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. 8383 _____ Федоров И.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование организации стандартных прерываний, обработчиков прерываний, способов выгрузки резидентных программ из памяти и загрузки. Построить обработчик прерываний сигналов таймера.

Ход 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.
3. Если прерывание установлено, то выводится соответствующее сообщение.
4. Выгрузка прерывания по соответствующему значению параметра в командной строке /un.

Исходный код **.EXE** модуля приведен в приложении А.

При запуске программы устанавливается новый обработчик прерывания 1Ch, при этом теперь выводится количество вызовов нового обработчика. Результаты работы программы представлены на рис. 1. Карта памяти до запуска программы представлена на рис. 2, после вызова - на рис. 3. (результат работы программы 3-й лабораторной для вывода карты памяти представлены в виде скриншота **.txt** файла, в который перенаправлен вывод).

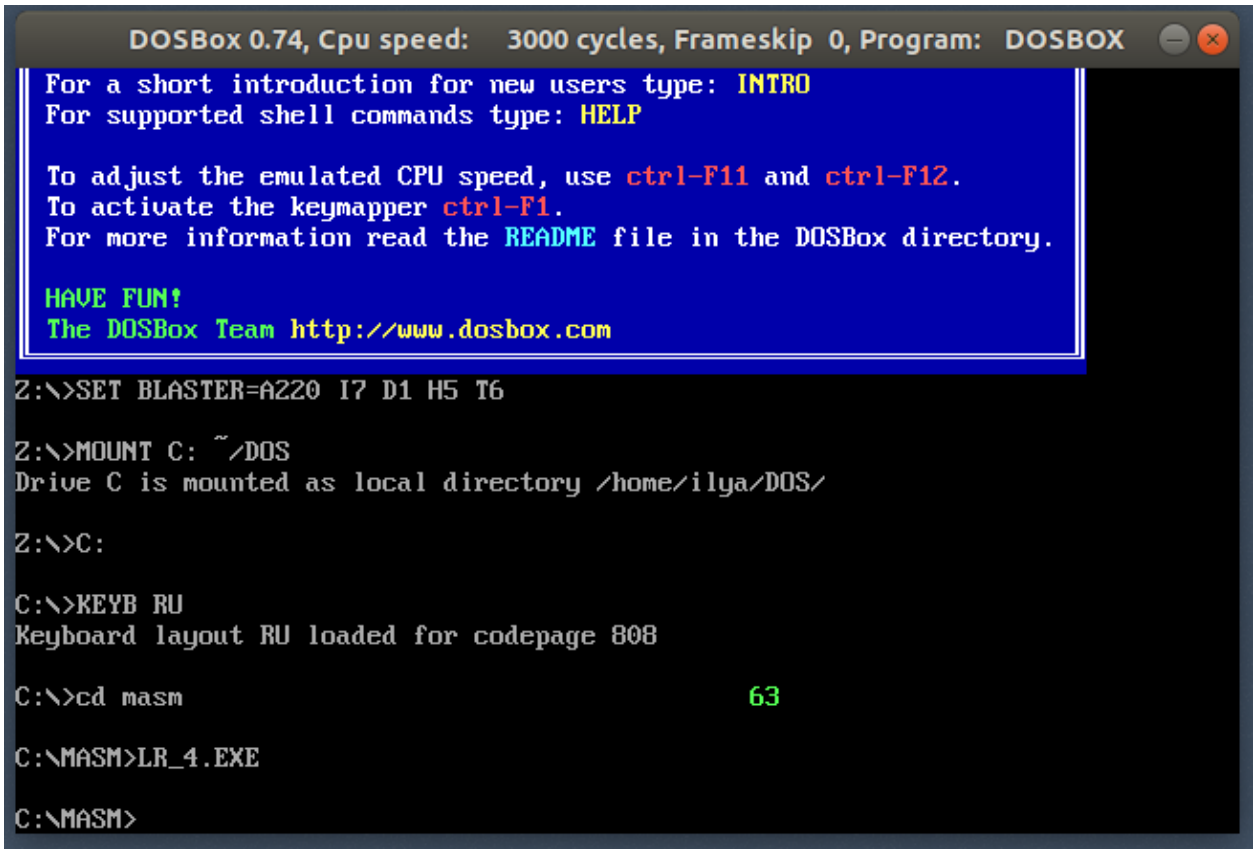

Рисунок 1 – Работа нового прерывания

```
Size available memory (b): 648912
Size extend memory (kb): 15360
--------------------------------
Num: 1
PSP address: 0008
Size (b): 16
Name:
--------------------------------
Num: 2
PSP address: 0000
Size (b): 64
Name:
--------------------------------
Num: 3
PSP address: 0040
Size (b): 256
Name:
--------------------------------
Num: 4
PSP address: 0192
Size (b): 144
Name:
--------------------------------
Num: 5
PSP address: 0192
Size (b): 648912
Name: LR3COM
```

