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 _____ Терехов А.Е.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

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 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.

Ход работы.

В ходе работы был написан EXE модуль, код которого представлен в приложении А, который проверяет установлено ли пользовательское прерывание с вектором 1ch. Если не установлено, то устанавливает резидентный обработчик для его обработки и выходит по функции 4ch прерывания 21h. Если установлено, то сообщает об этом и выходит аналогичным образом. По ключу /un можно выгрузить обработчик, восстановить исходный вектор прерываний и освободить память, занимаемую резидентом.

Проверка установки прерывания состоит в следующем. Считывается адрес, записанный в векторе прерывания, и сравнивается придуманная нами сигнатура, с сигнатурой расположенной на известном смещении.

Резидентный обработчик выполняет следующие действия. Он сохраняет значения изменяемых регистров при входе и восстанавливает при выходе, а также считает количество прерываний и выводит его на экран. Результат работы программы представлен на рисунке 1.

Рисунок 1. Результат запуска программы.

На рисунках 2а (доказательство работы обработчика), 2б (полный вывод программы), представлен результат работы программы из лабораторной работы №3, выводящей блоки управления памятью, как видно резидент продолжает работать и выводить на экран количество тиков. Также можно заметить, что появился еще один блок, которого не было в прошлой работе. Это блок написанного в данной работе резидента.

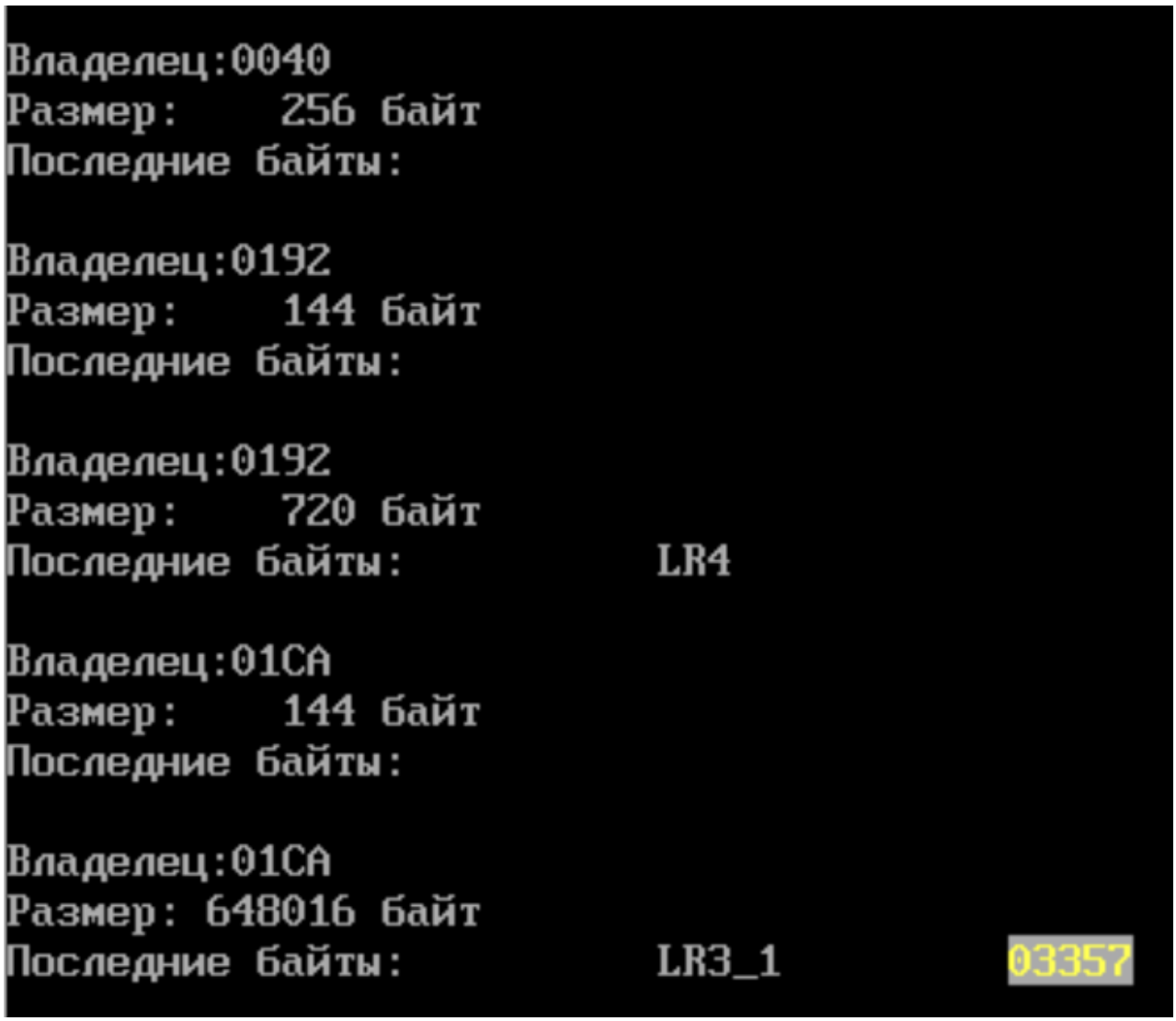

Рисунок 2а. Вывод MCB в окне DOSBOX.

