

**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. 8382

\_\_\_\_\_

Колногоров Д.Г.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2020

### **Цель работы.**

Построение обработчика прерывания сигналов таймера.

### **Выполнение работы.**

Был написан программный модуль типа **.EXE** (представлен в приложении А), 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.

Состояние памяти до загрузки обработчика прерывания представлено на рисунке 1.

```
available memory:648912 bytes
extended memory: 15360 bytes

01
owner: MS DOS
size:    16 bytes
last bytes:

02
owner: free
size:    64 bytes
last bytes:

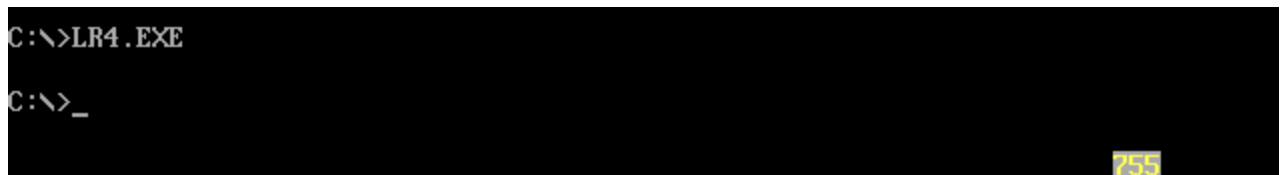
03
owner: 0040
size:    256 bytes
last bytes:

04
owner: 0192
size:    144 bytes
last bytes:

05
owner: 0192
size: 648912 bytes
last bytes: LR3 1
```

Рисунок 1 — состояние памяти до загрузки обработчика прерывания

На рисунке 2 представлен счётчик, отображаемый после загрузки обработчика прерывания.



The image shows a DOS command prompt window with a black background and white text. The first line shows the command 'C:\>LR4.EXE' being entered. The second line shows the prompt 'C:\>\_' followed by a yellow box containing the number '755'.

Рисунок 2 — отображаемый счётчик

На рисунке 3 представлено состояние памяти после загрузки обработчика прерывания.

```

available memory:648320 bytes
extended memory: 15360 bytes

01
owner: MS DOS
size:    16 bytes
last bytes:

02
owner: free
size:    64 bytes
last bytes:

03
owner: 0040
size:    256 bytes
last bytes:

04
owner: 0192
size:    144 bytes
last bytes:

05
owner: 0192
size:    416 bytes
last bytes: LR4

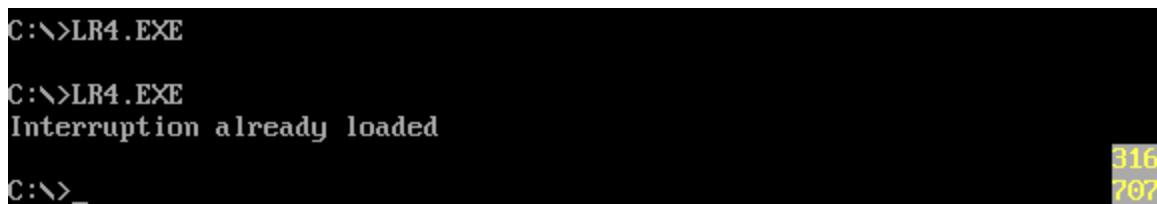
06
owner: 01B7
size:    144 bytes
last bytes:

07
owner: 01B7
size: 648320 bytes
last bytes: LR3 1

```

Рисунок 3 — состояние памяти после загрузки обработчика прерывания

На рисунках 4 и 5 представлено тестирование программы на корректность.



```

C:\>LR4.EXE
C:\>LR4.EXE
Interruption already loaded
C:\>_

```

Рисунок 4 — попытка повторной загрузки обработчика прерывания

```

C:\>LR4.EXE
C:\>LR4.EXE
Interrupt already loaded
C:\>LR4.EXE /un
Restored interruption
C:\>_

```

Рисунок 5 — выгрузка обработчика прерывания

На рисунке 6 представлено состояние памяти после выгрузки обработчика прерывания.

```

available memory:648912 bytes
extended memory: 15360 bytes

```

```

01
owner: MS DOS
size:    16 bytes
last bytes:

```

```

02
owner: free
size:    64 bytes
last bytes:

```

```

03
owner: 0040
size:    256 bytes
last bytes:

```

```

04
owner: 0192
size:    144 bytes
last bytes:

```

```

05
owner: 0192
size: 648912 bytes
last bytes: LR3 1

```

7

Рисунок 6 — состояние памяти после выгрузки обработчика прерывания

### Контрольные вопросы.

1) Как реализован механизм прерывания от часов?

