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. 9381 _____ Прибылов Н.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

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 и выполняет соответствующие действия.

В данной лабораторной работе 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.

Ход работы.

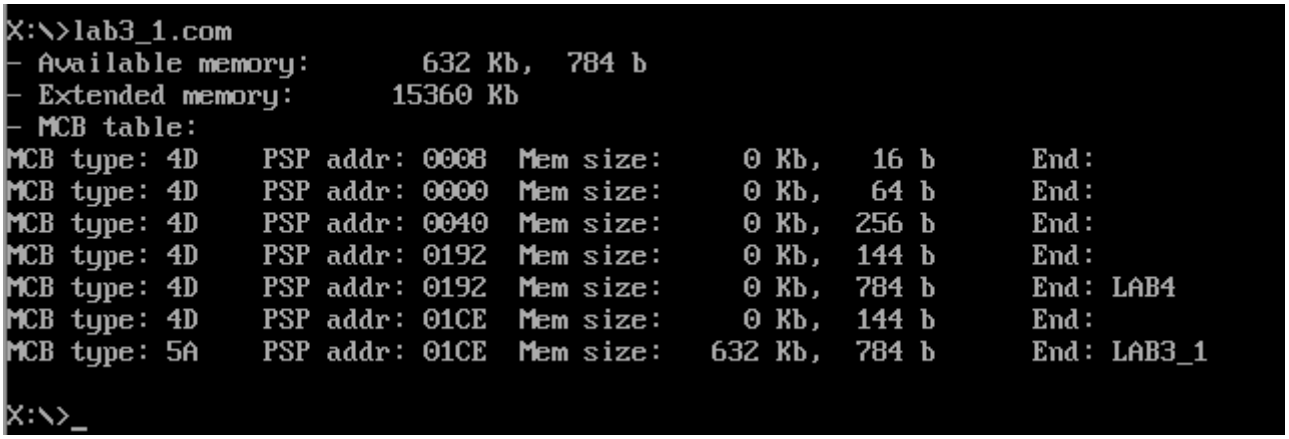
1. Был написан и отлажен программный модуль типа .EXE, который выполняет следующие функции:

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

2. Далее была запущена отлаженная программа, проверено, что резидентный обработчик прерывания 1Ch установлен. Сделано так, чтобы

работа прерывания отображалась на экране, а также проверено размещение прерывания в памяти. Для этого была запущена программа лабораторной работы №3, которая отображает карту памяти в виде списка блоков MCB.

Программа запущена



Программа из лаб. работы №3 запущена

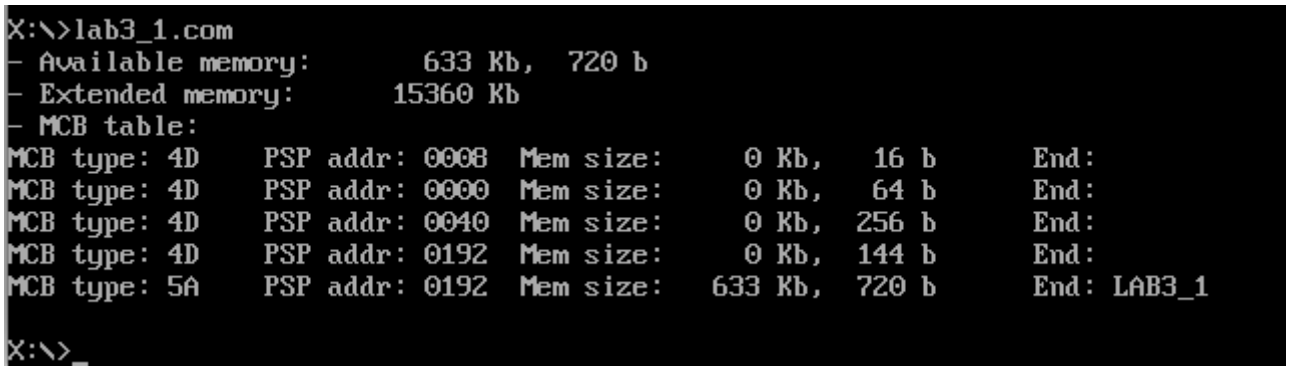
3. Была запущена отлаженная программа еще раз и проверено, что она определяет установленный обработчик прерываний.

Программа запущена повторно

4. Была запущена отлаженная программа с ключом выгрузки и проверено, что резидентный обработчик прерывания выгружен, то есть сообщения на

экран не выводятся, а занятая резидентом память освобождена. Для этого также была запущена программа лабораторной работы №3.

Программа запущена с ключом /un



Программа из лаб. работы №3 запущена

Описание функций и переменных.

