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: ИССЛЕДОВАНИЕ ОРГАНИЗАЦИИ УПРАВЛЕНИЯ ПАМЯТЬЮ

Студент гр. 0382 Сергеев Д.А.

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 3 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 4 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.

В ходе работы был взят шаблон из методических материалов, также были написаны процедуры:

- CHECK_UPLOAD_KEY – проверяет указан ли параметр /un при запуске программы
- UNLOAD – выгружает из памяти пользовательское прерывание, освобождает память и восстанавливает исходный вектор прерываний
- MY_INT – резидентное прерывание, которое загружается в память и подсчитывает количество вызванных прерываний, также выводит его на консоль
- IF_ALREADY_LOAD – проверяет, загружено ли прерывание

В ходе выполнения первого и второго шага задания была написана и отлажена программа, удовлетворяющая условиям:

```
C:\>lab4.exe
Loading of interruption went successfully

C:\>PROG1.COM
Available Memory in Bytes: 644400
Extended Memory in KiloBytes: 15360

MCB  List:
MCB #1 Address: 016F PSP TYPE:  Belong MS DOS       Size: 0001 SC/SD:
MCB #2 Address: 0171 PSP TYPE:  Free PSP            Size: 0004 SC/SD:
MCB #3 Address: 0176 PSP TYPE:  0040                Size: 0010 SC/SD:
MCB #4 Address: 0187 PSP TYPE:  0192                Size: 0009 SC/SD:
MCB #5 Address: 0191 PSP TYPE:  0192                Size: 010F SC/SD:  LAB4
MCB #6 Address: 02A1 PSP TYPE:  02AC                Size: 0009 SC/SD:
MCB #7 Address: 02AB PSP TYPE:  02AC                Size: 9D53 SC/SD:  PROG1

C:\>
```

Рисунок 1 – Демонстрация корректной работы программы

Используя программу из предыдущей лабораторной работы, можно увидеть, что наша программа загружена в память.

Рисунок 2 – Работа счётчика прерываний

Счётчик изменяется очень быстро, так как это прерывание от часов.

При повторном вызове написанной программы выводится сообщение о том, что наше прерывание уже загружено:

```
C:\>lab4.exe
Loading of interruption went successfully

C:\>PROG1.COM
Available Memory in Bytes: 644400
Extended Memory in KiloBytes: 15360

MCB  List:
MCB #1 Address: 016F PSP TYPE:  Belong MS DOS       Size: 0001 SC/SD:
MCB #2 Address: 0171 PSP TYPE:  Free PSP            Size: 0004 SC/SD:
MCB #3 Address: 0176 PSP TYPE:  0040                Size: 0010 SC/SD:
MCB #4 Address: 0187 PSP TYPE:  0192                Size: 0009 SC/SD:
MCB #5 Address: 0191 PSP TYPE:  0192                Size: 010F SC/SD:  LAB4
MCB #6 Address: 02A1 PSP TYPE:  02AC                Size: 0009 SC/SD:
MCB #7 Address: 02AB PSP TYPE:  02AC                Size: 9D53 SC/SD:  PROG1

C:\>lab4.exe
Interruption was already loaded

C:\>_
```

Рисунок 3 – Демонстрация повторного запуска программы

Далее программа была вызвана уже с параметром /un и это спровоцировало выгрузку написанного прерывания, освобождение памяти и восстановление исходного вектора прерываний:

