**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. 9382 |  | Рыжих Р.В. |
| 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.

**Задание.**

**Шаг 1.** Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .ЕХЕ, 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 4Сh прерывания int 21h.

3) Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Сh 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 4Сh 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.

**Ход работы**

**Шаг 1**. Был написан и отлажен программный модуль типа .EXE, который выполняет, данные в задании функции.

**Шаг 2**. Была запущена программа Lab5.exe. Видно, что обработчик прерываний работает успешно. Прерывание меня символы ‘D’, ‘W’, ‘H’ на „!“, „$“, „#“ соответственно.



Рис. 1: Демонстрация работы резидентного обработчика прерываний.

**Шаг 3.** Было проверено размещение прерывания в памяти.

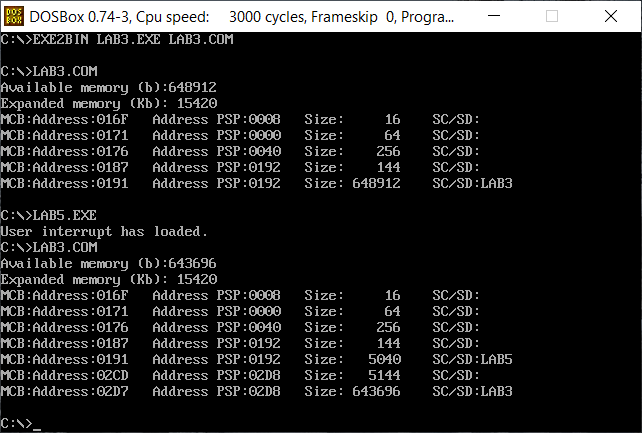


Рис. 2: Проверка размещения резидентного обработчика в памяти.

**Шаг 4.** Программа была повторно запущена, чтобы удостовериться, что программа определяет установленный обработчик прерываний.



Рис. 3: Проверка на то, что программа определяет установленный обработчик прерываний.

**Шаг 5.** Программа была запущена с ключом выгрузки /un для того, чтобы убедиться, что резидентный обработчик прерывания выгружен и память, занятая резидентом, освобождена.

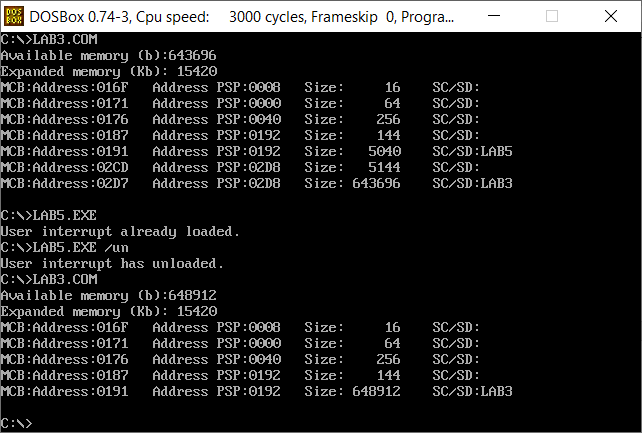


Рис. 4: Демонстрация того, что выгрузка резидентного обработчика прерываний происходит успешно.

**Ответы на контрольные вопросы**

1. Какого типа прерывания использовались в работе?

Использовались следующие прерывания:

• 09h и 16h – аппаратное прерывание

• 21h – программное прерывание.

2. Чем отличается скан-код от кода ASCII?

Скан-код – это уникальное число, однозначно определяющее нажатую

клавишу, в то время как ASCII – это код символа из таблицы ASCII.

**Выводы.**

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

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

**Lab5.asm:**

AStack SEGMENT STACK

DW 256 DUP(?)

