

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. 7381

Адамов Я.В.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2019

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.

Описание функций.

Название функции	Описание
PrintMsg	Печать строки, адрес которой помещен в DX.
ROUT	Обработчик прерываний, выводящий вместо символа тильда '~' смайлик '☺'.
CHECKING	Проверка, загружен ли обработчик прерываний.
SET_INTERRUPT	Установка нового обработчик прерывания с запоминанием данных для восстановления предыдущего обработчика прерываний.
DELETE_INTERRUPT	Удаление пользовательского прерывания, восстановление прерывание по умолчанию.

Описание структур данных.

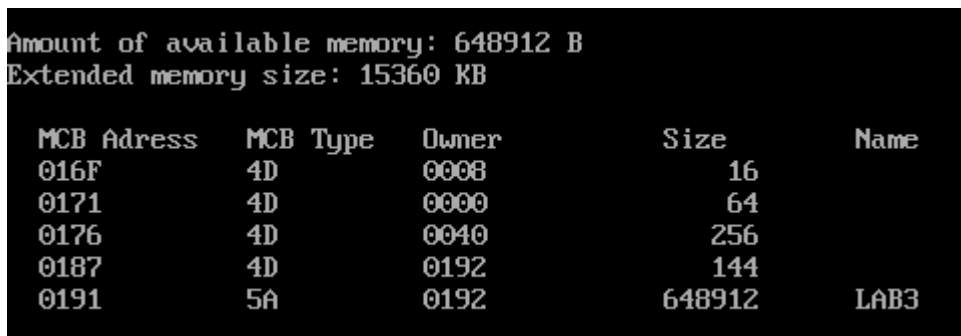
Название	Тип	Описание
wasloaded	db	Строка: «Interruption had already been loaded.».

unloaded	db	Строка: «Interrupt is restored.».
loading	db	Строка: «Interrupt is loaded.».

Описание работы утилиты.

Программа проверяет, установлено ли пользовательское прерывание с вектором 09h. Устанавливает обработчик прерываний, если он не установлен, в ином случае выводится соответствующее сообщение. Программа выгружает прерывания по соответствующему значению параметра в командной строке /un, восстановления стандартного вектора прерывания.

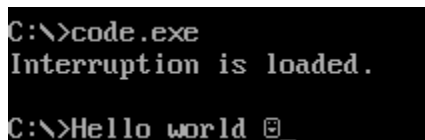
Состояние памяти до запуска программы получено с помощью утилиты, написанной в третьей лабораторной работе, и представлено на рис. 1. Демонстрация работы программы представлена на рис. 2. Состояние памяти после запуска программы представлено на рис. 3. Результат повторного запуска программы представлен на рис. 4. Запуск программы с ключом выгрузки /un представлен на рис. 5. Состояние памяти после выгрузки представлено на рис. 6.



Amount of available memory: 648912 B
Extended memory size: 15360 KB

MCB Address	MCB Type	Owner	Size	Name
016F	4D	0008	16	
0171	4D	0000	64	
0176	4D	0040	256	
0187	4D	0192	144	
0191	5A	0192	648912	LAB3

Рисунок 1 – состояние памяти до запуска программы.



```
C:\>code.exe
Interrupt is loaded.
C:\>Hello world
```

Рисунок 2 – демонстрация работы программы.

Amount of available memory: 647360 B
Extended memory size: 15360 KB

MCB Address	MCB Type	Owner	Size	Name
016F	4D	0008	16	
0171	4D	0000	64	
0176	4D	0040	256	
0187	4D	0192	144	
0191	4D	0192	1376	CODE
01E8	4D	01F3	1144	
01F2	5A	01F3	647360	LAB3

Рисунок 3 – состояние памяти после запуска программы.

```
C:\>CODE.EXE
Interruption had already been loaded.
```

Рисунок 4 – повторный запуск программы.

```
C:\>CODE.EXE /un
Interruption is restored.
```

Рисунок 5 – запуск программы с ключом выгрузки /un.

