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. 0382 _____ Морева Е.С.

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. На основе кода лабораторной работы №4 был написан и отлажен программный модуль типа .EXE, который выполняет следующие функции:

1) Проверяет, установлено ли пользовательское прерывание;

2) Устанавливает резидентную функцию для обработки прерывания;

3) Если резидентная функция уже установлена — выводит соответствующее сообщение;

4) Выгружает прерывание по значению параметра в командной строке: /un, восстанавливая стандартный вектор прерывания.

В данном случае обработчик пользовательского прерывания получает управление по прерыванию int 09h при нажатии клавиши на клавиатуре. Обрабатывается скан-код клавиши: если он совпадает с определенным заданным кодом, то символ будет заменён на другой, если не совпадает с заданным - управление вернётся к стандартному прерыванию.

Функции, реализованные в работе:

- ROUT — Обработчик прерывания (вместо символа «M» выводит знак «♠»);
- CHECK_USER_INT— Проверка, загружено ли пользовательское прерывание;
- SET_INT — Установка нового обработчика прерывания с запоминанием данных для восстановления предыдущего;
- PRINT — Осуществление вывода.

Шаг 2. Загрузка программы.

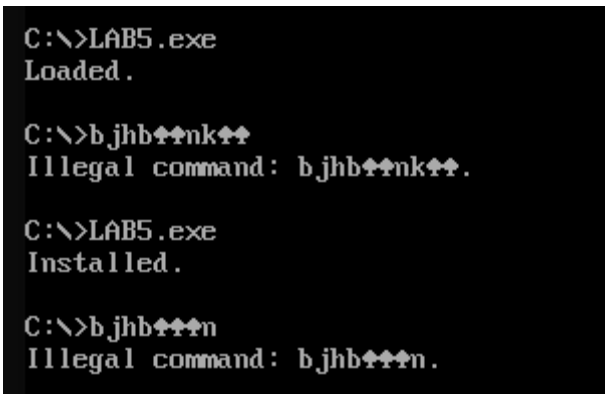

Рисунок 1 — Результат загрузки lab5.exe

Так как при вводе строки символы «M» были заменены на заданный, а те, что не были указаны в обработчике прерывания остались без изменения, можно сделать вывод о том, что пользовательское прерывание было успешно установлено.

Шаг 3. Проверка размещения прерывания в памяти.

Для того, чтобы удостовериться в корректности проверки — фиксируем вывод информации о состоянии блоков MCB перед установкой прерывания (пользуясь программой из лабораторной работы №3).

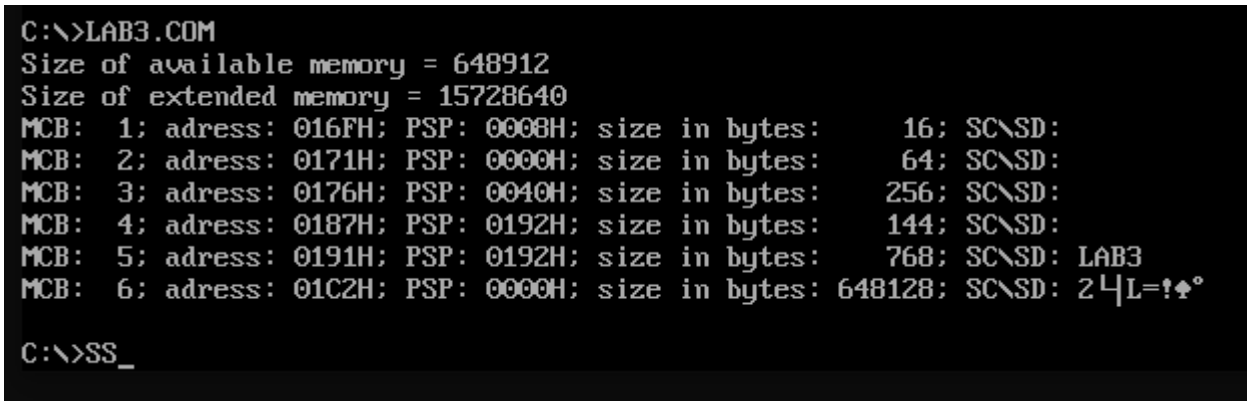

Рисунок 2 — Результат запуска lab3.com перед установкой прерывания

Следом вновь запускаем lab3.com после установки lab5.exe:

