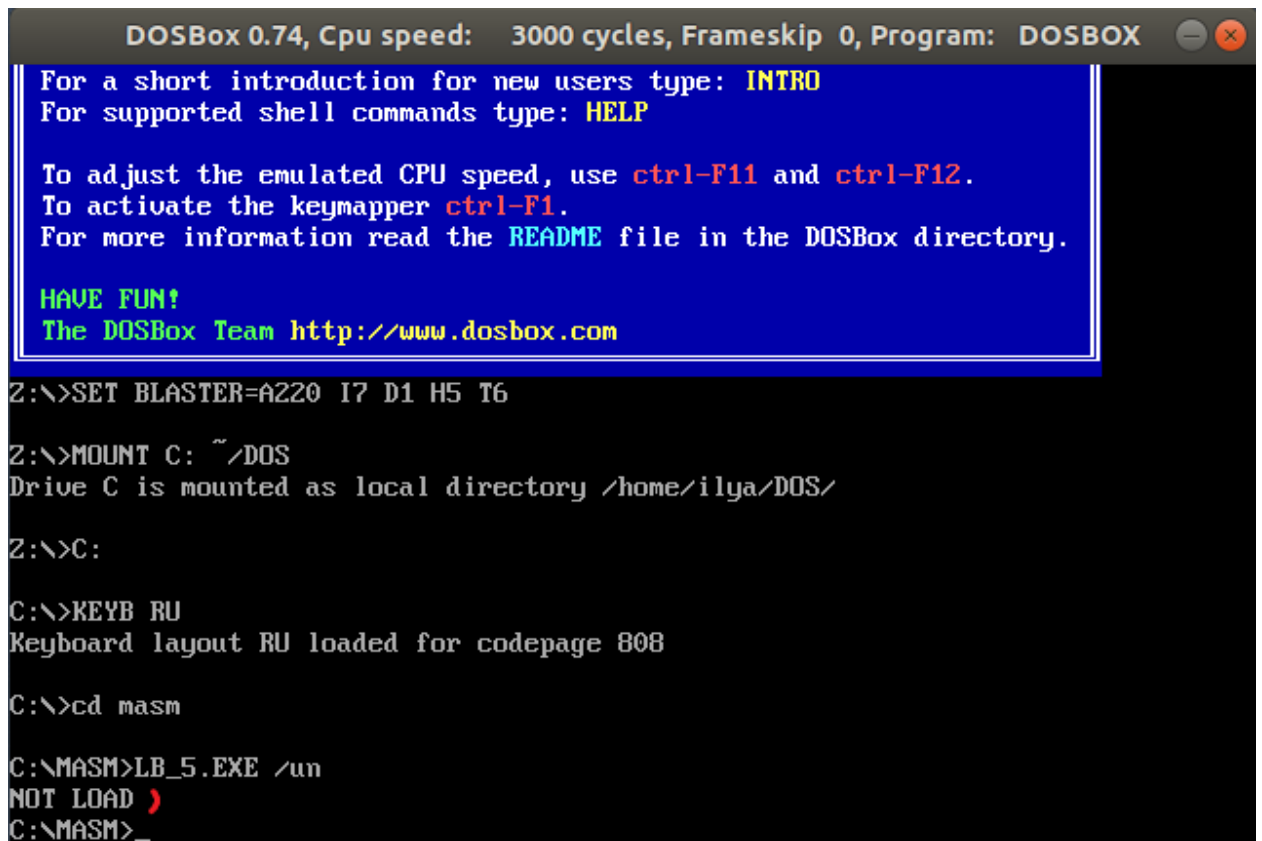
```
C:\MASM>LB_5.EXE
INTERRUPT ALREADY LOAD!
C:\MASM>

C:\MASM>LB_5.EXE /un
INTERRUPT UNLOAD!
C:\MASM>_
```

Рисунок 5 – Работа при наличии ключа выгрузки

```
|
Size available memory (b): 648912
Size extend memory (kb): 15360
-----
Num: 1
PSP address: 0008
Size (b): 16
Name:
-----
Num: 2
PSP address: 0000
Size (b): 64
Name:
-----
Num: 3
PSP address: 0040
Size (b): 256
Name:
-----
Num: 4
PSP address: 0192
Size (b): 144
Name:
-----
Num: 5
PSP address: 0192
Size (b): 648912
Name: LR3COM
```