Amount of available memory: 648912 B
Extended memory size: 15360 KB

MCB Address	MCB Type	Owner	Size	Name
016F	4D	0008	16	
0171	4D	0000	64	
0176	4D	0040	256	
0187	4D	0192	144	
0191	5A	0192	648912	LAB3

Рисунок 6 – состояние памяти после выгрузки.

Вывод.

В ходе выполнения лабораторной работы была исследована возможность встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

Ответы на контрольные вопросы.

Какого типа прерывания использовались в работе?

Аппаратные (1Ch) и программные (21h) прерывания.

Чем отличается скан-код от кода ASCII?

Код ASCII – код символа, скан-код – код клавиши (одна клавиша может генерировать несколько различных кодов в сочетании с другими клавишами, такими как *ctrl*, *alt*, *shift* и т.д.).

Приложение A. lab3.asm.

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

ORG 100H

START: JMP BEGIN

; _____

; Данные

AvailableMemory db 0dh,0ah,'Amount of available memory:
B',0dh,0ah,'\$'

ExtendedMemorySize db 'Extended memory size: KB',0dh,0ah,'\$'

TableHead db 0dh,0ah,' MCB Adress MCB Type Owner Size
Name ',0dh,0ah,'\$'

MCB 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
    push cx
    mov ah,al
    call TETR_TO_HEX
    xchg al,ah
    mov cl,4
    shr al,cl
    call TETR_TO_HEX
    pop cx
    ret
BYTE_TO_HEX      ENDP
```

```
WRD_TO_HEX PROC near
    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
    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_l
    or al,30h
    mov [si],al
end_l: pop dx
    pop cx
    ret
BYTE_TO_DEC ENDP

```


WRD_TO_DEC PROC near

```
    push cx
    push dx
    push ax
    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 ax,00h
    jbe end_1_2
    or al,30h
    mov [si],al
```

end_1_2:

```
    pop ax
    pop dx
    pop cx
    ret
```

WRD_TO_DEC ENDP

PrintMsg PROC near

```
    push ax
    mov ah,09h
    int 21h
```

```
    pop ax
    ret
PrintMsg ENDP
```

```
PrintAvailableMemory PROC near
```

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

```
    mov ah,04Ah
    mov bx,0FFFFh
    int 21h
    mov ax,10h
    mul bx
    lea si,AvailableMemory
    add si,35
    call WRD_TO_DEC
    lea dx,AvailableMemory
    call PrintMsg
```

```
    pop si
    pop dx
    pop bx
    pop ax
    ret
```

```
PrintAvailableMemory ENDP
```

PrintExtendedMemorySize PROC near

push ax

push bx

push dx

push si

mov al,30h

out 70h,al

in al,71h

mov bl,al

mov al,31h

out 70h,al

in al,71h

mov ah,al

mov al,bl

sub dx,dx

lea si,ExtendedMemorySize

add si,26

call WRD_TO_DEC

lea dx,ExtendedMemorySize

call PrintMsg

pop si

pop dx

pop bx

pop ax

ret

PrintExtendedMemorySize ENDP

PrintMCB PROC near

; Address

lea di,MCB

mov ax,es

add di,5

call WRD_TO_HEX

; Type

lea di,MCB

add di,15

xor ah,ah

mov al,es:[0]

call BYTE_TO_HEX

mov [di],al

inc di

mov [di],ah

; Owner

lea di,MCB

mov ax,es:[1]

add di,29

call WRD_TO_HEX

; Size

```

    lea di,MCB
    mov ax,es:[3]
    mov bx,10h
    mul bx
    add di,46
    push si
    mov si,di
    call WRD_TO_DEC
    pop si

```

```

; Name
lea di,MCB
add di,53
mov bx,0

```

```

print_:
    mov dl,es:[bx+8]
    mov [di],dl
    inc di
    inc bx
    cmp bx,8
    jne print_
    mov ax,es:[3]
    mov bl,es:[0]
    ret

```

```

PrintMCB ENDP

```

```

PrintMemoryManagementUnits PROC near

```

```

        lea dx,TableHead
        call PrintMsg
        mov ah,52h
        int 21h
        sub bx,2h
        mov es,es:[bx]
metka_1:
        call PrintMCB
        lea dx,MCB
        call PrintMsg
        mov cx,es
        add ax,cx
        inc ax
        mov es,ax
        cmp bl,4Dh
        je metka_1
        ret
PrintMemoryManagementUnits ENDP

```

; _____

; Код

```

BEGIN:
        call PrintAvailableMemory
        call PrintExtendedMemorySize
        call PrintMemoryManagementUnits

```

```
xor al,al  
mov ah,4ch  
int 21h
```

```
TESTPC ENDS  
END START
```

Приложение Б. code.asm.

