

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

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

Студент гр. 9383

Арутюнян С.Н.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Цель работы

Исследовать возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Написать пользовательский обработчик прерывания, который получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре и обрабатывает скан-код, осуществляя 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.

Выполнение работы

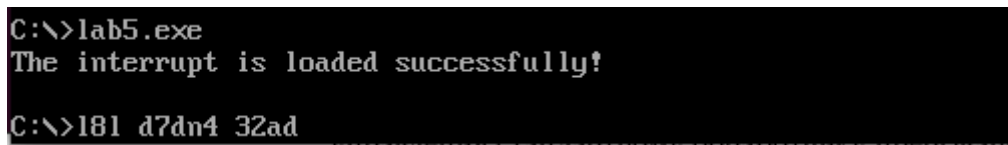
Шаг 1. Был написан и отлажен программный модуль типа .EXE, который выполняет следующие функции:

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

2. Устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и осуществляет выход по функции 4Ch прерывания int 21h.

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

4. Выгрузка прерывания по соответствующему значению параметра в командной строке \up. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождения памяти, занимаемой резидентом. Затем осуществляется выход по функции 4ch прерывания int 21h.



```
C:\>lab5.exe
The interrupt is loaded successfully!
C:\>181 d7dn4 32ad
```

Рис. 1. Работа установленного прерывания

```
C:\>LAB5.EXE
The interrupt is already loaded!
C:\>
```

Рис. 2. Пример обработки попытки повторной установки прерывания

Шаг 2. Отлаженная программа была запущена и я убедился, что резидентный обработчик прерывания 09h установлен, т. к. буквы от q до р начали заменяться на цифры:

```
C:\>LAB3_1.COM
Available memory size: 644288 bytes
Extended memory size: 246720 bytes
MCB #1: Address: 016F PSP address: 0008 Size: 16 SC/SD:
MCB #2: Address: 0171 PSP address: 0000 Size: 64 SC/SD:
MCB #3: Address: 0176 PSP address: 0040 Size: 256 SC/SD:
MCB #4: Address: 0187 PSP address: 0192 Size: 144 SC/SD:
MCB #5: Address: 0191 PSP address: 0192 Size: 4448 SC/SD: LAB5
MCB #6: Address: 02A8 PSP address: 02B3 Size: 1448 SC/SD:
MCB #7: Address: 02B2 PSP address: 02B3 Size: 644288 SC/SD: LAB3_1
C:\>
```

Рис. 4. Пример работы программы из лабораторной работы №3

Контрольные вопросы

1. Какого типа прерывания использовались в работе?

В работе использовались программные (21h и 10h, т. е. выход в DOS и получение информации о курсоре) и аппаратные прерывания (09h, т. е. прерывание клавиатуры).

2. Чем отличается скан код от кода ASCII?

Скан-код — код клавиши конкретной клавиатуры, подключенной к компьютеру. ASCII-код — общий для всех компьютеров код конкретного символа. Он нужен для стандартизации работы с символами на всех устройствах.

Заключение

В процессе выполнения лабораторной работы был изучен механизм взаимодействия клавиатуры с ОС DOS, изучен принцип работы скан-кодов, а также были получены навыки реализации собственных резидентных прерываний .

Приложение А

AStack SEGMENT STACK

dw 256 DUP(?) ; 1 килобайт

AStack ENDS

DATA SEGMENT

NEWLINE db 0dh, 0ah, '\$'

INT_ALREADY_LOADED db "The interrupt is already loaded!", 0dh, 0ah, '\$'

INT_IS_NOT_LOADED db "There is no our loaded interrupt!", 0dh, 0ah, '\$'

INT_LOADED_SUCCESS db "The interrupt is loaded successfully!", 0dh, 0ah, '\$'

INT_IS_RESTORED_SUCCECCFULLY db "The interrupt is restored successfully!", 0dh,
0ah, '\$'

