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

ЛАБОРАТОРНАЯ РАБОТА № 5

по дисциплине «Операционные системы»

ТЕМА: Сопряжение стандартного и пользовательского обработчиков прерываний.

Студент гр. 9381 _____ Николаев А.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

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.

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет такие же функции, как в программе ЛР 4, а именно:

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. Адрес точки входа в стандартный обработчик прерывания находится в теле пользовательского обработчика. Осуществляется выход по функции 4Ch прерывания int 21h.

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

Шаг 2. Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания 09h установлен. Работа прерывания проверяется введением различных символов, обрабатываемых установленным обработчиком и стандартным обработчиком.

Шаг 3. Также необходимо проверить размещение прерывания в памяти. Для этого запустите программу ЛР 3, которая отображает карту памяти в виде списка блоков МСВ. Полученные результаты поместите в отчет.

Шаг 4. Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

Шаг 5. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Шаг 6. Ответьте на контрольные вопросы.

Выполнение работы.

Был написан и отлажен программный модуль типа .EXE, который выполняет такие же функции, как в программе ЛР 4, а именно:

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. Адрес точки входа в стандартный обработчик прерывания находится в теле пользовательского обработчика. Осуществляется выход по функции 4Ch прерывания int 21h.

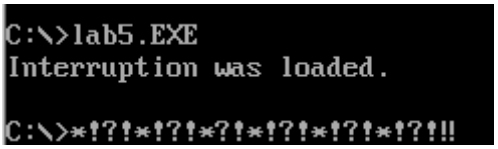
3) Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.

Программа заменяет следующие символы, вводимые с клавиатуры g, k, c на *, !, ? соответственно.

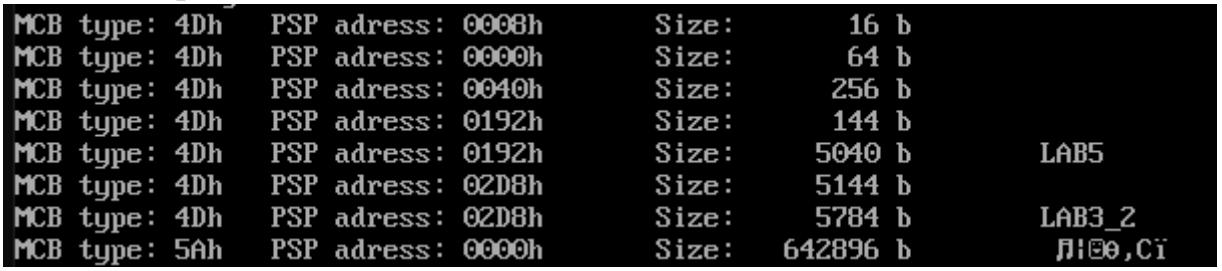
Была запущена lab3_1 для того, чтобы удостоверится, что прерывание не загружено.

```
C:\>lab3_1.COM
Amount of available memory:     648912 b
Size of extended memory:      15360 Kb
List of memory control blocks:
MCB type: 4Dh   PSP adress: 0008h       Size:       16 b
MCB type: 4Dh   PSP adress: 0000h       Size:       64 b
MCB type: 4Dh   PSP adress: 0040h       Size:      256 b
MCB type: 4Dh   PSP adress: 0192h       Size:      144 b
MCB type: 5Ah   PSP adress: 0192h       Size:   648912 b
```

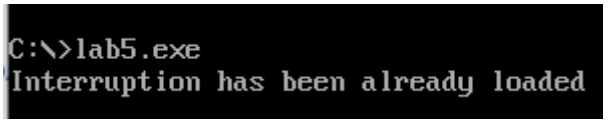
Была запущена программа и проверена ее работа, путем нажатия клавиш g, k, c.