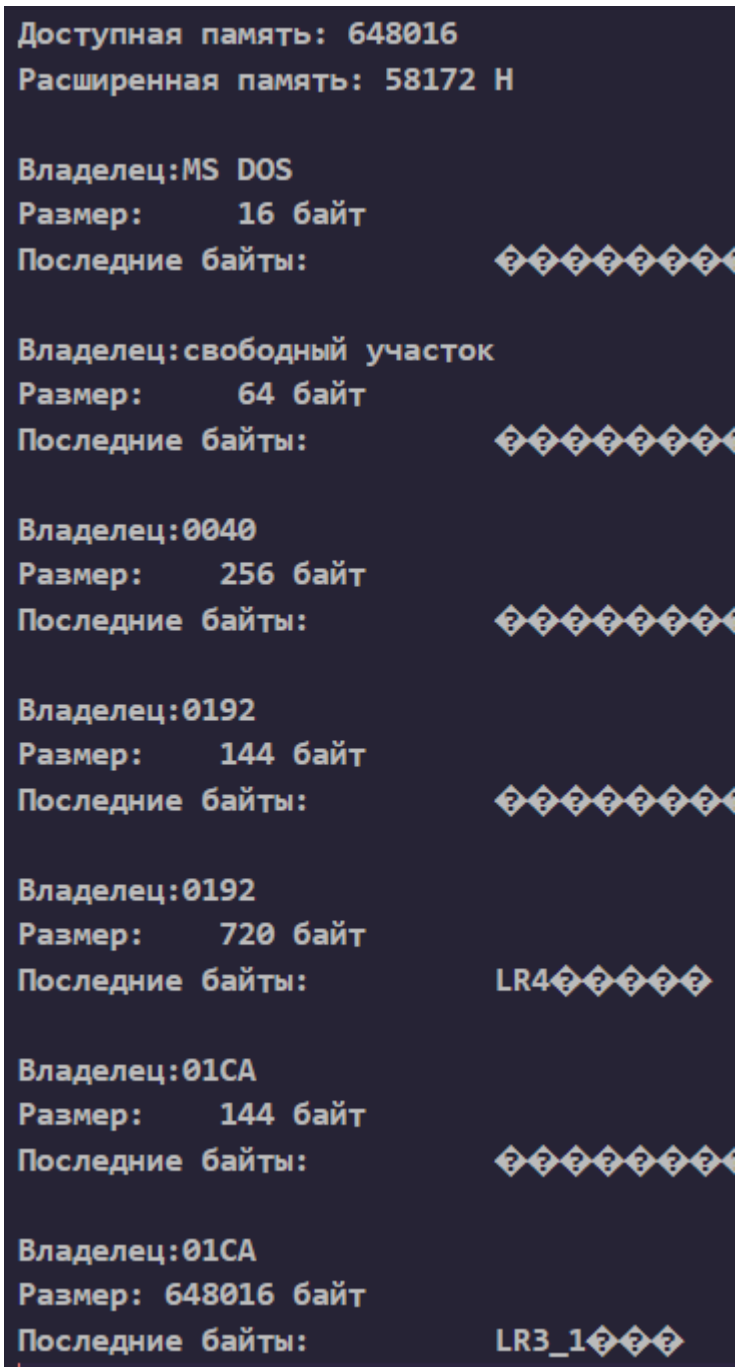

Рисунок 2б. Вывод MCB в файл.

При повторном запуске программы будет выведено сообщение о том, что обработчик уже загружен и повторно он загружаться не станет. На рисунке 3 представлено доказательство этого.

Рисунок 3. Результат повторного запуска программы.

При запуске программы с ключом /un обработчик будет выгружен (рисунок 4), и будет очищена память, которую он занимал (рисунок 5).

Рисунок 4. Результат запуска программы с ключом /un.