```
A AStack SEGMENT STACK
```

```
    DW 100h DUP(?)
```

```
AStack ENDS
```

```
; _____
```

```
DATA SEGMENT
```

```
wasloaded db 'Interruption had already been loaded.',0DH,0AH,'$'
```

```
unloaded db 'Interruption is restored.',0DH,0AH,'$'
```

```
loading db 'Interruption is loaded.',0DH,0AH,'$'
```

```
DATA ENDS
```

```
; _____
```

```
CODE SEGMENT
```

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

```
PrintMsg PROC near
```

```
    push ax
```

```
    mov ah,09h
```

```
    int 21h
```

```
    pop ax
```

```
    ret
```

```
PrintMsg ENDP
```



```

ROUT PROC far
    jmp ROUT_
_DATA:
    STACK_ dw 64 DUP (?)
    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
    _TILD db 29h

```

```

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

    mov ax,0040h
    mov es,ax
    mov al,es:[17h]

```

```
cmp al,00000010b
```

```
jnz NEXT
```

```
in al,60H
```

```
cmp al,_TILD
```

```
je TILD
```

NEXT:

```
pop es
```

```
pop ds
```

```
pop dx
```

```
mov ax,CS:KEEP_AX
```

```
mov sp,CS:KEEP_SP
```

```
mov ss,CS:KEEP_SS
```

```
jmp dword ptr cs:[KEEP_IP]
```

TILD:

```
mov cl,01h
```

```
jmp DO_REQ
```

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
```

ADDSYMB:

```
mov ah,05h
mov ch,00h
int 16h
or al,al
jz ROUT_END
cli
mov ax,es:[1Ah]
mov es:[1Ch],ax
sti
jmp ADDSYMB
```

ROUT_END:

```
pop es
pop ds
pop dx
pop ax
mov ax,KEEP_SS
mov ss,ax
mov sp,KEEP_SP
mov ax,KEEP_AX
iret
```

ROUT ENDP

CHECKING PROC

```
mov ah,35h
mov al,09h
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_INTERRUPT
mov dx,offset LAST_BYTE
mov cl,4
shr dx,cl
inc dx
add dx,CODE
sub dx,CS: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 BACK
cmp byte ptr es:[83h], 'u'
jne BACK
cmp byte ptr es:[84h], 'n'
je UNLOAD_
```

BACK:

```
pop ax
pop es
mov dx,offset wasloaded
call PrintMsg
ret
```

UNLOAD_:

```
pop ax
pop es
call DELETE_INTERRUPT
mov dx,offset unloaded
call PrintMsg
ret
```

CHECKING ENDP

```

SET_INTERRUPT PROC
    push dx
    push ds

    mov ah,35h
    mov al,09h
    int 21h
    mov CS:KEEP_IP,bx
    mov CS:KEEP_CS,es

    mov dx,offset ROUT
    mov ax,seg ROUT
    mov ds,ax
    mov ah,25h
    mov al,09h
    int 21h

    pop ds
    mov dx,offset loading
    call PrintMsg
    pop dx
    ret
SET_INTERRUPT ENDP

```

```

DELETE_INTERRUPT PROC

```

```
push ds
cli
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
sti
pop ds
ret
```

DELETE_INTERRUPT ENDP

BEGIN:

```
mov ax,DATA
mov ds,ax
```

```
mov KEEP_PSP,es  
call CHECKING  
xor al,al  
mov ah,4Ch  
int 21h
```

LAST_BYTE:

CODE ENDS

END BEGIN