

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

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

Студент гр. 9383

Хотяков Е.П.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Постановка задачи.

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.

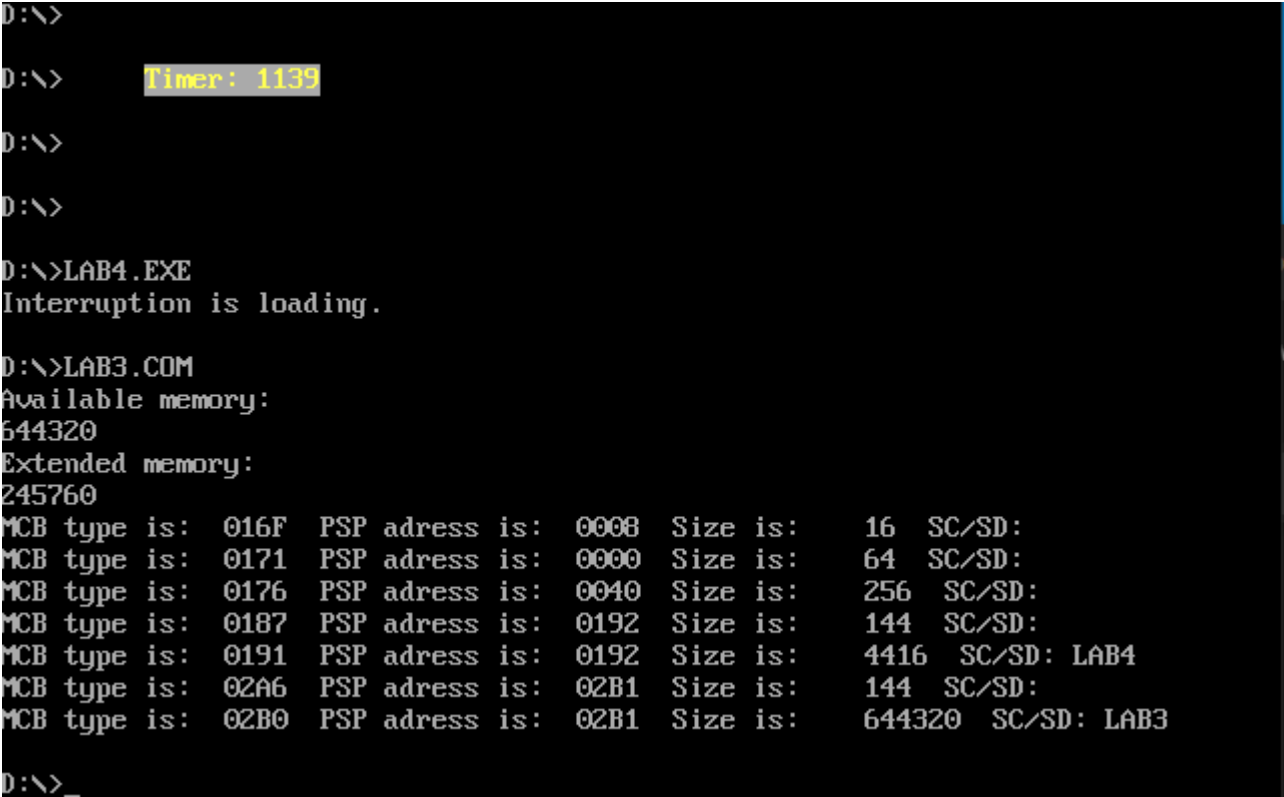
Шаг 4. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Шаг 5. Ответьте на контрольные вопросы.

Результаты исследования проблем.

Шаг 1. Был написан и отлажен программный модуль типа .EXE, который выполняет, выполняет требуемый функционал.

Шаг 2. Программа была отлажена и запущена. Проверено размещение прерывания в памяти(5ая строчка в таблице MCB).



```
D:\>
D:\>  Timer: 1139
D:\>
D:\>
D:\>LAB4.EXE
Interruption is loading.
D:\>LAB3.COM
Available memory:
644320
Extended memory:
245760
MCB type is: 016F  PSP adress is: 0008  Size is: 16  SC/SD:
MCB type is: 0171  PSP adress is: 0000  Size is: 64  SC/SD:
MCB type is: 0176  PSP adress is: 0040  Size is: 256  SC/SD:
MCB type is: 0187  PSP adress is: 0192  Size is: 144  SC/SD:
MCB type is: 0191  PSP adress is: 0192  Size is: 4416  SC/SD: LAB4
MCB type is: 02A6  PSP adress is: 02B1  Size is: 144  SC/SD:
MCB type is: 02B0  PSP adress is: 02B1  Size is: 644320  SC/SD: LAB3
D:\>_
```

