

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

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

Студентка гр. 9383

Лихашва А.Д.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Постановка задачи

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 .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 шагов лабораторной работы:

1 шаг:

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

2 шаг:

Была запущена программа lab.exe. Видно, что обработчик прерываний работает успешно. Прерывание меня символы 'D', 'W', 'H' на „!“, „\$“, „#“ соответственно.

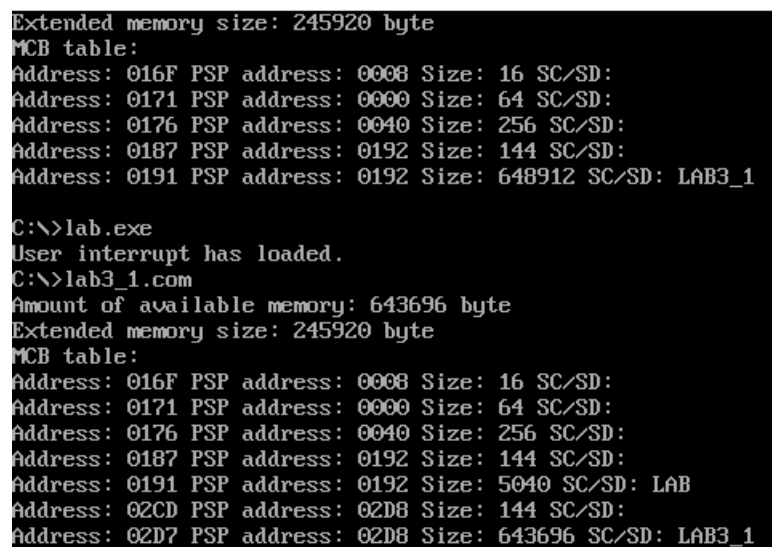


```
C:\>lab.exe
User interrupt has loaded.
C:\>asa$$$$!?!?#####_
```

Рисунок 1: Демонстрация работы резидентного обработчика прерываний

Шаг 3.

Было проверено размещение прерывания в памяти.



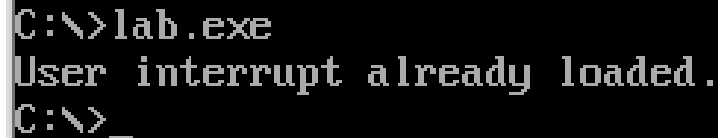
```
Extended memory size: 245920 byte
MCB table:
Address: 016F PSP address: 0008 Size: 16 SC/SD:
Address: 0171 PSP address: 0000 Size: 64 SC/SD:
Address: 0176 PSP address: 0040 Size: 256 SC/SD:
Address: 0187 PSP address: 0192 Size: 144 SC/SD:
Address: 0191 PSP address: 0192 Size: 648912 SC/SD: LAB3_1

C:\>lab.exe
User interrupt has loaded.
C:\>lab3_1.com
Amount of available memory: 643696 byte
Extended memory size: 245920 byte
MCB table:
Address: 016F PSP address: 0008 Size: 16 SC/SD:
Address: 0171 PSP address: 0000 Size: 64 SC/SD:
Address: 0176 PSP address: 0040 Size: 256 SC/SD:
Address: 0187 PSP address: 0192 Size: 144 SC/SD:
Address: 0191 PSP address: 0192 Size: 5040 SC/SD: LAB
Address: 02CD PSP address: 02D8 Size: 144 SC/SD:
Address: 02D7 PSP address: 02D8 Size: 643696 SC/SD: LAB3_1
```

Рисунок 2: Демонстрация работы резидентного обработчика прерываний

Шаг 4.

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

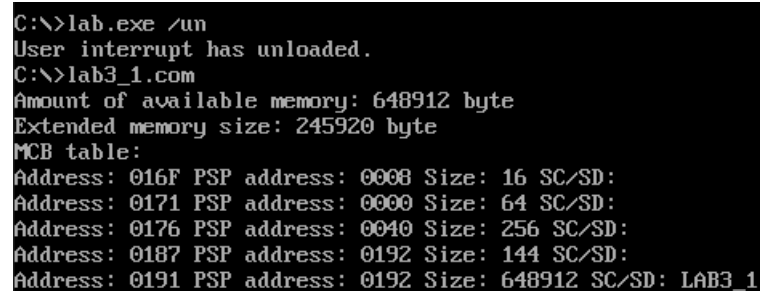


```
C:\>lab.exe
User interrupt already loaded.
C:\>
```

Рисунок 3: Демонстрация определения установленного обработчика прерывания при повторном запуске программы.

Шаг 5.

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



```
C:\>lab.exe /un
User interrupt has unloaded.
C:\>lab3_1.com
Amount of available memory: 648912 byte
Extended memory size: 245920 byte
MCB table:
Address: 016F PSP address: 0008 Size: 16 SC/SD:
Address: 0171 PSP address: 0000 Size: 64 SC/SD:
Address: 0176 PSP address: 0040 Size: 256 SC/SD:
Address: 0187 PSP address: 0192 Size: 144 SC/SD:
Address: 0191 PSP address: 0192 Size: 648912 SC/SD: LAB3_1
```

Рисунок 4: Демонстрация выгрузки резидентного обработчика прерываний

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.

Заключение.

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

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: lab.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
```

```
PRINT_STRING PROC NEAR
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
PRINT_STRING ENDP
```

```
INTERRUPT PROC FAR
    jmp interrupt_start
```

```
interrupt_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
```

```
interrupt_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 interrupt_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 interrupt_end
        mov ax, 0040h
        mov es, ax
        mov ax, es:[1ah]
        mov es:[1ch], ax

```

```
    jmp print_key
```

```
interrupt_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_I:
```

```
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 load_end
    mov IS_LOAD, 1
```

load_end:

```
    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 check_end
    cmp byte ptr es:[83h], 'u'
    jne check_end
    cmp byte ptr es:[84h], 'n'
    jne check_end
    mov IS_UNLOAD, 1
```

check_end:

```
    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_I
    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 PRINT_STRING
    jmp begin_end
```

load:

```
    mov dx, offset STRING_LOAD
    call PRINT_STRING
    call INTERRUPT_LOAD
    jmp  begin_end
```

unload:

```
    cmp  IS_LOAD, 1
    jne  not_loaded
    mov dx, offset STRING_UNLOAD
    call PRINT_STRING
    call INTERRUPT_UNLOAD
    jmp  begin_end
```

not_loaded:

```
    mov  dx, offset STRING_NOT_LOADED
    call PRINT_STRING
```

begin_end:

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

BEGIN ENDP

CODE ENDS

END BEGIN