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 _____ Тюленев Т.В.

Преподаватель _____ 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 MCB. 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.

Шаг 1. Был написан и отлажен программный модуль типа .EXE, который выполняет следующие функции:

1) Проверяет установлено ли пользовательское прерывание;

2) Устанавливает резидентную функцию для обработки прерывания;

3) Если резидентная функция уже установлена — выводит соответствующее сообщение;

4) Выгружает прерывание по значению параметра в командной строке: /un, восстанавливая стандартный вектор прерывания.

Пользовательское прерывание 1Ch — выводит информацию о количестве сигналов таймера с момента запуска программы.

Функции, реализованные в работе:

- setCurs — Установка курсора;
- getCurs — Возврат положения курсора;
- ROUT — Обработчик прерывания (считает и выводит число его вызовов от таймера);
- CHECK — Проверка, загружено ли пользовательское прерывание;
- SET_INT — Установка нового обработчика прерывания с запоминанием данных для восстановления предыдущего;
- PRINT — Осуществление вывода.

Шаг 2.

Для того, чтобы удостовериться в корректности дальнейшей работы прерывания — фиксируем вывод информации о состоянии блоков MCB перед установкой прерывания (пользуясь программой из лабораторной работы №3).

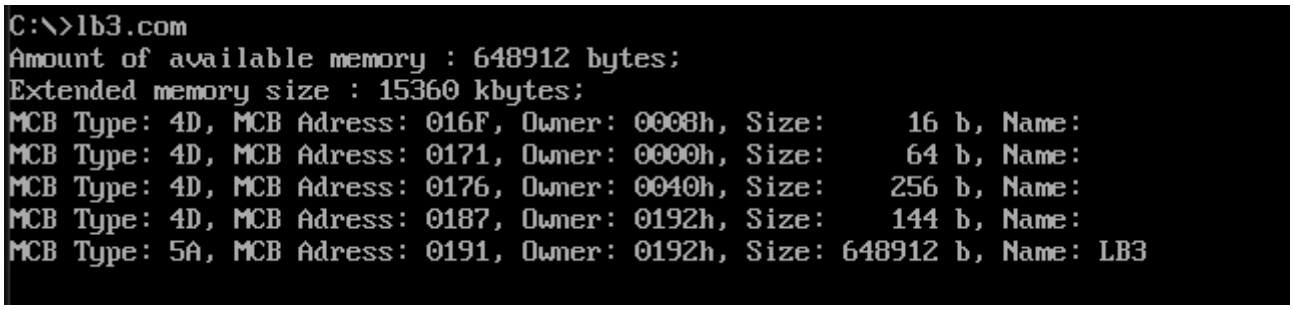

Рисунок 1 — Результат запуска lb3.com перед установкой прерывания.

Далее запускаем программу с обработкой пользовательского прерывания:

Рисунок 2 — Результат загрузки lb.exe.

Следом, вновь запускаем код 3-й лабораторной работы:

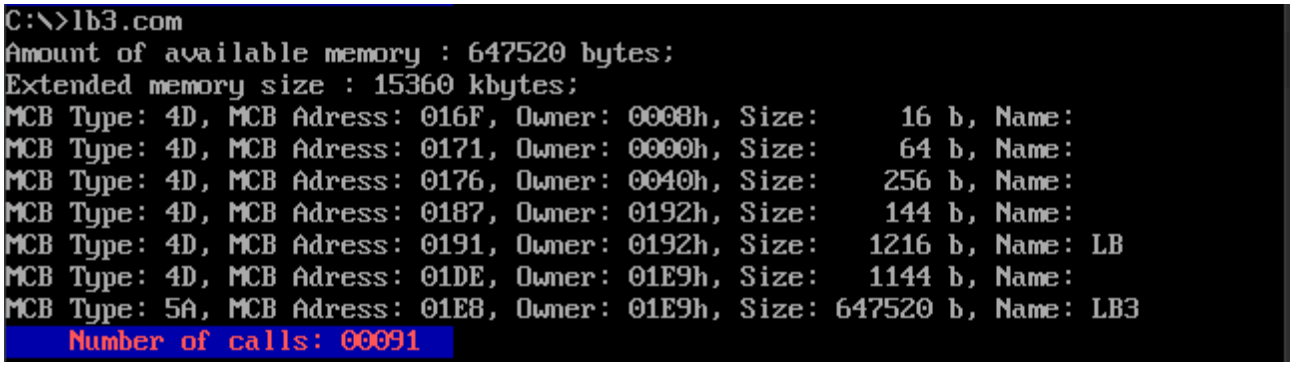

Рисунок 3 — Результат запуска lb3.com после установки прерывания.