AStack ENDS

DATA SEGMENT

IS\_LOAD DB 0

IS\_UNLOAD DB 0

STRING\_LOAD db "User interrupt has loaded.$"

STRING\_LOADED db "User interrupt already loaded.$"

STRING\_UNLOAD db "User interrupt has unloaded.$"

STRING\_NOT\_LOADED db "User interrupt is not loaded.$"

DATA ENDS

CODE SEGMENT

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

WRITESTRING PROC NEAR

push ax

mov ah, 09h

int 21h

pop ax

ret

WRITESTRING ENDP

INTERRUPT PROC FAR

jmp INTER\_START

KEEP\_DATA:

keep\_ip DW 0

keep\_cs DW 0

keep\_psp DW 0

keep\_ax DW 0

keep\_ss DW 0

keep\_sp DW 0

interrupt\_stack DW 256 DUP(0)

key DB 0

sign DW 1234h

INTER\_START:

mov keep\_ax, ax

mov keep\_sp, sp

mov keep\_ss, ss

mov ax, seg interrupt\_stack

mov ss, ax

mov ax, offset interrupt\_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

mov ds, ax

in al, 60h

cmp al, 20h

je KEY\_D

cmp al, 11h

je KEY\_W

cmp al, 23h

je KEY\_H

pushf

call dword ptr cs:keep\_ip

jmp INTER\_END

KEY\_D:

mov key, '!'

jmp NEXT

KEY\_W:

mov key, '$'

jmp NEXT

KEY\_H:

mov key, '#'

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

mov ch, 00h

int 16h

or al, al

jz INTER\_END

mov ax, 0040h

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

INTERRUPT endp

END\_ITTER:

CHECK\_LOAD PROC NEAR

push ax

push bx

push si

mov ah, 35h

mov al, 09h

int 21h

mov si, offset sign

sub si, offset INTERRUPT

mov ax, es:[bx + si]

cmp ax, sign

jne END\_LOAD

mov IS\_LOAD, 1

END\_LOAD:

pop si

pop bx

pop ax

ret

CHECK\_LOAD ENDP

CHECK\_UNLOAD PROC NEAR

push ax

push es

mov ax, keep\_psp

mov es, ax

cmp byte ptr es:[82h], '/'

jne END\_CHECK

cmp byte ptr es:[83h], 'u'

jne END\_CHECK

cmp byte ptr es:[84h], 'n'

jne END\_CHECK

mov IS\_UNLOAD, 1

END\_CHECK:

pop es

pop ax

ret

CHECK\_UNLOAD ENDP

INTERRUPT\_LOAD PROC NEAR

push ax

push bx

push cx

push dx

push ds

push es

mov ah, 35h

mov al, 09h

int 21h

mov keep\_cs, es

mov keep\_ip, bx

mov ax, seg INTERRUPT

mov dx, offset INTERRUPT

mov ds, ax

mov ah, 25h

mov al, 09h

int 21h

pop ds

mov dx, offset END\_ITTER

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

INTERRUPT\_LOAD ENDP

INTERRUPT\_UNLOAD PROC NEAR

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 INTERRUPT

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

INTERRUPT\_UNLOAD ENDP

BEGIN PROC

push ds

xor ax, ax

push ax

mov ax, data

mov ds, ax

mov keep\_psp, es

call CHECK\_LOAD

call CHECK\_UNLOAD

cmp IS\_UNLOAD, 1

je unload

mov al, IS\_LOAD

cmp al, 1

jne load

mov dx, offset STRING\_LOADED

call WRITESTRING

jmp FINISH

LOAD:

mov dx, offset STRING\_LOAD

call WRITESTRING

call INTERRUPT\_LOAD

jmp FINISH

UNLOAD:

cmp IS\_LOAD, 1

jne not\_loaded

mov dx, offset STRING\_UNLOAD

call WRITESTRING

call INTERRUPT\_UNLOAD

jmp FINISH

NOT\_LOADED:

mov dx, offset STRING\_NOT\_LOADED

call WRITESTRING

FINISH:

xor al, al

mov ah, 4ch

int 21h

BEGIN ENDP

CODE ENDS

END BEGIN