

**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. 9383

Моисейченко К. А.

Преподаватель

Ефремов М. А.

Санкт-Петербург

2021

## **Цель работы.**

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 MSB. 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

## **Основные теоретические положения.**

Резидентные обработчики прерываний - это программные модули, которые вызываются при возникновении прерываний определенного типа (сигнал таймера, нажатие клавиши и т.д.), которым соответствуют определенные вектора прерывания. Когда вызывается прерывание, процессор переключается на выполнение кода обработчика, а затем возвращается на выполнение прерванной программы. Адрес возврата в прерванную программу (CS:IP) запоминается в стеке вместе с регистром флагов. Затем в CS:IP загружается адрес точки входа программы обработки прерывания и начинается выполняться его код. Обработчик прерывания должен заканчиваться инструкцией IRET (возврат из прерывания).

Вектор прерывания имеет длину 4 байта. В первом хранится значение IP, во втором - CS. Младшие 1024 байта памяти содержат 256 векторов. Вектор для прерывания 0 начинается с ячейки 0000:0000, для прерывания 1 – с ячейки 0000:0004 и т.д.

Обработчик прерывание - это отдельная процедура, имеющая следующую структуру:

```
ROUT PROC FAR
    PUSH AX ; сохранение изменяемых регистров
    ...
    <действия по обработке прерывания>
    POP AX ; восстановление регистров
    ...
    MOV AL, 20H
    OUT 20H, AL
    IRET
ROUT ENDP
```

Две последние строки необходимы для разрешения обработки прерываний с более низкими уровнями, чем только что обработанное. Для установки написанного прерывания в поле векторов прерываний используется функция 25H прерывания 21H, которая устанавливает вектор прерывания на указанный адрес.

```
PUSH DS
MOV DX, OFFSET ROUT ; смещение для процедуры в DX
MOV AX, SEG ROUT ; сегмент процедуры
MOV DS, AX ; помещаем в DS
MOV AH, 25H ; функция установки вектора
```

```

MOV AL, 1CH ; номер вектора
INT 21H ; меняем прерывание
POP DS

```

Программа, выгружающая обработчик прерываний должна восстанавливать оригинальные векторы прерываний. Функция 35 прерывания 21H позволяет восстановить значение вектора прерывания, помещая значение сегмента в ES, а смещение в BX. Программа должна содержать следующие инструкции:

```

; -- хранится в обработчике прерываний
KEEP_CS DW 0 ; для хранения сегмента
KEEP_IP DW 0 ; и смещения прерывания
; -- в программе при загрузке обработчика прерывания
MOV AH, 35H ; функция получения вектора
MOV AL, 1CH ; номер вектора
INT 21H
MOV KEEP_IP, BX ; запоминание смещения
MOV KEEP_CS, ES ; и сегмента
; -- в программе при выгрузке обработчика прерываний
CLI
PUSH DS
MOV DX, KEEP_IP
MOV AX, KEEP_CS
MOV DS, AX
MOV AH, 25H
MOV AL, 1CH
INT 21H ; восстанавливаем вектор
POP DS
STI

```

Для того, чтобы оставить процедуру прерывания резидентной в памяти, следует воспользоваться функцией DOS 31h прерывания 21h. Эта функция оставляет память, размер которой указывается в качестве параметра, занятой, а остальную память освобождает и осуществляет выход в DOS. Функция 31h int 21h использует следующие параметры:

AH - номер функции 31h;

AL - код завершения программы;

DX - размер памяти в параграфах, требуемый резидентной программе.

Пример обращения к функции:

```

mov DX, offset LAST_BYTE ; размер в байтах от начала сегмента
mov CL, 4 ; перевод в параграфы
shr DX, CL
inc DX ; размер в параграфах
mov AH, 31h

