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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Операционные системы»

Тема: Обработка стандартных прерываний

Студент гр. 0382 _____ Крючков А.М.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2022

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.

Постановка задачи.

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.

Выгрузка прерывания по соответствующему значению параметра в командной строке /un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождении памяти, занимаемой резидентом. Затем осуществляется выход по функции 4Ch прерывания int 21h.

Для того чтобы проверить установку прерывания, можно поступить следующим образом. Прочитать адрес, записанный в векторе прерывания. Предположим, что этот адрес указывает на точку входа в установленный резидент. На определенном, известном смещении в теле резидента располагается сигнатура, некоторый код, который идентифицирует резидент.

Сравнив известное значение сигнатуры с реальным кодом, находящимся в резиденте, можно определить, установлен ли резидент. Если значения совпадают, то резидент установлен. Длину кода сигнатуры должна быть достаточной, чтобы сделать случайное совпадение маловероятным.

Программа должна содержать код устанавливаемого прерывания в виде удаленной процедуры. Этот код будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

1) Сохранить значения регистров в стеке при входе и восстановить их при выходе.

2) При выполнении тела процедуры анализируется скан-код.

3) Если этот код совпадает с одним из заданных, то требуемый код записывается в буфер клавиатуры.

4) Если этот код не совпадает ни с одним из заданных, то осуществляется передача управления стандартному обработчику прерывания.

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.

Исходный код модуля представлен в приложении А.

Шаг 1.

Был написан программный модуль типа .EXE. Для проверки наличия установки пользовательского прерывания с вектором 1ch была написана процедура is_int_installed. Если прерывание не установлено, то прерывание устанавливается при помощи функции install_int. Для этого сохраняется стэк в специально выделенное место памяти. Функция прерывания была помещена в самое начало программы, а сегмент данных и стэка после сегмента кода. Это было сделано для того, чтобы в памяти оставалась только функция прерывания. Если прерывание уже установлено, то выводится соответствующее сообщение, а программа завершается. Если в командную строку было передано сообщение о выгрузке прерывания, то при помощи помощи процедуры check_console проверяется наличие этого сообщения и при помощи процедуры uninstall_int прерывание выгружается.

Прерывание организует свой стек и анализирует скан-код. В случае нажатия left shift выводится s, left ctrl — c, z — O. В остальных случаях отрабатывает стандартное прерывание.

Шаг 2. Результат работы программы. Былли нажаты следующие клавиши:left shift, left ctrl, z, a, b, c, ctrl, shift.

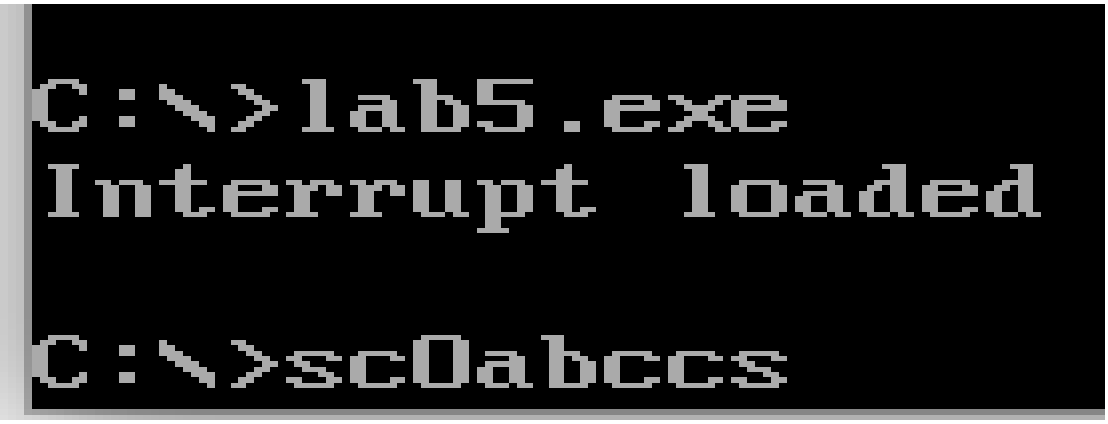

Рисунок 1 – Результат работы программы lab5.exe

Шаг 3. Был запущен код 3 лабораторной, чтобы показать размещение в памяти нашего прерывания.