С определённой периодичностью (примерно 18.2 раз в секунду) вызывается обработчик прерывания системного таймера, адрес которого находится в таблице векторов прерываний. Обработчик сохраняет состояние прерванного процесса и запрещает все прерывания от внешних устройств

(посредством сброса флага IF). Далее обработчик инкрементирует счётчик и сбрасывает его при переполнении (в данной работе после этого счётчик выводится на экран). Затем разрешается обработка прерываний от внешних устройств, восстановление состояния прерванного процесса и возврат в прерванную программу.

2) Какого типа прерывания использовались в работе?

Реализуемое в работе прерывание от системного таймера — аппаратное. При выполнении работы также использовались программные прерывания 21h и 10h.

### **Вывод.**

В ходе выполнения лабораторной работы был реализован обработчик прерываний сигналов системного таймера.

# ПРИЛОЖЕНИЕ А

## СОДЕРЖИМОЕ ФАЙЛА LR4.ASM

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, ES:NOTHING, SS:AStack

MY\_INT PROC FAR

    jmp MY\_INT\_START

MY\_INT\_DATA:

    KEEP\_CS    dw 0

    KEEP\_IP    dw 0

    KEEP\_PSP   dw 0

    SIGNATURE dw 1234h

    COUNTER    db '000\$'

MY\_INT\_START:

    push AX

    push BX

    push CX

    push DX

    push SI

    push DS

    push ES

    ; set DS to int's data segment

    mov AX, SEG KEEP\_CS

    mov DS, AX

INC\_COUNTER:

    xor CX, CX

    mov SI, offset COUNTER

    add SI, 2

INC\_DIGIT:

    mov AL, [SI]

    cmp AL, '9'

    je CARRY

    inc AL

    mov [SI], AL

    jmp INC\_DIGIT\_END

CARRY:

```

        mov AL, '0'
        mov [SI], AL

        cmp SI, offset COUNTER
        je CLEAR_COUNTER
        dec SI

        jmp INC_DIGIT
CLEAR_COUNTER:
        mov CX, 2
        clear_digit:
            mov AL, '0'
            mov [SI], AL
            inc SI
            loop clear_digit

INC_DIGIT_END:

SAVE_CURSOR:
    mov AH, 03h
    mov BH, 0
    int 10h
    push DX

SET_CURSOR:
    mov AH, 02h
    mov BH, 0
    mov DX, 1845h      ; DH=row, DL=col (18==last row)
    int 10h

PRINT_COUNTER:
    push ES
    push BP
    mov AX, seg KEEP_CS
    mov ES, AX
    mov BP, offset COUNTER
    mov AH, 13h
    mov AL, 1
    mov BH, 0
    mov CX, 3          ; string length
    int 10h
    pop ES

```



```

        pop BP

RESET_CURSOR:
        pop DX
        mov AH, 02h
        mov BH, 0
        int 10h

        pop ES
        pop DS
        pop SI
        pop DX
        pop CX
        pop BX
        pop AX

        mov AL, 20h
        out 20H, AL
        iret
MY_INT ENDP
MY_INT_END:

CHECK_INT PROC
        ; sets AX to 1 if interruption is already loaded
        ; otherwise sets AX to 0
        push BX
        push CX
        push DX
        push SI
        push ES

        ; get int's segment
        mov AH, 35h
        mov AL, 1Ch
        int 21h

        ; get signature's offset
        mov SI, offset SIGNATURE
        sub SI, offset MY_INT

        ; check signature

