МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Операционные системы»
Тема: Обработка стандартных прерываний

Студент гр. 8382 _____ Нечепуренко Н.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Построить резидентный обработчик прерываний сигналов таймера, который будет выводить на экран информацию о количестве прерываний.

Выполнение работы.

Для обработки прерываний сигналов таймера был написан исполняемый .exe модуль, запоминающий исходный вектор прерывания 1ch и подменяющий его на пользовательский обработчик. По завершению, обработчик остается в резидентной части оперативной памяти. Результат работы программы приведен на рисунке 1.

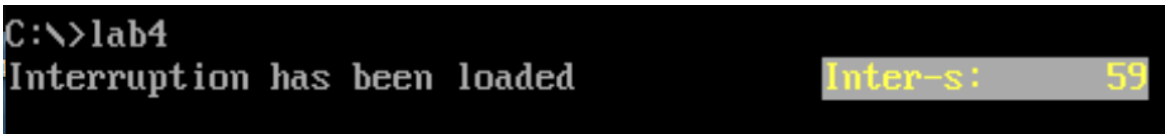

Рисунок 1 – Результат выполнения программы

Для восстановления исходного вектора прерывания необходимо запустить программы с ключом «/un». Результат такого вызова приведен на рисунке 2.

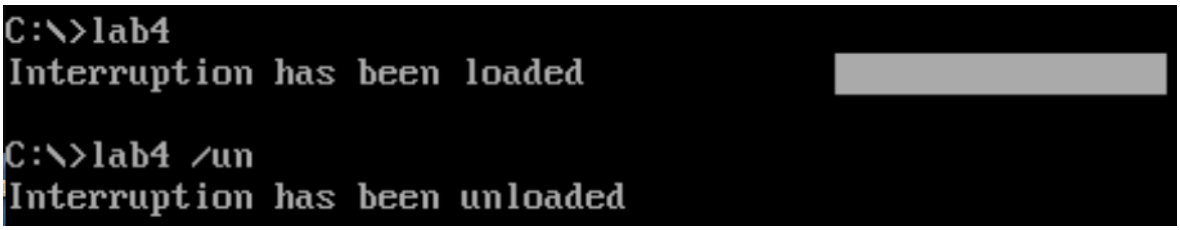

Рисунок 2 – Запуск программы с ключом «/un»

Проверим, что после восстановления вектора прерывания, резидентный обработчик очищает после себя память. Рассмотрим MCB блоки с помощью программы из лабораторной работы № 3. Результат приведен на рисунке 3

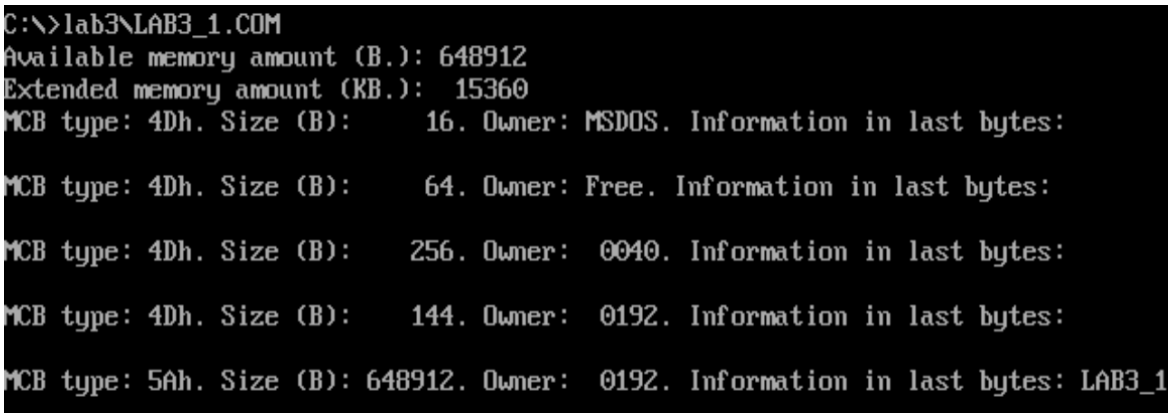

Рисунок 3 – MCB блоки операционной памяти

Запустим программу еще раз без ключа, отслеживая изменение блоков оперативной памяти. Результат представлен на рисунке 4.

