

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

Гудов Н.Р.

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.

Программный код см. в Приложении А

Шаг 1

Написан и отлажен программный модуль типа .EXE, выполняющий поставленные задачи:

- 1) Проверка установки пользовательского прерывания.
- 2) Установка резидентной функции для обработки прерывания.
- 3) Вывод сообщения, в случае, если прерывание уже установлено.
- 4) Выгрузка прерывания по значению параметра командной строки.

Разработаны следующие функции:

PRINT- процедура печати строки, с помощью функции 9h прерывания 21h.

INTER- обработчик прерывания. Использует функции 02h и 13h прерывания 10h для смещения курсора и вывода значения.

IS_LOADED- Прочитывает адрес из вектора прерывания. Сравнивает известное значение сигнатуры с реальным кодом в резиденте.

LOAD_INT- устанавливает обработчик прерывания, предварительно сохраняя первоначальный. Для выполнения этих действий используются функции 35h и 25 h прерывания 21h.

UNLOAD_INT- Возвращает первоначальный вариант прерывания. Для выполнения этих действий используются функции 35h и 25 h

прерывания 21h.

CHECK- проверка установки прерывания.

Шаг 2

Зафиксируем информацию о состоянии блоков MCB перед началом работы программы.(рис1)

```
B:\>lab31
Available size:      648912
Expanded size:      245760
MCB:01 Address:016F PSP address:0008 Size: 16 SD/SC:
MCB:02 Address:0171 PSP address:0000 Size: 64 SD/SC:
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: LAB31
```

Рисунок 1

Далее запустим программу с пользовательским прерыванием.
После нее снова выведем информацию о блоках MCB.(рис2)

```
B:\>lab4
-INTERRUPT LOADED!-
Calls : 2913
B:\>lab31
Available size:      647840
Expanded size:      245760
MCB:01 Address:016F PSP address:0008 Size: 16 SD/SC:
MCB:02 Address:0171 PSP address:0000 Size: 64 SD/SC:
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: 896 SD/SC: LAB4
MCB:06 Address:01CA PSP address:01D5 Size: 144 SD/SC:
MCB:07 Address:01D4 PSP address:01D5 Size: 647840 SD/SC: LAB31
```

Рисунок 2

Шаг 3

Запустим программу заново.(рис3)

```
B:\>lab4
-INTERRUPT ALREADY LOADED!-
```

Рисунок 3

Шаг 4

Вернем исходное прерывание и посмотрим на состояние блоков MCB.(рис4)

```
B:\>lab4 /un
- INTERRUPT UNLOADED!-!

B:\>lab31
Available size:      648912
Expanded size:      245760
MCB:01 Address:016F  PSP address:0008  Size:    16  SD/SC:
MCB:02 Address:0171  PSP address:0000  Size:    64  SD/SC:
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: LAB31
```

Рисунок 4

Заметим, что блоки, хранящие пользовательское прерывания удалены – прерывание выгружено.

Вопросы.

- 1) Как реализован механизм прерывания от часов?

Механизм прерывания от часов работает следующим образом:

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?

В работе использовались 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. Изучен механизм прерываний в DOS.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЕ КОДЫ ПРОГРАММ

Название файла: lab4.asm

```
Astack SEGMENT STACK
    DW 128 DUP(?)
Astack ENDS
```

```
DATA SEGMENT
    flag          DB 0
    msg_loaded     DB '-INTERRUPT LOADED!-', 0DH, 0AH, '$'
    msg_unloaded   DB '-INTERRUPT UNLOADED!-', 0DH, 0AH, '$'
    msg_already    DB '-INTERRUPT ALREADY LOADED!-', 0DH, 0AH, '$'
DATA ENDS
```

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

;**ПРОЦЕДУРЫ**

;------

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

;------

```
INTER 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 'Calls : 0000'
    INT_STACK    DB 128 dup(?)
```

```
    int_start:
    mov KEEP_SS, SS
    mov KEEP_SP, SP
    mov SP, seg INTER
    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

```

```

        ired
        int_last_byte:
INTER 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 INTER
    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 INTER
    mov AX, seg INTER
    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 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 ENDP
;-----
main PROC FAR
    push DS
    xor AX, AX
    mov AX, DATA
    mov DS, AX

    mov PSP, ES
    call CHECK
    cmp flag, 1
    je int_unload
    call IS_LOADED
    cmp flag, 0
    je int_load
    mov DX, OFFSET msg_already
    call PRINT
    jmp final

    int_load:
    mov DX, OFFSET msg_loaded
    call PRINT
    call LOAD_INT
    jmp final

    int_unload:
    call IS_LOADED
    cmp flag, 0
    je unloaded
    call UNLOAD_INT
    unloaded:
    mov DX, OFFSET msg_unloaded
    call PRINT

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

END main

```