

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)
КАФЕДРА МО ЭВМ

ОТЧЕТ
по лабораторной работе №4
по дисциплине «Операционные системы»
Тема: Обработка стандартных прерываний.

Студент гр. 0382

Афанасьев Н. С.

Преподаватели

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

В архитектуре компьютера существуют стандартные прерывания, за которыми закреплены определенные вектора прерываний. Вектор прерываний хранит адрес подпрограммы обработчика прерываний. При возникновении прерывания, аппаратура компьютера передает управление по соответствующему адресу вектора прерывания. Обработчик прерываний получает управление и выполняет соответствующие действия.

В лабораторной работе № 4 предлагается построить обработчик прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определенные интервалы времени и, при возникновении такого сигнала, возникает прерывание с определенным значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

Задание.

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет следующие функции:

- 1) Проверяет, установлено ли пользовательское прерывание с вектором 1Ch.
- 2) Устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и 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.
- 4) 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) Сохраняет стек прерванной программы (регистры SS и SP) в рабочих переменных и восстановить при выходе.
- 2) Организовать свой стек.
- 3) Сохранить значения регистров в стеке при входе и восстановить их при выходе.
- 4) При выполнении тела процедуры накапливать общее суммарное число прерываний и выводить на экран. Для вывода на экран следует использовать прерывание `int 10h`, которое позволяет непосредственно выводить информацию на экран.
- 5) Функция прерывания должна содержать только переменные, которые она использует.

Шаг 2. Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания `1Ch` установлен. Работа прерывания должна отображаться на экране, а 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 3. Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

Шаг 4. 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 5. Ответьте на контрольные вопросы.

Выполнение работы.

Шаг 1.

При работе были использованы/созданы следующие процедуры:

- TETR_TO_HEX, BYTE_TO_HEX, WRD_TO_HEX – процедуры, описанные в шаблоне, для перевода двоичных кодов в символы шестнадцатеричных чисел.
- PRINT – процедура для вывода в консоль строки, отступ на которую содержится в DX, на экран, используя функцию 09h прерывания 21h.
- IS_LOADED – процедура для проверки того, что пользовательский обработчик прерывания уже был установлен. Для этого сначала загружается вектор прерывания 1Ch через функцию 35h прерывания 21h, а затем проверяется сигнатура, установленная в созданном обработчике прерывания.
- LOAD_INT – процедура для установки обработчика прерывания. Сначала сохраняется оригинальный вектор прерывания, полученный через функцию 35h прерывания 21h. Затем устанавливается пользовательский обработчик прерывания через функцию 25h прерывания 21h. Затем используется функция 31h прерывания 21h, чтобы оставить процедуру прерывания резидентной в памяти.
- UNLOAD_INT – процедура для выгрузки пользовательского прерывания и возвращение оригинального, который запоминался на этапе установки, через

функции 35h и 25h прерывания 21h. Также происходит освобождение памяти через функцию 49h прерывания 21h.

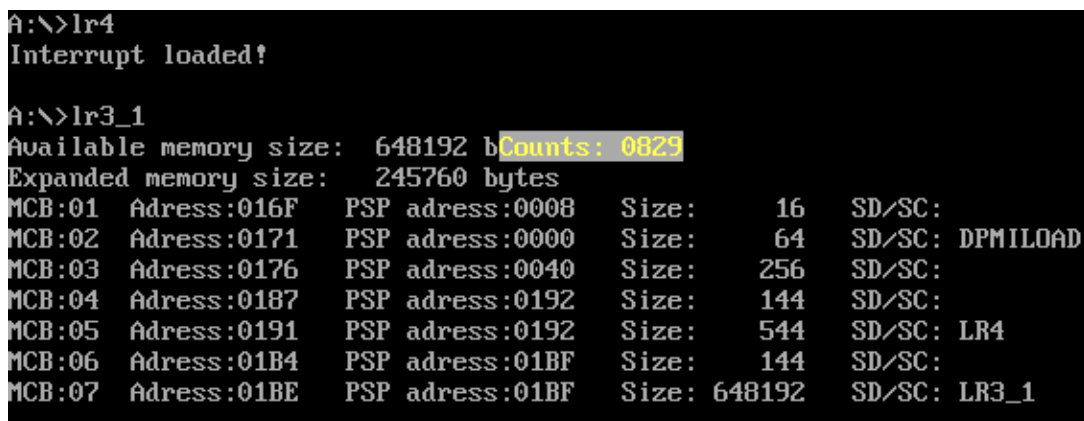
- CHECK_CMD_KEY – процедура для проверки ввода ключа “/un” при вызове программы через проверку байтов в блоке PSP, содержащего хвост командной строки.

