

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

Студент гр. 8383

Колмыков В.Д.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

Процедуры, используемые в работе.

Название процедуры	Описание
MAIN	Головная процедура
INTERRUPT	Обработчик прерывания
CHECK_INTERRUPT	Проверка, не загружено ли уже прерывание, которое требуется загрузить
WRITE	Вывод строки из DX
LOAD_INTERRUPT	Загрузка и сохранение обработчика прерывания в памяти
CHECK_PARAM	Проверка параметров командной строки
UNILOAD_INTERRUPT	Восстановление вектора прерывания и очищение памяти

Ход работы.

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, если она еще не установлена
- 3) Если она уже остановлена выводится соответствующее сообщение
- 4) Выгрузка прерывания по значению параметра в командной строке /un

Код программы приведен в приложении А.

Начальное состояние памяти, полученное при помощи программы из ЛР3 продемонстрировано на рис. 1.

Avaiable memory: 648912 bytes
Memory was free successfully
Extended memory: 15360 kbytes

MCB number 1
Owner: MS DOS
Area size: 16 bytes

MCB number 2
Owner: free
Area size: 64 bytes

MCB number 3
Owner: 0040
Area size: 256 bytes

MCB number 4
Owner: 0192
Area size: 144 bytes

MCB number 5
Owner: 0192
Area size: 1488 bytes
LR3_2

MCB number 6
Owner: free
Area size: 647408 bytes

Рисунок 1 – Состояние памяти до загрузки прерывания

Результат загрузки прерывания показан на рис. 2. На клавиатуре было набрано «qwer».



```
C:\>lr5
C:\>ULAD_
```

Рисунок 2 – Результат работы прерывания

Состояние памяти с загруженным прерыванием приведено на рис. 3.

```
Avaiable memory: 648016 bytes
Memory was free successfully
Extended memory: 15360 kbytes
```

```
MCB number 1
Owner: MS DOS
Area size: 16 bytes
```

```
MCB number 2
Owner: free
Area size: 64 bytes
```

```
MCB number 3
Owner: 0040
Area size: 256 bytes
```

```
MCB number 4
Owner: 0192
Area size: 144 bytes
```

```
MCB number 5
Owner: 0192
Area size: 720 bytes
LR5
```


```
MCB number 6
Owner: 01CA
Area size: 144 bytes
```

```
MCB number 7
Owner: 01CA
Area size: 1488 bytes
LR3_2
```

```
MCB number 8
Owner: free
Area size: 646512 bytes
```

Рисунок 3 – Состояние памяти с загруженным прерыванием

Результат повторной попытки загрузить прерывание показан на рис. 4.



```
C:\>lr5
Interrupt already exist
C:\>
```

Рисунок 4 – Результат повторного запуска программы

Результат выгрузки прерывания показан на рис. 5.



```
C:\>lr5 /un
C:\>qwer_
```

Рисунок 5 – Результат выгрузки прерывания из памяти

Состояние памяти после выгрузки прерывание показано на рис. 6.

Avaiable memory: 648912 bytes
Memory was free successfully
Extended memory: 15360 kbytes

MCB number 1
Owner: MS DOS
Area size: 16 bytes

MCB number 2
Owner: free
Area size: 64 bytes

MCB number 3
Owner: 0040
Area size: 256 bytes

MCB number 4
Owner: 0192
Area size: 144 bytes

MCB number 5
Owner: 0192
Area size: 1488 bytes
LR3_2

MCB number 6
Owner: free
Area size: 647408 bytes

Рисунок 6 – Состояние памяти после выгрузки прерывания из памяти

Ответы на контрольные вопросы.

1) Какого типа прерывания использовались в работе?

Был реализован обработчик для аппаратного прерывания (от клавиатуры), в коде программы также использовались программные прерывания (например 21h).

2) Чем отличается скан-код и ASCII код?

Скан-код – код клавиши, позволяющий опознавать нажатые клавиши драйверу клавиатуры. ASCII код – код символа для печати на экран.

Выводы.