DATA ENDS

CODE SEGMENT

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

PRINT_NEWLINE proc near

push ax

push dx

mov dx, offset NEWLINE

mov ah, 9h

int 21h

pop dx

pop ax

ret

PRINT_NEWLINE endp

WRITE_STRING proc near

push ax

mov ah, 9h

int 21h

pop ax

ret

WRITE_STRING endp

MY_INTERRUPT proc far

jmp start_interrupt

WITH_SHIFT db 0

CALL_STANDARD_INTERRUPT db 0

INTERRUPT_SIGN dw 7777h

INTERRUPT_KEEP_IP dw 0

INTERRUPT_KEEP_CS dw 0

INTERRUPT_PSP_ADDRESS dw 0

INTERRUPT_KEEP_SS dw 0

INTERRUPT_KEEP_SP dw 0

```
INTERRUPT_KEEP_AX    dw 0
INTERRUPT_MAIN_HANDLER dw 0
REQUIRED_KEY         db 0
SOME_STRING          db "lol what?", 0dh, 0ah, '$'
INTERRUPT_STACK dw 64 dup(?)
```

start_interrupt:

```
mov INTERRUPT_KEEP_SP, sp
mov INTERRUPT_KEEP_AX, ax
mov ax, ss
mov INTERRUPT_KEEP_SS, ax
```

```
mov sp, offset start_interrupt
mov ax, seg INTERRUPT_STACK
mov ss, ax
```

```
mov ax, INTERRUPT_KEEP_AX
```

```
push ax
push es
push dx
push cx
```

```
mov CALL_STANDARD_INTERRUPT, 1
mov WITH_SHIFT, 0
```

interrupt_handling:

```
mov ax, 40h
mov es, ax
mov ax, es:[17h]
```


; если зажат правый шифт

mov dx, ax

and dx, 1

cmp dx, 1

je write_with_shift

; если зажат левый шифт

mov dx, ax

and dx, 10b

cmp dx, 10b

jne process_required_key

write_with_shift:

mov WITH_SHIFT, 1

process_required_key:

in al, 60h

cmp al, 10h ; клавиша q

jl exit_interrupt

cmp al, 19h ; клавиша р

jg exit_interrupt

; переводим скан-код в цифру

add al, 20h

; данную клавишу обрабатываем мы, поэтому вызывать старый обработчик не нужно

mov CALL_STANDARD_INTERRUPT, 0

do_req:

push ax

; берем инфу из служебного порта клавиатуры

```
in al, 61h
mov ah, al
; устанавливаем бит разрешения
or al, 80h
; и кидаем его в служебный порт
out 61h, al
```

```
xchg ah, al
; записываем обратно исходное значение порта
out 61h, al
; посылаем сигнал конца прерывания ЧТО ЕСЛИ ЗАКОММЕНТИТЬ ЭТУ
```

ЧАСТЬ??????

```
mov al, 20h
out 20h, al
```

```
pop ax
```

```
; на этом этапе у нас в al лежит текущая буква в кодировке ASCII
; sub al, 65
```

print_entered_key:

```
mov ah, 5h
mov cl, al
mov ch, 0h
int 16h
```

exit_interrupt:

```
pop cx
pop dx
pop es
pop ax
```

```
mov sp, INTERRUPT_KEEP_SP
mov ax, INTERRUPT_KEEP_SS
mov ss, ax
mov ax, INTERRUPT_KEEP_AX
```

```
; посылаем сигнал "конец прерывания"
mov al, 20h
out 20h, al
```

```
; если нужно обработать не нашу клавишу, вызываем
cmp CALL_STANDARD_INTERRUPT, 1
jne interrupt_iret_exit
```

```
call_standard:
    jmp dword ptr cs:[INTERRUPT_KEEP_IP]
```

```
interrupt_iret_exit:
    iret
```

```
INTERRUPT_SIZE:
MY_INTERRUPT endp
```

```
CHECK_ALREADY_LOADED proc near
```

```
    push ax
    push dx
    push es
    push si
```

```
    mov cl, 0ah
    mov ah, 35h
```

```
mov al, 09h
```

```
int 21h
```

```
mov cl, 0
```

```
mov si, offset INTERRUPT_SIGN
```

```
sub si, offset MY_INTERRUPT
```

```
mov dx, es:[bx + si]
```

```
cmp dx, INTERRUPT_SIGN
```

```
jne check_loaded_exit
```

```
mov cl, 1h
```

```
check_loaded_exit:
```

```
pop si
```

```
pop es
```

```
pop dx
```

```
pop ax
```

```
ret
```

```
CHECK_ALREADY_LOADED endp
```

```
LOAD_INTERRUPT proc near
```

```
push ax
```

```
push es
```

```
push bx
```

```
push dx
```

```
; сначала проверяем, установлено ли прерывание в 09h
```

```
call CHECK_ALREADY_LOADED
```

```
cmp cl, 1
```