- INTERRUPT – процедура, являющаяся обработчиком прерывания, при каждом вызове смещает курсор через функцию 02h прерывания int 10h, увеличивает значение вызовов процедуры (значение хранится в строке), выводит значение с помощью функцию 13h прерывания int 10h, после чего восстанавливает позицию курсора.

В процедуре MAIN вызываются вышеописанные процедуры при выполнении соответствующих условий (необходимость загрузки или выгрузки прерывания) и выводятся соответствующие информативные сообщения.

Шаг 2.

Программа была запущена, результаты проверены. Для проверки размещения прерывания в памяти была использована программа из предыдущей лабораторной работы (см. рис. 1).



```
A:\>lr4
Interrupt loaded!

A:\>lr3_1
Available memory size: 648192 bCounts: 0829
Expanded memory size: 245760 bytes
MCB:01 Address:016F PSP address:0008 Size: 16 SD/SC:
MCB:02 Address:0171 PSP address:0000 Size: 64 SD/SC: DPMILOAD
MCB:03 Address:0176 PSP address:0040 Size: 256 SD/SC:
MCB:04 Address:0187 PSP address:0192 Size: 144 SD/SC:
MCB:05 Address:0191 PSP address:0192 Size: 544 SD/SC: LR4
MCB:06 Address:01B4 PSP address:01BF Size: 144 SD/SC:
MCB:07 Address:01BE PSP address:01BF Size: 648192 SD/SC: LR3_1
```

Рис.1 – Выполнение программы и проверка размещения в памяти

Как видно из результатов, программа работает (значения вызовов отображаются на экране) и размещение прерывания в памяти подтверждено (прерывание занимает 2 блока общим размером в 688 байт).

Шаг 3.

Программа была запущена несколько раз: 2 раза без ключа и один раз с ключом (см. рис. 2).

```
A:\>lr4
Interrupt loaded!

A:\>lr4
Interrupt already loaded!

A:\>lr4
Interrupt already loaded!

A:\>lr4 /un
Interrupt unloaded!

A:\>lr4
Interrupt loaded!

A:\>lr4 /un
Interrupt unloaded!
```

Рис.2 – Повторное выполнение программы: с ключом и без

Как можно заметить программа видит установленное прерывание и не устанавливает его повторно. При запуске с ключом программа восстанавливает оригинальный вектор прерывания.

Шаг 4.

После выгрузки прерывания была вновь запущена программа из прошлой лабораторной работы (см. рис. 3)

```
A:\>lr4
Interrupt loaded!

A:\>lr4 /un
Interrupt unloaded!

A:\>lr3_1
Available memory size: 648912 bytes
Expanded memory size: 245760 bytes
MCB:01 Address:016F PSP address:0008 Size: 16 SD/SC:
MCB:02 Address:0171 PSP address:0000 Size: 64 SD/SC: DPMILOAD
MCB:03 Address:0176 PSP address:0040 Size: 256 SD/SC:
MCB:04 Address:0187 PSP address:0192 Size: 144 SD/SC:
MCB:05 Address:0191 PSP address:0192 Size: 648912 SD/SC: LR3_1
```

Рис.3 – Выгрузка прерывания и удаление из памяти

Как можно видеть, в памяти не осталось блоков, связанных с пользовательским прерыванием.

Программный код см. в Приложении А

Вопросы.

1) Как реализован механизм прерывания от часов?

– Прерывание 1Ch вызывается с каждым тиком аппаратных часов (приблизительно 18.2 раз в секунду). Изначально прерывание указывает на команду IRET, но смещение может быть переопределено программой. Запоминается содержимое регистра флагов, а также CS:IP для возврата. Затем выполняется само прерывание, после чего управление возвращается прерванной программе.

2) Какого типа прерывания использовались в работе?

– int 10h (видеосервис – функция BIOS), int 21h (функции DOS), в том числе пользовательское прерывание по вектору 1Ch

Выводы.

В ходе выполнения лабораторной работы был построен обработчик прерываний сигналов таймера, который выводит информацию о количестве вызовов на экран.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЕ КОДЫ ПРОГРАММ

Название файла: lr4.asm

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

    INTERRUPT PROC FAR
        jmp int_start
        PSP                DW ?
        KEEP_CS            DW ?
        KEEP_IP            DW ?
        KEEP_SS            DW ?
        KEEP_SP            DW ?
        INT_ID             DW 0ABCDh
        COUNTER            DB 'Counts: 0000'
        INT_STACK          DB 128 dup(?)

        int_start:
            mov KEEP_SS, SS
            mov KEEP_SP, SP
            mov SP, seg INTERRUPT
            mov SS, SP
            mov SP, OFFSET int_start
            push DS
            push ES
            push AX
            push BX
            push CX
            push DX
            push SI
            push BP

            mov AH, 03h
            mov BH, 0
            int 10h
            push DX

            mov AH, 02h
            mov BH, 0
            mov DL, 20h
            mov DH, 5h
            int 10h

            mov SI, SEG COUNTER
            mov DS, SI
            mov SI, OFFSET COUNTER
            add SI, 7

            mov CX, 4
            num_loop:
                mov BP, CX
                mov AH, [SI+BP]
                inc AH
```



