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. 0382 Самулевич В.А.

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.

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 работы.

Шаг 1:

При написании требуемого .exe модуля были реализованы следующие функции:

- Select_mode - в зависимости от переданных аргументов определяет, будет происходить загрузка или выгрузка пользовательского прерывания. После того, как режим работы определен, происходит вызов соответствующих функций.
- load_interrupt – производит загрузку пользовательского обработчика вместо стандартного по номеру 09h.
- unload_interrupt – производит выгрузку пользовательского обработчика и очистку занимаемой им памяти.
- User_interrupt – пользовательский обработчик для прерывания с номером 09h. Он осуществляет замену клавиш стрелок на символы w (верхняя стрелка), a (левая стрелка), s (нижняя стрелка), d (правая стрелка).

Шаг 2:

Символы, выводящиеся на экран после вызова lab5.exe и нажатии различных стрелок, представлены на рисунке 1.

Рисунок 1 – результат работы lab5.exe.

Можно заметить, что замена стрелок на w, a, s, d прошла успешно. Отсюда следует, что резидентный обработчик прерывания 09h был установлен.

Шаг 3:

Карта памяти в виде блоков MCB (после вызова lab5.exe) представлена на рисунке 2.

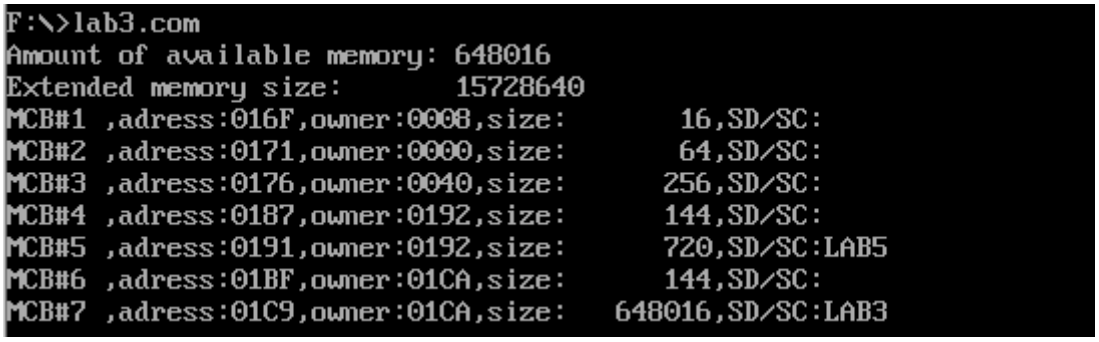

Рисунок 2 – карта памяти после вызова lab5.exe.

Как видно из рисунка, MCB#5 занят резидентом, принадлежащим lab5. Это указывает на успешную загрузку пользовательского обработчика в основную память.

Шаг 4:

Результат работы программы при ее повторном запуске представлен на рисунке 3.

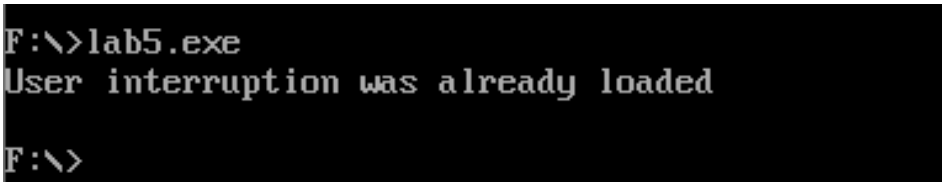

Рисунок 3 – результат повторного запуска программы.

Т.к. пользовательский обработчик уже был загружен, модуль ограничивается выводом соответствующего сообщения.

Шаг 5:

Результат работы lab5.exe с ключом выгрузки /un , а также обновлённая карта памяти представлены на рисунке 4.