Рисунок 6 – Состояние памяти после выгрузки резидента



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>MOUNT C: ~/DOS
Drive C is mounted as local directory /home/ilya/DOS/

Z:\>C:

C:\>KEYB RU
Keyboard layout RU loaded for codepage 808

C:\>cd masm

C:\MASM>LB_5.EXE /un
NOT LOAD
C:\MASM>_
```

Рисунок 7 – Попытка выгрузки обработчика без загрузки

Ответы на контрольные вопросы.

1) Какого типа прерывания использовались в работе?

Прерывание от клавиатуры - аппаратное. Были использованы программные int 21h.

2) Чем отличается скан-код от кода ASCII?

Скан-код это однобайтное число, младшие 7 битов которого представляют идентификационный номер, присвоенный каждой клавише (по сути порядковый номер). При нажатии клавиши int 09 считывает из порта 60h скан-код нажатия, и по таблице трансляции скан-кодов в коды ASCII получает символ. Коды ASCII - это байтные числа, которые соответствуют расширенному набору кодов ASCII (коды символов для вывода).

Выводы.

В ходе выполнения работы была написана программа, размещающая и выгружающая новый обработчик прерывания от клавиатуры. Была разобрана возможность встраивания пользовательского обработчика в стандартный.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

```
code segment
    assume cs:code, ds:data, ss:astack

RESIDENT_INTER PROC FAR
start:  jmp begin
        KEEP_IP dw 0
        KEEP_CS dw 0
        SEGMENT_ dw 0
        SIGNATURE_INTER dw 777h
        REQ_KEY1 db 2Dh      ; x
        REQ_KEY2 db 2Ch      ; Z
        mem dw 0
        char db 0
        ss_seg dw 0
        ss_offs dw 0
        LOCAL_STACK dw 64 dup (0)
        TOP_STACK=$
begin:
        mov ss_seg,ss
        mov ss_offs,sp
        mov mem, ax

        CLI                      ;сохранение кадра стека прерванной задачи
        mov ax,cs
        mov ss,ax
        mov sp,offset TOP_STACK
        STI

        mov ax, mem
        push es
        push ax
        push bx
        push cx
        push dx
verify_caps:
        mov AH,2
        int 16H                  ;байт статуса
        test AL,01000010B        ;проверка на caps и/или левый shift
        jnz call_std             ;если 0, то CAPS выключен

        xor ax, ax
        in al,60H                ;читать ключ
        cmp al,REQ_KEY1
        je do_req1
        cmp al, REQ_KEY2
        je do_req2
call_std:
        ;вызов стандартного
        pushf                    ;iret выталкивает ip,cs и flags
        call dword ptr cs:KEEP_IP
```



```

        jmp end_resident

do_req1:
    mov char, 0DEh
    jmp noise

do_req2:
    mov char, 0BAh
    jmp noise

noise:
    CLI
    push ax
    push dx
    mov ah, 2
    mov dl, 7
    int 21h
    pop dx
    pop ax
    jmp next_
    STI
next_:
    in al, 61H                ;взять значение порта управления
    клавиатурой
    mov ah, al                ; сохранить его
    or al, 80H                ;установить бит разрешения для клавиатуры
    out 61H, al               ; и вывести его в управляющий порт
    xchg ah, al               ; извлечь исходной значение порта
    out 61H, al               ;и записать его обратно
    mov al, 20H
    out 20H, al

print_bufer:
    mov ah, 05h
    mov cl, char
    mov ch, 00h
    int 16h
    or al, al
    jnz skip
    jmp end_resident

skip:
    CLI                        ;запрещаем прерывания
    SUB AX, AX                 ;обнуляем регистр
    MOV ES, AX                 ;добавочный сегмент - с начала памяти
    MOV AL, ES:[41AH]          ;берем указатель на голову буфера
    MOV ES:[41CH], AL          ;посылаем его в указатель хвоста
    STI                        ;разрешаем прерывания
    JMP print_bufer

end_resident:

    pop dx

```

```

    pop cx
    pop bx

    xor ax, ax
    mov ax, ss_seg
    mov ss, ax
    pop ax
    pop es
    mov sp, ss_offs
    mov AL, 20H
    OUT 20H, AL
    IRET
LAST_BYTE:
RESIDENT_INTER ENDP

```

```

WriteMsg PROC NEAR
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
WriteMsg ENDP

```