```

```
int 21h
```

Вывод на экран информации обработчиком прерываний осуществляется с помощью функций прерывания 10h.

```
;функция вывода символа из AL
outputAL proc
    push ax
    push bx
    push cx
    mov ah,09h ;писать символ с текущей позиции курсора
    mov bh,0 ;номер видео страницы
    mov cx,1 ;число экземпляров символа для записи
    int 10h ;выполнить функцию
    pop cx
    pop bx
    pop ax
    ret
;
;
;функция вывода строки по адресу ES:BP на экран
outputBP proc
    push ax
    push bx
    push dx
    push CX
    mov ah,13h ; функция
    mov al,1 ; sub function code
; 1 = use attribute in BL; leave cursor at end of string
    mov bh,0 ; видео страница
    mov dh,22 ; DH,DL = строка, колонка (считая от 0)
    mov dl,0
    int 10h
    pop CX
    pop dx
    pop bx
    pop ax
    ret
outputBP endp
;
; Установка позиции курсора
; установка на строку 25 делает курсор невидимым
setCurs proc
    push ax
    push bx
    push dx
    push CX
    mov ah,02h
    mov bh,0
    mov dh,22 ; DH,DL = строка, колонка (считая от 0)
    mov dl,0
    int 10h ; выполнение.
    pop CX
    pop dx
    pop bx
    pop ax
```

```

        ret
;
; 03H читать позицию и размер курсора
; вход: BH = видео страница
; выход: DH,DL = текущие строка, колонка курсора
; CH,CL = текущие начальная, конечная строки курсора
getCurs proc
    push ax
    push bx
    push dx
    push CX
    mov ah,03h
    mov bh,0
    int 10h ; выполнение.
; выход: DH,DL = текущие строка, колонка курсора
; CH,CL = текущие начальная, конечная строки курсора
    pop CX
    pop dx
    pop bx
    pop ax
    ret

```

### **Контрольные вопросы по лабораторной работе.**

- 1) Как реализован механизм прерывания от часов?
- 2) Какого типа прерывания использовались в работе?

### **Выполнение работы.**

#### **Шаг 1.**

Был написан и отлажен 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.

## Шаг 2.

Была запущена отлаженная программа lab4.exe. Работа прерывания отображается на экране.

```
Number of my interrupts: 0001er Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All rights reserved.

Object filename [lab4.OBJ]:
Source listing [NUL.LST]:
Cross-reference [NUL.CRF]:

49892 + 451226 Bytes symbol space free

0 Warning Errors
0 Severe Errors

C:\>link lab4.obj

Microsoft (R) Overlay Linker Version 3.64
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.

Run File [LAB4.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:

C:\>lab4.exe
Interrupt was loaded

C:\>_
```

Рисунок 1 - Результат работы программы lab4.exe

Для того, чтобы проверить размещение прерывания в памяти была запущена программа из ЛР №3, которая отображает карту памяти в виде списка блоков МСВ. Прерывание отобразилось в памяти.

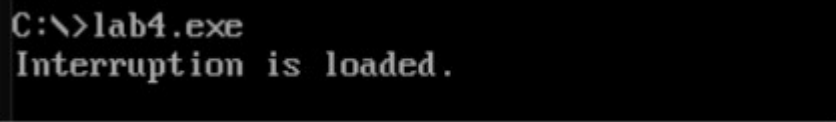
```
C:\>lab3_1.com
Amount of available memory: 648128 byte
Extended memory size: 245920 byte
MCB table:
Address: 016F PSP address: 0008 Size: 16 SC/SD:
Address: 0171 PSP address: 0000 Size: 64 SC/SD:
Address: 0176 PSP address: 0040 Size: 256 SC/SD:
Address: 0187 PSP address: 0192 Size: 144 SC/SD:
Address: 0191 PSP address: 0192 Size: 608 SC/SD: LAB4
Address: 01B8 PSP address: 01C3 Size: 144 SC/SD:
Address: 01C2 PSP address: 01C3 Size: 648128 SC/SD: LAB3_1
```



Рисунок 2 - Работа программы lab3\_1.com

Шаг 3.

Отлаженная программа была запущена еще раз. На экран вывелось сообщение о том, что прерывание уже загружено в память, т.е. программа определяет установленный обработчик прерываний.




```
C:\>lab4.exe
Interruption is loaded.
```

Рисунок 3 - Повторный запуск программы lab4.exe

Шаг 4.