Рисунок 2 – Карта памяти до размещения прерывания в памяти

```
Size available memory (b): 648160
Size extend memory (kb): 15360
--------------------------------
Num: 1
PSP address: 0008
Size (b): 16
Name:
--------------------------------
Num: 2
PSP address: 0000
Size (b): 64
Name:
--------------------------------
Num: 3
PSP address: 0040
Size (b): 256
Name:
--------------------------------
Num: 4
PSP address: 0192
Size (b): 144
Name:
--------------------------------
Num: 5
PSP address: 0192
Size (b): 576
Name: LR_4
--------------------------------
Num: 6
PSP address: 01C1
Size (b): 144
Name:
--------------------------------
Num: 7
PSP address: 01C1
Size (b): 648160
Name: LR3COM
```

Рисунок 3 – Карта памяти после размещения нового прерывания

Отлаженная программа при повторном запуске определяет установлен ли новый обработчик прерываний, проверяя сигнатуру (просто значение), расположенную в резиденте и идентифицирующую его. При совпадении выводится соответствующее сообщение. Результат приведен на рис. 4.

```
C:\MASM>LR_4.EXE                                        88

C:\MASM>LR_4.EXE
INTERRUPT ALREADY LOAD!
C:\MASM>
```

Рисунок 4 – Определение установки обработчика

При запуске программы с соответствующим ключом выгрузки `/un` программа выгружает резидентный обработчик, выводит соответствующее сообщение и освобождает память. Результат работы приведен на рис. 5,

состояние памяти после выгрузки приведено на рис. 6. Выгрузка не срабатывает, если обработчик не был размещен в памяти (см. рис. 7).

