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. 7381 Минуллин М.А.

Преподаватель Ефремов М.А.

Санкт-Петербург

2019

Цель работы.

В архитектуре компьютера существуют стандартные прерывания, за которыми закреплены определённые вектора прерываний. Вектор прерываний хранит адрес подпрограммы обработчика прерываний. При возникновении прерывания, аппаратура компьютера передаёт 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 определённые интервалы времени и, при возникновении такого сигнала, возникает прерывание с определённым значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

Необходимые сведения для составления программы.

Резидентные обработчик прерываний – это программные модули, которые вызываются при возникновении прерываний определённого типа (сигнал таймера, нажатие клавиши и т. д.), которым соответствуют определённые 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 начинает 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.

Ход работы.

Был написан и отлажен .EXE модуль, выполняющий следующие функции:

1. Проверяется, установлено ли пользовательское прерывание с вектором 1Ch.

2. Устанавливается резидентная функция для обработки прерывания и настраивается вектор прерываний, если прерывание не установлено, осуществляется выход через функцию 2Ch прерывания 21h.

3. Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход через функцию 2Ch прерывания 21h.

4. Выгружается прерывание по соответствующему значению параметра в командной строке /un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождения памяти, занимаемой резидентом. Осуществляется выход через функцию 4Ch прерывания 21h.

Была запущена полученная программа (рис. 1).

```
C:\>LAB4.EXE
Resident was loaded                    Count of interrupt: 0087
```

Рисунок 1 – Резидентная программа лабораторной работы №4.

Для проверки размещения прерывания в памяти была запущена программа из лабораторной работы №3, отображающей карту памяти в виде блоков MCB (рис. 2).

```
C:\>LAB3_1.COM
Available memory (B): 647632
Extended memory (KB): 15360
| MCB Type | PSP Address | Size | SC/SD |
     4D          0008          16
     4D          0000          64
     4D          0040         256
     4D          0192         144
     4D          0192        1104   LAB4
     4D          01E2        1144
     5A          01E2      647632   LAB3_1Count of interrupt: 0899
```

Рисунок 2 – Резидентная программа на карте памяти.

Произведена попытка повторного запуска программы лабораторной работы №4 (рис. 3). Выводится сообщение о том, что программа уже была запущена.

```
C:\>LAB4.EXE
Resident has already been loaded       Count of interrupt: 1481
```

Рисунок 3 – Попытка запустить программу дважды.

Выполнен запуск программы лабораторной работы №4 с ключом выгрузки (рис. 4).

```
C:\>LAB4.EXE /un
Resident was unloaded
```

Рисунок 4 – Выгрузка резидентной программы.

Проверка отсутствия программы-резидента на карте памяти с помощью повторного запуска программы из лабораторной работы №3 (рис. 5).

```
C:\>LAB3_1.COM
Available memory (B): 648912
Extended memory (KB): 15360
| MCB Type | PSP Address | Size | SC/SD |
     4D          0008          16
     4D          0000          64
     4D          0040         256
     4D          0192         144
     5A          0192      648912   LAB3_1
```

Рисунок 5 – Проверка отсутствия резидентной программы.

Контрольные вопросы.

В: Как реализован механизм прерывания от часов?

О: Прерывание по таймеру вызывается автоматически по каждому тику аппаратных часов каждые 55 миллисекунд. После вызова сохраняется содержимое регистров и определяется источник прерывания, по номеру которого определяется смещение в таблице векторов прерываний. Полученный адрес сохраняется в регистр CS:IP. После этого управление передаётся по этому адресу, т. е. выполняется запуск обработчика прерываний и происходит его выполнение. После выполнения происходит возврат управления прерванной программе.

В: Какого типа прерывания использовались в работе?

О: В данной лабораторной работе использовались аппаратные прерывания, прерывания функций MS DOS и прерывания функций BIOS.

Выводы.