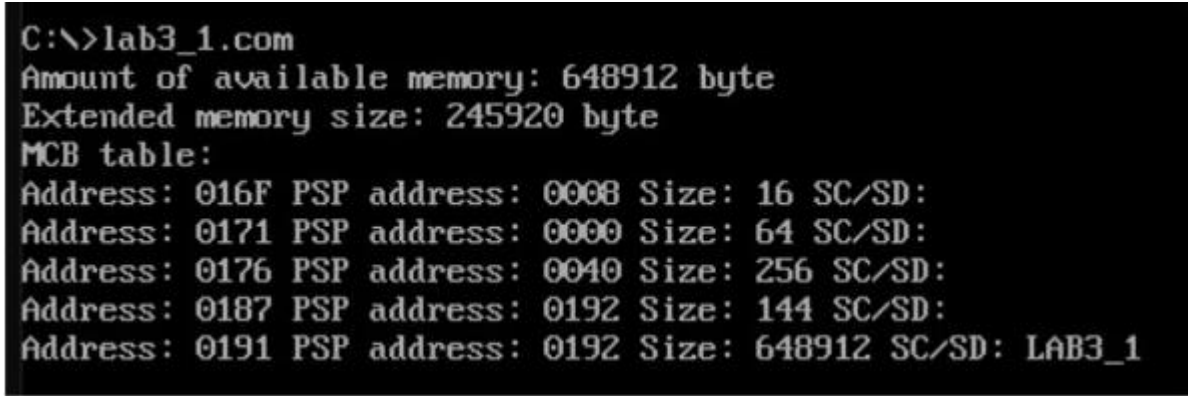
Отлаженная программа была запущена с ключом выгрузки /un. Вывелось сообщение, что стандартный обработчик прерываний был восстановлен.



```
C:\>lab4.exe /UN
Interruption was restored.
```

Рисунок 4 - Запуск программы lab4.exe с ключом выгрузки /un

Чтобы убедиться, что память, занятая резидентом освобождена, была запущена программы из ЛР №3.



```
C:\>lab3_1.com
Amount of available memory: 648912 byte
Extended memory size: 245920 byte
MCB table:
Address: 016F PSP address: 0008 Size: 16 SC/SD:
Address: 0171 PSP address: 0000 Size: 64 SC/SD:
Address: 0176 PSP address: 0040 Size: 256 SC/SD:
Address: 0187 PSP address: 0192 Size: 144 SC/SD:
Address: 0191 PSP address: 0192 Size: 648912 SC/SD: LAB3_1
```

Рисунок 5 - Повторный запуск программы lab3\_1.com

### **Ответы на контрольные вопросы.**

1) Как реализован механизм прерывания от часов?

Каждый такт из таймера вычитается определенное значение. Когда значение достигает 9, возникает прерывание от таймера. При возникновении прерывания процессор запоминает в стеке адрес возврата (CS:IP) и регистр флагов. Затем в CS:IP загружается адрес обработчика прерывания и выполняется его код. В конце регистры восстанавливаются, и процессор возвращается на выполнение прерванной программы.

2) Какого типа прерывания использовались в работе?

В данной работе использовались аппаратное прерывание 21h с вектором 1Ch, а также пользовательские прерывания 10h и 21h.

### **Выводы.**

Была написана программа обработчика прерываний таймера, изучены обработка стандартных прерываний, методы загрузки программы-резидента, а также его выгрузка из памяти.

**ПРИЛОЖЕНИЕ А**  
**ИСХОДНЫЙ КОД ПРОГРАММЫ**

# ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab3\_1.asm

```
AStack      SEGMENT STACK
DB 256 DUP(?)
AStack ENDS

DATA SEGMENT

flag DB 0
interrupt_was_loaded_string DB 'Interrupt was loaded', 0DH, 0AH, '$'
interrupt_was_unloaded_string DB 'Interrupt was unloaded', 0DH,
0AH, '$'
interrupt_not_loaded_string DB 'Interrupt has not been loaded', 0DH,
0AH, '$'
interrupt_already_loaded_string DB 'Interrupt has already been
loaded', 0DH, 0AH, '$'

DATA ENDS

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

MY_INTERRUPT PROC far
jmp start_interrupt

ID DW 0FFFFh
PSP DW ?

KEEP_IP DW 0
KEEP_CS DW 0
KEEP_SS DW 0
KEEP_SP DW 0
KEEP_AX DW 0

INT_COUNTER DB 'Number of iterrups: 0000$'

INT_STACK DW 128 DUP (?)
END_INT_STACK:

start_interrupt:
mov KEEP_SS, SS
mov KEEP_SP, SP
mov KEEP_AX, AX

mov AX, CS
mov SS, AX
mov SP, OFFSET END_INT_STACK

push BX
push CX
push DX
```

