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

ОТЧЕТ

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

Студент гр. 9383 _____ Гладких А.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

ПОСТАНОВКА ЗАДАЧИ

Цель работы.

Исследовать возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Написать пользовательский обработчик прерывания, который получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре и обрабатывает скан-код, осуществляя 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.

Сведения о функциях и структурах данных управляющей программы.

Процедура	Описание
WRITEWRD	Функция печати строки на экран
WRITEBYTE	Функция печати символа на экран
ENDLINE	Функция печати символов переноса строки
OUTPUTAL	Вывод на экран содержимого регистра AL
OUTPUTBP	Вывод на экран содержимого регистра BP
MY_INTERRUPTION	Функция прерывания
CHECK_CLI_OPT	Проверка наличия параметров командной строки
CHECK_LOADED	Проверка загрузки пользовательского прерывания
LOAD_INTERRUPTION	Функция загрузки пользовательского прерывания в таблицу прерываний
UNLOAD_INTERRUPTION	Функция возвращения стандартного прерывания
MAIN	Главная функция программы

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.

Исходный код.

Исходный код представлен в приложении А.

РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ ПРОБЛЕМ

Шаг 1. Был написан и отлажен программный модуль типа .EXE, который выполняет поставленные в задании функции. Прерывание заменяет символы «q», «w» на символы «a», «z» или на «A», «Z», если также была нажата клавиша «Shift».

Шаг 2. Написанный модуль был отлажен и запущен. Резидентный обработчик прерываний был установлен и размещен в памяти.

```
D:\>lab.exe
Interruption is loaded successfully

D:\>azerty
Illegal command: azerty.

D:\>lab.exe /un
Interruption is restored

D:\>qwerty
```

Рисунок 1 - Иллюстрация работы .EXE-модуля

```
D:\>lab.exe
Interruption is loaded successfully

D:\>lab3.com
Available Memory (bytes):644240
Extended Memory (kbytes):15360

MCB Table:
MCB #1 Address: 016F  PSP TYPE: belongs MSDOS       Size: 0001  SC/SD:
MCB #2 Address: 0171  PSP TYPE: free area           Size: 0004  SC/SD: DPMILOA
MCB #3 Address: 0176  PSP TYPE: 0040                Size: 0010  SC/SD:
MCB #4 Address: 0187  PSP TYPE: 0192                Size: 0009  SC/SD:
MCB #5 Address: 0191  PSP TYPE: 0192                Size: 0119  SC/SD: LAB
MCB #6 Address: 02AB  PSP TYPE: 02B6                Size: 0009  SC/SD:
MCB #7 Address: 02B5  PSP TYPE: 02B6                Size: 9D49  SC/SD: LAB3
```

Рисунок 2 - Иллюстрация работы .EXE-модуля и корректной установки и размещения прерывания

Шаг 3. Программа корректно определяет установленный обработчик прерываний.