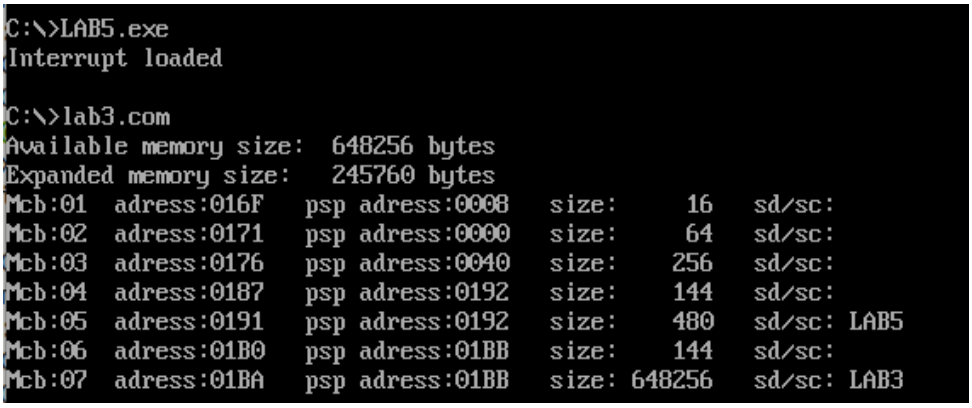

Рисунок 2 – память в виде списка блоков MCB

Шаг 4. Отлаженная программа была запущена еще раз. Можно убедиться, что программа определяет установленный обработчик прерываний.

Рисунок 3 – повторный запуск lab5.exe

Шаг 5. При запуске программы с ключом \un выводится сообщение об выгрузке прерывания. После запуска lab3.com можно увидеть что ни один блок памяти больше не содержит в себе lab5.

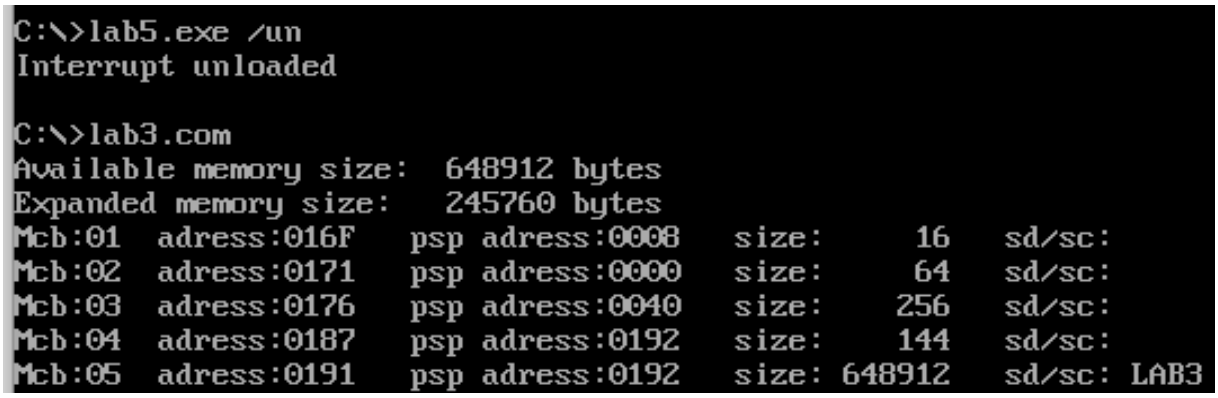

Рисунок 4 – запуск lab5.exe с ключом \un и запуск lab3.com

Контрольные вопросы.

1. Какого типа прерывания использовались в работе?

Программные - 21h и аппаратные 09h и 16h.

2. Чем отличается скан-код от кода ASCII?

Скан код – это код, с помощью которого драйвер клавиатуры опознаёт нажатую клавишу. Код ASCII – это код каждого символа в таблице ASCII. Не для всех нажатых клавиш существует код в таблице ASCII.

Выводы.