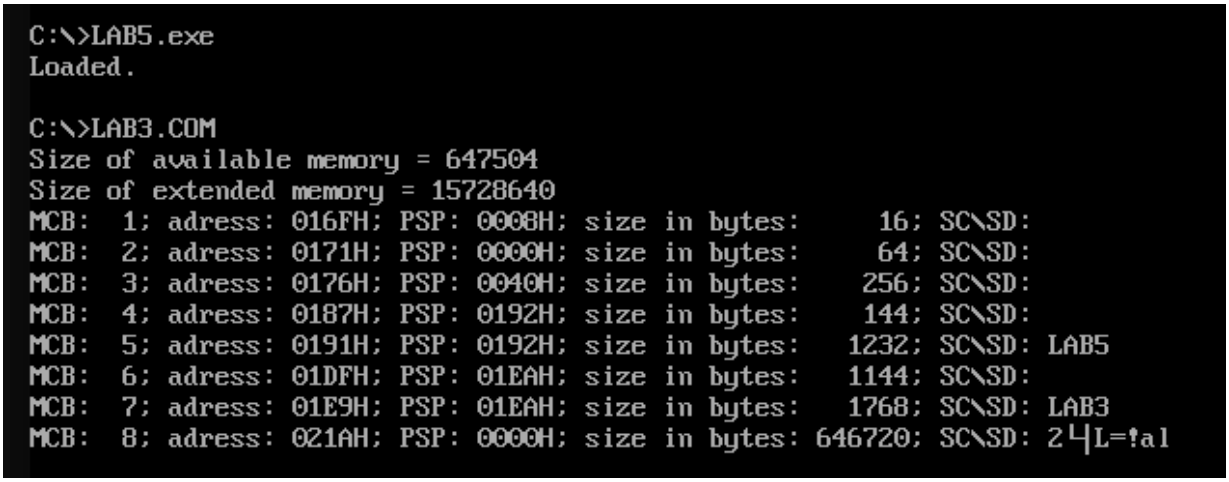

Рисунок 3 — Результат загрузки lab3.com после установки прерывания

Исходя из вывода программы видно, что в памяти появились блоки MCB обработчика прерывания (lab5.exe), что свидетельствует о том, что пользовательское прерывание успешно загружено в память.

Шаг 4. Повторный запуск программы.

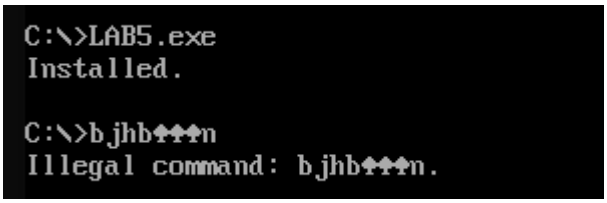

Рисунок 4 — Результат повторной загрузки lb.exe

В результате прерывание не было установлено повторно, о чем говорит соответствующее сообщение.

Шаг 5. Восстановление прерывания по умолчанию.

Рисунок 5 — Результат запуска lab5.exe с ключом выгрузки Символы

введённой строки не были изменены, следовательно, пользовательское прерывание было выгружено и стандартный вектор рассматриваемого прерывания восстановлен.

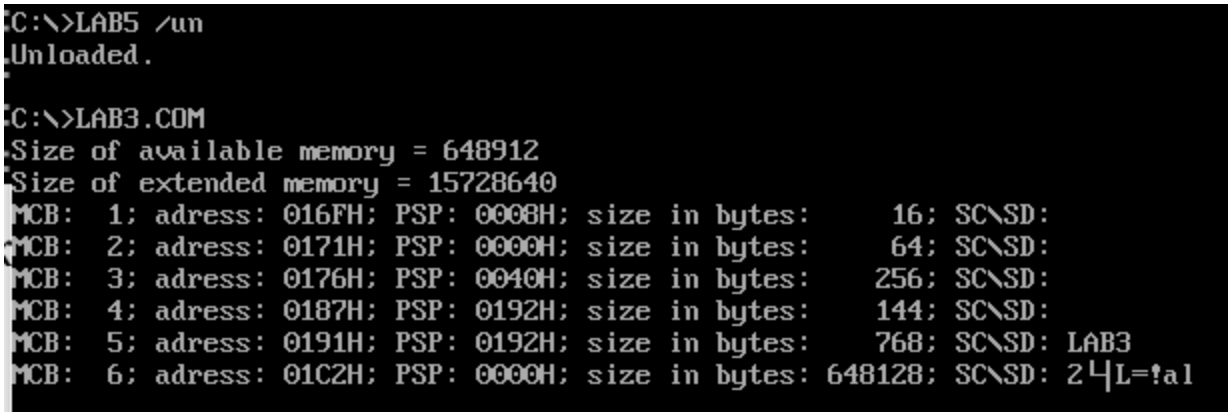

Рисунок 6 — Результат запуска lab3.com после выгрузки прерывания

Вывод программы lab3.com демонстрирует то, что блоки памяти, в которых хранилось пользовательское прерывание — удалены, значит память, занимаемая резидентом — освобождена, что так же указывает на успешную выгрузку.

Исходный код программ см. в приложении А

Контрольные вопросы.

1) Какого типа прерывания использовались в работе?

- Ответ: В работе использовались программные прерывания (int 21h) и аппаратные (int 16h, int 09h).

2) Чем отличается скан код от кода ASCII?

- Ответ: Скан код – код клавиши на клавиатуре; Код ASCII – код символа согласно таблице ASCII.

Выводы.