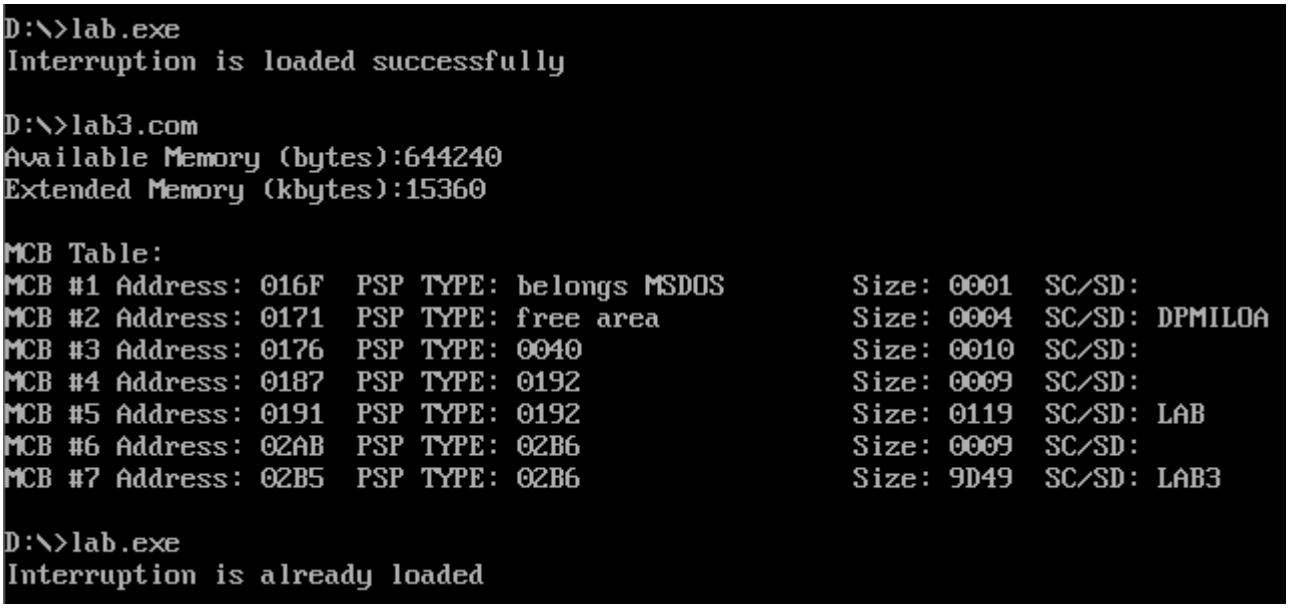

Рисунок 3 - Иллюстрация корректного определения установленного обработчика прерываний

Шаг 4. Была запущена отлаженная программа с ключом выгрузки. Резидентный обработчик был выгружен. Память была успешна освобождена.

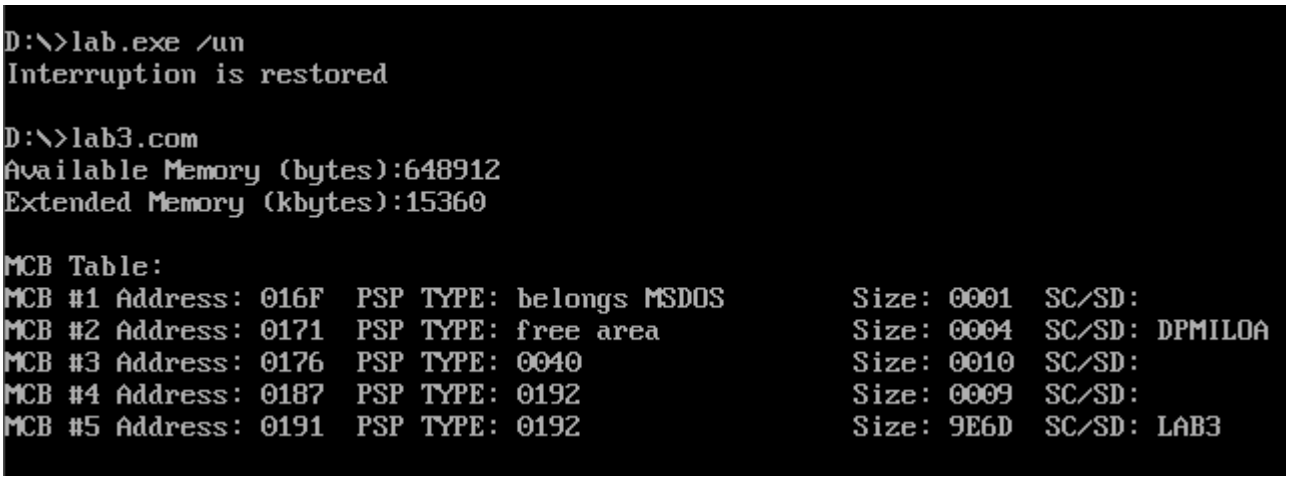

Рисунок 4 - Иллюстрация корректной выгрузки обработчика прерывания

Шаг 5. Была произведена оценка результатов и были отвечены контрольные вопросы:

1. ***Какого типа прерывания использовались в работе?***

   Ответ: были использованы: 09h, 16h – аппаратное прерывание, 10h, 21h – пользовательские.

2. ***Чем отличается скан-код от кода ASCII?***

Ответ: скан-код — код клавиши клавиатуры, который обработчик прерываний от клавиатуры преобразует в некий код символа — например, код символа ASCII таблицы.

Выводы.