Исходя из вывода программы lb3.com видно, что в памяти остались блоки MCB обработчика прерывания (lb.exe), что свидетельствует о том, что пользовательское прерывание успешно загружено в память.

Шаг 3.

Рисунок 4 — Результат повторной загрузки lb.exe.

В результате, прерывание не было установлено повторно, о чем говорит соответствующее сообщение.

Шаг 4.

Рисунок 5 — Результат запуска lb.exe с ключом выгрузки.

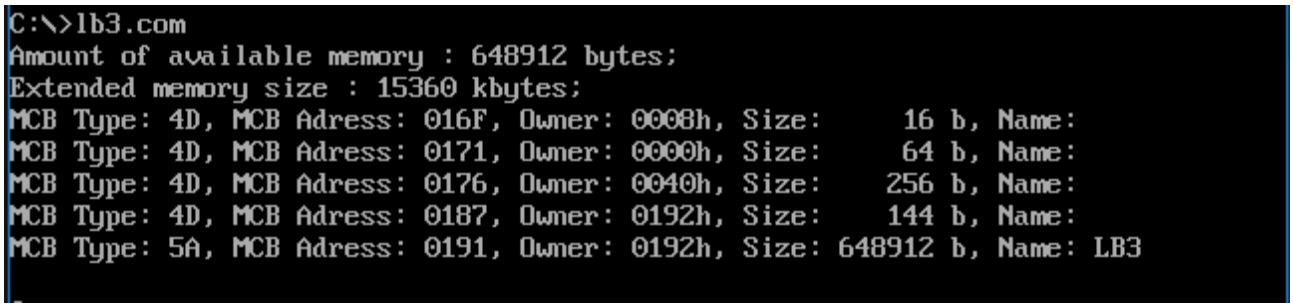

Рисунок 6 — Результат запуска lb3.com после выгрузки прерывания.

Вывод программы lb3.com демонстрирует то, что блоки памяти, в которых хранилось пользовательское прерывание — удалены, следовательно, и само прерывание выгружено.

Исходный код программ см. в приложении А

Контрольные вопросы.

1) Как реализован механизм прерывания от часов?

Механизм прерывания от часов реализован так:

При возникновении сигнала часов (сигналы генерируются аппаратурой через

определенные интервалы времени) — возникает прерывание с определенным значением вектора: сохраняется состояние регистров текущей программы, для последующего возвращения, из вектора прерывания считываются CS и IP обработчика, и следом управление передаётся функции, чья точка входа записана в соответствующий вектор прерывания. Прерывание обрабатывается и затем управление вновь переходит к прерванной программе.

2) Какого типа прерывания использовались в работе?

В работе использовались программные прерывания (int 10h, int 21h) и аппаратные (int 1Ch).

Выводы.

В ходе выполнения был изучен механизм обработки прерываний в DOS.

