

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

ОТЧЕТ
по лабораторной работе №5
по дисциплине «Операционные системы»
Тема: Сопряжение стандартного и пользовательского обработчиков
прерываний

Студент гр. 8382

Колногоров Д.Г.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

Выполнение работы.

Был написан программный модуль типа **.EXE** (представлен в приложении А), который выполняет следующие функции:

1) Проверяет, установлено ли пользовательское прерывание с вектором 09h.

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:

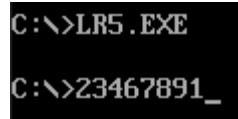
04
owner: 0192
size:    144 bytes
last bytes:

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

1

Рисунок 1 — состояние памяти до загрузки обработчика прерывания

На рисунке 2 представлен результат загрузки обработчика прерывания с последующим введением на клавиатуре строки «123456789».



```
C:\>LR5.EXE
C:\>123456789_
```

Рисунок 2 — результат загрузки обработчика прерывания

На рисунке 3 представлено состояние памяти после загрузки обработчика прерывания.

```

available memory:648368 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:    368 bytes
last bytes: LR5

06
owner: 01B4
size:    144 bytes
last bytes:

07
owner: 01B4
size: 648368 bytes
last bytes: LR3 1

```

1

Рисунок 3 — состояние памяти после загрузки обработчика прерывания

На рисунках 4 и 5 представлено тестирование программы на корректность.

```

C:\>LR5.EXE

C:\>234567891
Illegal command: 234567891.

C:\>LR5.EXE
Interruption already loaded

C:\>

```

Рисунок 4 — попытка повторной загрузки обработчика прерывания

```

C:\>LR5.EXE

C:\>234567891
Illegal command: 234567891.

C:\>LR5.EXE
Interruption already loaded

C:\>LR5.EXE /un
Restored interruption

C:\>123456789_

```

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

```

1

Рисунок 6 — состояние памяти после выгрузки обработчика прерывания

Контрольные вопросы.

1) Какого типа прерывания использовались в работе?

Реализуемое в работе прерывание от клавиатуры — аппаратное. При выполнении работы также использовалось программное прерывание 21h.

2) Чем отличается скан код от кода ASCII?

Скан код — это код, присвоенный каждой клавише, с помощью которого драйвер клавиатуры распознает, какая клавиша была нажата. ASCII-код — код символа в таблице ASCII.

Вывод.

В ходе выполнения лабораторной работы была исследована возможность встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

ПРИЛОЖЕНИЕ А

СОДЕРЖИМОЕ ФАЙЛА LR5.ASM

CODE SEGMENT

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

MY_INT PROC FAR

 jmp MY_INT_START

MY_INT_DATA:

 KEEP_IP dw 0

 KEEP_CS dw 0

 KEEP_PSP dw 0

 SIGNATURE dw 1234h

 INT_STACK dw 64 dup(?)

 KEEP_SS dw 0

 KEEP_SP dw 0

 KEEP_AX dw 0

MY_INT_START:

 mov KEEP_SS, SS

 mov KEEP_SP, SP

 mov KEEP_AX, AX

 mov AX, SEG INT_STACK

 mov SS, AX

 mov AX, offset INT_STACK

 add AX, 128 ; add stack size

 mov SP, AX

 push BX

 push CX

 push DX

 push SI

 push DS

 push ES

 ; set DS to int's data segment

 mov AX, SEG KEEP_IP

 mov DS, AX

 ; check if key matches

```

in AL, 60h
cmp AL, 02h
jl KEY_DID_NOT_MATCH
cmp AL, 0Ah
jg KEY_DID_NOT_MATCH
jmp KEY_MATCHED

KEY_DID_NOT_MATCH:
    ; call original vec and exit int
    pushf
    call DWORD PTR CS:KEEP_IP
    jmp MY_INT_END

```

```

KEY_MATCHED:
    ; increase digit
    sub AL, 01h
    mov CL, '1'
    add CL, AL
    cmp CL, '9'
    jle NO_OVERFLOW
    mov CL, '1'
NO_OVERFLOW:

    ; some required stuff
    in AL, 61h
    mov AH, AL
    or AL, 80h
    out 61h, AL
    xchg AH, AL
    out 61h, AL
    mov AL, 20h
    out 20h, AL

    ; write char into keyboard buffer
    mov AH, 05h
    mov CH, 00h
    int 16h

```

```

MY_INT_END:
    pop ES
    pop DS

```



```

    pop SI
    pop DX
    pop CX
    pop BX

    mov AX, KEEP_SS
    mov SS, AX
    mov SP, KEEP_SP
    mov AX, KEEP_AX

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

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, 09h
    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, 09h
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, 09h
    int 21h
    pop DS

    ; make resident
    mov DX, offset MY_INT_LAST_BYTE
    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, 09h
int 21h

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

mov DX, ES:[BX+SI]          ; ip
mov AX, ES:[BX+SI+2]       ; cs
push DS
mov DS, AX
mov AH, 25h
mov AL, 09h
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

```