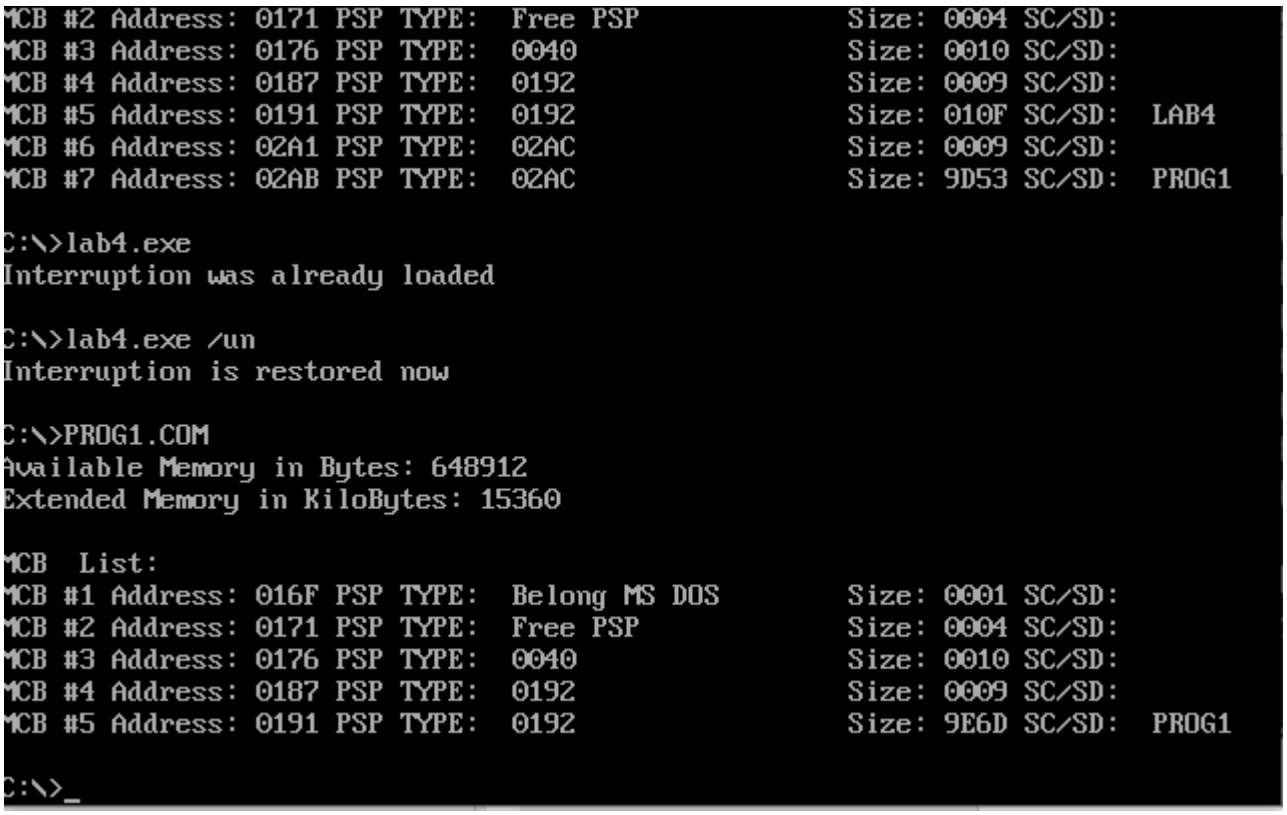

Рисунок 4 – Демонстрация деактивации прерывания

Исходный программный код смотрите в приложении А.

Контрольные вопросы.

1. Как реализован механизм прерывания от часов?

Ответ: прерывание от часов (прерывание int 1Ch) вызывается с помощью обработчика аппаратного прерывания от системного таймера int 08h. Происходит это около 18 раз в секунду.

При вызове данного прерывания прежде всего сохраняются значения регистров IP и CS для дальнейшего возвращения в программу.

Далее по номеру источника прерывания определяется смещение (адрес) вызываемого вектора, он записывается в регистры IP и CS, после чего обработчик прерывания вызывается по этому сохраненному адресу.

В конце управление возвращается прерванной программе от обработчика прерываний.

2. Какого типа прерывания использовались в программе?

Ответ: в данной работе были использованы следующие прерывания: 1Ch – это аппаратное прерывание, а также 10h и 21h – это программные прерывания.

Выводы.

В ходе проделанный работы был построен собственный обработчик прерываний сигналов таймера. Также были получены дополнительные знания о работе с памятью (резидентный обработчик может быть загружен и выгружен из памяти).

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab4.asm

```
ASTACK SEGMENT STACK
   DW 200 DUP(?)
ASTACK ENDS

DATA SEGMENT
    already_load_str db 'Interruption was already loaded', 0DH, 0AH, '$'
    success_load_str db 'Loading of interruption went successfully', 0DH, 0AH,
'$'
    not_load_str db 'Interruption isnt load', 0DH, 0AH, '$'
    restored_str db 'Interruption is restored now', 0DH, 0AH, '$'
DATA ENDS

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

;-----------------------------------
    PRINT_MESSAGE  PROC  NEAR
        push AX
        mov AH, 9
        int 21h
        pop AX
        ret
    PRINT_MESSAGE  ENDP
;-----------------------------------

;-----------------------------------
    SET_CURSOR PROC NEAR
        mov ah, 02h
        mov bh, 0h
        mov dh, 0h
        mov dl, 0h
        int 10h
        ret
    SET_CURSOR ENDP

    GET_CURSOR PROC NEAR
        mov ah, 03h
        mov bh, 0
        int 10h
        ret
    GET_CURSOR ENDP
;-----------------------------------

;-----------------------------------
    MY_INT PROC FAR
        jmp begin_proc
        counter db 'Interruption counter: 0000$'
        signature dw 7777h
        keep_ip dw 0
        keep_cs dw 0
        address_of_psp dw ?
        keep_ss dw 0
```

```
    keep_sp dw 0
    keep_ax dw 0
    my_stack dw 16 dup(?)

begin_proc:
    mov keep_sp, SP
    mov keep_ax, AX
    mov AX, SS
    mov keep_ss, AX
    mov AX, keep_ax
    mov SP, offset begin_proc
    mov AX, seg my_stack
    mov SS, AX
    push AX
    push CX
    push DX

    call GET_CURSOR
    push DX
    call SET_CURSOR
    push SI
    push CX
    push DS
    push BP
    mov AX, seg counter
    mov DS, AX
    mov SI, offset counter
    add SI, 21
    mov CX, 4

loop_for_count:
    mov BP, CX
    mov AH, [SI+BP]
    inc AH
    mov [SI+BP], AH
    cmp AH, 3ah
    jne for_print
    mov AH, 30h
    mov [SI+BP], AH
    loop loop_for_count

for_print:
    pop BP
    pop DS
    pop CX
    pop SI
    push ES
    push BP
    mov AX, seg counter
    mov ES,AX
    mov AX, offset counter
    mov BP,AX
    mov AH, 13h
    mov AL, 00h
    mov CX, 26
    mov BH,0
    int 10h
    pop BP
```