Переменные:

Название	Описание
PSP_ADDRESS_1	Хранит старое значение ES до того, как программа была оставлена резидентной в памяти
KEEP_CS	Хранит сегмент прерывания
KEEP_IP	Хранит смещение прерывания
NEW_INTERRUPT_SET	Хранит количество вызванных прерываний
INT_COUNT	Строка "Interrupts call count: 0000"
STRING_INT_NOT_LOADED	Строка "Interrupt was not loaded."

STRING_INT_RESTORED	Строка "Interrupt was restored."
STRING_INT_ALREADY_LOADED	Строка "Interrupt is already loaded."
STRING_INT_IS_LOADING	Строка "Interrupt is loading now."

Функции:

NEW_INTERRUPT	Собственный обработчик прерывания. Выводит количество прерываний, которые были вызваны.
IS_INTERRUPT_SET	Проверяет, установлен ли вектор прерывания
CHECK_COMMAND_PROMT	Проверяет параметр un
LOAD_INTERRUPT	Устанавливает новые обработчики прерывания
UNLOAD_INTERRUPT	Восстанавливает сохранённые заранее обработчики прерываний и выгружает резидентную программу
PRINT_STRING	Печатает строку на экран

Контрольные вопросы.

1. Как реализован механизм прерывания от часов?

Принимается сигнал прерывания (примерно каждые 54 мс), запоминаются значения регистров, по номеру источника прерывания в таблице векторов определяется смещение, сохраняются адреса 2 байта в IP и 2 байта в CS. Далее выполняется прерывание по сохранённому адресу и восстанавливается информация прерванного процесса, управление возвращается прерванной программе.

2. Какого типа прерывания использовались в работе?

Аппаратные прерывания, прерывания DOS (21h), прерывания BIOS (10h).

Выводы.

Был построен обработчик прерывания от сигналов таймера. Изучены дополнительные функции работы с памятью: установка программы-резидента и его выгрузка из памяти.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab4.asm

```
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:STACK

NEW_INTERRUPT PROC FAR
    jmp START_FUNC

    PSP_ADDRESS_0       dw 0
    PSP_ADDRESS_1       dw 0
    KEEP_CS             dw 0
    KEEP_IP             dw 0
    NEW_INTERRUPT_SET   dw 0FEDCh
    INT_COUNT           db 'Interrupts call count: 0000  $'

    KEEP_SS             dw ?
    KEEP_SP             dw ?
    KEEP_AX             dw ?
    INT_STACK           dw 64 dup (?)
    END_INT_STACK       dw ?

START_FUNC:
    mov     KEEP_SS, ss
    mov     KEEP_SP, sp
    mov     KEEP_AX, ax
    mov     ax, cs
    mov     ss, ax
    mov     sp, offset END_INT_STACK

    push    ax
    push    bx
    push    cx
    push    dx

    mov     ah, 03h
    mov     bh, 00h
    int     10h
    push    dx

    mov     ah, 02h
    mov     bh, 00h
    mov     dx, 0220h
    int     10h

    push    si
    push    cx
    push    ds
    mov     ax, SEG INT_COUNT
    mov     ds, ax
    mov     si, offset INT_COUNT
    add     si, 1Ah
```

```
    mov     ah,[si]
    inc     ah
    mov     [si], ah
    cmp     ah, 3Ah
    jne     END_CALC
    mov     ah, 30h
    mov     [si], ah

    mov     bh, [si - 1]
    inc     bh
    mov     [si - 1], bh
    cmp     bh, 3Ah
    jne     END_CALC
    mov     bh, 30h
    mov     [si - 1], bh

    mov     ch, [si - 2]
    inc     ch
    mov     [si - 2], ch
    cmp     ch, 3Ah
    jne     END_CALC
    mov     ch, 30h
    mov     [si - 2], ch

    mov     dh, [si - 3]
    inc     dh
    mov     [si - 3], dh
    cmp     dh, 3Ah
    jne     END_CALC
    mov     dh, 30h
    mov     [si - 3],dh

END_CALC:
    pop     ds
    pop     cx
    pop     si

    push    es
    push    bp
    mov     ax, SEG INT_COUNT
    mov     es, ax
    mov     ax, offset INT_COUNT
    mov     bp, ax
    mov     ah, 13h
    mov     al, 00h
    mov     cx, 1Dh
    mov     bh, 0
    int     10h
    pop     bp
    pop     es

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

    pop     dx
    pop     cx
```

```
    pop     bx
    pop     ax

    mov     ss, KEEP_SS
    mov     ax, KEEP_AX
    mov     sp, KEEP_SP
    mov     AL, 20H
    out     20H, AL

    iret
NEW_INTERRUPT ENDP

NEED_MEM_AREA PROC
NEED_MEM_AREA ENDP

IS_INTERRUPT_SET PROC NEAR
    push    bx
    push    dx
    push    es

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

    mov     dx, es:[bx + 11]
    cmp     dx, 0FEDCh
    je      INT_IS_SET
    mov     al, 00h
    jmp     POP_REG

INT_IS_SET:
    mov     al, 01h
    jmp     POP_REG

POP_REG:
    pop     es
    pop     dx
    pop     bx

    ret
IS_INTERRUPT_SET ENDP

CHECK_COMMAND_PROMT PROC NEAR
    push    es

    mov     ax, PSP_ADDRESS_0
    mov     es, ax

    mov     bx, 0082h

    mov     al, es:[bx]
    inc     bx
    cmp     al, '/'
    jne     NULL_CMD

    mov     al, es:[bx]
    inc     bx
    cmp     al, 'u'
```

```
    jne     NULL_CMD

    mov     al, es:[bx]
    inc     bx
    cmp     al, 'n'
    jne     NULL_CMD

    mov     al, 0001h
NULL_CMD:
    pop     es

    ret
CHECK_COMMAND_PROMT ENDP

LOAD_INTERRUPT PROC NEAR
    push    ax
    push    bx
    push    dx
    push    es

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

    mov     KEEP_IP, bx
    mov     KEEP_CS, es

    push    ds
    mov     dx, offset NEW_INTERRUPT
    mov     ax, seg NEW_INTERRUPT
    mov     ds, ax

    mov     ah, 25h
    mov     al, 1Ch
    int     21h
    pop     ds

    mov     dx, offset STRING_INT_IS_LOADING
    call    PRINT_STRING

    pop     es
    pop     dx
    pop     bx
    pop     ax

    ret
LOAD_INTERRUPT ENDP

UNLOAD_INTERRUPT PROC NEAR
    push    ax
    push    bx
    push    dx
    push    es

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