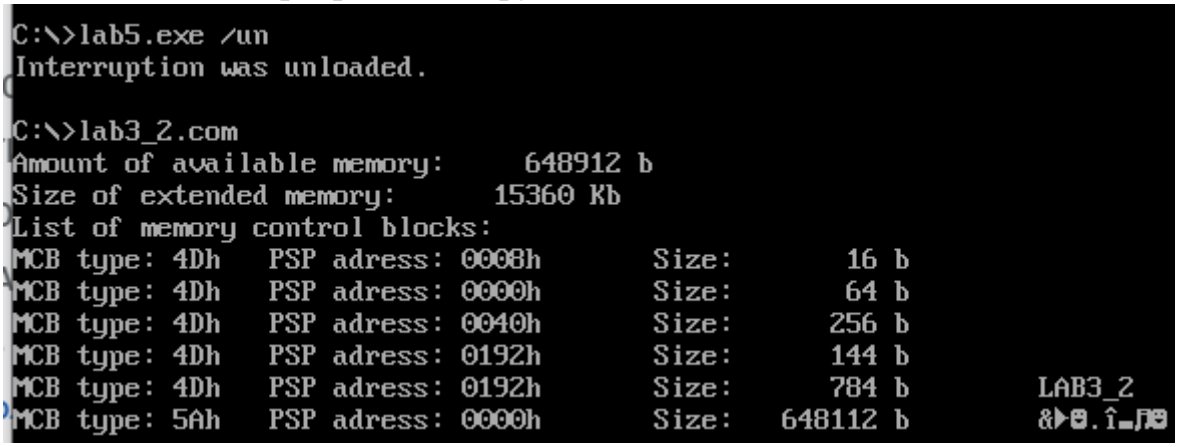
Был запущен lab3_2 для проверки размещения прерывания в памяти. Мы можем убедиться, что резидент находится в памяти.



Если запустить программу повторно, программа выдаст сообщение, что обработчик прерываний уже загружен.



Запустим программу с параметром /un и убедимся, что обработчик прерывания выгружен. Нажатия клавиш должны обрабатываться стандартно и программа lab3_2 должна показать, что программа выгружена из памяти.



Ответы на контрольные вопросы:

1. Какого типа прерывания использовались в работе?

DOS (int 21h)

BIOS (16h, 09h) .

2. Чем отличается скан код от кода ASCII?

С помощью скан-кода драйвер клавиатуры распознает, какая именно клавиша была нажата.

ASCII код – это код из таблицы всех имеющихся символов, которые могут быть выведены в консоль.

Выводы.

В процессе выполнения данной работы производилось исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Было написано пользовательское прерывание от клавиатуры, которое анализирует скан-коды, выполняет вывод сообщения результата нажатия, а при несовпадении скан-кода передает управление стандартному обработчику.