; получение курсора

```
mov AH, 3h
mov BH, 0h
int 10h
push DX
```

; установка курсора

```
mov AH, 2h
mov BH, 0h
mov DH, 2h
    mov DL, 5h
int 10h
```

```
    push BP
push SI
push CX
push DS
```

```
    mov AX, seg INT_COUNTER
    mov DS, AX
    mov SI, offset INT_COUNTER
    add SI, 20
    mov CX, 4
```

```
incr:
mov BP, CX
mov AH, [SI+BP]
inc AH
mov [SI+BP], AH
cmp AH, 3Ah
jne good
mov AH, 30h
mov [SI+BP], AH
loop incr
```

```
good:
    pop DS
pop CX
pop SI
```

```
push ES
mov DX, SEG INT_COUNTER
mov ES, DX
mov BP, OFFSET INT_COUNTER
mov AH, 13h
mov AL, 0h
mov CX, 24
mov DX, 0h
int 10h
```

```
pop ES
pop BP
```

; возврат курсора

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

```
pop DX
pop CX
pop BX
```

```
mov AX, KEEP_SS
mov SS, AX
mov AX, KEEP_AX
mov SP, KEEP_SP
mov AL, 20h
out 20h, AL
```

```
iret
end_my_interrupt:
MY_INTERRUPT     endp
```

LOAD PROC near

```
push    AX
push    CX
push    DX
```

```
mov     AH, 35h
mov     AL, 1Ch
int     21h
mov     KEEP_IP, BX
mov     KEEP_CS, ES
```

```
push    DS
mov     DX, OFFSET MY_INTERRUPT
mov     AX, SEG MY_INTERRUPT
mov     DS, AX
mov     AH, 25h
mov     AL, 1Ch
int     21h
pop     DS
```

```
mov     DX, OFFSET END_INT_STACK
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      AX
ret
```

LOAD endp

UNLOAD PROC near

```
push     AX
push     DX
push     SI
push     ES

cli
push     DS
mov      AH, 35h
mov      AL, 1Ch
int      21h
mov      SI, OFFSET KEEP_CS
sub      SI, OFFSET MY_INTERRUPT
mov      DX, ES:[BX+SI+2]
mov      AX, ES:[BX+SI]
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      ES
pop      SI
pop      DX
pop      AX
ret
```

UNLOAD endp

LOAD\_FLAG PROC near

```
push     AX
```

```

        mov     AL, ES:[82h]
        cmp     AL, '/'
        jne     end_load_flag
        mov     AL, ES:[83h]
        cmp     AL, 'u'
        jne     end_load_flag
        mov     AL, ES:[84h]
        cmp     AL, 'n'
        jne     end_load_flag
        mov     flag, 1h
end_load_flag:
pop     AX

```

```
LOAD_FLAG endp
```

```
IS_LOAD PROC near
```

```

        push    AX
        push    DX
        push    SI
        mov     flag, 1h
        mov     AH, 35h
        mov     AL, 1Ch
        int     21h
        mov     SI, OFFSET ID
        sub     SI, OFFSET MY_INTERRUPT
        mov     DX, ES:[BX+SI]
        cmp     DX, 0FFFFh
        je      loading
        mov     flag, 0
loading:
        pop     SI
        pop     DX
        pop     AX
        ret

```

```
IS_LOAD endp
```

```
PRINT_STR PROC near
```

```

        push    AX
        mov     AH, 09h
        int     21h
        pop     AX
        ret

```

```
PRINT_STR endp
```

```
MAIN PROC far
```



```

        mov     AX, DATA
        mov     DS, AX
        mov     PSP, ES
        mov     flag, 0
        call    LOAD_FLAG
        cmp     flag, 1
        je      un

; loading

        call    IS_LOAD
        cmp     flag, 0
        je      notld
        mov     DX, OFFSET interrupt_already_loaded_string
        call    PRINT_STR
        jmp     fin

notld:   mov     DX, OFFSET interrupt_was_loaded_string
        call    PRINT_STR
        call    LOAD
        jmp     fin

; unloading

un:      call    IS_LOAD
        cmp     flag, 0
        jne     alrld
        mov     DX, OFFSET interrupt_not_loaded_string
        call    PRINT_STR
        jmp     fin

alrld:   call    UNLOAD
        mov     DX, OFFSET interrupt_was_unloaded_string
        call    PRINT_STR

fin:     mov     AX, 4Ch      ; завершение
        int     21h

MAIN    endp
CODE    ENDS
END     MAIN

```