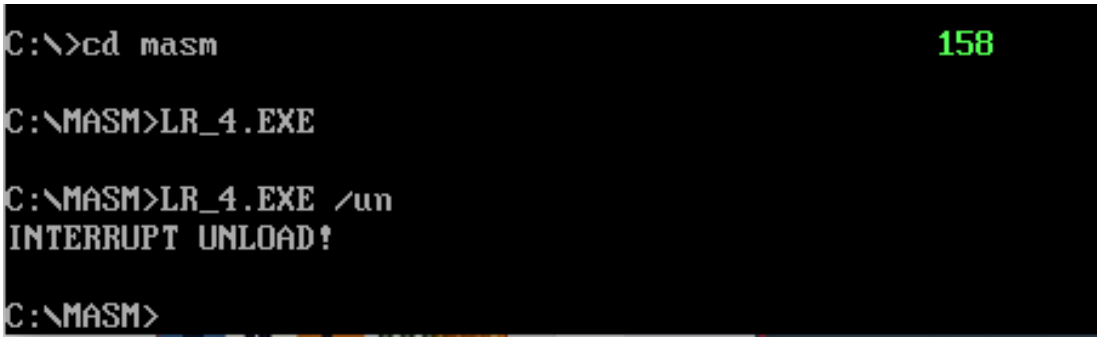

Рисунок 5 – Работа при наличии ключа выгрузки

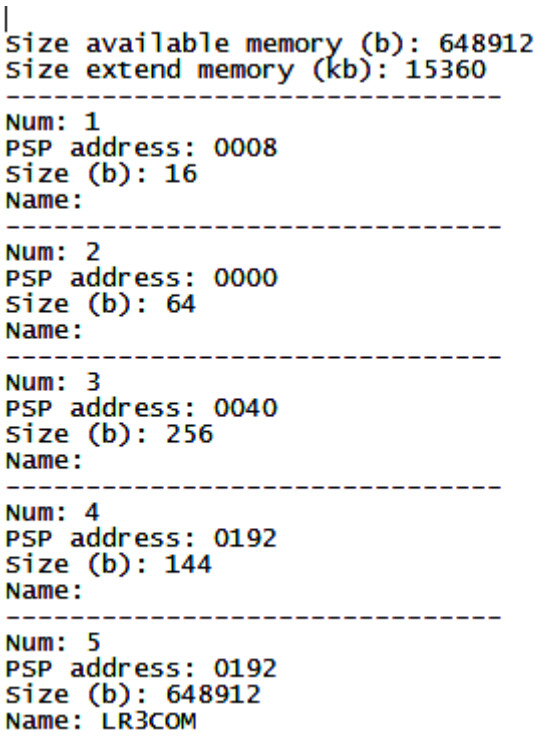

Рисунок 6 – Состояние памяти после выгрузки резидента

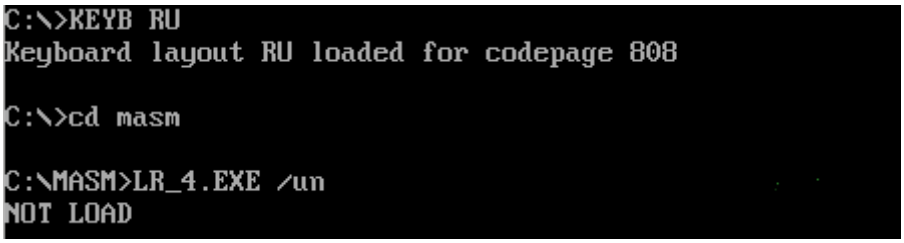

Рисунок 7 – Попытка выгрузки без загрузки

Ответы на контрольные вопросы.

1) Как реализован механизм прерывания от часов?

Таймер вырабатывает сигналы с частотой 18.206 Гц, вызывающий обработчик прерывания (с помощью таблицы вектором прерываний). Этот обработчик с каждым вызовом увеличивает на единицу содержимое двухсловной ячейки, расположенной в области данных BIOS, в нее при начальной загрузки компьютера считывает показания часов реального времени.

18.2 раза в секунду вызывается обработка аппаратного прерывания таймера, которая ненадолго останавливает процессор и выполняет ряд действий: в начале проверяется счетчик на переполнение (который хранится в обработчике), затем сохраняется кадр стека прерванной задачи и необходимые при работе регистры, далее увеличивается суммарное число прерываний, которое выводится на экран. В конце восстанавливаются регистры и кадр стека прерванной задачи.

2) Какого типа прерывания использовались в работе?

Использовались программные прерывания int (21h и 10h), аппаратное 1Ch.

Выводы.

В ходе выполнения работы была написана программа, размещающая и выгружающая новый обработчик прерывания системного таймера, проверяющая его наличие и освобождающая память при вызове с соответствующим ключом.

