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. 8382 _____ Наконечная А. Ю.

Преподаватель _____ Ефремов М. А.

Санкт-Петербург
2020

Цель работы.

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

Постановка задачи.

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет такие же функции, как в программе ЛР 4, а именно:

Программа должна содержать код устанавливаемого прерывания в виде удаленной процедуры. Этот код будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

- Сохранить значения регистров в стеке при входе и восстановить их при выходе.
- При выполнении тела процедуры анализируется скан-код.
- Если этот код совпадает с одним из заданных, то требуемый код записывается в буфер клавиатуры.
- Если этот код не совпадает ни с одним из заданных, то осуществляется передача управления стандартному обработчику прерывания.

Шаг 2. Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания 09h установлен. Работа прерывания проверяется введением различных символов, обрабатываемых установленным обработчиком и стандартным обработчиком.

Шаг 3. Также необходимо проверить размещение прерывания в памяти. Для этого запустите программу ЛР 3, которая отображает карту памяти в виде с писка блоков МСВ. Полученные результаты поместите в отчет.

Шаг 4. Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

Шаг 5. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Выполнение работы.

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 передаётся стандартному прерыванию.

В результате выполнения была получена следующая информация (рисунок 1 - 3):

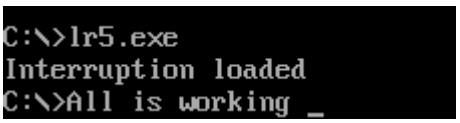

Рисунок 1 - Прерывание загружено в память и пример вывода для нескольких нажатий левого shift

