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. 8382 Мирончик П.Д.

Преподаватель Ефремов М.А.

Санкт-Петербург

2020

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.

Ход работы.

Была реализована программа, которая реализует следующие функции:

- Проверяет, установлено ли пользовательское прерывание с вектором 1Ch.
- Устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и осуществляет выход по функции 4Ch прерывания int 21h.
- Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.
- Осуществляет выгрузку 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.

1. Вывод программы lab3_1.com (программа из 3-й лабораторной работы, показывающая информацию о блоках памяти)

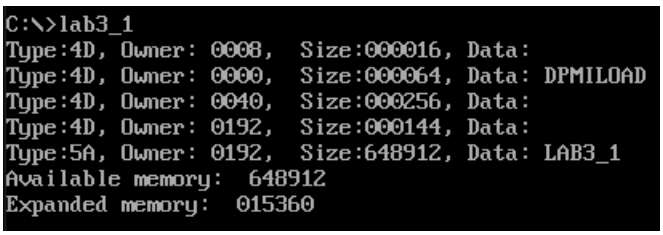

Рис. 1. Вывод блоков MCB перед установкой прерывания

2. Вывод программы lab4 – установка прерывания

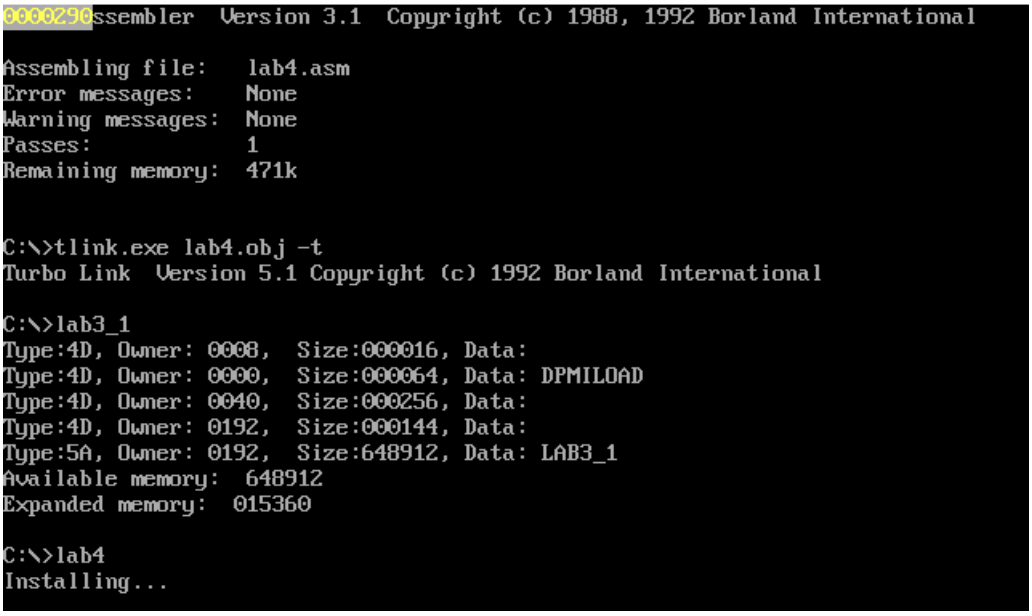

Рис. 2. Вывод программы lab4 при установке прерывания

и вывод lab3_1 сразу после этого

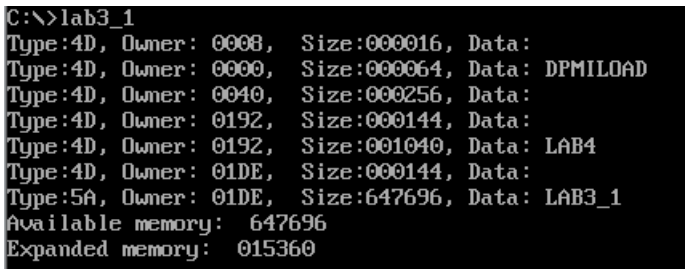

Рис.3. Вывод программы lab3_1 при установленном прерывании

Видно, что в памяти остались блоки памяти программы lab4, где хранится код функции прерывания

