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»

Студентка гр. 6383 _____ Михеева Е. Е.

Преподаватель _____ Губкин А. Ф.

Санкт-Петербург
2018

Цель работы.

Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Пользовательский обработчик прерывания получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре. Он обрабатывает скан-код и осуществляет определенные действия, если скан-код совпадает с определенными кодами, которые он должен обрабатывать. Если скан-код не совпадает с этими кодами, то управление передается стандартному прерыванию.

Последовательность действий, выполняемых программой:

1) Проверяется хвост командной строки.
2) Если он равен «/un», тогда проверяется (по сигнатуре «HANDLE») был ли загружен наш обработчик, если он был загружен, то утилита возвращает предыдущий обработчик, освобождает память и завершается; иначе программа завершается с соответствующим сообщением об ошибке.
3) Иначе проверяется по сигнатуре был ли загружен наш обработчик и, если наш обработчик загружен не был, утилита запоминает адрес предыдущего обработчика, устанавливает свой обработчик и остается в памяти по завершению. В противном случае следует сообщение об ошибке.

Действия, выполняемые обработчиком прерывания 09h:

1) Считать скан-код нажатой клавиши, если это не скан-код клавиши „[“ - вернуть управление стандартному обработчику.
2) Иначе, прочитать флаги состояния, если не выставлен флаг, соответствующий правому SHIFT, то вывести в буфер клавиатуры символ „[“
3) Иначе, вывести в буфер клавиатуры символ „D“.

Сведения о функциях и структурах данных управляющей программы.

Название процедуры	Описание
PRINT	Выводит на экран строку
ROUT	Пользовательский обработчик

	прерывания
CHECK_INT	Проверяет, установлено ли пользовательское прерывание
SET_INT	Устанавливает пользовательское прерывание в поле векторов прерываний
outputAL	Вывод символа с кодом AL в текущее положение курсора
DELETE_INT	Выход из пользовательского прерывания
EXIT_FROM_INTER	Проверка наличия /un

Таблица 1. Описание функций.

Название переменной	**Тип**	**Описание**
INT_IS_LOADED	db	Строка-сообщение для вывода информации о том, что пользовательское прерывание уже установлено
UNLOADED	db	Строка-сообщение для вывода информации о том, что пользовательское прерывание выгружено
INT_LOAD	db	Строка-сообщение для вывода информации о том, что пользовательское прерывание установлено
STRN	db	Строка, которая используется для определения: было ли установлено пользовательское прерывание или нет
KEEP_CS	dw	Переменная для сохранения значения в регистре CS
KEEP_IP	dw	Переменная для сохранения значения в регистре IP
KEEP_PSP	dw	Переменная для сохранения сегментного адреса PSP

Таблица 2. Описание переменных.

Ход работы.

Шаг 1.

Рис. 1. Результат работы программы 5.exe. SHIFT+ «[» заменяется на «D»

Шаг 2.

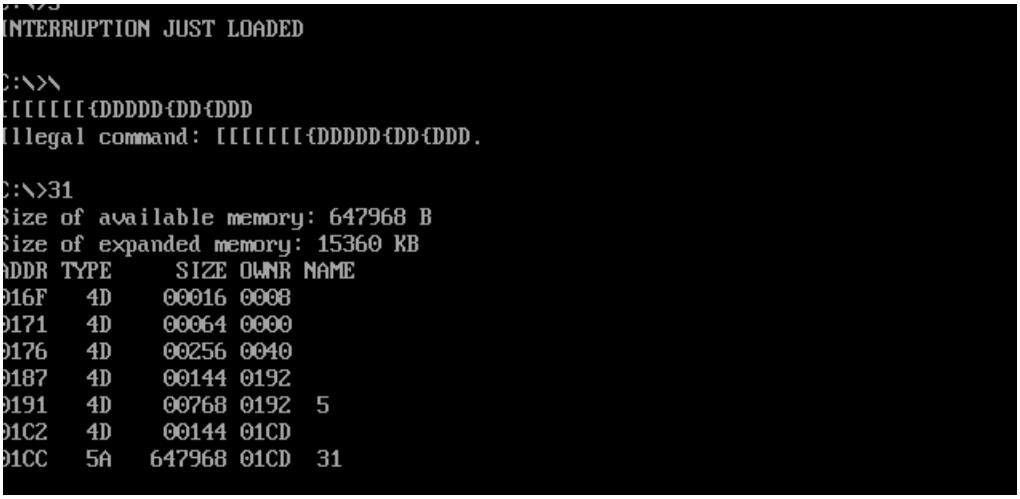

Рис. 2. Проверка загрузки прерывания.