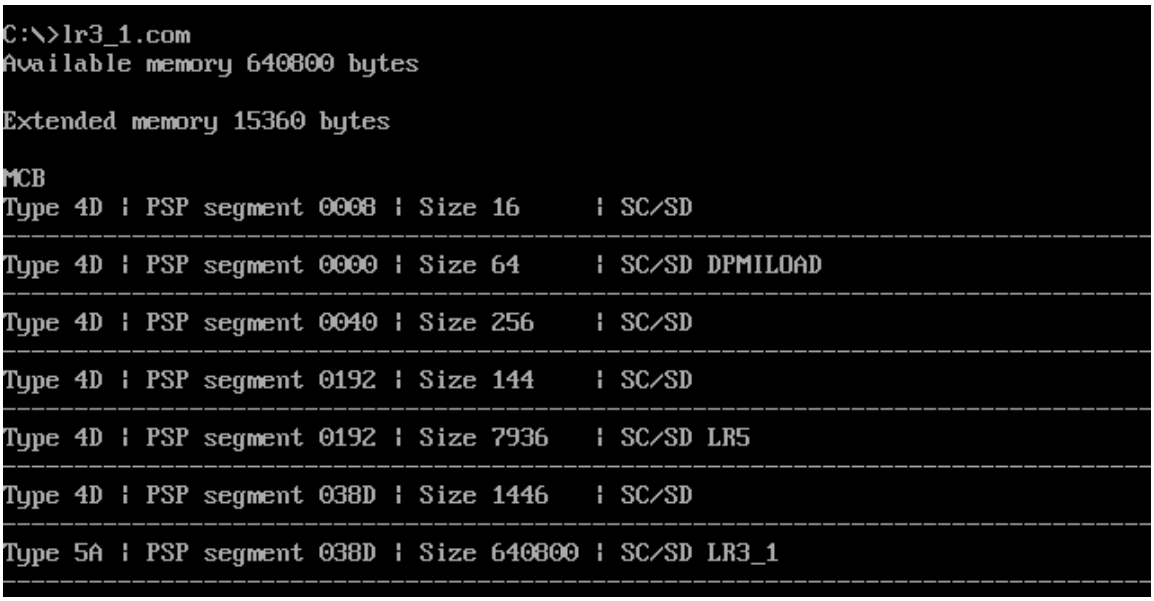

Рисунок 2 — Расположение в памяти

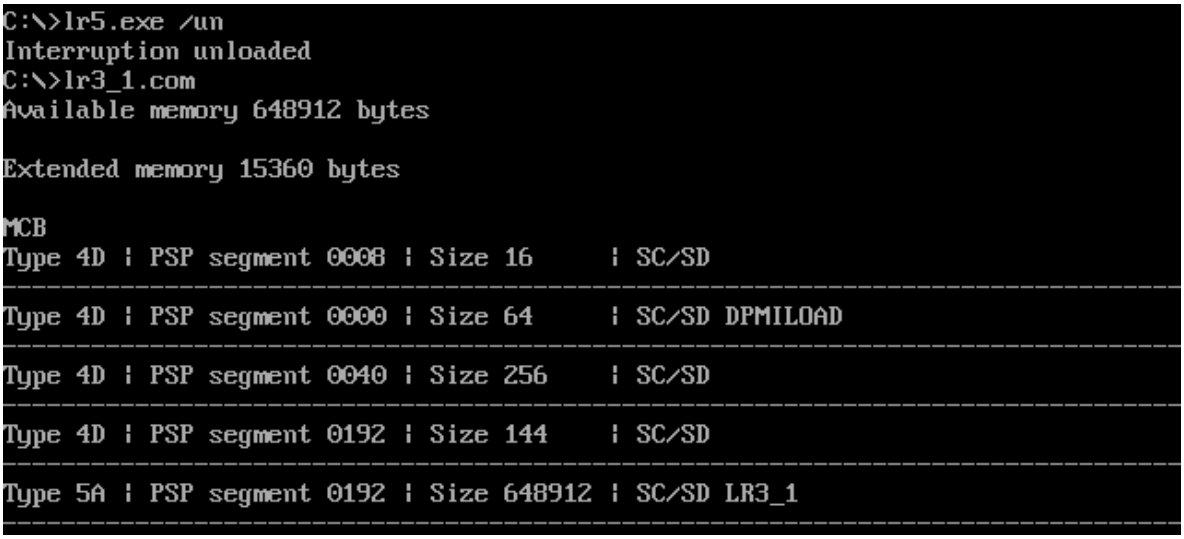

Рисунок 3 — Выгрузка прерывания

Ответы на контрольные вопросы.

1) Какого типа прерывания использовались в работе?

В данной работе использовались пользовательские прерывания – int 21h, прерывания BIOS (int 16h), аппаратные прерывания (09h).

2) Чем отличается скан-код от кода ASCII?

Скан-код – код, присвоенный каждой клавише, с помощью которого драйвер клавиатуры распознает, какая клавиша была нажата.

Код ASCII – это уникальный код для каждого символа.

Выводы.

В ходе лабораторной работы была исследованы возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