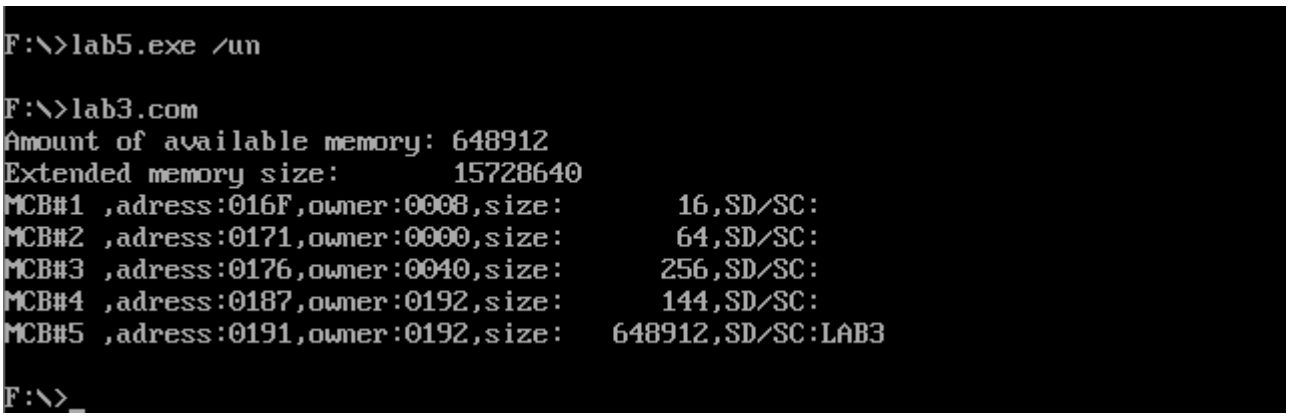

Рисунок 4 – результат работы модуля с ключом выгрузки, вместе с обновленной картой памяти.

Как видно из рисунка, не существует ни одного блока MCB, который бы принадлежал lab5.Это говорит о том, что выгрузка пользовательского обработчика из основной памяти прошла успешно.

Шаг 6:

Контрольные вопросы:

1) Какого типа прерывания использовались в работе?
   Программные(21h) и аппаратные(09h,16h)

2) Чем отличается скан-код от кода ASCII?
   Скан-код – код, присвоенный каждой клавише клавиатуры. Отсюда следует, что он часто не связан с каким-либо одним символом из таблицы ASCII: например, скан код клавиши A может представлять, как строчную, так и заглавную букву, в зависимости от флагов в байтах состояния. Также есть скан- коды не имеющие символьного представления - shift, delete и т.д.

   Разработанный код см. в приложении А.

Выводы.