Были исследованы возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Был написан пользовательский обработчик прерывания, который получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре и обрабатывает скан-код, осуществляя определенные действия, если скан-код совпадает с определенными кодами, которые он должен обрабатывать. В случае же, если скан-код не совпадает с этими кодами, управление передается стандартному прерыванию.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab.asm

```
ASTACK SEGMENT STACK
   DW 200 DUP(?)
ASTACK ENDS

DATA SEGMENT
        interruption_already_loaded_string db 'Interruption is already loaded', 0DH, 0AH, '$'
        interruption_loaded_successfully_string db 'Interruption is loaded successfully', 0DH, 0AH, '$'
        interruption_not_loaded_string db 'Interruption is not loaded', 0DH, 0AH, '$'
        interruption_restored_string db 'Interruption is restored', 0DH, 0AH, '$'
        test_string db 'test', 0DH, 0AH, '$'
DATA ENDS

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

    WRITEWRD  PROC  NEAR
        push ax
        mov ah, 9
        int 21h
        pop ax
        ret
    WRITEWRD  ENDP

    WRITEBYTE  PROC  NEAR
        push ax
        mov ah, 02h
        int 21h
        pop ax
        ret
    WRITEBYTE  ENDP
```

```
ENDLINE PROC NEAR
    push ax
    push dx

    mov dl, 0dh
    call WRITEBYTE

    mov dl, 0ah
    call WRITEBYTE

    pop dx
    pop ax
    ret
ENDLINE ENDP

OUTPUTAL PROC NEAR
    push ax
    push bx
    push cx
    mov ah, 09h ;писать символ с текущей позиции курсора
    mov bh, 0 ;номер видео страницы
    mov cx, 1 ;число экземпляров символа для записи
    int 10h ;выполнить функцию
    pop cx
    pop bx
    pop ax
    ret
OUTPUTAL ENDP

OUTPUTBP PROC NEAR
    push ax
    push bx
    push dx
    push CX
    mov ah,13h ; функция
    mov al, 0 ; sub function code
    ; 1 = use attribute in BL; leave cursor at end of string
    mov bh,0 ; видео страница
    mov dh,22 ; DH,DL = строка, колонка (считая от 0)
    mov dl,0
    int 10h
    pop CX
```

```
    pop dx
    pop bx
    pop ax
    ret
OUTPUTBP ENDP

MY_INTERRUPTION PROC FAR
    jmp start

    STD_KEY db 0h
    SHIFT_PRESSED db 0
    interruption_signature dw 7777h

    int_keep_ip dw 0
    int_keep_cs dw 0
    psp_address dw ?
    int_keep_ss dw 0
    int_keep_sp dw 0
    int_keep_ax dw 0
    IntStack dw 64 dup(?)

start:
    mov int_keep_sp, sp
    mov int_keep_ax, ax
    mov ax, ss
    mov int_keep_ss, ax

    mov sp, OFFSET start
    mov ax, seg IntStack
    mov ss, ax

    mov ax, int_keep_ax

    push ax ;сохранение изменяемого регистра
    push cx ;сохранение изменяемого регистра
    push dx ;сохранение изменяемого регистра

    ;Само прерывание

    mov STD_KEY, 0h
    mov SHIFT_PRESSED, 0h
```

```
    mov ax, 40h
    mov es, ax
    mov ax, es:[17h]
    and ax, 11b
    cmp ax, 0h
    je read_symbol
    mov SHIFT_PRESSED, 1h

read_symbol:
    in al, 60h ;читать ключ
    cmp al, 10h ;это требуемый код?
    je key_q ; да, активизировать обработку
    cmp al, 11h
    je key_w

    ; нет, уйти на исходный обработчик
    mov STD_KEY, 1h
    jmp interuption_end

key_q:
    mov al, 'a'
    jmp do_req

key_w:
    mov al, 'z'
    jmp do_req

do_req:
    push ax
    ;следующий код необходим для отработки аппаратного прерывания
    in al, 61H  ; взять значение порта управления клавиатурой
    mov ah, al  ; сохранить его
    or al, 80h  ; установить бит разрешения для клавиатуры
    out 61H, al ; и вывести его в управляющий порт
    xchg ah, al ; извлечь исходное значение порта
    out 61H, al ; и записать его обратно
    mov al, 20H ; послать сигнал "конец прерывания"
    out 20H, al ; контроллеру прерываний 8259
    pop ax

    cmp SHIFT_PRESSED, 0h
    je print_key
```

```
        sub al, 20h

    print_key:
        mov ah, 05h ; Код функции
        mov cl, al ; Пишем символ в буфер клавиатуры
