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 _____ Любимов В.А.

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.

Описание функций и структуры данных

1. MY_INTERRUPTION – реализация требуемого прерывания. Заменяет символы нажатия на клавиши ‘q’, ‘w’, ‘e’ на символы нажатия на ‘0’, ‘9’, ‘8’ соответственно.
2. IS_INTER_WORKING – проверяет, выполняется ли наше прерывание.
3. UNLOAD_INTER – выгружает наше прерывание и восстанавливает исходное состояние обработчика.
4. LOAD_INTER – загружает наше прерывание.
5. PRINT_MES – при помощи функции 9h из прерывания 21h выводит строку на экран.
6. IS_END_CMD – проверяет условие выгрузки нашего прерывания, то есть была ли программа запущена с флагом ‘/un’.

Ход выполнения работы

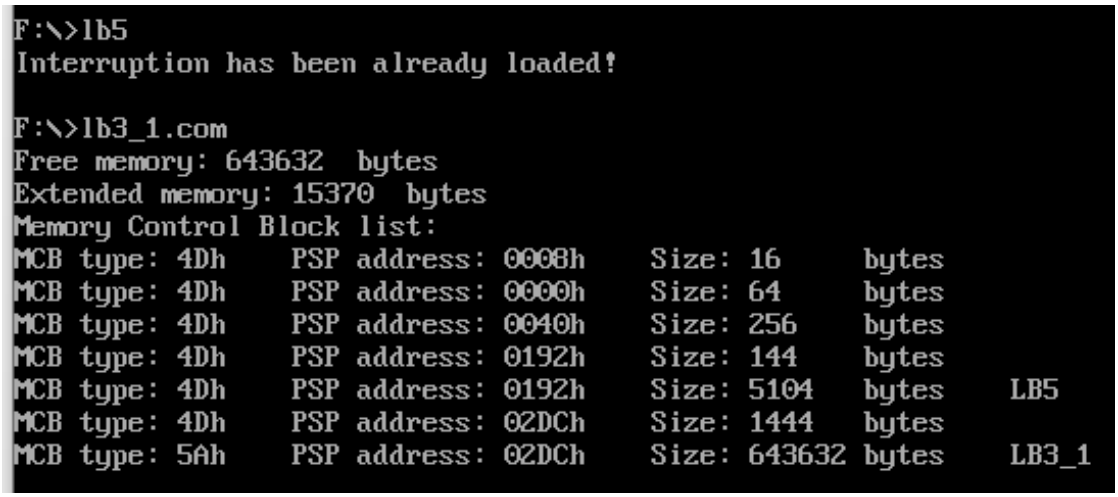
1. Разработан и написан .EXE модуль, реализующий следующие возможности:
   - Проверка установки пользовательского прерывания с вектором 09h.

- Если прерывание не установлено, то устанавливается резидентная функция для обработки прерывания и настраивает вектор прерываний. Производится выход по функции 4Ch прерывания 21h.
- Если прерывание установлено, то выводится сообщение об этом и производится выход по функции 4Ch прерывания 21h.
- Пользовательское прерывание заменяет символы нажатия на клавиши ‘q’, ‘w’, ‘e’ на символы нажатия на клавиши ‘0’, ‘9’, ‘8’ соответственно.
- Если программа запущена с флагом /un, то пользовательское прерывание выгружается, то есть восстанавливается стандартный вектор прерываний и освобождается память, занятая резидентом. Производится выход по функции 4Ch прерывания 21h.