Были изучены возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

Разработан пользовательский обработчик прерывания при нажатии клавиши на клавиатуре.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: lab5.asm

```
AAStack SEGMENT STACK
        DW 256 DUP(?)

AStack ENDS

DATA SEGMENT
LOADED db 'Loaded.' , 0DH, 0AH, '$'
INSTALLED db 'Installed.', 0DH, 0AH, '$'
UNLOADED db 'Unloaded.' , 0DH, 0AH, '$'
DATA ENDS

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

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

ROUT PROC FAR
        jmp start_func
        SIGN db '0000'
        KEEP_PSP dw 0
        KEEP_IP dw 0
        KEEP_CS dw 0
        KEEP_SS dw 0
        KEEP_SP dw 0
        KEEP_AX dw 0
        value db 0

        INTERRUPT_STACK dw 128 dup (?)
        END_INT_STACK dw ?

start_func:
        mov KEEP_SS, ss
        mov KEEP_SP, sp
        mov KEEP_AX, ax
        mov ax, cs
        mov ss, ax
        mov sp, offset END_INT_STACK

        push ax
```

```
        push bx
        push cx
        push dx
        push si
        push es
        push ds

        mov ax, seg value
        mov ds, ax


        in al, 60h
        cmp al, 32h
        je do_req


        pushf
        call dword ptr cs:KEEP_IP
        jmp int_final

do_req:
        in al, 61h
        mov ah, al
        or al, 80h
        out 61h, al
        xchg ah, al
        out 61h, al
        mov al, 20h
        out 20h, al

print_key:
        mov ah, 05h
        mov cl, 06h
        mov ch, 00h
        int 16h
        or al, al
        jz int_final
        mov ax, 40h
        mov es, ax
        mov ax, es:[1ah]
        mov es:[1ch], ax
        jmp print_key

int_final:

        pop ds
        pop es
        pop si
        pop dx
        pop cx
        pop bx
```

```
        pop ax

        mov ax, KEEP_SS
        mov ss, ax
        mov ax, KEEP_AX
        mov sp, KEEP_SP

        mov al, 20h
        out 20h, al
        iret
FINAL:
ROUT ENDP


CHECK_USER_INT PROC NEAR
        push ax
        push bx
        push si

   mov AH, 35h       ; функция получения вектора
   mov AL, 09h       ; номер вектора
   int 21h
   mov SI, offset SIGN
   sub SI, offset ROUT
   mov AX,'00'
        cmp AX, ES:[BX+SI]
   jne UNLOAD
   cmp AX, ES:[BX+SI+2]
   je LOAD
UNLOAD:
   call SET_INT

   mov DX, offset FINAL  ; размер в байтах от начала сегмента
   mov CL, 4             ; перевод в параграфы
   shr DX, CL
   inc DX                ; размер в параграфах
   add DX, CODE
   sub DX, KEEP_PSP
   xor AL, AL
   mov AH, 31h
        int 21h

LOAD:
   push ES
   push AX
   mov AX, KEEP_PSP
   mov ES, AX
   cmp byte ptr ES:[82h],'/'
   jne INST
   cmp byte ptr ES:[83h],'u'
   jne INST
```

```
    cmp byte ptr ES:[84h],'n'
    je UNL
INST:
    pop AX
    pop ES
    mov DX, offset INSTALLED
    call PRINT

        pop si
        pop bx
        pop ax
    ret
UNL:
    pop AX
    pop ES

    call INT_UN
    mov DX, offset UNLOADED
    call PRINT

        pop si
        pop bx
        pop ax
    ret
CHECK_USER_INT ENDP

SET_INT PROC NEAR
        push AX
        push BX
        push DX
        push ES
    push DS
    mov AH, 35h        ; функция получения вектора
    mov AL, 09h        ; номер вектора
    int 21h
    mov KEEP_IP, BX    ; запоминание смещения
    mov KEEP_CS, ES    ; и сегмента

    mov dx, offset ROUT ; смещение для процедуры в DX
    mov ax, seg ROUT    ; сегмент процедуры
        mov DS, AX         ; помещаем в DS
    mov AH, 25h        ; функция установки вектора
    mov AL, 09h        ; номер вектора
    int 21h            ; меняем прерывание
    pop DS
    mov DX, offset LOADED
    call PRINT


    pop ES
        pop DX
```

```
        pop BX
        pop AX
    ret
SET_INT ENDP

INT_UN PROC NEAR
        cli
        push ax
        push bx
        push dx
        push si
        push es
        push ds

        mov ah, 35h
        mov al, 09h
        int 21h

        mov si, offset KEEP_IP
        sub si, offset ROUT
        mov dx, es:[bx + si]
        mov ax, es:[bx + si + 2]

        mov ds, ax
        mov ah, 25h
        mov al, 09h
        int 21h
        pop ds

        mov ax, es:[bx + si - 2]
        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 es
        pop si
        pop dx
        pop bx
        pop ax

        ret
INT_UN ENDP
```

```
MAIN PROC FAR
   mov AX, DATA
   mov DS, AX
         mov KEEP_PSP, ES
   call CHECK_USER_INT
   xor AL, AL
   mov AH, 4Ch
   int 21h
MAIN ENDP
CODE ENDS
   END MAIN
```