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

```
code segment
       assume cs:code, ds:data, ss:astack

RESIDENT_INTER PROC FAR
start:  jmp begin
    KEEP_IP dw 0
       KEEP_CS dw 0
    SEGMENT_ dw 0
    COUNT dw 0
       SIGNATURE_INTER dw 777h
    mem dw 0
       ss_seg dw 0
       ss_offs dw 0
       LOCAL_STACK dw 64 dup (0)
       TOP_STACK=$
begin:
   mov ss_seg,ss
   mov ss_offs,sp
   mov mem, ax

   CLI               ;сохранение кадра стека прерванной задачи
   mov ax,cs
   mov ss,ax
   mov sp,offset TOP_STACK
   STI

   mov ax, mem
   push ax
   push bx
   push cx
   push dx
getCurs:
   mov ah,03h
   mov bh,0h
   int 10h
  ;выход: DH,DL = текущие строка, колонка
   push dx
setCurs:
   mov ah,02h
   mov bh,0h
   mov dh,14h
   mov dl,30h   ;DH,DL = строка, колонка
   int 10h      ;выполнение

   push bx
   push ax
   inc COUNT
   mov ax, COUNT
   xor cx,cx
   mov bx,10
print_count:
   xor dx,dx
   div bx
   push dx
```

```
   inc cx
   test ax,ax
   jnz print_count  ;ZF = 0
   mov ah, 02h
print_num:
   pop dx
   add dl, '0'
   int 21h
loop print_num
end_inter:
   pop bx
   pop ax
   pop dx
retCurs:
   mov ah,02h
   mov bh,0h
   int 10h
   pop dx
   pop cx
   pop bx
   xor ax, ax
   mov ax, ss_seg
   mov ss, ax
   pop ax
   mov sp, ss_offs

   mov AL, 20H
   OUT 20H, AL
   IRET
LAST_BYTE:
RESIDENT_INTER ENDP




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


;----------------------
outputAL PROC
       ;call setCurs
       push ax
       push bx
       push cx
       mov ah, 09h   ;писать символ в текущей позиции курсора
       mov bh, 0     ;номер видео страницы
       mov cx, 1     ;число экземпляров символа для записи
       int 10h      ;выполнить функцию
       pop cx
       pop bx
```

```
       pop ax
       ret
outputAL ENDP
;---------------------

VERIFY_TAIL PROC near
   push ax
   push cx
   push es
   mov al, es:[81h+1]     ; es на psp
   cmp al, '/'
   jne error_tail
   mov al, es:[81h+2]
   cmp al, 'u'
   jne error_tail
   mov al, es:[81h+3]
   cmp al, 'n'
   jne error_tail
   inc bool_var
   jmp end_p
error_tail:
   mov bool_var, 0
end_p:
   pop es
   pop cx
   pop ax
   ret
VERIFY_TAIL ENDP



VERIFY_LOADING PROC
   push ax
   push bx
   push si
   push es
   mov AH, 35h
   mov AL, 1Ch;Номер прерывания
   int 21h
   mov si, offset SIGNATURE_INTER
   sub si, offset RESIDENT_INTER
   mov ax, es:[bx+si]
   cmp ax, SIGNATURE
   jne end_
   inc bool_resident
end_:
   pop es
   pop si
   pop bx
   pop ax
   ret
VERIFY_LOADING ENDP


DO_RESIDENT PROC
       push ax
       push bx
```

```
       push dx
       push es
    ;------------------------
       MOV AH, 35h        ; функция получения вектора
       MOV AL, 1CH        ; номер вектора
       INT 21h
       MOV KEEP_IP, BX     ; запоминание смещения
       MOV KEEP_CS, ES     ; и сегмента вектора прерывания
       ;---------------установка прерывания
    PUSH DS
       MOV DX, OFFSET RESIDENT_INTER       ; смещение для процедуры в
DX
       MOV AX, SEG RESIDENT_INTER          ; сегмент процедуры
       MOV DS, AX                    ; помещаем в DS
       MOV AH, 25H                   ; функция установки вектора
       MOV AL, 1CH                   ; номер вектора
       INT 21H                       ; меняем прерывание
       POP DS
       mov DX, offset LAST_BYTE
       add DX, 10Fh                  ;100h + 15(Fh) при /16 в большую
сторону
       mov CL, 4
       shr DX, CL
       inc DX
       xor AX, AX
       mov AH, 31h
       int 21h
       pop es
       mov dx, offset message1
       call WriteMsg
       pop dx
       pop bx
       pop ax
       ret
DO_RESIDENT ENDP



UNLOAD_RESIDENT PROC
        push AX
        push BX
        push DX
        push DS
        push ES
        mov AH, 35h
        mov AL, 1Ch
        int 21h
     mov dx, es:KEEP_IP
     mov ax, es:KEEP_CS
     CLI
        push DS
        mov DS, AX
        mov AH, 25h
        mov AL, 1Ch
        int 21h
        pop DS
        STI
```

```
     mov ax, es:SEGMENT_
        mov es, ax
        push ES
        mov AX, ES:[2Ch]
        mov ES, AX
        mov AH, 49h   ;функция освобождения памяти
        int 21h
        pop ES
        mov AH, 49h
        int 21h
        pop ES
        pop DS
        mov dx, offset message4
        call WriteMsg
        pop DX
        pop BX
        pop AX
       ret
UNLOAD_RESIDENT ENDP


MAIN PROC FAR
       push ds
       xor ax, ax
       push ax
       mov ax, data
       mov ds, ax
       mov SEGMENT_, es
    call VERIFY_TAIL
       call VERIFY_LOADING
       cmp bool_var, 0
if_:je else_unload
    cmp bool_resident, 0
       je not_load
       call UNLOAD_RESIDENT
       jmp end_main
not_load:
       mov dx, offset message2
       call WriteMsg
       jmp end_main
else_unload:
       cmp bool_resident, 0
       ja already
       call DO_RESIDENT
       jmp end_main
already:
       mov dx, offset message3
       call WriteMsg
       jmp end_main
end_main:
       xor AL, AL
       mov AH, 4Ch
       int 21h
MAIN ENDP


code ENDS
```

```
data segment
 bool_resident db 0
 bool_var db 0
 message1 db 'INTERRUPT LOAD!',10,13,'$'
 message2 db 'NOT LOAD$'
 message3 db 'INTERRUPT ALREADY LOAD!$'
 message4 db 'INTERRUPT UNLOAD!',10,13,'$'
 SIGNATURE dw 777h
data ends

astack segment stack
dw 128 dup(?)
astack ends

END MAIN
```