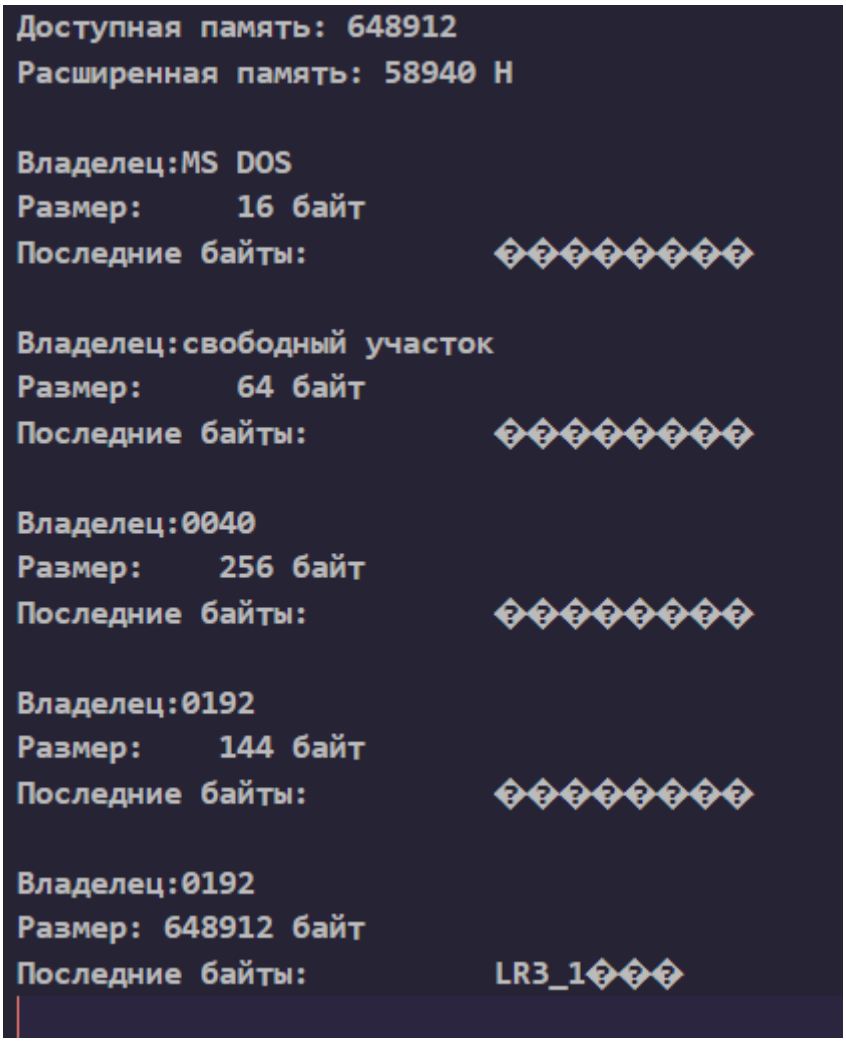

Рисунок 5. Рисунок 2б. Вывод МСВ в файл после выгрузки обработчика.

Ответы на вопросы.

1) Как реализован механизм прерывания от часов?

Примерно 18 раз в секунду происходит сохранение регистров и определяется источник прерывания, по номеру которого определяется смещение в таблице векторов. На стеке также сохраняется адрес возврата и регистр флагов. Из таблицы векторов прерывания берется адрес входа в обработчик, который загружается в CS:IP. Происходит выполнение действий обработчика. После этого восстанавливаются регистры и по сохраненному адресу возврата управление возвращается прерванной программе.

2) Какого типа прерывания использовались в работе?

Аппаратное – 1ch.

Программные – 21h (DOS) и 10h (BIOS).

Вывод.

В результате работы был реализован резидентный обработчик прерываний сигналов таймера. В программе предусмотрена установка обработчика, защита повторной установки и выгрузка обработчика. Скриншоты делались не по порядку, поэтому на них такие значения счетчика.