Разработана программа, загружающая и выгружающая пользовательское прерывание от системного таймера в память.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: lb.asm

```
AStack SEGMENT STACK
        DW 100 DUP(?)
AStack ENDS

DATA SEGMENT
LOADING db 'User interrupt loaded...' , 0DH, 0AH, '$'
LOADED db 'User interrupt installed!', 0DH, 0AH, '$'
UNLOADED db 'User interrupt unloaded!' , 0DH, 0AH, '$'
DATA ENDS

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

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

setCurs PROC
;Установка              позиции
        курсора push AX
        push  BX
        push DX
        push CX
        mov AH,02h
        mov BH,0
        int 10h  ; выполнение.
        pop CX
        pop DX
        pop  BX
        pop AX
        ret
setCurs ENDP

getCurs PROC
; 03H читать позицию и размер курсора
;  вход: BH = видео страница
; выход: DH,DL = текущие строка, колонка курсора
;       CH,CL = текущие начальная, конечная строки курсора
```

```
        push AX
        push BX
        push CX
        mov AH,03h
        mov BH,0
        int 10h    ; выполнение.
        pop CX
        pop BX
        pop AX
        ret
getCurs ENDP

ROUT PROC FAR
   jmp _ROUT

        SIGN db '0000'
        KEEP_CS dw 0    ; для хранения сегмента
        KEEP_IP dw 0    ; и смещения прерывания
        KEEP_PSP dw 0
        VAL db 0
        _STACKdw 100 dup (0)
        KEEP_SS dw 0
        KEEP_AX         dw 0
        KEEP_SP dw 0
        COUNT db '   Number of calls: 00000    ','$'

_ROUT:
        mov KEEP_SS, SS
        mov KEEP_AX, AX
        mov KEEP_SP, SP
        mov AX, seg
        _STACK mov SS, AX
        mov SP, 0
        mov               AX,
        KEEP_AX push AX
        push DX
        push DS
        push ES
        cmp    VAL,
        1 je RES
        call getCurs
        push DX
        mov DH, 23 ; DH,DL = строка, колонка (считая от 0)
        mov DL, 0
        call setCurs
ROUT_SUM:
        push SI
        push CX
        push DS
```

```
        push AX
        mov AX, seg COUNT
        mov DS, AX
        mov BX, offset
        COUNT add BX, 21
        mov SI, 3
lp:
        mov AH,[BX+SI]
        add AH, 1
        cmp AH, 58
        jne ROUT_NEXT
        mov AH, 48
        mov [BX+SI], AH
        sub SI, 1
        cmp SI, 0
        jne lp
ROUT_NEXT:
        mov [BX+SI], AH
        pop DS
        pop SI
        pop BX
        pop AX
        push ES
        push BP
        mov AX, seg COUNT
        mov ES, AX
        mov AX, offset
        COUNT mov BP, AX
        mov AH, 13h
        mov AL, 1      ; sub function code
        mov CX, 28     ;число экземпляров символа для записи
        mov BH, 0      ;номер видео страницы
        int 10h        ;выполнить функцию
        pop BP
        pop ES
        pop DX
        call setCurs
        jmp FINAL
RES
:       cli
        mov DX, KEEP_IP
        mov AX, KEEP_CS
        mov DS, AX
        mov AH, 25h
        mov AL, 1CH
        int 21H     ; восстанавливаем вектор
        mov ES, KEEP_PSP
        mov ES, ES:[2Ch]
        mov AH, 49h
```

```
        int 21h
        mov ES, KEEP_PSP
        mov AH, 49h
        int 21h
        sti
FINAL:
        pop  ES
        pop DS
        pop DX
        pop AX
        mov AX, KEEP_SS
        mov SS, AX
        mov  SP,  KEEP_SP
        mov AX, KEEP_AX

        mov AL, 20H
        out 20H,AL

        iret
ROUT ENDP

CHECK PROC
        mov AH, 35h       ; функция получения
        вектора mov AL, 1Ch ; номер вектора
        int 21h
        mov SI, offset SIGN
        sub SI, offset ROUT
        mov AX,'00'
        cmp AX, ES:[BX+SI]
        jne UNLOAD
        cmp AX, ES:[BX+SI+2]
        je LOAD
UNLOAD:
        call SET_INT
        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
        add DX, CODE
        sub DX, KEEP_PSP
        xor AL, AL
        mov AH, 31h
        int 21h
LOAD:
        push ES
        push AX
        mov AX, KEEP_PSP
        mov ES, AX
        cmp byte ptr ES:[82h],'/'
```

```
        jne stop
        cmp byte ptr ES:
        [83h],'u' jne stop
        cmp byte ptr ES:
        [84h],'n' je _UNLOAD
stop
:       pop AX
        pop ES
        mov DX, offset
        LOADED call PRINT
        ret
_UNLOAD:
        pop AX
        pop ES
        mov byte ptr ES:[BX+SI+10], 1
        mov DX, offset UNLOADED
        call PRINT
        ret
CHECK ENDP

SET_INT PROC
        push DX
        push DS
        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

        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
        mov DX, offset
        LOADING call PRINT
        pop DX
        ret
SET_INT ENDP

Main PROC FAR
        mov AX, DATA
        mov DS, AX
        mov KEEP_PSP, ES
        call CHECK
        xor AL, AL
        mov AH, 4Ch
```

```
        int 21h
    Main ENDP

        LAST_BYTE:
    CODE ENDS
      END Main
```

Название файла: lb3.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING,
    SS:NOTHING ORG   100H
START: jmp BEGIN
; ДАННЫЕ

AM db 'Amount of available memory :    bytes;', 0DH, 0AH, '$'
EM db 'Extended memory size :    kbytes;', 0DH, 0AH, '$'
MCB db 'MCB Type:   , MCB Adress:    , Owner:    h, Size:    b, Name: $'
END_STR db 0DH, 0AH,'$'