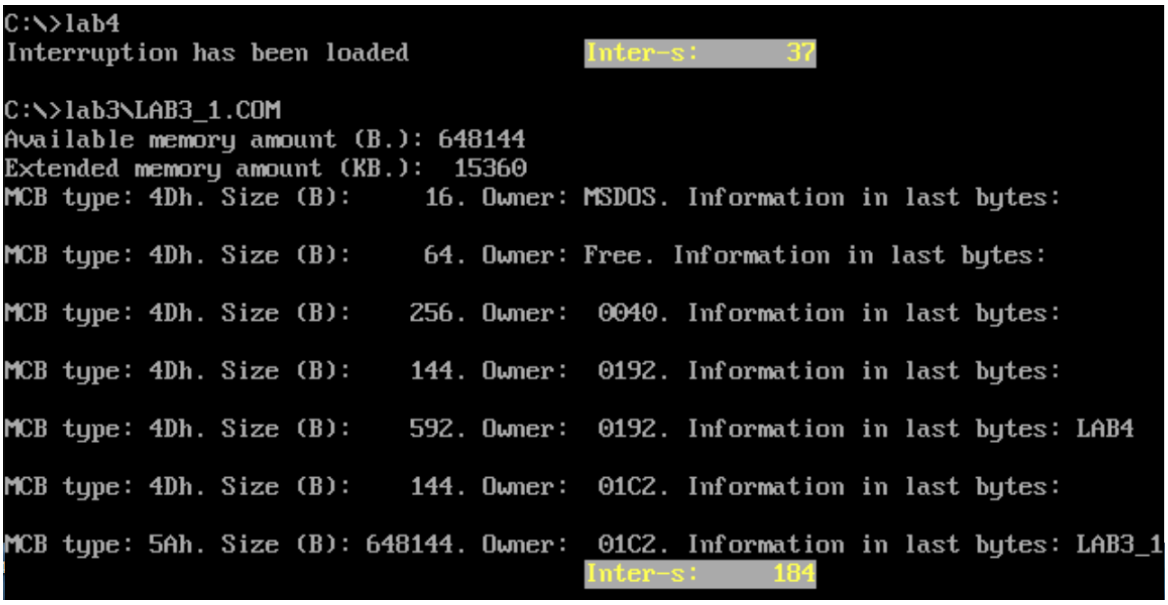

Рисунок 4 – MCB блоки после запуска резидента

Заметим появление двух новых блоков. Теперь запустим программу с ключом «/un» и еще раз рассмотрим блоки оперативной памяти (см. рис. 5).