je int_already_exists

; сохраняем информацию об изначальном прерывании

mov INTERRUPT_PSP_ADDRESS, es

mov ah, 35h

mov al, 09h

int 21h

mov INTERRUPT_KEEP_CS, es

mov INTERRUPT_KEEP_IP, bx

; загружаем прерывание

push ds

mov dx, offset MY_INTERRUPT

mov ax, seg MY_INTERRUPT

mov ds, ax

mov ah, 25h

mov al, 09h

int 21h

pop ds

; оповещаем о том, что все ок

mov dx, offset INT_LOADED_SUCCESS

call WRITE_STRING

; остаемся резидентными

mov dx, offset INTERRUPT_SIZE ; LEA DX, INTERRUPT_SIZE?

mov cl, 4

shr dx, cl

inc dx

add dx, 100h

```
xor ax, ax
mov ah, 31h
int 21h
```

```
jmp load_int_exit
```

```
int_already_exists:
    mov dx, offset INT_ALREADY_LOADED
    call WRITE_STRING
```

```
load_int_exit:
```

```
    pop dx
    pop bx
    pop es
    pop ax
```

```
ret
```

```
LOAD_INTERRUPT endp
```

```
UNLOAD_INTERRUPT proc near
```

```
    push ax
    push si
    push dx
```

```
    ; проверяем, установлено ли прерывание
    call CHECK_ALREADY_LOADED
    cmp cl, 0
```

je interrupt_is_not_loaded

; отключаем прерывания

cli

push ds

push es

; достаем адрес текущего загруженного прерывания

mov ah, 35h

mov al, 09h

int 21h

; достаем из загруженного прерывания инфу о предыдущем прерывании

mov si, offset INTERRUPT_KEEP_IP

sub si, offset MY_INTERRUPT

mov dx, es:[bx + si]

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

mov ds, ax

; заменяем текущее прерывание тем прерыванием, которое он заменил

mov ah, 25h

mov al, 09h

int 21h

mov ax, es:[bx + si + 4]

mov es, ax

push es

mov ax, es:[2ch]

mov es, ax

mov ah, 49h

int 21h

pop es

mov ah, 49h

int 21h

pop es

pop ds

; включаем прерывания обратно

sti

mov dx, offset INT_IS_RESTORED_SUCCECCFULLY

call WRITE_STRING

jmp unload_int_exit

interrupt_is_not_loaded:

mov dx, offset INT_IS_NOT_LOADED

call WRITE_STRING

unload_int_exit:

pop dx

pop si

pop ax

ret

UNLOAD_INTERRUPT endp

CHECK_INPUT proc near

push ax

push bx

push es

mov ah, 62h

int 21h

; в bx - адрес начала PSP

mov es, bx

mov al, es:[80h]

cmp al, 4

; если количество символов != 3, то нужно загрузить прерывание

jne interrupt_set_label

; иначе, если было передано \up, то выгружаем

mov al, es:[82h]

cmp al, '\'

jne interrupt_set_label

mov al, es:[83h]

cmp al, 'u'

jne interrupt_set_label

mov al, es:[84h]

cmp al, 'n'

jne interrupt_set_label

call UNLOAD_INTERRUPT

jmp check_input_exit

interrupt_set_label:

call LOAD_INTERRUPT

check_input_exit:

pop es

pop bx

pop ax

ret

CHECK_INPUT endp

Main proc far

mov ax, DATA

mov ds, ax

; проверка аргументов командной строки

call CHECK_INPUT

; выход в DOS

xor al, al

mov ah, 4ch

int 21h

Main endp

CODE ENDS

END Main