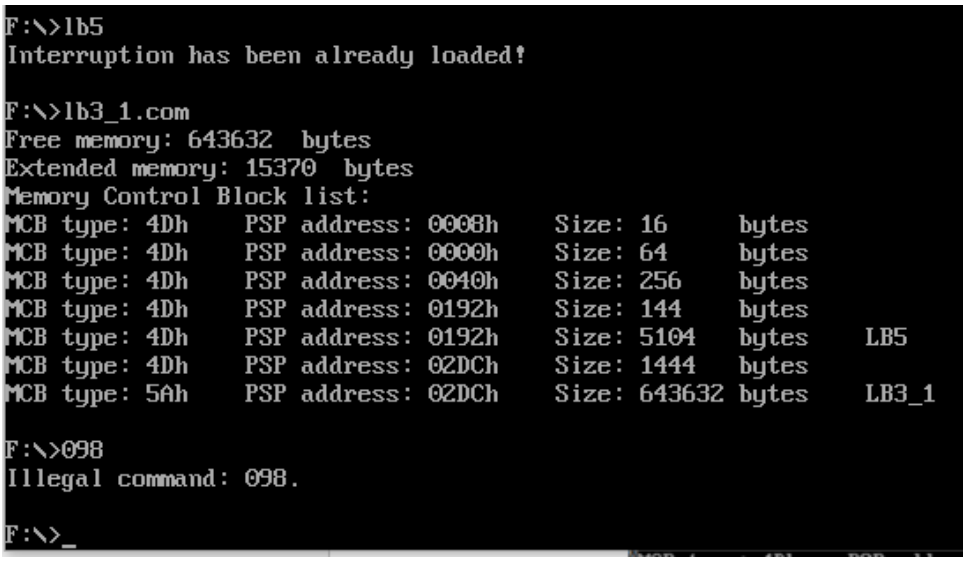
2. Запускаем программу. Прерывание загружается и выводится строка с информацией об этом. Также выведен список блоков MCB до и после запуска.

```
Free memory: 648912  bytes
Extended memory: 15370  bytes
Memory Control Block list:
MCB type: 4Dh    PSP address: 0008h    Size: 16     bytes
MCB type: 4Dh    PSP address: 0000h    Size: 64     bytes
MCB type: 4Dh    PSP address: 0040h    Size: 256    bytes
MCB type: 4Dh    PSP address: 0192h    Size: 144    bytes
MCB type: 5Ah    PSP address: 0192h    Size: 648912 bytes    LB3_1

F:\>lb5
Interruption is loaded!

F:\>lb3_1.com
Free memory: 643632  bytes
Extended memory: 15370  bytes
Memory Control Block list:
MCB type: 4Dh    PSP address: 0008h    Size: 16     bytes
MCB type: 4Dh    PSP address: 0000h    Size: 64     bytes
MCB type: 4Dh    PSP address: 0040h    Size: 256    bytes
MCB type: 4Dh    PSP address: 0192h    Size: 144    bytes
MCB type: 4Dh    PSP address: 0192h    Size: 5104   bytes    LB5
MCB type: 4Dh    PSP address: 02DCh    Size: 1444   bytes
MCB type: 5Ah    PSP address: 02DCh    Size: 643632 bytes    LB3_1
```

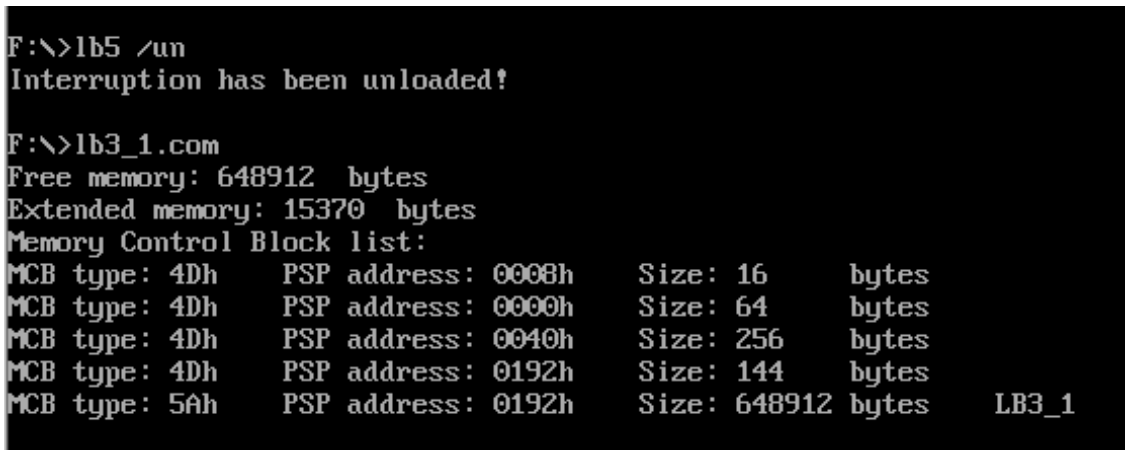
3. Запускаем программу повторно. Выведено сообщение о том, что прерывание уже загружено. Обработчик прерываний корректно определяется, то есть остался резидентным в памяти.