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

        pop DX
        pop CX
        pop AX
        mov keep_ax, AX
        mov SP, keep_sp
        mov AX, keep_ss
        mov SS, AX
        mov AX, keep_ax
        mov AL, 20h
        out 20h, AL
        iret
    my_int_last:
    MY_INT ENDP
;-----------------------------------

;-----------------------------------
    CHECK_UPLOAD_KEY PROC NEAR
        push AX
        push BP
        mov CL, 0h
        mov BP, 81h
        mov AL, ES:[BP + 1]
        cmp AL, '/'
        jne exit
        mov AL, ES:[BP + 2]
        cmp AL, 'u'
        jne exit
        mov AL, ES:[BP + 3]
        cmp AL, 'n'
        jne exit
        mov CL, 1h

    exit:
        pop BP
        pop AX
        ret
    CHECK_UPLOAD_KEY ENDP

    IF_ALREADY_LOAD PROC NEAR
        push AX
        push DX
        push ES
        push SI
        mov CL, 0h
        mov AH, 35h
        mov AL, 1ch
        int 21h
        mov SI, offset signature
        sub SI, offset MY_INT
        mov DX, ES:[BX+SI]
        cmp DX, signature
        jne if_end
```

```
    mov CL, 1h

if_end:
    pop SI
    pop ES
    pop DX
    pop AX
    ret
IF_ALREADY_LOAD ENDP

LOAD PROC NEAR
    push AX
    push CX
    push DX
    call IF_ALREADY_LOAD
    cmp CL, 1h
    je already_load
    mov address_of_psp, ES
    mov AH, 35h
    mov AL, 1ch
    int 21h
    mov keep_cs, ES
    mov keep_ip, BX
    push ES
    push BX
    push DS
    lea DX, MY_INT
    mov AX, seg MY_INT
    mov DS, AX
    mov AH, 25h
    mov AL, 1ch
    int 21h
    pop DS
    pop BX
    pop ES
    mov DX, offset success_load_str
    call PRINT_MESSAGE
    lea DX, my_int_last
    mov CL, 4h
    shr DX, CL
    inc DX
    add DX, 100h
    xor AX,AX
    mov AH, 31h
    int 21h
    jmp end_load

already_load:
    mov DX, offset already_load_str
    call PRINT_MESSAGE

end_load:
    pop DX
    pop CX
    pop AX
    ret
LOAD ENDP
```

```
    UNLOAD PROC NEAR
        push AX
        push SI
        call IF_ALREADY_LOAD
        cmp CL, 1h
        jne not_load
        cli
        push DS
        push ES
        mov AH, 35h
        mov AL, 1ch
        int 21h
        mov SI, offset keep_ip
        sub SI, offset MY_INT
        mov DX, ES:[BX+SI]
        mov AX, ES:[BX+SI+2]
        mov DS, AX
        mov AH, 25h
        mov AL, 1ch
        int 21h
        mov AX, ES:[BX+SI+4]
        mov ES, AX
        push ES
        mov AX, ES:[2ch]
        mov ES, AX
        mov AH, 49h
        int 21h
        pop ES
        mov AH, 49h
        int 21h
        pop ES
        pop DS
        sti
        mov DX, offset restored_str
        call PRINT_MESSAGE
        jmp end_unload

    not_load:
        mov DX, offset not_load_str
        call PRINT_MESSAGE

    end_unload:
        pop SI
        pop AX
        ret
    UNLOAD ENDP
;-----------------------------------

;-----------------------------------
    MAIN PROC FAR
        mov AX, DATA
        mov DS, AX
        call CHECK_UPLOAD_KEY
        cmp CL, 0h
        jne unload_key
        call LOAD
        jmp end_main
```

```
    unload_key:
        call UNLOAD

    end_main:
        xor AL, AL
        mov AH, 4ch
        int 21h
    MAIN ENDP

CODE ENDS

END MAIN
```