3. Повторный запуск lab4

Рис.4. Вывод lab4 при установленноп прерывании

Прерывание не было повторно установлено

4. Восстановление прерывания по умолчанию

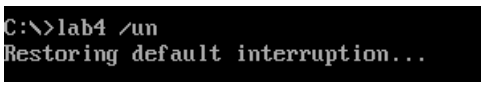

Рис.5. Восстановление прерывания по умолчанию

а также вывод lab3_1, который показывает, что блоки памяти, в которых хранилось прерывание, удалены

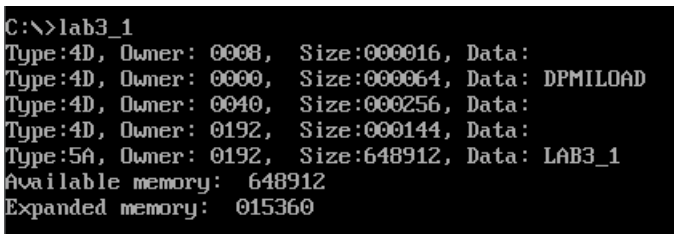

Рис.6. Вывод lab3_1 после восстановление прерывания

5. Повторная попытка очистить прерывание

Рис.7. Вывод программы lab4 при попытке восстановить прерывание, когда оно уже установлено по умолчанию

Ответы на вопросы.

1. Как реализован механизм прерывания от часов?

Примерно 18,2 раз в секунду (на каждый тик аппаратных часов) BIOS вызывает прерывание 1Ch. При этом в стек сохраняется адрес возврата (CS:IP)

и регистр флагов, после чего в CS:IP записывается адрес прерывания и, соответственно, начинает выполняться код прерывания.

2. Какого типа прерывания использовались в работе?

В работе использовались аппаратные (1Ch) прерывания и программные прерывания DOS и BIOS (21h и 10h).

Выводы

Был построен обработчик прерываний сигналов таймера, выполняющий приращение счетчика и вывод его на экран в виде строки. В ходе выполнения работы были получены знания об организации обработки стандартных прерываний, а также исследовано размещение прерываний в памяти

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
CODE    SEGMENT
        ASSUME cs:CODE, ds:CODE, ss:NOTHING
        org 100h

START: jmp BEGIN

INSTALLING_STR          DB      'Installing...',10,13,'$'
ALREADY_INSTALLED_STR   DB      'Interraption is already installed',10,13
,'$'
NOT_INSTALLED_STR       DB      'Interruption is not installed',10,13,'$'
RESTORING_STR           DB      'Restoring default interruption...',10,13
,'$'

INT_TIMER PROC FAR
        jmp INT_TIMER_begin

 INT_TIMER_data:
        ; data
        INT_TIMER_sign          DW      1234h
        INT_TIMER_stack         DB      100h DUP(0)
        INT_TIMER_keep_int_cs   DW      0
        INT_TIMER_keep_int_ip   DW      0
        INT_TIMER_keep_ss       DW      0
        INT_TIMER_keep_sp       DW      0
        INT_TIMER_count         DW      0
        INT_TIMER_count_string  DB      '0000000$'

        ; code
INT_TIMER_begin:
        push ds
        push ax
        mov ax, cs
        mov ds, ax

        mov INT_TIMER_keep_ss, ss
        mov INT_TIMER_keep_sp, sp

        mov ax, cs
        mov ss, ax
        mov sp, offset INT_TIMER_stack
        add sp, 100h

        push es
        push bp
        push cx
        push bx
        push dx
        push di
        push si

        ; increment current count
        inc INT_TIMER_count
```

```
        mov ax, INT_TIMER_count
        xor dx, dx
        mov di, offset INT_TIMER_count_string
        add di, 6
        mov si, offset INT_TIMER_count_string
        call WRD_TO_DEC

