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. 7383 _____ Иолшина В.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2019

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 определённые действия, если скан-код совпадает с определёнными кодами, которые он должен обрабатывать. Если скан- код не совпадает с этими кодами, то управление передаётся стандартному прерыванию.

Таблица 1 - Описание процедур.

Название процедуры	Назначение
PRINT	вызывает функцию печати строки
ROUT	пользовательский обработчик прерываний, считающий и печатающий количество его вызовов
CHECKING	проверяет загруженность обработчика прерываний
SET_INTERRUPT	устанавливает новый обработчик прерывания, запоминая данные для восстановления предыдущего обработчика прерываний
DELETE_INTERRUPT	удаляет пользовательское прерывание, восстанавливает прерывание по умолчанию

Последовательность действий, выполняемых программой.

1. Проверка, установлено ли пользовательское прерывание с вектором 09h.
2. Установка обработчика прерываний, если он не установлен, и выход.
3. Если прерывание установлено, то вывод соответствующего сообщения и выход.

4. Выгрузка прерывания по соответствующему значению параметра в командной строке /un, выход в DOS.

Результат выполнения программы представлен на рис. 1-6.

1. Состояние памяти до запуска программы lr5.exe представлено на рис.1, в качестве программы изображена работа файла lr3_1.com.

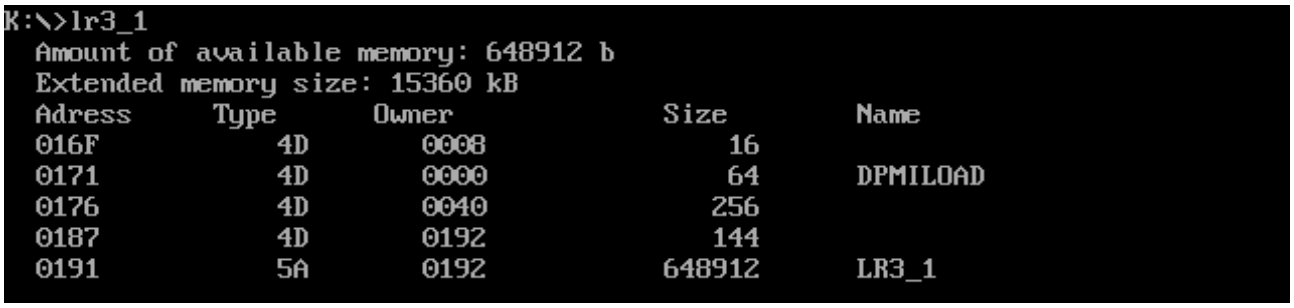

Рисунок 1 – Результат выполнения программы lr3_1.com

2. На рис. 2 представлен запуск программы lr5.exe

Рисунок 2 – Результат выполнения программы 5.exe

3. Проверим работу прерывания нажатием Ctrl – C и другие клавиши. Действие программы показано на рис. 3.

Рисунок 3 – Результат выполнения программы lr5.exe

4. Проверим размещение прерывания в памяти в виде блоков MCB. Выполнение программы lr3_1.com показано на рис. 4.

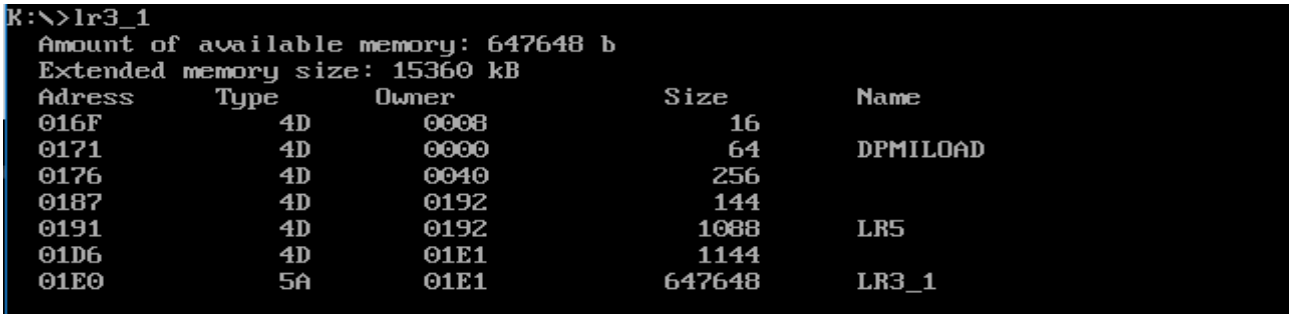

Рисунок 4 – Результат выполнения программы lr3_1.com

5. Запустим программу lr5.exe ещё раз, но с ключом выгрузки /un (см. рис.5):

