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 _____ Соседков К.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

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.

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 МСВ. 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.

При выполнении работы 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.

Для проверки размещения прерывания в памяти использовалась программа из лабораторной работы №3.

Результаты работы программы представлены на рисунках 1-4 ниже.

Рисунок 1. После загрузки прерывания

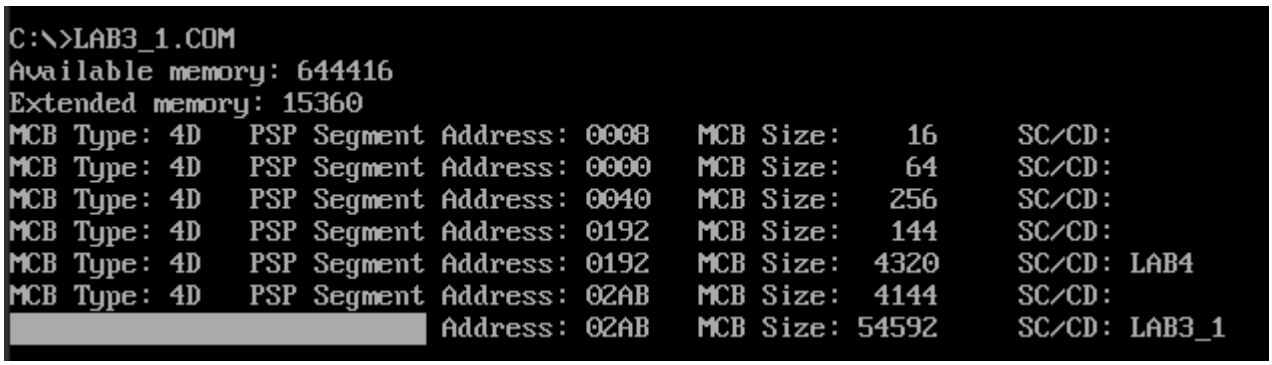

Рисунок 2. Память после загрузки прерывания

Рисунок 3: После выгрузки прерывания

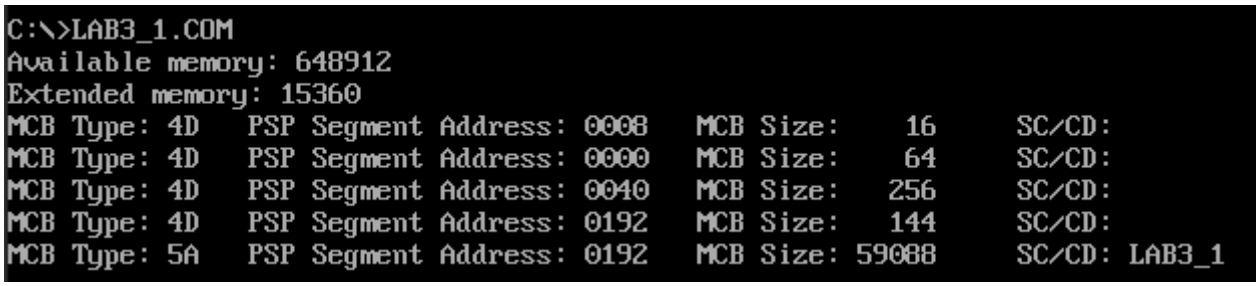

Рисунок 4: Память после выгрузки прерывания

Контрольные вопросы.

1) Как реализован механизм прерывания от часов?

1.1) принимается сигнал прерывания

1.2) сохраняются значения регистров

1.3) по номеру источника прерывания в таблице векторов определяется смещение

1.4) запоминается адрес 2 байта в IP и 2 байта в CS

1.5) выполняется прерывание по сохранённому адресу

1.6) восстанавливается информация прерванного процесса и управление возвращается прерванной программе

2) Какого типа прерывания использовались в работе?

2.1) аппаратные(пример: прерывание от часов - 1ch)

2.2) программные(DOS(21h), BIOS(10h))

Выводы.