Рисунок 1 – Демонстрация корректной работы резидентного обработчика прерываний

Шаг 3. Программа была повторно запущена, чтобы удостовериться, что программа определяет установленный обработчик прерываний.

```
D:\>  
Timer: 2900  
D:\>  
  
D:\>LAB4.EXE  
Interruption is loading.  
  
D:\>LAB3.COM  
Available memory:  
644320  
Extended memory:  
245760  
MCB type is: 016F PSP adress is: 0008 Size is: 16 SC/SD:  
MCB type is: 0171 PSP adress is: 0000 Size is: 64 SC/SD:  
MCB type is: 0176 PSP adress is: 0040 Size is: 256 SC/SD:  
MCB type is: 0187 PSP adress is: 0192 Size is: 144 SC/SD:  
MCB type is: 0191 PSP adress is: 0192 Size is: 4416 SC/SD: LAB4  
MCB type is: 02A6 PSP adress is: 02B1 Size is: 144 SC/SD:  
MCB type is: 02B0 PSP adress is: 02B1 Size is: 644320 SC/SD: LAB3  
  
D:\>LAB4.EXE  
Interruption has already loaded.  
  
D:\>
```

Рисунок 2 – Демонстрация корректного определения установленного обработчика прерывания при повторном запуске программы.

Шаг 4. Программа была запущена с ключом выгрузки, чтобы убедиться, что резидентный обработчик прерывания выгружен и память, занятая резидентом освобождена.

```

D:\>LAB4.EXE
Interruption has already loaded.

D:\>LAB4.EXE \un
Interruption has already loaded.

D:\>LAB4.EXE /un
Interruption was unloaded.

D:\>LAB3.COM
Available memory:
648912
Extended memory:
245760
MCB type is: 016F PSP adress is: 0008 Size is: 16 SC/SD:
MCB type is: 0171 PSP adress is: 0000 Size is: 64 SC/SD:
MCB type is: 0176 PSP adress is: 0040 Size is: 256 SC/SD:
MCB type is: 0187 PSP adress is: 0192 Size is: 144 SC/SD:
MCB type is: 0191 PSP adress is: 0192 Size is: 648912 SC/SD: LAB3
D:\>

```

Рисунок 3 – Демонстрация корректной выгрузки резидентного обработчика прерываний.

По итогам выполнения работы можно ответить на контрольные вопросы:

1. Как реализован механизм прерывания от часов?

Прерывание INT 1Ch вызывается обработчиком аппаратного прерывания от таймера INT 08h приблизительно 18,2 раза в секунду.

Сначала запоминаются значения регистров, определяется смещение по номеру источника прерывания в таблице векторов (2 байта в IP, два – в CS). Вызывается обработчик прерывания по сохраненному адресу.

В конце управление передается обратно от обработчика прерывания к прерванной программе.

2. Какого типа прерывания использовались в работе?

1Ch – аппаратное прерывание, 10h и 21h – программное прерывание, вызываемое командой int.

Выводы.

Построен собственный обработчик прерываний сигналов таймера. Получены дополнительные знания о работе с памятью (резидентный обработчик может быть загружен и выгружен из памяти).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

lab4.asm:

```
MY_STACK SEGMENT STACK
    DW 64 DUP(?)
MY_STACK ENDS

DATA SEGMENT
    INT_NOT_LOAD DB 'INTERRUPTION DID NOT LOAD.', 0DH, 0AH, '$'
    INT_IS_UNLOADED DB 'INTERRUPTION WAS UNLOADED.', 0DH, 0AH, '$'
    INT_LOADED DB 'INTERRUPTION HAS ALREADY LOADED.', 0DH, 0AH, '$'
    INT_IS_LOADING DB 'INTERRUPTION IS LOADING.', 0DH, 0AH, '$'
DATA ENDS

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

WRITE_STRING PROC NEAR
    PUSH AX
    MOV AH, 9H
    INT 21H
    POP AX
    RET
WRITE_STRING ENDP

START:
ROUT PROC FAR
    JMP START_PROC
    SAVED_PSP DW 0
    SAVED_IP DW 0
    SAVED_CS DW 0
    SAVED_SS DW 0
    SAVED_SP DW 0
    SAVED_AX DW 0

    INDEX DW 1337H
    TIMER DB 'TIMER: 0000$'
```

```

        BSTACK DW 64 DUP(?)
START_PROC:
        MOV SAVED_SP, SP
        MOV SAVED_AX, AX
        MOV AX, SS
        MOV SAVED_SS, SS

        MOV AX, SAVED_AX

        MOV SP, OFFSET START_PROC

        MOV AX, SEG BSTACK
        MOV SS, AX

        PUSH BX
        PUSH CX
        PUSH DX

        MOV AH, 3H
        MOV BH, 0H
        INT 10H
        PUSH DX

        PUSH SI
        PUSH CX
        PUSH DS
        PUSH AX
        PUSH BP

        MOV AX, SEG TIMER
        MOV DS, AX
        MOV SI, OFFSET TIMER

        ADD SI, 6
        MOV CX, 4

TIMER_INC:
        MOV BP, CX