Шаг 3.

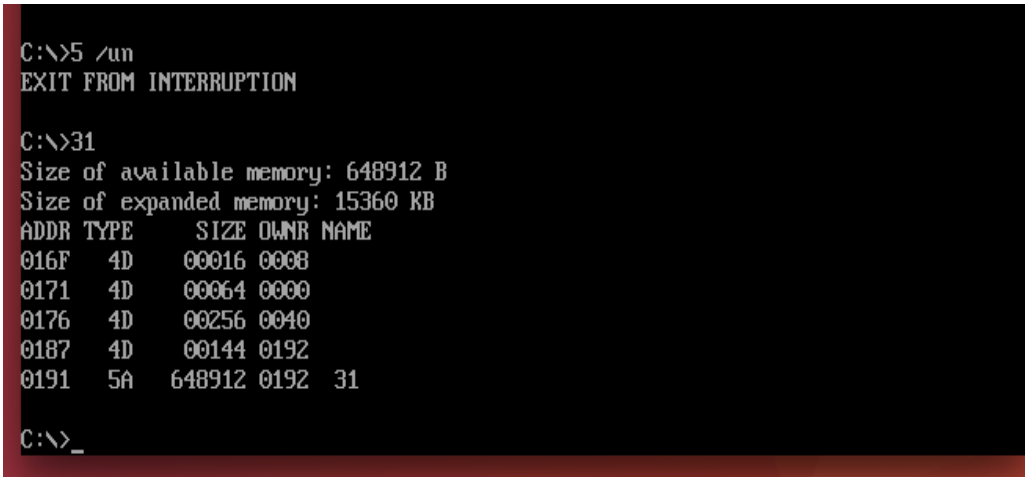

Рис. 3. Проверка выгрузки резидентного обработчика прерывания.

Ответы на контрольные вопросы.

1. **Какого типа прерывания использовались в работе?**

Ответ: В данной лабораторной работе использовались: аппаратные прерывания (int 09h) и программные прерывания (int 21h, int 10h).

2. **Чем отличается скан-код от кода ASCII?**

Ответ.

ASCII коды — коды символов в потоке.

Скан-код — код клавиши, который посылает контроллер клавиатуры при нажатии этой клавиши. Скан-коды привязаны к каждой клавише на аппаратном уровне и не зависят от состояния индикаторов Caps Lock, Num Lock и Scroll Lock, а также управляющих клавиш.

Вывод.

В ходе лабораторной работы было исследованы особенности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