```
    cli
    push    ds
    mov     dx, es:[bx + 9]
    mov     ax, es:[bx + 7]

    mov     ds, ax
    mov     ah, 25h
    mov     al, 1Ch
    int     21h
    pop     ds
    sti

    mov     dx, offset STRING_INT_RESTORED
    call    PRINT_STRING

    push    es
    mov     cx, es:[bx + 3]
    mov     es, cx
    mov     ah, 49h
    int     21h
    pop     es

    mov     cx, es:[bx + 5]
    mov     es, cx
    int     21h

    pop     es
    pop     dx
    pop     bx
    pop     ax

    ret
UNLOAD_INTERRUPT ENDP

PRINT_STRING PROC NEAR
    push    ax
    mov     ah, 09h
    int     21h
    pop     ax
    ret
PRINT_STRING ENDP

MAIN_PROGRAM PROC FAR
    mov     bx, 02Ch
    mov     ax, [bx]
    mov     PSP_ADDRESS_1, ax
    mov     PSP_ADDRESS_0, ds
    sub     ax, ax
    xor     bx, bx

    mov     ax, DATA
    mov     ds, ax

    call    CHECK_COMMAND_PROMT
    cmp     al, 01h
    je      UNLOAD_START

    call    IS_INTERRUPT_SET
```

```
        cmp     al, 01h
        jne     INTERRUPT_IS_NOT_LOADED

        mov     dx, offset STRING_INT_ALREADY_LOADED
        call    PRINT_STRING
        jmp     EXIT_PROGRAM

        mov     ah,4Ch
        int     21h

    INTERRUPT_IS_NOT_LOADED:
        call    LOAD_INTERRUPT

        mov     dx, offset NEED_MEM_AREA
        mov     cl, 04h
        shr     dx, cl
        add     dx, 1Bh

        mov     ax, 3100h
        int     21h

    UNLOAD_START:
        call    IS_INTERRUPT_SET
        cmp     al, 00h
        je      INT_IS_NOT_SET
        call    UNLOAD_INTERRUPT
        jmp     EXIT_PROGRAM

    INT_IS_NOT_SET:
        mov     dx, offset STRING_INT_NOT_LOADED
        call    PRINT_STRING
        jmp     EXIT_PROGRAM

    EXIT_PROGRAM:
        mov     ah, 4Ch
        int     21h
    MAIN_PROGRAM ENDP

    CODE ENDS
    STACK SEGMENT STACK
        db 64 DUP(?)
    STACK ENDS

    DATA SEGMENT
         STRING_INT_NOT_LOADED         db "Interrupt was not loaded.",
0dh, 0ah, '$'
        STRING_INT_RESTORED          db "Interrupt was restored.", 0dh,
0ah, '$'
         STRING_INT_ALREADY_LOADED    db "Interrupt is already loaded.",
0dh, 0ah, '$'
         STRING_INT_IS_LOADING         db "Interrupt is loading now.",
0dh, 0ah, '$'
    DATA ENDS

    END MAIN_PROGRAM
```