; ПРОЦЕДУРЫ
;.....
TETR_TO_HEX PROC near
    and AL, 0Fh
    cmp AL, 09
    jbe NEXT
    add AL, 07
NEXT:  add AL, 30h
    ret
TETR_TO_HEX ENDP
;.....-
BYTE_TO_HEX PROC
near
; байт в AL переводится в два символа 16-го числа в AX
    push CX
    mov AH, AL
    call TETR_TO_HEX
    xchg AL, AH
    mov CL, 4
    shr AL, CL
    call TETR_TO_HEX ; в AL старшая
    цифра pop CX     ; в AH младшая
    ret
BYTE_TO_HEX ENDP
;.....
WRD_TO_HEX PROC
near
; перевод в 16 с/с 16-ти разрядного числа
; в AX- число, в DI- адрес последнего символа
```

```
push BX
mov BH, AH
```

```
        call BYTE_TO_HEX
        mov [DI], AH
        dec DI
        mov [DI], AL
        dec DI
        mov AL, BH
        call BYTE_TO_HEX
        mov [DI], AH
        dec DI
        mov [DI], AL
        pop BX
        ret
WRD_TO_HEX ENDP
;--------------------------------------------------------
BYTE_TO_DEC PROC near
; перевод в 10 с/с, в SI- адрес поля младшей цифры
        push CX
        push DX
        xor AH, AH
        xor DX, DX
        mov CX, 10
loop_bd: div CX
        or DL, 30h
        mov [SI], DL
        dec SI
        xor DX, DX
        cmp AX, 10
        jae loop_bd
        cmp AL, 00h
        je end_1
        or AL, 30h
        mov [SI], AL
end_1:  pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP
;--------------------------------------
; КОД
WRD_TO_DEC PROC near
        push CX
        push DX
        mov CX, 10
loop_wd:
        div CX
        or DL, 30h
        mov [SI], DL
        dec SI
        xor DX, DX
        cmp AX, 10
```

```
        jae loop_wd
        cmp AL, 00h
        je end_2
        or  AL,  30h
        mov [SI], AL
end_2
:       pop DX
        pop CX
        ret
WRD_TO_DEC ENDP

PRINT PROC near
        mov AH, 09h
        int 21h
        ret
PRINT ENDP

_AM PROC near
        mov SI, offset AM
        add SI, 34
        mov AH, 4Ah
        mov BX, 0FFFFh
        int 21h
        mov AX, 16
        mul BX
        call WRD_TO_DEC
        mov   DX,   offset
        AM call PRINT
        ret
_AM ENDP

_EM PROC near
        mov SI, offset EM
        add SI, 27
        sub DX, DX

        mov AL, 30h ; запись адреса ячейки CMOS
        out 70h, AL
        in AL, 71h     ; чтение младшего байта
        mov BL, AL   ; размер расширенной памяти
        mov AL, 31h ; запись адреса ячейки CMOS
        out 70h, AL
        in AL, 71h     ; чтение старшего байта
                       ; размера расширенной памяти
        mov AH, AL
        mov Al, BL
        call WRD_TO_DEC
        mov   DX,   offset
        EM call PRINT
```

```
        ret
_EM ENDP

_MCB PROC  near
        mov AH, 52h
        int 21h
        sub BX, 2
        mov AX, ES:[BX]
        mov ES, AX
Chain:
;Type
        mov DI, offset MCB
        add DI, 10
        mov AX, ES:[00h]
        call BYTE_TO_HEX
         mov [DI], AL
         add DI, 1
        mov [DI], AH
;Adress
        mov DI, offset MCB
        add DI, 29
        mov AX, ES
        call WRD_TO_HEX
;Owner
        mov DI, offset MCB
        add DI, 42
         mov AX, ES:[01h]
        call WRD_TO_HEX
;Size
        mov SI, offset MCB
         add SI, 57
        mov AX, ES:[03h]
        mov BX, 16
        mul BX
         call WRD_TO_DEC
;Print
        mov DX, offset MCB
        call PRINT
;Name
          mov DI, offset MCB
        add DI, 58
        mov BX, 8
        mov CX, 7
        cycle:
        mov DL, ES:[BX]
                mov AH, 02h
                int 21h
                add BX, 1
                loop cycle
```

```
        mov AL, ES:[0h]
        cmp AL, 5ah
        je final

        mov BX, ES
        mov AX, ES:[03h]
        add AX, BX
        add AX, 1
        mov ES, AX
        mov DX, offset END_STR
        call PRINT
        jmp Chain
final:
      ret
_MCB ENDP

BEGIN
:       call   _AM
        call   _EM
        call _MCB
; Выход в DOS
        xor AL, AL
        mov AH, 4Ch
        int 21h
ENDING:
TESTPC  ENDS
        END START
```