Было произведено исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab5.asm

```
AStack SEGMENT STACK
    DW 12 DUP(?)
AStack ENDS

DATA SEGMENT
    FLAG DB "/un",0DH
    already_loaded_message DB "User interruption was already loaded",
0Dh,0AH,'$'
    already_unloaded_message DB "user interrupt not set",0Dh,0Ah,'$'

DATA ENDS

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

User_interrupt PROC FAR

jmp interrupt_start

    SIGNATURE DW 5555
    KEEP_PSP DW 0
    KEEP_SS DW 0
    KEEP_SP DW 0
    KEEP_IP DW 0
    KEEP_CS DW 0
    FIRST_BYTE_FLAG DB 0
    INT_STACK DW 100 DUP(0)

interrupt_start:
    mov KEEP_SS, SS
    mov KEEP_SP,SP
    mov SP, SEG INT_STACK
    mov SS,SP
    mov SP,offset interrupt_start


    push AX
    push BX
    push CX
    push ES


 in AL,60h
    cmp AL,48h
    je up
    cmp AL,4bh
    je left
    cmp AL,50h
    je down
    cmp AL,4dh
    je right
```

```
base_handler:
    pushf
    call DWORD PTR CS:KEEP_IP
    jmp interrupt_end



up:
    mov CL,'w'
    jmp load_into_buffer

down:
    mov CL,'s'
    jmp load_into_buffer

left:
    mov CL,'a'
    jmp load_into_buffer

right:
    mov CL,'d'

load_into_buffer:
    in AL,61h
    mov AH,AL
    or AL,80h
    out 61h,AL
    xchg AH,AL
    out 61h,AL
    mov AL,20h
    out 20h,AL
load_attempt:
    mov AH,05h
    mov CH,0
    int 16h
    cmp AL,0
    je interrupt_end
    mov AX,40h
    mov ES,AX
    mov AX,ES:[1ah]
    mov ES:[1ch],AX
    jmp load_attempt

 interrupt_end:
    pop ES
    pop CX
    pop BX
    pop AX


    mov SS,KEEP_SS
    mov SP,KEEP_SP
    mov AL,20h
    out 20h,AL
    iret
User_interrupt ENDP
```

```
Main PROC FAR
    push DS
    sub AX,AX
    push AX
    mov AX,DATA
    mov DS,AX
    mov WORD PTR KEEP_PSP,ES
    call Select_mode

    ret
Main ENDP

Select_mode PROC NEAR
    push AX
    push SI
    push DI

    mov AL,32
    mov DI,081h

skip_spaces:
    SCASB
    je skip_spaces

    dec DI
    mov SI,offset  FLAG

cmp_loop:
    mov AL, ES:[DI]
    cmp AL, DS:[SI]
    jne load_mode
    cmp AL,0dh
    je unload_mode
    inc SI
    inc DI
    jmp cmp_loop

load_mode:
    call load_interrupt
    jmp select_mode_end

unload_mode:
    call unload_interrupt

select_mode_end:

    pop DI
    pop SI
    pop AX

    ret
Select_mode ENDP

load_interrupt PROC NEAR
    push SI
```

```
    push AX
    push BX
    push CX
    push DX
    push ES

    mov AH,35h
    mov AL,09h
    int 21h
    mov SI,offset SIGNATURE
    mov AX, ES:[SI]
    cmp AX,5555
    je already_loaded
    mov KEEP_IP, BX
    mov KEEP_CS, ES

    push DS
    mov DX, offset User_interrupt
    mov AX, seg User_interrupt
    mov DS, AX
    mov AH, 25h
    mov AL, 09h
    int 21h
    pop DS
    mov DX, offset Main
    mov CL, 4
    shr DX, CL
    inc DX
    mov AX, CS
    sub AX, KEEP_PSP
    add DX, AX
    mov AX,0
    mov AH, 31h
    int 21h
    jmp load_end
already_loaded:
    mov DX, offset already_loaded_message
    call print
load_end:

    pop ES
    pop DX
    pop CX
    pop BX
    pop AX
    pop SI
    ret
load_interrupt ENDP



unload_interrupt PROC NEAR
    push AX
    push BX
    push DX
    push ES
    push SI
```

```
    mov AH,35h
    mov AL,09h
    int 21h
    mov SI,offset SIGNATURE
    mov AX, es:[SI]
    cmp AX,5555
    jne already_unloaded
    CLI
    push DS
    mov DX,ES:KEEP_IP
    mov AX,ES:KEEP_CS
    mov DS,AX
    mov AH,25h
    mov AL,09h
    int 21h
    pop DS
    mov SI,offset KEEP_PSP
    mov AX,ES:[SI]
    mov ES,AX
    push ES
    mov AX,ES:[2ch]
    mov ES,AX
    mov AH,49h
    int 21h
    pop ES
    int 21h
    STI
    jmp unload_end

already_unloaded:
    mov DX,offset already_unloaded_message
    call print
unload_end:
    pop SI
    pop ES
    pop DX
    pop BX
    pop AX
    ret
unload_interrupt ENDP


print PROC NEAR
    push AX
    mov AH,09h
    int 21h
    pop AX
    ret
print ENDP


CODE ENDS
    END Main
```