```

```

    mov AX, 1
    mov BX, ES:[BX+SI]
    mov CX, SIGNATURE
    cmp BX, CX
    je CHECK_INT_END
    mov AX, 0

CHECK_INT_END:
    pop ES
    pop DX
    pop SI
    pop CX
    pop BX
    ret
CHECK_INT ENDP

CHECK_TAIL PROC
    ; sets AX to 1 if tail starts with '/un'
    ; otherwise sets AX to 0
    mov AX, 0

    cmp byte ptr ES:[82h], '/'
    jne CHECK_TAIL_END
    cmp byte ptr ES:[83h], 'u'
    jne CHECK_TAIL_END
    cmp byte ptr ES:[84h], 'n'
    jne CHECK_TAIL_END

    mov AX, 1

CHECK_TAIL_END:
    ret
CHECK_TAIL ENDP

LOAD_INT PROC
    push AX
    push BX
    push CX
    push DX
    push DS
    push ES

```

```

; save old int
mov AH, 35h
mov AL, 1Ch
int 21h
mov KEEP_IP, BX
mov KEEP_CS, ES

; set new int
push DS
mov DX, offset MY_INT
mov AX, seg MY_INT
mov DS, AX
mov AH, 25h
mov AL, 1Ch
int 21h
pop DS

; make resident
mov DX, offset MY_INT_END
shr DX, 1
shr DX, 1
shr DX, 1
shr DX, 1
add DX, 11h
inc DX
mov AX, 0
mov AH, 31h
int 21h

pop ES
pop DS
pop DX
pop CX
pop BX
pop AX

ret
LOAD_INT ENDP

UNLOAD_INT PROC

```

```

push AX
push BX
push CX
push DX
push SI
push ES
push DS

cli

; get int's segment
mov AH, 35h
mov AL, 1Ch
int 21h

; get int's data offset
mov SI, offset KEEP_CS
sub SI, offset MY_INT

mov AX, ES:[BX+SI]          ; cs
mov DX, ES:[BX+SI+2]      ; ip
push DS
mov DS, AX
mov AH, 25h
mov AL, 1Ch
int 21h
pop DS

; free memory
mov AX, ES:[BX+SI+4]      ; saved PSP
mov ES, AX
push ES
mov AX, ES:[2Ch]          ; env variables seg
mov ES, AX
mov AH, 49h
int 21h                    ; free env variables seg
pop ES
mov AH, 49H
int 21h                    ; free resident program

sti

```

```

    pop DS
    pop ES
    pop SI
    pop DX
    pop CX
    pop BX
    pop AX

    ret
UNLOAD_INT ENDP

MAIN PROC
    PUSH DS
    SUB AX, AX
    PUSH AX
    MOV AX, DATA
    MOV DS, AX

    mov KEEP_PSP, ES ; save PSP to free it later

    call CHECK_TAIL
    mov BX, AX          ; BX=tail.startswith("/un")
    call CHECK_INT      ; AX=1 if int is loaded

    cmp BX, 1
    jne LOAD
    UNLOAD:
        cmp AX, 1
        jne NOT_LOADED
        call UNLOAD_INT
        mov DX, offset STR_RESTORE
        mov AH, 09h
        int 21h
        jmp CHECK_END
    LOAD:
        cmp AX, 1
        je LOADED
        call LOAD_INT
        mov DX, offset STR_LOAD
        mov AH, 09h

```

```

        int 21h
        jmp CHECK_END
LOADED:
        mov DX, offset STR_EXISTS
        mov AH, 09h
        int 21h
        jmp CHECK_END
NOT_LOADED:
        mov DX, offset STR_NOT_EXISTS
        mov AH, 09h
        int 21h
CHECK_END:


MAIN_END:
xor AL, AL
mov AH, 4Ch
int 21h
MAIN ENDP


CODE ENDS


DATA SEGMENT
    STR_EXISTS      db "Interruptio already loaded",10,13,"$"
    STR_NOT_EXISTS  db "Interruptio isn't loaded",10,13,"$"
    STR_LOAD        db "Interruptio successfully loaded",10,13,"$"
    STR_RESTORE     db "Restored interruptio",10,13,"$"
DATA ENDS


AStack SEGMENT STACK
    DW 200 DUP(?)
AStack ENDS


END MAIN

```