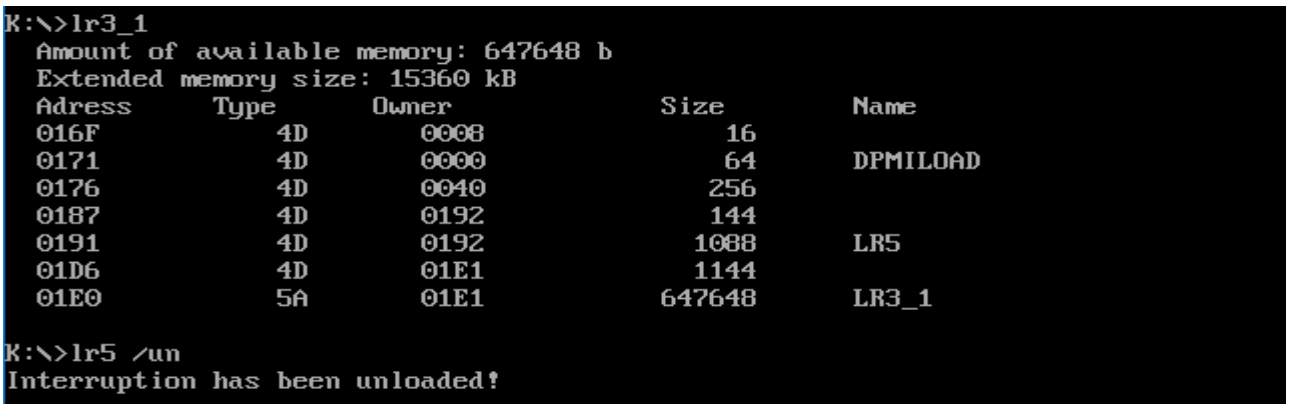

Рисунок 5 – Результат выполнения программы lr5.exe с ключом /un

6. Запустим программу lr3_1.com, чтобы убедиться в том, что память освобождена.

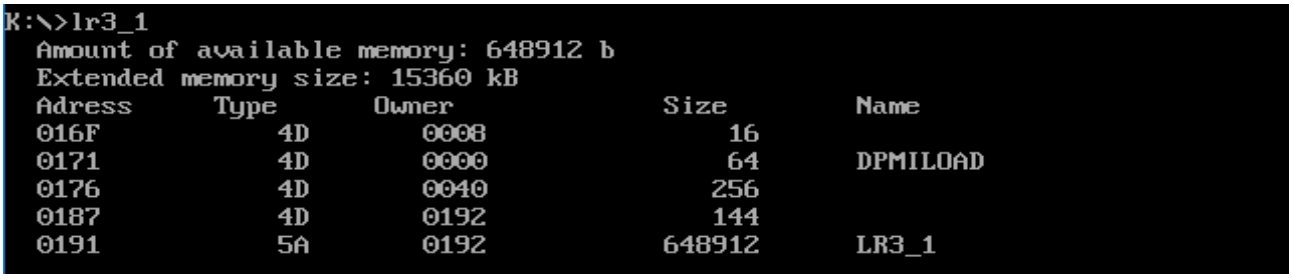

Рисунок 6 – Результат повторного выполнения программы lr3_1.com

Выводы.

В процессе выполнения данной лабораторной работы была исследована возможность встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Также был создан программный модуль, устанавливающий пользовательский обработчик прерываний от клавиатуры, который проверяет, установлено ли пользовательское прерывание с вектором 09h, устанавливает резидентную функцию для обработки прерывания, выгружает пользовательское прерывание по соответствующему значению параметра командной строки «/un», а также при нажатии определённой клавиши выводит заданный символ.

Ответы на контрольные вопросы.

1. Какого типа прерывания использовались в работе?

В данной лабораторной работе использовались аппаратные прерывание (int 09h - пользовательское прерывание выполняется при каждом нажатии и отпускании клавиши) и программные прерывания (int 21h и int 16h - интерфейс прикладного уровня с клавиатурой).

2. Чем отличается скан-код от кода ASCII?

Код ASCII является уникальным числовым представлением какого-либо символа. Скан-код – код, присвоенный каждой клавише, с помощью которого драйвер клавиатуры распознаёт, какая клавиша была нажата. ASCII символы не связаны напрямую с клавиатурой, обработчик прерываний от клавиатуры может по-разному обрабатывать скан-коды, в том числе и записывать ASCII-код в буфер клавиатуры.