Приложение А. Код программы

```
.286

ASTACK SEGMENT stack
db 100h dup(?)
ASTACK ENDS

DATA SEGMENT
    INT_LOAD DB 'INTERRUPTION JUST LOADED',0AH,0DH,'$'
    INT_IS_LOADED DB 'INTERRUPTION IS ALREADY
LOADED !!!',0AH,0DH,'$'
    EXIT_FROM_INTERR DB 'EXIT FROM
INTERRUPTION',0AH,0DH,'$'
    INT_NOT_LOADED DB 'INTERRUPTION NOT
LOADED!!!',0AH,0DH,'$'

DATA ENDS

CODE SEGMENT
     ASSUME CS:CODE, DS:CODE, ES:CODE, SS:ASTACK

PRINT PROC NEAR
    pusha
    push ds
    mov ax, DATA
    mov ds, ax
```

```
    mov ah, 09h
    int 21h
    pop ds
    popa

    ret
PRINT ENDP

outputAL PROC NEAR

    mov ah, 09h
    mov cx, 1
    mov bh, 0
    int 10h

    ret
outputAL ENDP

ROUT PROC FAR
    jmp entry
    STRN DB 'ABC'
    entry:
    pusha
    push es
    push ds

    ;getCurs

    mov ah,03h
    mov bh, 0
    int 10h
```

```
    push dx
    push cx

    call outputAL

    ; setCurs
    pop cx
    pop dx
    mov ah, 02h
    int 10h

    inc cs:COUNTER

    ; end of code
    pop ds
    pop es
    popa
    mov al, 20h
    out 20h, al
    iret
    KEEP_CS DW 0
    KEEP_IP DW 0
    KEEP_PSP DW 0h
    COUNTER DW 0h
    last_byte:
ROUT ENDP


SET_INT  PROC NEAR
    pusha
    push ds
```

```
    push es

    mov ah, 35h
    mov al, 1Ch
    int 21h

    mov cs:KEEP_IP, bx
    mov cs:KEEP_CS, es

    mov ax, SEG ROUT
    mov ds, ax
    mov dx, offset ROUT
    mov ah, 25h
    mov al, 1Ch
    int 21h
    ;end
    mov dx, offset last_byte
    shr dx, 4
    inc dx
    add dx, CODE
    sub dx, cs:KEEP_PSP
    mov ah, 31h
    int 21h

    pop es
    pop ds
    popa

    ret
SET_INT ENDP
```

```
CHECK_INT PROC NEAR ; if set then al=1, else al=0
    cli
    pusha
    push ds
    push es

    mov ah, 35h
    mov al, 1Ch
    int 21h

    mov dx, es:KEEP_IP
    mov ax, es:KEEP_CS

    cmp word ptr es:STRN+0, 'BA'
    jne no
    cmp byte ptr es:STRN+2, 'C'
    jne no

    yes:
    pop es
    pop ds
    popa
    mov al, 1
    sti
    jmp end_ch
    no:
    pop es
    pop ds
    popa
    mov al, 0
    sti
```

```
    end_ch:

    ret
CHECK_INT ENDP

DELETE_INT PROC NEAR
    cli
    pusha
    push ds
    push es
    mov ah, 35h
    mov al, 1Ch
    int 21h

    push es
    mov dx, es:KEEP_IP
    mov ax, es:KEEP_CS
    mov ds, ax
    mov ah, 25h
    mov al, 1Ch
    int 21h

    pop es
    mov es, es:KEEP_PSP
    push es
    mov es, es:[2Ch]
    mov ah, 49h
    int 21h

    pop es
    mov ah, 49h
```

```
    int 21h
    pop es
    pop ds
    popa
    sti

    ret
DELETE_INT ENDP

EXIT_FROM_INTER PROC NEAR
  pusha
  cmp byte ptr ES:[82H], '/'
  jne exit_from_int
  cmp byte ptr es:[83H], 'u'
  jne exit_from_int
  cmp byte ptr es:[84H], 'n'
  jne exit_from_int

  popa
  mov al, 1
  jmp end_ex
  exit_from_int:
  popa
  mov al, 0
  end_ex:

  ret
EXIT_FROM_INTER ENDP


BEGIN:
```

```
push DS
sub AX,AX
push AX
mov cs:KEEP_PSP, ds

;checking /un to exit or not

call EXIT_FROM_INTER
cmp al, 1
je end_int
call CHECK_INT
cmp ax, 0
jne loading
mov dx, offset INT_LOAD
call PRINT
call SET_INT
jmp exit

;already loaded
loading:
mov dx, offset INT_IS_LOADED
 call PRINT
jmp exit
call CHECK_INT
cmp al, 1
jne already_loaded
;exit from int
 end_int:
 call DELETE_INT
 mov dx, offset EXIT_FROM_INTERR
 call PRINT
```

```
    jmp exit
    already_loaded:
     mov dx, offset INT_LOAD
     call PRINT

    exit:
    xor AL,AL
    mov AH,4Ch
    int 21H
CODE ENDS
END BEGIN
```