```

        mov [SI+BP], AH
        cmp AH, 3Ah
        jne num_loop_end
        mov AH, 30h
        mov [SI+BP], AH
        loop num_loop
num_loop_end:

mov BP, SEG COUNTER
mov ES, BP
mov BP, OFFSET COUNTER
mov AH, 13h
mov AL, 1
mov BH, 0
mov CX, 12
int 10h

mov AH, 02h
mov BH, 0
pop DX
int 10h

pop BP
pop SI
pop DX
pop CX
pop BX
    pop AX
    pop ES
pop DS
mov SP, KEEP_SS
mov SS, SP
mov SP, KEEP_SP
mov AL, 20h
out 20h, AL
iret
    int_last_byte:
INTERRUPT ENDP

IS_LOADED PROC NEAR
    push AX
    push BX
    push DX
    push SI

    mov FLAG, 1
    mov AH, 35h
    mov AL, 1Ch
    int 21h
    mov SI, OFFSET INT_ID
    sub SI, OFFSET INTERRUPT
    mov DX, ES:[BX+SI]
    cmp DX, 0ABCDh
    je loaded
    mov FLAG, 0

    loaded:
    pop SI

```

```

    pop DX
    pop BX
    pop AX
    ret
IS_LOADED ENDP

LOAD_INT PROC NEAR
    push DS
    push ES
    push AX
    push BX
    push CX
    push DX

    MOV AH, 35h
    MOV AL, 1Ch
    INT 21h
    MOV KEEP_IP, BX
    MOV KEEP_CS, ES

    mov DX, offset INTERRUPT
    mov AX, seg INTERRUPT
    mov DS, AX
    mov AH, 25h
    mov AL, 1Ch
    int 21h

    mov DX, offset int_last_byte
    mov CL, 4
    shr DX, CL
    inc DX
    mov AX, CS
    sub AX, PSP
    add DX, AX
    xor AX, AX
    mov AH, 31h
    int 21h

    pop DX
    pop CX
    pop BX
    pop AX
    pop ES
    pop DS
    ret
LOAD_INT ENDP

UNLOAD_INT PROC NEAR
    push DS
    push ES
    push AX
    push BX
    push DX

    cli
    mov AH, 35h
    mov AL, 1Ch
    int 21h

```

```

        mov DX, ES:[offset KEEP_IP]
        mov AX, ES:[offset KEEP_CS]
        mov DS, AX
        mov AH, 25h
        mov AL, 1Ch
        int 21h

        mov AX, ES:[offset PSP]
        mov ES, AX
        mov DX, ES:[2ch]
        mov AH, 49h
        int 21h
        mov ES, DX
        mov AH, 49h
        int 21h
        sti

        pop DX
        pop BX
        pop AX
        pop ES
        pop DS
        ret
UNLOAD_INT ENDP

CHECK_CMD_KEY PROC NEAR
    push AX

    mov FLAG, 0
    mov AL, ES:[82h]
    cmp AL, '/'
    jne no_key
    mov AL, ES:[83h]
    cmp AL, 'u'
    jne no_key
    mov AL, ES:[84h]
    cmp AL, 'n'
    jne no_key
    mov FLAG, 1

    no_key:
    pop AX
    ret
CHECK_CMD_KEY ENDP

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

main PROC FAR
    push DS
    xor AX, AX
    mov AX, DATA
    mov DS, AX

```

```

    mov PSP, ES
    call CHECK_CMD_KEY
    cmp FLAG, 1
    je int_unload

    call IS_LOADED
    cmp FLAG, 0
    je int_load
    mov DX, OFFSET ALREADY_LOADED_MSG
    call PRINT
    jmp exit

    int_load:
        mov DX, OFFSET LOADED_MSG
        call PRINT
        call LOAD_INT
        jmp exit

    int_unload:
        call IS_LOADED
        cmp FLAG, 0
        je unloaded
        call UNLOAD_INT
        unloaded:
            mov DX, OFFSET UNLOADED_MSG
            call PRINT

    exit:
        pop DS
        mov AH, 4Ch
        int 21h
main ENDP
CODE ENDS

ASTACK SEGMENT STACK
    DW 128 DUP(?)
ASTACK ENDS

DATA SEGMENT
    FLAG DB 0
    LOADED_MSG DB 'Interrupt loaded!', 0DH, 0AH, '$'
    UNLOADED_MSG DB 'Interrupt unloaded!', 0DH, 0AH, '$'
    ALREADY_LOADED_MSG DB 'Interrupt already loaded!', 0DH, 0AH, '$'
DATA ENDS
END main

```