ПРИЛОЖЕНИЕ А

5.asm

```
        STACK SEGMENT STACK
        DW 100 DUP (?)
STACK ENDS


DATA SEGMENT
        wasloaded DB 'Interruption had been set earlier!',0DH,0AH,'$'
        unloaded DB 'Interruption has been unloaded!',0DH,0AH,'$'
        loading DB 'Interruption has been set!',0DH,0AH,'$'
DATA ENDS


CODE SEGMENT
        ASSUME CS:CODE, DS:DATA, ES:DATA, SS:STACK
START: JMP BEGIN


PRINT PROC NEAR ; обработчик прерывания
        push ax
        mov ah, 09h
        int 21h
        pop ax
        ret
PRINT ENDP


ROUT PROC FAR
        jmp ROUT_
_DATA:
        STACK_ DW 64 DUP (?)
        SIGN DB '0000'
        KEEP_IP DW 0
        KEEP_CS DW 0
        KEEP_PSP DW 0
        KEEP_SS DW 0
        KEEP_AX DW 0
        KEEP_SP DW 0
        _TILD DB 29h

ROUT_:
        mov KEEP_SS, ss
        mov KEEP_AX, ax
        mov KEEP_SP, sp
```

```
        mov ax, seg STACK_
        mov ss, ax
        mov sp, 0
        mov ax, KEEP_AX

        mov ax, 0040h
        mov es, ax
        mov al, es:[17h]
        cmp al, 00000010b
        jnz NEXT
        in al, 60H

        cmp al, _TILD
        je TILD

NEXT:
        pop ES
        pop DS
        pop DX
        mov ax, CS:KEEP_AX
        mov sp, CS:KEEP_SP
        mov ss, CS:KEEP_SS
        jmp dword ptr cs:[KEEP_IP]

TILD:
        mov cl, '_'
        jmp DO_REQ

DO_REQ:
        push ax
        in al, 61h
        mov ah, al
        or al, 80h
        out 61h, al
        xchg ah, al
        out 61h, al
        mov al, 20h
        out 20h, al
        pop ax

ADDSYMB:
        mov ah, 05h
        mov ch, 00h
        int 16h
        or al, al
        jz ROUT_END
```

```
        CLI
        mov ax,es:[1Ah]
        mov es:[1Ch],ax
        STI
        jmp ADDSYMB

ROUT_END:
        pop es
        pop ds
        pop dx
        pop ax
        mov AX, KEEP_SS
        mov SS, AX
        mov SP,KEEP_SP
        mov AX,KEEP_AX
        iret
ROUT ENDP


CHECKING PROC ; проверка прерывания
        mov ah,35h
        mov al,09h
        int 21h
        mov si, offset SIGN
        sub si, offset ROUT

        mov ax,'00'
        cmp ax,es:[bx+si]
        jne UNLOAD
        cmp ax,es:[bx+si+2]
        je LOAD

UNLOAD:
        call SET_INTERRUPT
        mov dx,offset LAST_BYTE
        mov cl,4
        shr dx,cl
        inc dx
        add dx,CODE
        sub dx,CS:KEEP_PSP
        xor al,al
        mov ah,31h
        int 21h

LOAD:
        push es
```

```
        push ax
        mov ax,KEEP_PSP
        mov es,ax
        cmp byte ptr es:[82h],'/'
        jne BACK
        cmp byte ptr es:[83h],'u'
        jne BACK
        cmp byte ptr es:[84h],'n'
        je UNLOAD_

BACK:
        pop ax
        pop es
        mov dx,offset wasloaded
        call PRINT
        ret

UNLOAD_:
        pop ax
        pop es
        call DELETE_INTERRUPT
        mov dx,offset unloaded
        call PRINT
        ret
CHECKING endp


SET_INTERRUPT PROC ; добавление нового прерывания
        push dx
        push ds

        mov ah,35h
        mov al,09h
        int 21h
        mov CS:KEEP_IP,bx
        mov CS:KEEP_CS,es

        mov dx,offset ROUT
        mov ax,seg ROUT
        mov ds,ax
        mov ah,25h
        mov al,09h
        int 21h

        pop ds
        mov dx,offset loading
```

```
        call PRINT
        pop dx
        ret
SET_INTERRUPT ENDP


DELETE_INTERRUPT PROC ;удаление прерывания
        push ds
        CLI
        mov dx,ES:[BX+SI+4]
        mov ax,ES:[BX+SI+6]
        mov ds,ax
        mov ax,2509h
        int 21h
        push es
        mov ax,ES:[BX+SI+8]
        mov es,ax
        mov es,es:[2Ch]
        mov ah,49h
        int 21h
        pop es
        mov es,ES:[BX+SI+8]
        mov ah, 49h
        int 21h
        STI
        pop ds
        ret
DELETE_INTERRUPT ENDP


BEGIN:
        mov AX,DATA
        mov DS,AX
        mov KEEP_PSP,ES
        call CHECKING
        xor AL,AL
        mov AH,4Ch
        int 21H
LAST_BYTE:
        CODE ENDS
        END START
```