ПРИЛОЖЕНИЕ А
КОД lab5.asm

```
LB5 segment
assume  cs:LB5, ds:DATA, ss:MY_STACK

MY_STACK  segment stack
 dw  256 dup(0)
MY_STACK  ends

INTER PROC FAR
    jmp  Start

intData:
    key_value db 0
    new_stack dw 256 dup(0)
    signature dw 6666h
    keep_ip dw 0
    keep_cs dw 0
    keep_psp dw 0
    keep_ax dw 0
    keep_ss dw 0
    keep_sp dw 0

Start:
    mov keep_ax, ax
    mov keep_sp, sp
    mov keep_ss, ss
    mov ax, seg new_stack
    mov ss, ax
    mov ax, offset new_stack
    add ax, 256
    mov sp, ax

    push ax
    push bx
    push cx
    push dx
    push si
    push es
    push ds
    mov ax, seg key_value
    mov ds, ax

    in al, 60h
```

```
    cmp al, 22h  ;g
    je val_g
    cmp al, 25h  ;k
    je val_k
    cmp al, 2Eh   ;c
    je val_c

    pushf
    call dword ptr cs:keep_ip
    jmp inter_end

val_g:
    mov key_value, '*'
    jmp next
val_k:
    mov key_value, '!'
    jmp next
val_c:
    mov key_value, '?'

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

print_key:
    mov ah, 05h
    mov cl, key_value
    mov ch, 00h
    int 16h
    or   al, al
    jz   inter_end
    mov ax, 40h
    mov es, ax
    mov ax, es:[1ah]
    mov es:[1ch], ax
    jmp print_key


inter_end:
    pop  ds
```

```
    pop  es
    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
INTER endp
 _end:


is_int_loaded proc
    push ax
    push bx
    push si

    mov  ah, 35h
    mov  al, 09h
    int  21h
    mov  si, offset signature
    sub  si, offset INTER
    mov  ax, es:[bx + si]
    cmp   ax, signature
    jne  end_proc
    mov  is_load, 1

end_proc:
    pop  si
    pop  bx
    pop  ax
    ret
    is_int_loaded endp

load_int  proc
    push ax
    push bx
    push cx
    push dx
```

```
    push es
    push ds

    mov ah, 35h
    mov al, 09h
    int 21h
    mov keep_cs, es
    mov keep_ip, bx
    mov ax, seg INTER
    mov dx, offset INTER
    mov ds, ax
    mov ah, 25h
    mov al, 09h
    int 21h
    pop ds

    mov dx, offset _end
    mov cl, 4h
    shr dx, cl
    add  dx, 10fh
    inc dx
    xor ax, ax
    mov ah, 31h
    int 21h

    pop es
    pop dx
    pop cx
    pop bx
    pop ax
ret
load_int  endp


unload_int proc
    cli
    push ax
    push bx
    push dx
    push ds
    push es
    push si

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

```
    mov si, offset keep_ip
    sub si, offset INTER
    mov dx, es:[bx + si]
    mov ax, es:[bx + si + 2]

    push ds
    mov ds, ax
    mov ah, 25h
    mov al, 09h
    int 21h
    pop ds

    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

    sti

    pop si
    pop es
    pop ds
    pop dx
    pop bx
    pop ax

ret
unload_int endp


is_unload_  proc
    push ax
    push es

    mov ax, keep_psp
    mov es, ax
    cmp byte ptr es:[82h], '/'
    jne end_unload
    cmp byte ptr es:[83h], 'u'
    jne end_unload
```

```
    cmp byte ptr es:[84h], 'n'
    jne end_unload
    mov is_un, 1

end_unload:
    pop es
    pop ax
 ret
is_unload_ endp


PRINT proc near
    push ax
    mov ah, 09h
    int 21h
    pop ax
ret
PRINT endp


begin proc
    push ds
    xor ax, ax
    push ax
    mov ax, DATA
    mov ds, ax
    mov keep_psp, es

    call is_int_loaded
    call is_unload_
    cmp is_un, 1
    je unload
    mov al, is_load
    cmp al, 1
    jne load
    mov dx, offset str_loaded
    call PRINT
    jmp end_begin

load:
    mov dx, offset str_load
    call PRINT
    call load_int
    jmp  end_begin

unload:
```

```
    cmp  is_load, 1
    jne  not_loaded
    mov dx, offset str_unload
    call PRINT
    call unload_int
    jmp  end_begin

not_loaded:
    mov  dx, offset str_not_loaded
    call PRINT

end_begin:
    xor al, al
    mov ah, 4ch
    int 21h
begin endp
LB5 ends

DATA segment
    IS_LOAD         db
0
    IS_UN           db
0
    STR_LOAD        db      "Interruption was loaded.",
0dh, 0ah, "$"
    STR_LOADED      db      "Interruption has been already
loaded",      0dh, 0ah, "$"
    STR_UNLOAD      db      "Interruption was unloaded.",
0dh, 0ah, "$"
    STR_NOT_LOADED  db      "Interruption is not loaded.",
0dh, 0ah, "$"
DATA ends
end begin
```