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

Студент гр. 9381 _____ Прибылов Н.А.
Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

Цель работы.

Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Пользовательский обработчик прерываний получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре. Он обрабатывает скан-код и осуществляет определенные действия, если скан-код совпадает с определёнными кодами, которые он должен обрабатывать. Если скан-код не совпадает с этими кодами, то управление передается стандартному прерыванию.

Ход работы.

1) Написан и отлажен программный модуль типа .EXE, который выполняет такие же функции, как и в лабораторной работе №4, а именно:

- Проверяет, установлено ли пользовательское прерывание с вектором 09h.
- 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 int21h.
- Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.

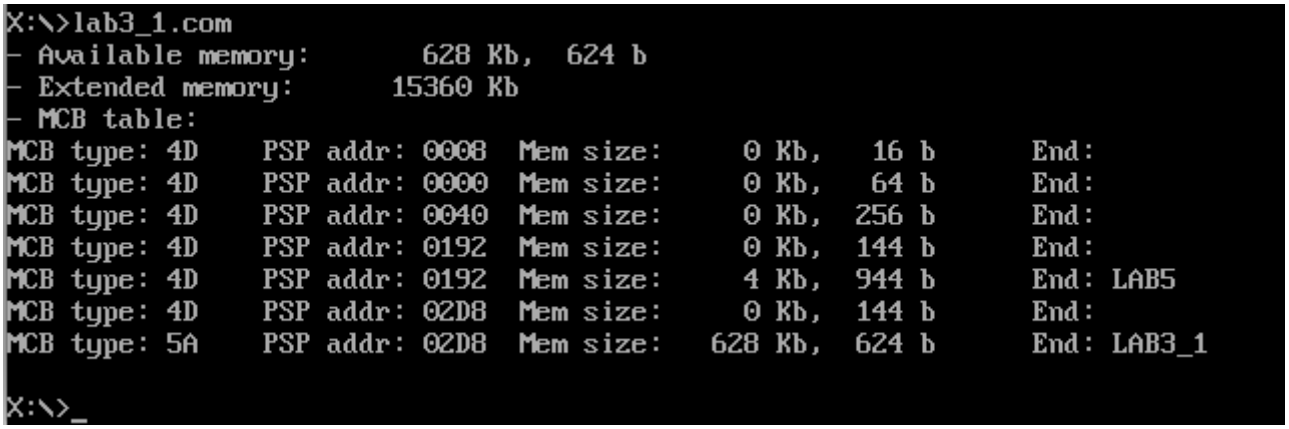
2. Была запущена отлаженная программа и проверено, что резидентный обработчик прерывания 09h установлен (заменяет символы, вводимые с клавиатуры: h → @, p → #, e → !).

```
X:\>lab5.exe
Interrupt is loading now.

X:\>@a##y n!w y!ar_
```

