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. 9382 _____ Голубева В.П.

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.

Порядок выполнения работы

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 згот 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 згот 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 згого 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.

Прерывание заменяет символы «w», «e», «r» на «1», «2», «3» соответственно. В Рисунке 1 демонстрируется, что после введения слова «result», оно заменилось на «32sult». А после выгрузки при вводе того же слова, оно ввелось без изменений.

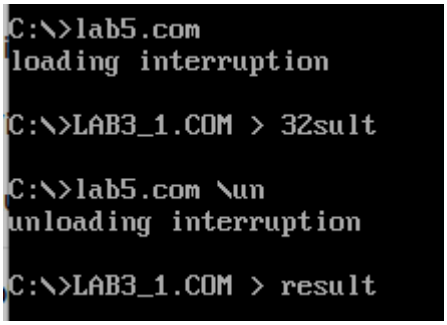

Рисунок 1. Работа прерывания

Проверим состояние памяти после загрузки прерывания — запустим программу из лабораторной номер 3. Результат можно посмотреть в Рисунке 2.

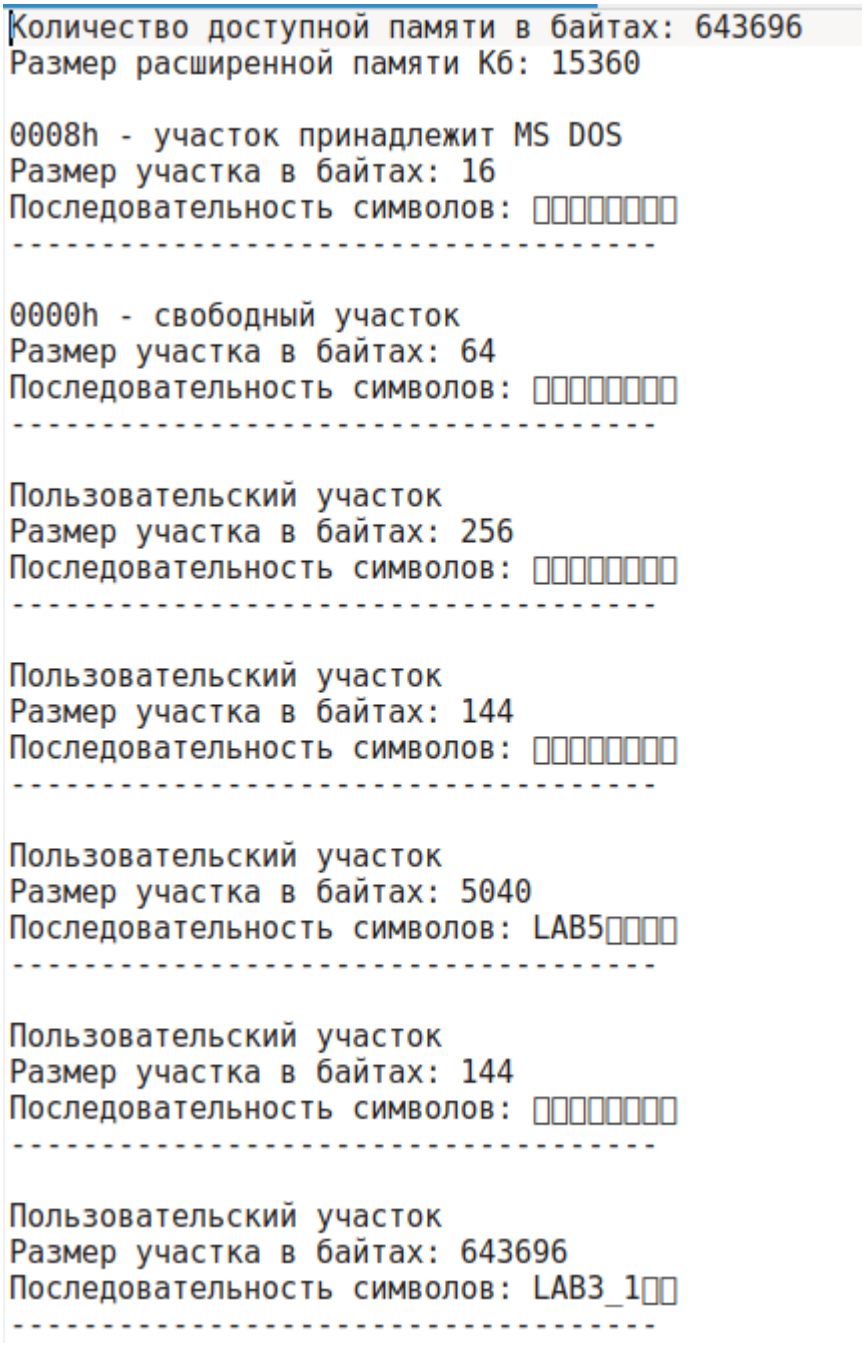

Рисунок 2. Состояние памяти после загрузки прерывания

Затем выгрузим преывание и посмотрим на состояние памяти снова. Результат можно посмотреть в Рисунке 3.