ПРИЛОЖЕНИЕ А

```
CODE SEGMENT
ASSUME cs:CODE, ds:DATA, ss:AStack
ROUT PROC far
    jmp handler_start
    HANDLER_SIGN DW 4200h
    KEEP_IP DW 0
    KEEP_CS DW 0
    KEEP_PSP DW 0
    KEEP_SS DW 0
    KEEP_SP DW 0
    KEEP_AX DW 0
    COUNT DW 0
    M_COUNT DB '00000',0Dh,0Ah,'$'
    HANDLER_STACK DW 100 DUP(0)
    stack_top:
handler_start:
    mov KEEP_SS, ss
    mov KEEP_SP, sp
    mov KEEP_AX, ax
    mov ax, seg HANDLER_STACK
    mov ss, ax
    lea sp, stack_top
    push ax
    push bx
    push cx
    push dx
    push si
    push ds
    push bp
    push es

    mov ax, seg HANDLER_STACK
    mov ds, ax
    inc COUNT
    mov ax, COUNT
    xor dx, dx
    lea si, M_COUNT
    add si, 4
    call WRD_TO_DEC
;getCurs
    mov ah, 03h
    mov bh, 0
    int 10h
    push dx
;setCurs
    mov ah, 02h
    mov bh, 0
    mov dh, 22
    mov dl, 40
    int 10h
;output
    mov ax, seg M_COUNT
    mov es, ax
    lea bp, M_COUNT
    mov ah, 13h
```

```
    mov al, 1
    mov bh, 0
    mov cx, 5
    int 10h

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

    pop es
    pop bp
    pop ds
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    mov sp, KEEP_SP
    mov ax, KEEP_SS
    mov ss, ax
    mov ax, KEEP_AX
    mov al, 20h
    out 20h, al
    iret
ROUT ENDP
LAST_BYTE:

CHECK_IS_LOAD proc near
    push ax
    push bx
    push si
    push dx
    push es
    mov ah, 35h
    mov al, 1ch
    int 21h
    lea si, HANDLER_SIGN
    sub si, offset ROUT
    mov ax, es:[bx+si]
    cmp ax, 4200h
    jne END_CHECK
    mov IS_LOAD, 1
END_CHECK:
    pop es
    pop dx
    pop si
    pop bx
    pop ax
    ret
CHECK_IS_LOAD ENDP

WRD_TO_DEC proc near
   push ax
   push cx
   push DX
```

```
   mov cx,10
loop_wd:
   div cx
   or DL,30h
   mov [SI],DL
   dec SI
   xor DX,DX
   cmp ax,0
   jnz loop_wd
end_l1:
   pop DX
   pop cx
   pop ax
   ret
WRD_TO_DEC ENDP


LOAD proc near
    push ax
    push bx
    push cx
    push dx
    push ds
    push es

    mov ah, 35h
    mov al, 1ch
    int 21h
    mov KEEP_IP, bx
    mov KEEP_CS, es

    push ds
    lea dx, ROUT
    mov ax, seg ROUT
    mov ds, ax
    mov ah, 25h
    mov al, 1ch
    int 21h
    pop ds

    lea dx, LAST_BYTE
    mov cl, 4
    shr dx, cl
    add dx, 16h
    inc dx
    mov ah, 31h
    int 21h

    pop es
    pop ds
    pop dx
    pop cx
    pop bx
    pop ax
    ret
LOAD ENDP
```

```
UNLOAD proc near
    cli
    push ax
    push bx
    push dx
    push es
    push si
    push ds

    mov ah, 35h
    mov al, 1ch
    int 21h
    lea si, KEEP_IP
    sub si, offset ROUT

    mov dx, es:[bx+si]
    mov ax, es:[bx+si+2]
    mov ds, ax
    mov ah, 25h
    mov al, 1ch
    int 21h
    pop ds

    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

end_unload:
    pop si
    pop es
    pop dx
    pop bx
    pop ax
    sti
    ret
UNLOAD ENDP

WRITE proc near
        push ax
        mov ah,09h
        int 21h
        pop ax
        ret
WRITE ENDP

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

```
    mov KEEP_PSP, es
    call CHECK_IS_LOAD
    mov al, IS_LOAD
    cmp al, 1
    je loaded
    jmp n_loaded
loaded:
    cmp byte ptr es:[81h+1], '/'
    jne end_main_loaded
    cmp byte ptr es:[81h+2], 'u'
    jne end_main_loaded
    cmp byte ptr es:[81h+3], 'n'
    jne end_main_loaded
    call UNLOAD
    jmp end_main_not_loaded
n_loaded:
    cmp byte ptr es:[81h+1], '/'
    jne load_handler
    cmp byte ptr es:[81h+2], 'u'
    jne load_handler
    cmp byte ptr es:[81h+3], 'n'
    jne load_handler
    jmp end_main_not_loaded
load_handler:
    call LOAD

end_main_loaded:
    lea dx, M_ALREADY_LOADED
    call WRITE
    jmp exit
end_main_not_loaded:
    lea dx, M_NOT_LOADED
    call WRITE
exit:
    xor al, al
    mov ah, 4ch
    int 21h
MAIN ENDP
CODE ENDS
AStack SEGMENT STACK
    DW 100h DUP(?)
AStack ENDS

DATA SEGMENT
    IS_LOAD DB 0
    M_LOADED DB 'ЋЎa Ў®взЁЄ § Jar¦Ґ-!',0Dh,0Ah,'$'
    M_NOT_LOADED DB 'ЋЎa Ў®взЁЄ ўлJar¦Ґ-!',0Dh,0Ah,'$'
    M_ALREADY_LOADED DB 'ЋЎa Ў®взЁЄ г¦Ґ § Jar¦Ґ-!' ,0Dh,0Ah,'$'
DATA ENDS

END MAIN
```