        mov ch, 00h ;
        int 16h ;
        or al, al ; проверка переполнения буфера
        jz interuption_end ; если не переполнен идем в конец прерывания
        mov ax, 0040h
        mov es, ax
        mov ax, es:[1ah]
        mov es:[1ch], ax
        jmp print_key

    interuption_end:
        ;Конец прерывания

        pop dx ;восстановление регистра
        pop cx ;восстановление регистра
        pop ax ;восстановление регистра

        mov sp, int_keep_sp
        mov ax, int_keep_ss
        mov ss, ax
        mov ax, int_keep_ax

        mov al, 20h     ;разрешаем обработку прерываний
        out 20h, al     ;с более низкими уровнями

        cmp STD_KEY, 1h
        jne interuption_iret

             jmp  dword  ptr  cs:[int_keep_ip]  ;переход  на  первоначальный
обработчик
    interuption_iret:
        iret ;конец прерывания

    MY_INTERRUPTION ENDP
    interruption_last_byte:

    CHECK_CLI_OPT PROC near
```

```
        push ax
    push bp

    mov cl, 0h

    mov bp, 81h

        mov al,es:[bp + 1]
        cmp al,'/'
        jne lafin

        mov al,es:[bp + 2]
        cmp al,'u'
        jne lafin

        mov al,es:[bp + 3]
        cmp al,'n'
        jne lafin

        mov cl, 1h

lafin:
    pop bp
        pop ax
        ret
CHECK_CLI_OPT ENDP

CHECK_LOADED PROC NEAR
    push ax
    push dx
    push es
    push si

    mov cl, 0h

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

    mov si, offset interruption_signature
    sub si, offset MY_INTERRUPTION
    mov dx, es:[bx + si]
```

```
    cmp dx, interruption_signature
    jne checked

    mov cl, 1h ;already loaded

checked:
    pop si
    pop es
    pop dx
    pop ax
    ret
CHECK_LOADED ENDP

LOAD_INTERRUPTION PROC near
        push ax
    push cx
        push dx


        call CHECK_LOADED
        cmp cl, 1h
        je int_already_loaded

    mov psp_address, es

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

    mov int_keep_cs, es
mov int_keep_ip, bx

  push es
    push bx
        push ds

        lea dx, MY_INTERRUPTION
        mov ax, SEG MY_INTERRUPTION
        mov ds, ax

        mov ah, 25h
        mov al, 09h
```

```
        int 21h

        pop ds
    pop bx
    pop es

    mov dx, offset interruption_loaded_successfully_string
        call WRITEWRD

        lea dx, interruption_last_byte
        mov cl, 4h
        shr dx, cl
        inc dx ;dx - size in paragraphs

        add dx, 100h

        xor ax,ax

        mov ah, 31h
        int 21h

    jmp fin_load_interruption

int_already_loaded:
  mov dx, offset interruption_already_loaded_string
    call WRITEWRD

fin_load_interruption:
        pop dx
    pop cx
        pop ax
        ret
LOAD_INTERRUPTION ENDP

UNLOAD_INTERRUPTION PROC near
        push ax
        push si

        call CHECK_LOADED
        cmp cl, 1h
        jne interruption_is_not_loaded
```

```
    cli

    push ds
    push es

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

    mov si, offset int_keep_ip
sub si, offset MY_INTERRUPTION
mov dx, es:[bx + si]
mov ax, es:[bx + si + 2]
      mov ds, ax

    mov ah, 25h
    mov al, 09h
    int 21h

    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

    pop es
    pop ds

    sti

    mov dx, offset interruption_restored_string
        call WRITEWRD
```

```
        jmp int_unloaded

    interruption_is_not_loaded:
        mov dx, offset interruption_not_loaded_string
        call WRITEWRD

    int_unloaded:
            pop si
            pop ax
            ret
    UNLOAD_INTERRUPTION ENDP

    MAIN PROC FAR
            mov   ax, DATA
            mov   ds, ax

        call CHECK_CLI_OPT
        cmp cl, 0h
        jne opt_unload

        call LOAD_INTERRUPTION
        jmp main_end

    opt_unload:

        call UNLOAD_INTERRUPTION

    main_end:
        xor al, al
        mov ah, 4ch
        int 21h

    MAIN ENDP


CODE ENDS

END MAIN
```