```



```

MOV AH, [SI+BP]
INC AH
CMP AH, 3AH
JL TIMER_INC_END
MOV AH, 30H
MOV [SI+BP], AH

LOOP TIMER_INC

TIMER_INC_END:
MOV [SI+BP], AH

POP BP
POP AX
POP DS
POP CX
POP SI

PUSH ES
PUSH BP

MOV AX, SEG TIMER
MOV ES, AX
MOV AX, OFFSET TIMER
MOV BP, AX
MOV AH, 13H
MOV AL, 00H
MOV DH, 02H
MOV DL, 09H
MOV CX, 11
MOV BH, 0
INT 10H

POP BP
POP ES

;RETURN CURSOR
POP DX

```

```

MOV AH,02H
MOV BH,0H
INT 10H

POP DX
POP CX
POP BX

MOV SAVED_AX, AX
MOV SP, SAVED_SP
MOV AX, SAVED_SS
MOV SS, AX
MOV AX, SAVED_AX

MOV AL, 20H
OUT 20H, AL

IRET
END_ROUT:
ROUT ENDP

IF_NEED_UNLOAD PROC NEAR
    PUSH AX
    PUSH ES

    MOV AL,ES:[81H+1]
    CMP AL,'/'
    JNE END_IF_NEED_UNLOAD

    MOV AL,ES:[81H+2]
    CMP AL,'U'
    JNE END_IF_NEED_UNLOAD

    MOV AL,ES:[81H+3]
    CMP AL,'N'
    JNE END_IF_NEED_UNLOAD

    MOV CL,1H

```

```

END_IF_NEED_UNLOAD:
    POP ES
    POP AX
    RET
IF_NEED_UNLOAD ENDP

LOAD_ROUT PROC NEAR
    PUSH AX
    PUSH DX

    MOV SAVED_PSP, ES

    MOV AH, 35H
    MOV AL, 1CH
    INT 21H
    MOV SAVED_IP, BX
    MOV SAVED_CS, ES

    PUSH DS
    LEA DX, ROUT
    MOV AX, SEG ROUT
    MOV DS, AX
    MOV AH, 25H
    MOV AL, 1CH
    INT 21H
    POP DS

    LEA DX, END_ROUT
    MOV CL, 4H
    SHR DX, CL
    INC DX
    ADD DX, 100H
    XOR AX, AX
    MOV AH, 31H
    INT 21H

```

```

        POP DX
        POP AX
        RET
LOAD_ROUT ENDP

UNLOAD_ROUT PROC NEAR
        PUSH AX
        PUSH SI

        CLI
        PUSH DS
        MOV AH,35H
        MOV AL,1CH
        INT 21H

        MOV SI,OFFSET SAVED_IP
        SUB SI,OFFSET ROUT
        MOV DX,ES:[BX+SI]
        MOV AX,ES:[BX+SI+2]
        MOV DS,AX
        MOV AH,25H
        MOV AL,1CH
        INT 21H
        POP DS

        MOV AX,ES:[BX+SI-2]
        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 AX
        RET
UNLOAD_ROUT ENDP

IF_LOADED PROC NEAR
        PUSH AX
        PUSH SI

        PUSH ES
        PUSH DX

        MOV AH, 35H
        MOV AL, 1CH
        INT 21H

        MOV SI, OFFSET INDEX
        SUB SI, OFFSET ROUT
        MOV DX, ES:[BX+SI]
        CMP DX, INDEX
        JNE END_IF_LOADED
        MOV CH, 1H

END_IF_LOADED:
        POP DX
        POP ES
        POP SI
        POP AX
        RET
IF_LOADED ENDP

MAIN PROC FAR
        PUSH DS
        PUSH AX
        MOV AX, DATA
        MOV DS, AX

```

```

    CALL IF_NEED_UNLOAD
    CMP CL, 1H
    JE NEED_UNLOAD

    CALL IF_LOADED
    CMP CH, 1H
    JE PRINT_ROUT_IS_ALREADY_SET
    MOV DX, OFFSET INT_IS_LOADING
    CALL WRITE_STRING
    CALL LOAD_ROUT
    JMP EXIT

NEED_UNLOAD:
    CALL IF_LOADED
    CMP CH, 1H
    JNE PRINT_ROUT_CANT_BE_UNLOADED
    CALL UNLOAD_ROUT
    MOV DX, OFFSET INT_IS_UNLOADED
    CALL WRITE_STRING
    JMP EXIT

PRINT_ROUT_CANT_BE_UNLOADED:
    MOV DX, OFFSET INT_NOT_LOAD
    CALL WRITE_STRING
    JMP EXIT

PRINT_ROUT_IS_ALREADY_SET:
    MOV DX, OFFSET INT_LOADED
    CALL WRITE_STRING
    JMP EXIT

EXIT:
    MOV AH, 4CH
    INT 21H

MAIN ENDP
CODE ENDS
END MAIN

```