В процессе выполнения данной лабораторной работы был изучен механизм работы прерывания 09h и считывания введённых клавиш. Проведена работа по созданию резидентного пользовательского обработчика прерывания, заменяющего действия, производимые при нажатии клавиш left shift, left ctrl, z.

ПРИЛОЖЕНИЕ А.

Исходный код

lab5.asm:

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

interrupt proc far
jmp start
psp dw 0
keep_ip dw 0
keep_cs dw 0
keep_ss dw 0
keep_sp dw 0
keep_ax dw 0
key_sym db 0
int_id dw 07777
int_stack db 50 dup(" ")
start:

mov keep_ax, ax
mov ax, ss
mov keep_ss, ax
mov keep_sp, sp
mov ax, seg int_stack
mov ss, ax
mov sp, offset start

push ax
push bx
push cx
push dx

in al, 60h
cmp al, 2ah ;shift
je shift
cmp al, 1dh;ctrl
je ctrl
cmp al, 2ch; z
je z
call dword ptr cs:keep_ip
jmp exit_int

shift:
mov key_sym, 's'
jmp next_key
ctrl:
mov key_sym, 'c'
jmp next_key
z:
mov key_sym, 'O';suzdaem

next_key:
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_sym
mov ch, 00h
int 16h
or al, al
jz exit_int
mov ax, 40h
mov es, ax
mov ax, es:[1ah]
mov es:[1ch], ax
jmp print_key

exit_int:
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

last_byte:

interrupt endp

is_int_installed proc near
push ax
push bx
push dx
push si

mov int_installed, 1
mov ah, 35h
mov al, 09h
int 21h
mov si, offset int_id
sub si, offset interrupt
mov dx, es:[bx+si]
cmp dx, 07777
je loaded
mov int_installed, 0

loaded:
pop si
pop dx
pop bx
pop ax
ret
```

```
is_int_installed endp

install_int proc near
push ds
push es
push ax
push bx
push cx
push dx

mov ah, 35h
mov al, 09h
int 21h
mov keep_ip, bx
mov keep_cs, es

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

;сохраняем наше прерывание до last_byte
mov dx, offset last_byte
mov cl,4
shr dx,cl
inc dx
mov ax, cs
sub ax, psp
add dx, ax
xor ax, ax
mov ah,31h
int 21h

pop dx
pop cx
pop bx
pop ax
pop es
pop ds
ret
install_int endp

uninstall_int proc near
push ds
push es
push ax
push bx
push dx

cli
mov ah,35h
mov al,09h
int 21h
mov dx, es:[offset keep_ip]
mov ax, es:[offset keep_cs]
mov ds, ax
mov ah, 25h
mov al, 09h
```

```
int 21h

mov ax, es:[offset psp]
mov es, ax
mov dx, es:[2ch]
mov ah, 49h
int 21h
mov es, dx
mov ah, 49h
int 21h
sti

pop dx
pop bx
pop ax
pop es
pop ds
ret
uninstall_int endp

check_console proc near
push ax

mov int_installed, 0
mov al, es:[82h]
cmp al, '/'
jne no_key
mov al, es:[83h]
cmp al, 'u'
jne no_key
mov al, es:[84h]
cmp al, 'n'
jne no_key
mov int_installed, 1

no_key:
pop     ax
ret
check_console endp

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

main proc far
push ds
xor ax, ax
mov ax, data
mov ds, ax

mov psp, es
call check_console

cmp int_installed, 1
je int_unload
```

```
call is_int_installed
cmp int_installed, 0
je int_load
mov dx, offset already_loaded_msg
call print
jmp exit

int_load:
mov dx, offset loaded_msg
call print
call install_int
jmp exit

int_unload:
call is_int_installed
cmp int_installed, 0
je unloaded
call uninstall_int
unloaded:
mov dx, offset unloaded_msg
call print

exit:
pop ds
mov ah, 4ch
int 21h
main endp
code ends

astack segment stack
dw 128 dup(?)
astack ends

data segment
int_installed                 db 0
loaded_msg         db 'Interrupt loaded',0dh,0ah,'$'
unloaded_msg       db 'Interrupt unloaded',0dh,0ah,'$'
already_loaded_msg db 'Interrupt already loaded',0dh,0ah,'$'
data ends
end main
```