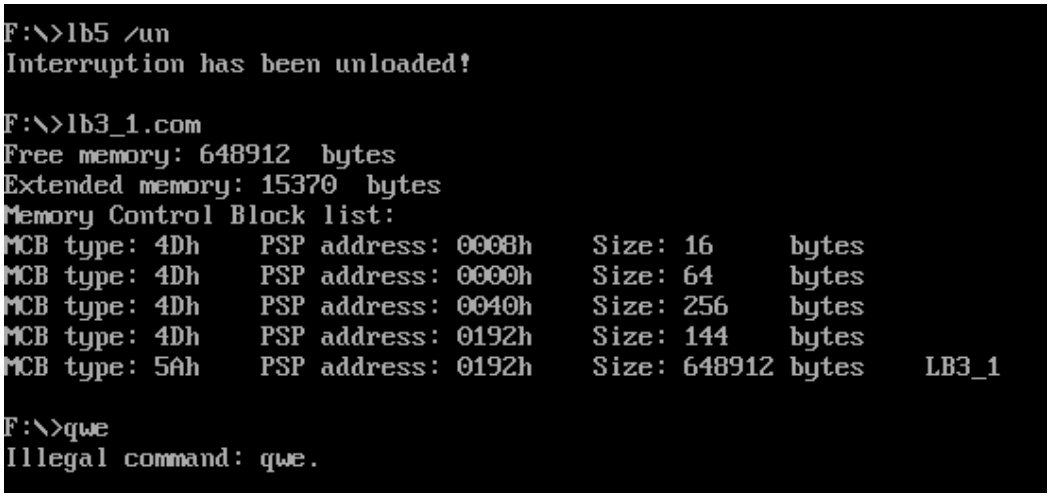
4. Вводим последовательность символов “qwe”. Вместо них будет введено “098”, так как прерывание запущено.



5. Программа запущена с ключом выгрузки /un. Выведенный список блоков управления памятью показывает, что обработчик прерывания выгружен и память освобождена. Счётчик более не отображается.



6. Вводим последовательность символов “qwe”. Будет введено “qwe”, так как прерывание выгружено.



Ответы на контрольные вопросы

1. Какого типа прерывания использовались в работе?

Аппаратное прерывание 9h, генерируемое при каждом нажатии на клавишу на клавиатуре. Программные прерывания функций BIOS (16h) и функций DOS (21h).

2. Чем отличается скан код от кода ASCII?

Скан-код является номером клавиши на клавиатуре, чтобы драйвер клавиатуры мог распознать какая клавиша нажата. Код ASCII является численной интерпретацией символа для таблицы ASCII. Итого, скан-код –

номер клавиши, а код ASCII – численной интерпретацией символа, для которого клавиша может и не существовать клавиша на клавиатуре.