В ходе выполнения лабораторной работы была реализована программа загружающая и выгружающая прерывание от клавиатуры.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

```
CODE SEGMENT
    assume CS:CODE, DS:DATA, SS:STACKK, ES:NOTHING

    INTERRUPT PROC FAR
        jmp INTERRUPT_START
        SYMB db 0
        INTERRUPT_ID dw 0804h
        SAVE_AX dw 0
        SAVE_SS dw 0
        SAVE_SP dw 0
        KEEP_IP dw 0
        KEEP_CS dw 0
        PSP_SEGMENT DW 0
        INTERRUPTION_STACK dw 128 dup(0);Свой стек для обработчика

    INTERRUPT_START:
        mov SAVE_AX, AX
        mov SAVE_SP, SP
        mov SAVE_SS, SS
        mov AX, SEG INTERRUPTION_STACK
        mov SS, AX
        mov AX, offset INTERRUPTION_STACK
        add AX, 256
        mov SP, AX
        push BX
        push CX
        push DX
        push SI
        push DS
        push BP
        push ES
        mov AX, SEG SYMB
        mov DS, AX

        in AL, 60h
        cmp AL, 10h
        je Q_BUTTON
        cmp AL, 11h
        je W_BUTTON
        cmp AL, 12h
        je E_BUTTON
        cmp AL, 13h
        je R_BUTTON
        pushf
        call DWORD PTR CS:KEEP_IP
        jmp INTERRUPT_END

    Q_BUTTON:
        mov SYMB, 'V'
        jmp AFTER_SYMB_SELECT
    W_BUTTON:
        mov SYMB, 'L'
        jmp AFTER_SYMB_SELECT
    E_BUTTON:
        mov SYMB, 'A'
        jmp AFTER_SYMB_SELECT
    R_BUTTON:
        mov SYMB, 'D'
```

```

AFTER_SYMB_SELECT:
    in AL, 61h
    mov AH, AL
    or AL, 80h
    out 61h, AL
    xchg AL, AL
    out 61h, AL
    mov AL, 20h
    out 20h, AL

WRITE_KEYBOARD_BUFF:
    mov AH, 05h
    mov CL, SYMB
    mov CH, 00h
    int 16h
    or AL, AL
    jz INTERRUPT_END

    mov AX, 0040h
    mov ES, AX
    mov AX, ES:[1Ah]
    mov ES:[1Ch], AX
    jmp WRITE_KEYBOARD_BUFF

INTERRUPT_END:
    pop ES
    pop BP
    pop DS
    pop SI
    pop DX
    pop CX
    pop BX
    mov SP, SAVE_SP
    mov AX, SAVE_SS
    mov SS, AX
    mov AX, SAVE_AX
    mov AL, 20h
    out 20h, AL
    IRET
    ret

INTERRUPT ENDP
END_OF_INTERRUPT:
;
CHECK_INTERRUPT PROC
    push AX
    push BX
    push ES
    push SI

    mov AH, 35h
    mov AL, 09h;Номер прерывания
    int 21h
    mov SI, offset INTERRUPT_ID
    sub SI, offset INTERRUPT
    mov AX, ES:[BX + SI]
    cmp AX, 0804h
    jne CHECK_INTERRUPT_END
    mov INTERRUPT_LOADED, 1

CHECK_INTERRUPT_END:
    pop SI
    pop ES

```



```

        pop BX
        pop AX
        ret
CHECK_INTERRUPT ENDP
;
WRITE      PROC NEAR
        PUSH AX
        MOV  AH, 09H
        INT 21H
        POP AX
        RET
WRITE      ENDP
;
LOAD_INTERRUPT PROC
        push AX
        push BX
        push CX
        push DX
        push DS
        push ES

        mov AH, 35h
        mov AL, 09h
        int 21h
        mov KEEP_CS, ES
        mov KEEP_IP, BX
        push DS
        mov DX, offset INTERRUPT
        mov AX, SEG INTERRUPT
        mov DS, AX
        mov AH, 25h
        mov AL, 09h
        int 21h
        pop DS
        mov DX, offset END_OF_INTERRUPT
        add DX, 10Fh
        mov CL, 4h
        shr DX, CL
        inc DX
        xor AX, AX
        mov AH, 31h
        int 21h

        pop ES
        pop DS
        pop DX
        pop CX
        pop BX
        pop AX
        ret
LOAD_INTERRUPT ENDP
;
CHECK_PARAM PROC
        push AX
        push ES

        mov AX, PSP_SEGMENT
        mov ES, AX
        cmp byte ptr ES:[82h], '/'
        jne CHECK_PARAM_END
        cmp byte ptr ES:[83h], 'u'
        jne CHECK_PARAM_END
        cmp byte ptr ES:[84h], 'n'

```

```

        jne CHECK_PARAM_END
        mov UN_PARAM, 1

CHECK_PARAM_END:
        pop ES
        pop AX
        ret
CHECK_PARAM 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 INTERRUPT
        mov DX, ES:[BX + SI];Смещение
        mov AX, ES:[BX + SI + 2];Сегмент
        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

        pop SI
        pop ES
        pop DS
        pop DX
        pop BX
        pop AX
        STI
        ret
UNLOAD_INTERRUPT ENDP

;
MAIN PROC
        push DS
        xor AX, AX
        push AX
        mov AX, DATA
        mov DS, AX
        mov PSP_SEGMENT, ES

        call CHECK_INTERRUPT
        call CHECK_PARAM
        cmp UN_PARAM, 1

```

```

        je MAIN_UNLOAD
        mov AL, INTERRUPT_LOADED
        cmp AL, 1
        jne MAIN_LOAD
        mov DX, offset STR_INTERRUPT_EXIST
        call WRITE
        jmp MAIN_END
MAIN_LOAD:
        call LOAD_INTERRUPT
        jmp MAIN_END
MAIN_UNLOAD:
        cmp INTERRUPT_LOADED, 1
        jne NOT_EXIST
        call UNLOAD_INTERRUPT
        jmp MAIN_END
NOT_EXIST:
        mov DX, offset STR_NOT_EXIST
        call WRITE
MAIN_END:
        xor AL, AL
        mov AH, 4Ch
        int 21h
MAIN ENDP
; _____
CODE ENDS

STACKK SEGMENT STACK
        dw 128 dup(0)
STACKK ENDS

DATA SEGMENT
        INTERRUPT_LOADED db 0
        UN_PARAM db 0
        STR_INTERRUPT_EXIST db 'Interrupt already exist$'
        STR_NOT_EXIST db 'Interrupt does not exist$'
DATA ENDS
END MAIN

```