        ; write current count
        call GET_CURSOR
        push dx

        mov dh, 0
        mov dl, 0
        call SET_CURSOR

        mov ax, ds
        mov es, ax
        mov bp, offset INT_TIMER_count_string
        add bp, 0
        mov ah, 13h
        mov al, 1h
        mov cx, 7
        mov bh, 0
        int 10h

        pop dx
        call SET_CURSOR

        pop si
        pop di
        pop dx
        pop bx
        pop cx
        pop bp
        pop es

        mov ss, INT_TIMER_keep_ss
        mov sp, INT_TIMER_keep_sp
        pop ax
        pop ds

        mov al, 20h
        out 20h, al
        iret
INT_TIMER ENDP

GET_CURSOR PROC near
; move current cursor's position to DX
        push ax
        push bx

        mov bh, 0h
        mov ah, 03h
        int 10h

        pop bx
        pop ax
        ret
```

```
GET_CURSOR ENDP

SET_CURSOR PROC near
; set cursor's position, stored in DX, to screen
        push ax
        push bx

        mov bh, 0h
        mov ah, 02h
        int 10h

        pop bx
        pop ax
        ret
SET_CURSOR ENDP

WRD_TO_DEC PROC near
; input ax - value
;       di - lower num address
;       si - address of highest available num position (DI-max), or 0 if
;            prefix isn't need
;
; converts AX to DEC and writes to di address (to DI, DI-1, DI-2, ...)
        push bx
        push dx
        push di
        push si
        push ax

        mov bx, 10
        WRD_TO_DEC_loop:
                div bx
                add dl, '0'
                mov [di], dl
                xor dx, dx
                dec di
                cmp ax, 0
                jne WRD_TO_DEC_loop

        cmp si, 0
        je WRD_TO_DEC_no_prefix
        cmp si, di
        jge WRD_TO_DEC_no_prefix
        WRD_TO_DEC_prefix_loop:
                mov dl, '0'
                mov [di], dl
                dec di
                cmp di, si
                jl WRD_TO_DEC_prefix_loop

WRD_TO_DEC_no_prefix:
        pop ax
        pop si
        pop di
        pop dx
        pop bx
        ret
WRD_TO_DEC ENDP
```

```
LOAD_INT PROC NEAR
        mov ah, 35h
        mov al, 1Ch
        int 21h
        mov INT_TIMER_keep_int_ip, bx
        mov INT_TIMER_keep_int_cs, es

        mov dx, offset INT_TIMER
        mov ah, 25h
        mov al, 1Ch
        int 21h

        mov dx, offset PROGRAM_END_BYTE
        mov cl, 4
        shr dx, cl
        inc dx
        mov ah, 31h
        int 21h
LOAD_INT ENDP

RELOAD_INT PROC NEAR
        push dx
        push ds
        push es
        push bx

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

        mov ax, es
        mov ds, ax
        mov dx, INT_TIMER_keep_int_ip
        mov ax, INT_TIMER_keep_int_cs
        mov ds, ax
        mov ah, 25h
        mov al, 1Ch
        int 21h

        push es
        mov ax, es:[2Ch]
        mov es, ax
        mov ah, 49h
        int 21h
        pop es
        int 21h

        pop bx
        pop es
        pop ds
        pop dx
        ret
RELOAD_INT ENDP

CHECK_INT PROC NEAR
        push ax
        push bx
```

```
        push es

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

        push ds
        mov ax, es
        mov ds, ax
        mov ax, INT_TIMER_sign
        cmp ax, 1234h
        pop ds

        pop es
        pop bx
        pop ax
        ret
CHECK_INT ENDP

BEGIN:
        cmp byte ptr es:[81h+1], '/'
        jne LOAD_IF_NEED
        cmp byte ptr es:[81h+2], 'u'
        jne LOAD_IF_NEED
        cmp byte ptr es:[81h+3], 'n'
        jne LOAD_IF_NEED

        call CHECK_INT
        jne NOT_INSTALLED
        call RELOAD_INT
        mov dx, offset RESTORING_STR
        mov ah, 09h
        int 21h
        jmp EXIT

NOT_INSTALLED:
        mov dx, offset NOT_INSTALLED_STR
        mov ah, 09h
        int 21h
        jmp EXIT

LOAD_IF_NEED:
        call CHECK_INT
        je INSTALLED
        mov dx, offset INSTALLING_STR
        mov ah, 09h
        int 21h
        call LOAD_INT
        jmp EXIT

INSTALLED:
        mov dx, offset ALREADY_INSTALLED_STR
        mov ah, 09h
        int 21h
        jmp EXIT

EXIT:
        xor     al,al
```

```
        mov     ah,4Ch
        int     21h

PROGRAM_END_BYTE:
CODE    ENDS
END START
```