В ходе выполнения лабораторной работы была написана программа, в которой был реализован обработчик прерываний сигналов таймера, а также были изучены дополнительный функции для работы с памятью: установка и выгрузка из памяти программы-резидента.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ ТЕКСТ .EXE МОДУЛЯ

```
MY_STACK segment STACK
    DW 64 dup (?)
MY_STACK ends

DATA segment
    message_1 db 'Resident was loaded', 13, 10, '$'
    message_2 db 'Resident has already been loaded', 13, 10, '$'
    message_3 db 'Resident was unloaded', 13, 10, '$'
DATA ends

CODE segment
    assume cs:CODE, ds:DATA, es:DATA, ss:MY_STACK

setCurs     proc
    push    ax
    push    bx
    push    cx
    mov     ah, 02h
    mov     bh, 00h
    int     10h
    pop     cx
    pop     bx
    pop     ax
    ret
setCurs     endp

getCurs     proc
    push    ax
    push    bx
    push    cx
    mov     ah, 03h
    mov     bh, 00h
    int     10h
    pop     cx
    pop     bx
    pop     ax
    ret
getCurs     endp

printSTR    proc
    push    es
    push    bp
    mov     ax, SEG COUNT
    mov     es, ax
    mov     ax, offset COUNT
```

```
    mov     bp, ax
    mov     ah, 13h
    mov     al, 00h
    mov     cx, 25
    mov     bh, 0
    mov     bl, 13
    int     10h
    pop     bp
    pop     es
    ret
printSTR    endp

ROUT    proc    far
    jmp     ROUT_START

    identifier  db '0000'
    KEEP_IP     dw 0
    KEEP_CS     dw 0
    KEEP_PSP    dw 0
    flag        db 0
    KEEP_SS     dw 0
    KEEP_AX     dw 0
    KEEP_SP     dw 0
    COUNT       db 'Count of interrupt: 0000 $'
    inter_stack dw 64 dup (?)
    end_stack   dw 0

ROUT_COUNT:
    push    si
    push    cx
    push    ds
    mov     ax, SEG COUNT
    mov     ds, ax
    mov     si, offset COUNT
    add     si, 23
    mov     ah, [si]
    add     ah, 1
    mov     [si], ah
    cmp     ah, 58
    jne     END_COUNT
    mov     ah, 48
    mov     [si], ah
    mov     bh, [si - 1]
    add     bh, 1
    mov     [si-1], bh
    cmp     bh, 58
    jne     END_COUNT
    mov     bh, 48
    mov     [si - 1], bh
    mov     ch, [si - 2]
    add     ch, 1
```

```
    mov     [si - 2], ch
    cmp     ch, 58
    jne     END_COUNT
    mov     ch, 48
    mov     [si - 2], ch
    mov     dh, [si - 3]
    add     dh, 1
    mov     [si - 3], dh
    cmp     dh, 58
    jne     END_COUNT
    mov     dh, 48
    mov     [si - 3], dh

END_COUNT:
    pop     ds
    pop     cx
    pop     si
    call    printSTR
    pop     dx
    call    setCurs
    jmp     END_ROUT

ROUT_START:
    mov     KEEP_AX, ax
    mov     KEEP_SS, ss
    mov     KEEP_SP, sp
    mov     ax, cs
    mov     ss, ax
    mov     sp, offset end_stack
    mov     ax, KEEP_AX
    push    dx
    push    ds
    push    es
    cmp     flag, 1
    je      ROUT_REC
    call    getCurs
    push    dx
    mov     dh, 22
    mov     dl, 39
    call    setCurs
    jmp     ROUT_COUNT

ROUT_REC:
    cli
    mov     dx, KEEP_IP
    mov     ax, KEEP_CS
    mov     ds, ax
    mov     ah, 25h
    mov     al, 1Ch
    int     21h
    mov     es, KEEP_PSP
```

```
    mov     es, es:[2Ch]
    mov     ah, 49h
    int     21h
    mov     es, KEEP_PSP
    mov     ah, 49h
    int     21h
    sti

END_ROUT:
    pop     es
    pop     ds
    pop     dx
    mov     ss, KEEP_SS
    mov     sp, KEEP_SP
    mov     ax, KEEP_AX
    iret
ROUT    endp

SET_INTERRUPT   proc
    push    dx
    push    ds
    mov     ah, 35h
    mov     al, 1Ch
    int     21h
    mov     KEEP_IP, bx
    mov     KEEP_CS, es
    mov     dx, offset ROUT
    mov     ax, seg ROUT
    mov     ds, ax
    mov     ah, 25h
    mov     al, 1Ch
    int     21h
    pop     ds
    mov     dx, offset message_1
    call    PRINT
    pop     dx
    ret
SET_INTERRUPT   endp

BASE_FUNC   proc
    mov     ah, 35h
    mov     al, 1Ch
    int     21h

    mov     si, offset identifier
    sub     si, offset ROUT

    mov     ax, '00'
    cmp     ax, es:[bx + si]
    jne     NOT_LOADED
    cmp     ax,es:[bx + si + 2]
```

```
    jne     NOT_LOADED
    jmp     LOADED

NOT_LOADED:
    call    SET_INTERRUPT
    mov     dx, offset LAST_BYTE
    mov     cl, 4
    shr     dx, cl
    inc     dx
    add     dx, CODE
    sub     dx, KEEP_PSP
    xor     al, al
    mov     ah, 31h
    int     21h

LOADED:
    push    es
    push    ax
    mov     ax, KEEP_PSP
    mov     es, ax
    mov     al, es:[81h + 1]
    cmp     al,'/'
    jne     NOT_UNLOAD
    mov     al, es:[81h + 2]
    cmp     al,'u'
    jne     NOT_UNLOAD
    mov     al, es:[81h + 3]
    cmp     al,'n'
    je      UNLOAD

NOT_UNLOAD:
    pop     ax
    pop     es
    mov     dx, offset message_2
    call    PRINT
    ret

UNLOAD:
    pop     ax
    pop     es
    mov     byte ptr es:[bx + si + 10], 1
    mov     dx, offset message_3
    call    PRINT
    ret
BASE_FUNC   endp

PRINT   proc    near
    push    ax
    mov     ah, 09h
    int     21h
    pop     ax
```

```
    ret
PRINT   endp

MAIN proc Far
    mov     ax,DATA
    mov     ds,ax
    mov     KEEP_PSP,es
    call    BASE_FUNC
    xor     al,al
    mov     ah,4Ch
    int     21H
LAST_BYTE:
MAIN    endp

CODE    ends

END MAIN
```