```

;-----
outputAL PROC
    ;call setCurs
    push ax
    push bx
    push cx
    mov ah, 09h    ;писать символ в текущей позиции курсора
    mov bh, 0      ;номер видео страницы
    mov cx, 1      ;число экземпляров символа для записи
    int 10h        ;выполнить функцию
    pop cx
    pop bx
    pop ax
    ret
outputAL ENDP
;-----

```

```

VERIFY_TAIL PROC near
    push ax
    push cx
    push es
    mov al, es:[81h+1]    ; es на psp
    cmp al, '/'
    jne error_tail

```

```

    mov al, es:[81h+2]
    cmp al, 'u'
    jne error_tail
    mov al, es:[81h+3]
    cmp al, 'n'
    jne error_tail
    inc bool_var
    jmp end_p
error_tail:
    mov bool_var, 0
end_p:
    pop es
    pop cx
    pop ax
    ret
VERIFY_TAIL ENDP

```

```

VERIFY_LOADING PROC
    push ax
    push bx
    push si
    push es
    mov AH, 35h
    mov AL, 09H;Номер прерывания
    int 21h
    mov si, offset SIGNATURE_INTER
    sub si, offset RESIDENT_INTER
    mov ax, es:[bx+si]
    cmp ax, SIGNATURE
    jne end_
    inc bool_resident
end_:
    pop es
    pop si
    pop bx
    pop ax
    ret
VERIFY_LOADING ENDP

```

```

DO_RESIDENT PROC
    push ax
    push bx
    push dx
    push es
    ;-----
    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 вектора прерывания

```

```

;-----установка прерывания
PUSH DS
MOV DX, OFFSET RESIDENT_INTER      ; смещение для
процедуры в DX
MOV AX, SEG RESIDENT_INTER          ; 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 LAST_BYTE
add DX, 10Fh                         ;100h + 15(Fh) при /16 в
большую сторону
mov CL, 4
shr DX, CL
inc DX
xor AX, AX
mov AH, 31h
int 21h
pop es
mov dx, offset message1
call WriteMsg
pop dx
pop bx
pop ax
ret
DO_RESIDENT ENDP

```

```

UNLOAD_RESIDENT PROC
    push AX
    push BX
    push DX
    push DS
    push ES
    mov AH, 35h
    mov AL, 09H
    int 21h
    mov dx, es:KEEP_IP
    mov ax, es:KEEP_CS
    CLI
    push DS
    mov DS, AX
    mov AH, 25h
    mov AL, 09H
    int 21h
    pop DS
    STI
    mov ax, es:SEGMENT_
    mov es, ax
    push ES
    mov AX, ES:[2Ch]

```

```

        mov ES, AX
        mov AH, 49h    ;функция освобождения памяти
        int 21h
        pop ES
        mov AH, 49h
        int 21h
        pop ES
        pop DS
        mov dx, offset message4
        call WriteMsg
        pop DX
        pop BX
        pop AX
        ret
UNLOAD_RESIDENT ENDP

```

```

MAIN PROC FAR
    push ds
    xor ax, ax
    push ax
    mov ax, data
    mov ds, ax
    mov SEGMENT_, es
    call VERIFY_TAIL
    call VERIFY_LOADING
    cmp bool_var, 0
if_:je else_unload
    cmp bool_resident, 0
    je not_load
    call UNLOAD_RESIDENT
    jmp end_main
not_load:
    mov dx, offset message2
    call WriteMsg
    jmp end_main
else_unload:
    cmp bool_resident, 0
    ja already
    call DO_RESIDENT
    jmp end_main
already:
    mov dx, offset message3
    call WriteMsg
    jmp end_main
end_main:
    xor AL, AL
    mov AH, 4Ch
    int 21h
MAIN ENDP

```

code ENDS

```

data segment
    bool_resident db 0
    bool_var db 0
    message1 db 'INTERRUPT LOAD!',10,13,'$'
    message2 db 'NOT LOAD$'
    message3 db 'INTERRUPT ALREADY LOAD!$'
    message4 db 'INTERRUPT UNLOAD!',10,13,'$'
    SIGNATURE dw 777h
data ends

astack segment stack
dw 128 dup(?)           ;: для исключения возможного взаимного влияния
системных и пользовательских
astack ends             ; прерываний рекомендуется отвести в
программе под стек не менее 1K байт.

END MAIN

```