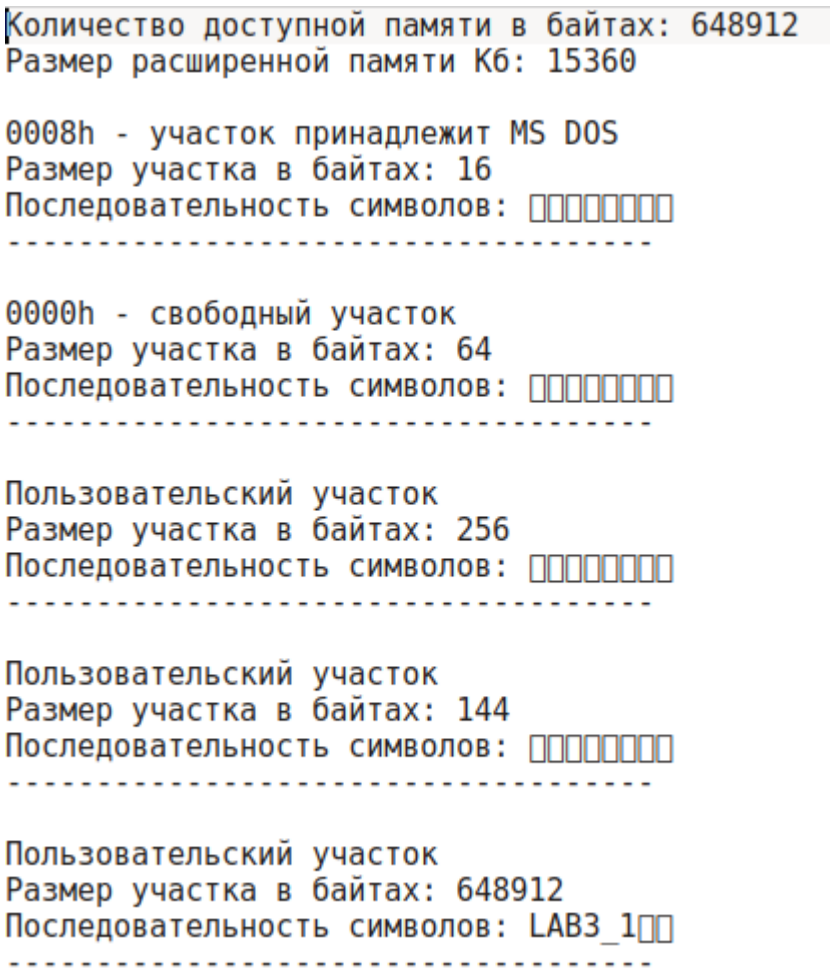

Рисунок 3. Состояние памяти после выгрузки прерывания

Прерывание корректно загружается, выполняется и выгружается с освобождением памяти.

Ответы на контрольные вопросы

Контрольные вопросы по лабораторной работе №5

1) Какого типа прерывания использовались в работе?

Ответ: прерывания функций DOS(21h), прерывания функций BIOS(16h, 09h).

2) Чем отличается скан код от кода ASCII?

Ответ: Код ASCII – это код символа из таблицы ASCII, а скан-код – код,

присвоенный каждой клавише, с помощью которого драйвер клавиатуры распознает, какая клавиша была нажата.

Выводы.

Были исследованы возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Пользовательский обработчик прерывания получаел управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре. Он обрабатывал скан-код и осуществлял замену определённых символов. Были получены навыки работы по сопряжению различных типов прерываний.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

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

stacks  segment stack
 dw  256 dup(0)
stacks  ends

data segment
    str_load db  "interruption has loaded",0dh,0ah,"$"
    str_loaded db  "interruption  already loaded",0dh,0ah,"$"
    str_unload db  "interruption has unloaded",0dh,0ah,"$"
    str_not_loaded  db  "interruption is not loaded",0dh,0ah,"$"
    is_load  db  0
    is_un db  0
data ends

interruption proc far
    jmp  start
interruptiondata:
    key_value db 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
    new_stack dw 256 dup(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, 11h
    je key_w
    cmp al, 12h
    je key_e
    cmp al, 13h
    je k_r

    pushf
    call dword ptr cs:keep_ip
    jmp end_interr

key_w:
    mov key_value, '1'
    jmp next_key
key_e:
    mov key_value, '2'
    jmp next_key
k_r:
    mov key_value, '3'
```

```
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_value
    mov ch, 00h
    int 16h
    or     al, al
    jz     end_interr
    mov ax, 0040h
    mov es, ax
    mov ax, es:[1ah]
    mov es:[1ch], ax
    jmp print_key

end_interr:
    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
interruption endp
 _end:

is_interr_load proc
    push ax
    push bx
    push si

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

eeendis_l:
    pop  si
    pop  bx
    pop  ax
    ret
    is_interr_load endp

    int_load  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 interruption
    mov dx, offset interruption
    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
int_load  endp

unload_interrupt 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 interruption
    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_interrupt endp

is_unload_  proc
    push ax
    push es

    mov ax, keep_psp
    mov es, ax
    cmp byte ptr es:[82h], '\'
    jne eeendun
    cmp byte ptr es:[83h], 'u'
    jne eeendun
    cmp byte ptr es:[84h], 'n'
    jne eeendun
    mov is_un, 1

eeendun:
    pop es
    pop ax
 ret
is_unload_ endp

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

begin proc
    push ds
    xor ax, ax
    push ax
    mov ax, data
    mov ds, ax
    mov keep_psp, es
```

```
    call is_interr_load
    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_str
    jmp eeend
load:
    mov dx, offset str_load
    call print_str
    call int_load
    jmp  eeend
unload:
    cmp  is_load, 1
    jne  not_loaded
    mov dx, offset str_unload
    call print_str
    call unload_interrupt
    jmp  eeend
not_loaded:
    mov  dx, offset str_not_loaded
    call print_str
eeend:
    xor al, al
    mov ah, 4ch
    int 21h
begin endp
code ends
end begin
```