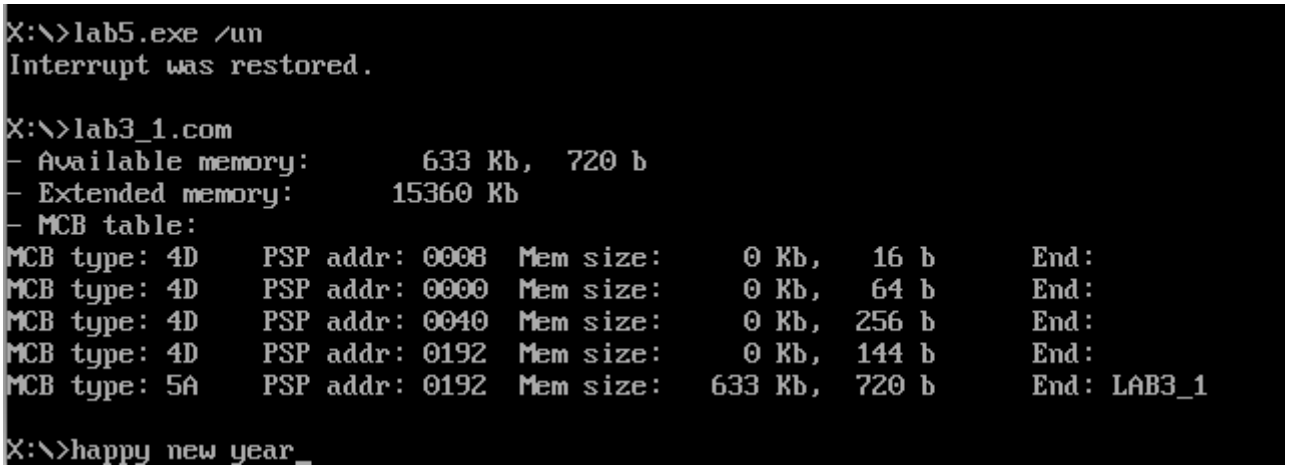
Программа запущена

3. Было проверено размещение прерывания в памяти. Для этого запущена программа из лаб. работы №3, которая отображает карту памяти в виде списка блоков MCB.



Резидент находится в памяти

4. Была запущена отлаженная программа с ключом выгрузки и проверено, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а занятая резидентом память освобождена. Для этого также была запущена программа лабораторной работы №3.



Резидент выгружен из памяти

Контрольные вопросы.

1. Какого типа прерывания использовались в работе?

Прерывания DOS (int 21h), прерывания BIOS (09h, 16h).

2. Чем отличается скан код от кода ASCII?

Скан-код – это код, который клавиатура передаёт системе. Тем самым система определяет, какая клавиша (или комбинация клавиш) была нажата. ASCII-код – это таблица кодировок для печатных символов.

Таким образом скан-код – это номер клавиши, а ASCII-код – код, соответствующий обозначению на этой клавише.

Выводы.

В ходе 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

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab5.asm

```
CODE SEGMENT
ASSUME  CS:CODE, DS:DATA, SS:STACK

STACK SEGMENT stack
    dw 256 dup(0)
STACK ends

new_interrupt PROC FAR
    jmp Start

intData:
    key_value   db 0
    new_stack   dw 256 dup(0)
    signature   dw 2468h
    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, 12h     ;e
    je      key_e
    cmp     al, 19h     ;p
    je      key_p
    cmp     al, 23h     ;h
    je      key_h

    pushf
```

```
    call    dword ptr cs:keep_ip
    jmp     end_interrupt

key_e:
    mov     key_value, '!'
    jmp     next_key
key_p:
    mov     key_value, '#'
    jmp     next_key
key_h:
    mov     key_value, '@'

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_interrupt
    mov     ax, 40h
    mov     es, ax
    mov     ax, es:[1ah]
    mov     es:[1ch], ax
    jmp     print_key


end_interrupt:
    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
new_interrupt 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 new_interrupt
    mov     ax, es:[bx + si]
    cmp     ax, signature
    jne     end_proc
    mov     IS_LOADED, 1

end_proc:
    pop     si
    pop     bx
    pop     ax
    ret
    is_int_loaded 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 new_interrupt
    mov     dx, offset new_interrupt
    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 new_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
unload_interrupt endp


is_unloaded_  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_UNLOADED, 1

end_unload:
    pop     es
    pop     ax
    ret
is_unloaded_ 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_unloaded_
    cmp     IS_UNLOADED, 1
    je      unload
    mov     al, IS_LOADED
    cmp     al, 1
    jne     load
    mov     dx, offset STRING_INT_ALREADY_LOADED
    call    PRINT
    jmp     end_begin

load:
    mov     dx, offset STRING_INT_IS_LOADING
    call    PRINT
    call    int_load
    jmp     end_begin

unload:
    cmp     IS_LOADED, 1
    jne     not_loaded
    mov     dx, offset STRING_INT_RESTORED
    call    PRINT
    call    unload_interrupt
    jmp     end_begin

not_loaded:
    mov     dx, offset STRING_INT_NOT_LOADED
    call    PRINT

end_begin:
```

```
        xor     al, al
        mov     ah, 4ch
        int     21h
    begin endp
    CODE ends

    DATA SEGMENT
        IS_LOADED                   db 0
        IS_UNLOADED                 db 0
         STRING_INT_IS_LOADING        db "Interrupt is loading now.",
0dh, 0ah, "$"
         STRING_INT_ALREADY_LOADED   db "Interrupt is already loaded.",
0dh, 0ah, "$"
         STRING_INT_RESTORED         db "Interrupt was restored.", 0dh,
0ah, "$"
         STRING_INT_NOT_LOADED         db "Interrupt was not loaded.",
0dh, 0ah, "$"
    DATA ends
    end begin
```