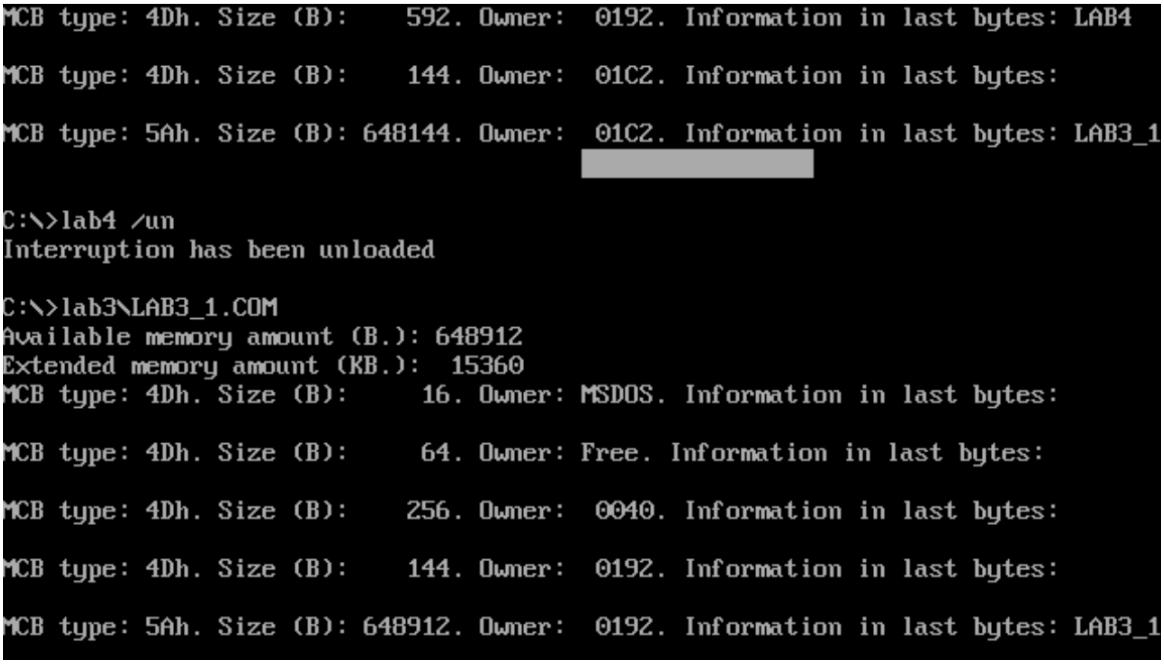

Рисунок 5 – MCB блоки после выгрузки резидента

В результате чего можно сделать вывод о корректности работы программы.

Контрольные вопросы.

Как реализован механизм прерываний от часов?

Каждые 55 миллисекунд (~18.2 раз в секунду) аппаратные часы генерируют прерывание, которое может быть обработано пользовательским обработчиком (int 1ch). Для обработки низкоуровневых программных прерываний (например, прерывание от клавиатуры) необходимо выполнить две команды `mov al, 20h` и `out 20h, al`, получив корректный обработчик.

Какого типа прерывания использовались в работе?

В работе использовалось прерывание операционной системы DOS 21h, прерывание BIOS 10h, отвечающее за видео режим, вывод символов, строк и графических примитивов. В обработчике прерываний запрещен вызов функций DOS, потому что аппаратное прерывание может прийти в любой момент, даже когда выполняется какая-либо функция DOS, что может привести к конфликту. По этой причине и было использовано прерывание 10h. Также было использовано прерывание DOS 31h для размещения обработчика в резидентной

памяти и аппаратное прерывание 1ch, которое перехватывал обработчик и выводил на экран результат.

Выводы.