Вывод

В ходе выполнения работы была изучена возможность встраивания пользовательского прерывания в стандартный обработчик прерываний от клавиатуры.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

lab5.asm

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

MYSTACK  SEGMENT STACK
 DW  256 DUP(0)
MYSTACK  ENDS

DATA SEGMENT
       no_inter_loaded_mes db "Interruption is not loaded!", 0DH, 0AH, '$'
       unload_mes db "Interruption has been unloaded!", 0DH, 0AH, '$'
       already_load_mes db "Interruption has been already loaded!", 0DH, 0AH, '$'
       load_mes db "Interruption is loaded!", 0DH, 0AH, '$'
    is_inter_loaded db 0
    is_inter_unloaded db 0
DATA ENDS

MY_INTERRUPTION PROC FAR
    jmp  Start

inter_data:
    key_value db 0
    inter_signature DW 0fedch
    new_stack DW 256 DUP(0)
    keep_ip DW 0
    keep_cs DW 0
    psp_state 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, 10h
    je key_is_q

    cmp al, 11h
    je key_is_w

    cmp al, 12h
```

```
    je key_is_e

    pushf
    call DWord ptr CS:keep_ip
    jmp end_of_my_interruption

key_is_q:
    mov key_value, '0'
    jmp next_key

key_is_w:
    mov key_value, '9'
    jmp next_key

key_is_e:
    mov key_value, '8'

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

show_key:
    mov ah, 05h
    mov cl, key_value
    mov ch, 00h
    int 16h
    or        al, al
    jz        end_of_my_interruption
    mov ax, 0040h
    mov es, ax
    mov ax, es:[1ah]
    mov es:[1ch], ax
    jmp show_key

end_of_my_interruption:
    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
MY_INTERRUPTION ENDP

UNLOAD_INTER PROC NEAR
    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 MY_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_INTER ENDP

AN_END:

IS_INTER_WORKING PROC NEAR
    push ax
    push bx
    push si

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

    mov  si, offset inter_signature
    sub  si, offset MY_INTERRUPTION
    mov  ax, es:[bx + si]
    cmp       ax, inter_signature
    jne  check_end
    mov  is_inter_loaded, 1

check_end:
    pop  si
    pop  bx
    pop  ax
    ret
IS_INTER_WORKING ENDP
```

```
LOAD_INTER PROC NEAR
    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 MY_INTERRUPTION
    mov dx, offset MY_INTERRUPTION
    mov ds, ax
    mov ah, 25h
    mov al, 09h
    int 21h
    pop ds

    mov dx, offset AN_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
LOAD_INTER  ENDP

IS_END_CMD PROC NEAR
       push   es

       mov    ax, psp_state
       mov    es, ax

       mov    al, es:[81h+1]
       cmp    al, '/'
       jne    wrong_cmd

       mov    al, es:[81h+2]
       cmp    al, 'u'
       jne    wrong_cmd

       mov    al, es:[81h+3]
       cmp    al, 'n'
       jne    wrong_cmd

       mov    is_inter_unloaded, 0001h
wrong_cmd:
       pop es
       ret
IS_END_CMD ENDP

PRINT_MES PROC NEAR
       push   ax
       mov    ah, 09h
```

```
       int       21h
       pop    ax
       ret
PRINT_MES ENDP

MAIN_PROC PROC
    push ds
    xor ax, ax
    push ax
    mov ax, DATA
    mov ds, ax
    mov psp_state, es

    call IS_INTER_WORKING
    call IS_END_CMD

    cmp  is_inter_unloaded, 1
    je unload_state

    mov al, is_inter_loaded
    cmp al, 1
    jne load_state
    mov dx, offset already_load_mes
    call PRINT_MES
    jmp end_main

load_state:
    mov dx, offset load_mes
    call PRINT_MES
    call LOAD_INTER
    jmp  end_main

unload_state:
    cmp  is_inter_loaded, 1
    jne  no_load_state
    mov dx, offset unload_mes
    call PRINT_MES
    call UNLOAD_INTER
    jmp  end_main

no_load_state:
    mov  dx, offset no_inter_loaded_mes
    call PRINT_MES

end_main:
    xor al, al
    mov ah, 4ch
    int 21h
MAIN_PROC ENDP
CODE ENDS
end MAIN_PROC
```