При выполнении лабораторной работы был реализован обработчик прерывания от сигналов таймера.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: lab4.asm

```
ASTACK segment stack
	dw 256 dup(?)
ASTACK ends



DATA segment
	int_already_loaded db 'Interruption_already_load',0dh,0ah,0dh,0ah,'$'
	interruption_loaded db 'Interruption_load',0dh,0ah,0dh,0ah,'$'
	interruption_delete db 'Interruption_was_delete',0dh,0ah,0dh,0ah,'$'
DATA ends



CODE segment
   assume cs:CODE, ds:DATA, ss:ASTACK


CUSTOM_INTERRUPTION proc far
	jmp start

	KEEP_CS dw 0
	KEEP_IP dw 0
	KEEP_PSP dw 0
	KEEP_SS dw 0
	KEEP_SP dw 0
	KEEP_AX dw 0
```

```
        int_counter db 'interruption_counter: 0000$'
        int_sig dw 9999h
        int_seg dw 16 dup(?)

start:
        mov KEEP_SP,sp
        mov KEEP_AX,ax
        mov ax,ss
        mov KEEP_SS,ax

        mov ax,KEEP_AX
        mov sp,offset start
        mov ax,seg int_seg
        mov ss,ax

        push ax
        push cx
        push dx

        call getCurs
        push dx
        call setCurs

        push si
        push cx
        push ds
        push bp

        mov ax,seg int_counter
        mov ds,ax
        mov si,offset int_counter
        add si,21
        mov cx,4
```

```
loop_:
        mov bp,cx
        mov ah,[si+bp]
        inc ah
        mov [si+bp],ah
        cmp ah,3ah
        jne update
        mov ah,30h
        mov [si+bp],ah
        loop loop_

update:
        pop bp
        pop ds
        pop cx
        pop si

        push es
        push bp

        mov ax,seg int_counter
        mov es,ax
        mov ax,offset int_counter
        mov bp,ax

        mov ah,13h
    mov al,0
    mov bh,0
    mov cx,26
    int 10h

        pop bp
        pop es
        pop dx
```

```
        mov ah,2
        mov bh,0
        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
LAST:
CUSTOM_INTERRUPTION endp




outputAL proc
        push ax
        push bx
        push cx
        mov ah,09h
        mov bh,0
        mov cx,1
        int 10h
        pop cx
        pop bx
        pop ax
```

```
        ret
outputAL endp



outputBP proc
        push ax
        push bx
        push dx
        push cx
        mov ah,13h
        mov al,1
        mov bh,0
        mov dh,22
        mov dl,0
        int 10h
        pop cx
        pop dx
        pop bx
        pop ax
        ret
outputBP endp



setCurs proc
        mov ah,02h
        mov bh,0
        mov dh,22
        mov dl,0
```

```
        int 10h
        ret
setCurs endp



getCurs proc
        mov ah,03h
        mov bh,0
        int 10h
        ret
getCurs endp



UNLOAD_CUSTOM_INTERRUPTION proc
        cli
        push ds
        push es

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

        mov si,offset KEEP_IP
        sub si,offset CUSTOM_INTERRUPTION
        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 interruption_delete
        call PRINT
        ret
UNLOAD_CUSTOM_INTERRUPTION endp



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

```
        pop ax
        ret
PRINT endp



CHECK_CMD proc far
   mov al, es:[81h+1]
        cmp al, '/'
        jne set_zero

        mov al, es:[81h+2]
        cmp al, 'u'
        jne set_zero

        mov al, es:[81h+3]
        cmp al, 'n'
        jne set_zero

   mov ax, 1h
   jmp check_cmd_exit

set_zero:
   mov ax, 0h

check_cmd_exit:
   ret
CHECK_CMD endp
```

```
IS_LOADED proc far
        mov ah, 35h
        mov al, 1ch
        int 21h

        mov si, offset int_sig
        sub si, offset CUSTOM_INTERRUPTION
        mov dx, es:[bx + si]
        cmp dx, int_sig
        jne not_loaded
        mov ax, 1h
    jmp is_loaded_exit

not_loaded:
    mov ax, 0h

is_loaded_exit:
    ret
IS_LOADED endp




LOAD_CUSTOM_INTERRUPTION proc far
        mov KEEP_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,CUSTOM_INTERRUPTION
        mov ax,seg CUSTOM_INTERRUPTION
        mov ds,ax

        mov ah,25h
        mov al,1ch
        int 21h

        pop ds
        pop bx
        pop es

        mov dx,offset interruption_loaded
        call PRINT

        lea dx,LAST
        mov cl,4h
        shr dx,cl
        inc dx

        add dx,100h
        xor ax,ax

        mov ah,31h
        int 21h

        ret
LOAD_CUSTOM_INTERRUPTION endp
```

```
MAIN proc far
        mov ax, DATA
        mov ds, ax

    push es
    call IS_LOADED
    cmp ax, 0h
    jne check_cmd_un

    call LOAD_CUSTOM_INTERRUPTION
    pop es
    jmp exit

check_cmd_un:
    pop es
    call CHECK_CMD
    cmp ax, 0h
    je already_loaded
    call UNLOAD_CUSTOM_INTERRUPTION
    jmp exit

already_loaded:
        mov dx,offset int_already_loaded
        call PRINT

exit:
        xor al,al
        mov ah,4ch
        int 21h
MAIN endp

CODE ends
end main
```