В результате выполнения лабораторной работы были получены навыки написания собственных обработчиков стандартных прерываний (в данном случае прерываний таймера), разработки модулей резидентных программ, а также обработки ключей командной строки.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ.

```
codeseg segment
    assume cs:codeseg, ss:astack, ds:dataseg

main proc far
    jmp resident_start
resident_data:
    inter_signature dw 1337h
    keep_cs dw 0
    keep_ip dw 0
    keep_psp dw 0
    keep_ss dw 0
    keep_sp dw 0
    keep_ax dw 0
    count dw 0
    msg_count db "Inter-s:        ", 13, 10, "$"
    inter_stack dw 30 dup("?")
resident_start:
    mov keep_ss, ss
    mov keep_sp, sp
    mov keep_ax, ax; stack is not properly tuned, so we have to save ax
this way
    mov ax, seg inter_stack
    mov ss, ax
    add ax, 100h
    mov sp, ax

    push bx
    push cx
    push dx
    push si
    push ds
    push bp
    push es

    mov ax, seg resident_data
    mov ds, ax
resident_work:
    inc count
```

```
    mov ax, count
    xor dx, dx
    mov si, offset msg_count
    add si, 15
    call WRD_TO_DEC

resident_get_cursor:
    mov ah, 03h
    mov bh, 0
    int 10h
    push dx
resident_set_cursor:
    mov ah, 02h
    mov bh, 0
    mov dh, 22
    mov dl, 40
    int 10h

resident_print:
    mov ax, seg msg_count
    mov es, ax
    mov bp, offset msg_count
    mov ah, 13h
    mov al, 1
    mov bh, 0
    mov cx, 10h
    int 10h

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

resident_final:
    pop es
    pop bp
    pop ds
    pop si
    pop dx
    pop cx
    pop bx
```

```
    mov sp, KEEP_SP
    mov ax, KEEP_SS
    mov ss, ax
    mov ax, KEEP_AX
    mov al, 20h
    out 20h, al
    iret

main endp
resident_part_end:

WRD_TO_DEC PROC NEAR
            push  cx
            push  dx
            mov   cx,10
loop_b: div       cx
            or          dl,30h
            mov   [si],dl
            dec   si
            xor   dx,dx
            cmp   ax,10
            jae   loop_b
            cmp   al,00h
            je          endl
            or          al,30h
            mov   [si],al
endl: pop   dx
            pop   cx
            ret
WRD_TO_DEC ENDP


init proc far
    mov ax, dataseg
    mov ds, ax
    mov keep_psp, es
    call check_un
    mov ax, un_flag
    cmp ax, 0
    jne init_reset
    call set_inter
    jmp init_final
```

```
init_reset:
    call reset_inter
init_final:
    mov ax, 4c00h
    int 21h
    ret
init endp

set_inter proc near
    push ax
    push bx
    push dx
    push di
    push cx
    push ds
    push es


set_inter_get_prev:
    mov ax, 351ch
    int 21h
    mov keep_cs, es
    mov keep_ip, bx

set_inter_set_new:
    push ds
    mov dx, offset main
    mov ax, seg main
    mov ds, ax
    mov ah, 25h
    mov al, 1ch
    int 21h
    pop ds

    mov di, offset msg_inter_loaded
    call print

set_inter_make_resident:
    mov dx, offset resident_part_end
    xor cx, cx
    mov cl, 4
    shr dx, cl
```

```
    add dx, 16h
    inc dx
    mov ah, 31h
    int 21h
set_inter_final:
    pop es
    pop ds
    pop cx
    pop di
    pop dx
    pop bx
    pop ax
    ret
set_inter endp


reset_inter proc near
    push ax
    push bx
    push dx
    push ds
    push es
    push si
    push di
reset_inter_get_prev:
    mov ax, 351ch
    int 21h
    mov si, offset inter_signature
    sub si, offset main
    mov ax, es:[bx+si] ; get resident_data
    cmp ax, 1337h ; check signature to be equal 1337h
    jne reset_inter_final

reset_inter_restore:
    cli
    push ds
    mov dx, es:[bx+si+4]; ip
    mov ax, es:[bx+si+2]; cs
    mov ds, ax
    mov ax, 251ch
    int 21h
    pop ds
```

```
    sti
reset_inter_free_memory:
    mov ax, es:[bx+si+6]; keep_psp
    mov es, ax
    push es
    mov ax, es:[2ch]; there is adr in psp of memory needs to be free
    mov es, ax
    mov ah, 49h
    int 21h
    pop es
    mov ah, 49h
    int 21h
    mov di, offset msg_inter_unloaded
    call print

reset_inter_final:
    pop di
    pop si
    pop es
    pop ds
    pop dx
    pop bx
    pop ax
    ret
reset_inter ENDP

check_un proc near
    push ax
    push es
    mov ax, keep_psp
    mov es, ax
    ; check cmd tail
    cmp byte ptr es:[81h+1], "/"
    jne check_un_final
    cmp byte ptr es:[81h+2], "u"
    jne check_un_final
    cmp byte ptr es:[81h+3], "n"
    jne check_un_final
    cmp byte ptr es:[81h+4], 13
    jne check_un_final
    mov ax, 1
    mov un_flag, ax
```

```
check_un_final:
    pop es
    pop ax
    ret
check_un endp

print proc near
    ; prints di content
    push dx
    push ax
    mov ah, 9h
    mov dx, di
    int 21h
    pop ax
    pop dx
    ret
print endp

codeseg ends

astack segment stack
    dw 100 dup("?")
astack ends

dataseg segment
    un_flag dw 0
    msg_inter_loaded db "Interruption has been loaded", 13, 10, "$"
    msg_inter_unloaded db "Interruption has been unloaded", 13, 10, "$"
dataseg ends

end init
```