ПРИЛОЖЕНИЕ А

Исходный код программы

lr5.asm

```
AStack SEGMENT STACK
    dw 100h dup(?)
AStack ENDS

DATA SEGMENT
     LOADED_STR db 'Interruption loaded $'
     NOT_LOADED_STR db 'Interruption unloaded $'
DATA ENDS

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:AStack
     LOADED db ?
     ID dw 0FFFh
     KEEP_AX dw ?
     KEEP_SS dw ?
     KEEP_SP dw ?
     KEEP_IP dw ?
     KEEP_CS dw ?
     PSP dw ?
ROUT PROC FAR
     jmp START
     INTER_STR db 'All is working $'
     INDEX db 0
     STACK_INTER dw 100h dup(?)
     END_STACK_INTER db ?
START:
     mov KEEP_SS, SS
     mov KEEP_SP, SP
     mov KEEP_AX, AX
     mov AX, CS
     mov SS, AX
     mov SP, offset END_STACK_INTER
     push BX
     push CX
     push DX
     push SI
     push DS
     push BP
     push ES
    in AL, 60h
    cmp AL, 2Ah ;левый shift
           je DO_REQ
     call dword ptr cs:KEEP_IP;
    jmp ENDING;
```

```
DO_REQ:
    in AL, 61h
     mov AH, AL
     or AL, 80h
     out 61h, AL
     xchg AH, AL
     out 61h, AL
     mov AL, 20h
     out 20h, AL
    xor BX, BX
     mov BL, INDEX
WRITE:
     mov AH, 05h
     mov CL, INTER_STR[BX]
     cmp CL, '$'
          je END_STR
     mov CH, 00h
     int 16h
     or AL, AL
          jnz SKIP
     inc BL
     mov INDEX, BL
     jmp ENDING
SKIP:
     mov AX, 0C00h
     int 21h
     jmp WRITE
END_STR:
     mov INDEX, 0
ENDING:
     pop ES
     pop BP
     pop DS
     pop SI
     pop DX
     pop CX
     pop BX
     mov SP, KEEP_SP
     mov AX, KEEP_SS
     mov SS, AX
     mov AX, KEEP_AX
     mov AL, 20h
     out 20h, AL
     iret
ROUT ENDP
LAST_BYTE:

LOAD_INTER PROC near
     push AX
```

```
      push CX
      push DX
      mov AH, 35h
      mov AL, 1Ch
      int 21h
      mov KEEP_IP, BX
      mov KEEP_CS, ES
      push DS
      mov DX, offset ROUT
      mov AX, seg ROUT
      mov DS, AX
      mov AH, 25h
      mov AL, 1ch
      int 21h
      pop DS
      mov DX, offset LAST_BYTE
      mov CL, 4h
      shr DX, CL
      add DX, CODE
      inc DX
      xor AX, AX
      mov AH, 31h
      int 21h
      pop DX
      pop CX
      pop AX
      ret
LOAD_INTER ENDP

UNLOAD_INTER PROC near
      push AX
      push BX
      push DX
      push DI
      cli
      mov AH, 35h
      mov AL, 1Ch
      int 21h
      mov DI, offset KEEP_IP
      sub DI, offset ROUT
      mov DX, ES:[BX+DI]
      mov AX, ES:[BX+DI+2]
      push DS
      mov DS, AX
      mov AH, 25h
      mov AL, 1Ch
      int 21h
      pop DS
      mov AX, ES:[BX+DI+4]
```

```
      mov ES, AX
      push ES
      mov AX, ES:[2Ch]
      mov ES, AX
      mov AH, 49h
      int 21h
      pop ES
      mov AH, 49h
      int 21h
      sti
      pop DI
      pop DX
      pop BX
      pop AX
      ret
UNLOAD_INTER ENDP

CHECK_UNLOAD PROC near
      mov AX, PSP
      mov ES, AX
      cmp byte ptr ES:[81h+1], '/'
            jne NOT_UN
      cmp byte ptr ES:[81h+2], 'u'
            jne NOT_UN
      cmp byte ptr ES:[81h+3], 'n'
            jne NOT_UN
      mov DX, offset NOT_LOADED_STR
      mov AH, 09h
     int 21h
      call UNLOAD_INTER
      jmp STOP
NOT_UN:
      mov AL, LOADED
      cmp AL, 1
            je STOP
      mov DX, offset LOADED_STR
      mov AH, 09h
     int 21h
      call LOAD_INTER
STOP:
      ret
CHECK_UNLOAD ENDP

CHECK_LOAD PROC near
      push ES
      mov AH, 35h
      mov AL, 1Ch
      int 21h
      mov DX, ES:[BX-2]
```

```
      pop ES
      cmp DX, ID
            je GO
      jmp EXIT
      GO:
            mov LOADED, 1
      EXIT:
            ret
CHECK_LOAD ENDP

MAIN PROC far
      mov AX, DATA
      mov DS, AX
      mov AX, ES
      mov PSP, AX
      call CHECK_LOAD
      call CHECK_UNLOAD

      ; Выход в DOS
      xor AL,AL
      mov AH,4Ch
      int 21H
MAIN ENDP
CODE ENDS
END MAIN
```