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. 9383 _____ Лысова А.М.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

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.

Задание.

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:

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

2) Если прерывание не установлено то, устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний. Адрес точки входа в стандартный обработчик прерывания находится в теле пользовательского обработчика. 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.

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.

Для того чтобы проверить установку прерывания, можно поступить следующим образом. Прочитать адрес, записанный в векторе прерывания. Предположим, что этот адрес указывает на точку входа в установленный резидент. На определенном, известном смещении в теле резидента

располагается сигнатура, некоторый код, который идентифицирует резидент. Сравнив известное значение сигнатуры с реальным кодом, находящимся в резиденте, можно определить, установлен ли резидент. Если значения совпадают, то резидент установлен. Длину кода сигнатуры должна быть достаточной, чтобы сделать случайное совпадение маловероятным.

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

1) Сохранить значения регистров в стеке при входе и восстановить их при выходе.

2) При выполнении тела процедуры анализируется скан-код.

3) Если этот код совпадает с одним из заданных, то требуемый код записывается в буфер клавиатуры.

4) 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 списка 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.

Шаг 6. Ответьте на контрольные вопросы.

Необходимые теоретические положения:

Клавиатура содержит микропроцессор, который воспринимает каждое нажатие на клавишу и посылает скан-код в порт микросхемы интерфейса с периферией. Когда сканкод поступает в порт, то вызывается аппаратное прерывание клавиатуры (int 09h). Процедура обработки этого прерывания считывает номер клавиши из порта 60h, преобразует номер клавиши в соответствующий код, выполняет установку флагов в байтах состояния, загружает номер клавиши и полученный код в буфер клавиатуры. В прерывании клавиатуры можно выделить три основных шага:

1. Прочитать скан-код и послать клавиатуре подтверждающий сигнал.

2. Преобразовать скан-код в номер кода или в установку регистра статуса клавишпереключателей.

3. Поместить код клавиши в буфер клавиатуры.

Текущее содержимое буфера клавиатуры определяется указателями на начало и конец записи. Расположение в памяти необходимых данных представлено в таблице.

Адрес в памяти	**Размер в байтах**	**Содержимое**
0040:001A	2	Адрес начала буфера клавиатуры
0040:001C	2	Адрес конца буфера клавиатуры
0040:001E	32	Буфер клавиатуры
0040:0017	2	Байты состояния

Флаги в байтах состояния устанавливаются в 1, если нажата соответствующая клавиша или установлен режим.

В момент вызова прерывания скан-код будет находиться в порте 60h. Поэтому сначала надо этот код прочитать командой IN и сохранить на стеке.

Затем используется порт 61H, чтобы быстро послать сигнал подтверждения микропроцессору клавиатуры. Надо просто установить бит 7 в 1, а затем сразу изменить его назад в 0. Заметим, что бит 6 порта 61H управляет сигналом часов клавиатуры. Он всегда должен быть установлен в 1, иначе клавиатура будет выключена. Эти адреса портов применимы и к AT, хотя он и не имеет микросхемы интерфейса с периферией 8255.

Сначала скан-код анализируется на предмет того, была ли клавиша нажата (код нажатия) или отпущена (код освобождения). Код освобождения состоит из двух байтов: сначала 0F0H, а затем скан-код. Все коды освобождения отбрасываются, кроме случая клавиш-переключателей, для которых делаются соответствующие изменения в байтах их статуса. С другой стороны, все коды нажатия обрабатываются. При этом опять могут изменяться байты статуса клавиш-переключателей. В случае же символьных кодов, надо проверять байты статуса, чтобы определить, например, что скан-код 30 соответствует нижнему или верхнему регистру буквы A. После того как введенный символ идентифицирован, процедура ввода с клавиатуры должна найти соответствующий ему код ASCII или расширенный код. Приведенный пример слишком короток, чтобы рассмотреть все случаи. В общем случае скан-коды сопоставляются элементам таблицы данных, которая анализируется инструкцией XLAT. XLAT принимает в AL число от 0 до 255, а возвращает в AL 1-байтное значение из 256-байтной таблицы, на которую указывает DS:BX. Таблица может находиться в сегменте данных. Если в AL находился скан-код 30, то туда будет помещен из таблицы байт номер 30 (31-й байт, так как отсчет начинается с нуля). Этот байт в таблице должен быть установлен равным 97, давая код ASCII для "a". Конечно для получения заглавной A нужна другая таблица, к которой обращение будет происходить, если статус сдвига установлен. Или заглавные буквы могут храниться в другой части той же таблицы, но в этом случае к скан-коду надо будет добавлять смещение, определяемое статусом клавиш-переключателей.

Номера кодов должны быть помещены в буфер клавиатуры. Процедура должна сначала проверить, имеется ли в буфере место для следующего символа. Буфер устроен как циклическая очередь. Ячейка памяти 0040:001A содержит указатель на голову буфера, а 0040:001C - указатель на хвост. Эти словные указатели дают смещение в области данных BIOS (которая начинается в сегменте 40H) и находятся в диапазоне от 30 до 60. Новые символы вставляются в ячейки буфера с более старшими адресами, а когда достигнута верхняя граница, то следующий символ переносится в нижний конец буфера. Когда буфер полон, то указатель хвоста на 2 меньше указателя на голову - кроме случая, когда указатель на голову равен 30 (начало области буфера), а в этом случае буфер полон, когда указатель хвоста равен 60. Для вставки символа в буфер, надо поместить его в позицию, на которую указывает хвост буфера и затем увеличить указатель хвоста на 2; если указатель хвоста был равен 60, то надо изменить его значение на 30.

Выполнение работы:

Прерывание заменяет буквы «a», «b» и «c» на «1», «2» и «3», соответственно, а также выводит «*» на экран при нажатии Esc.

Рисунок 1: Исполнение прерывания

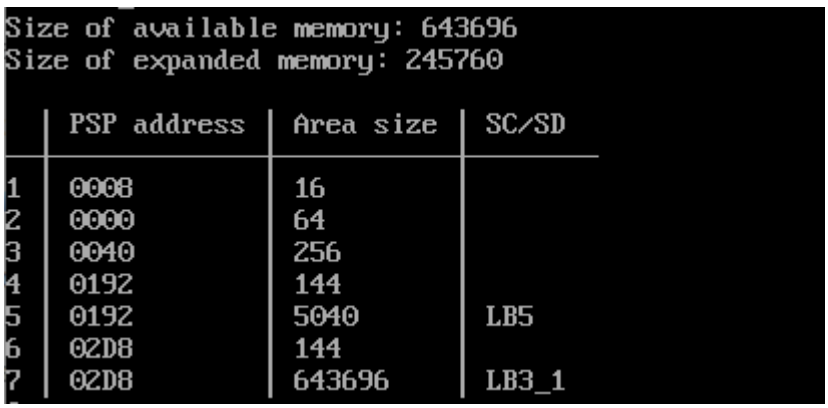

Рисунок 2: Запуск ЛР3 сразу после загрузки прерывания.

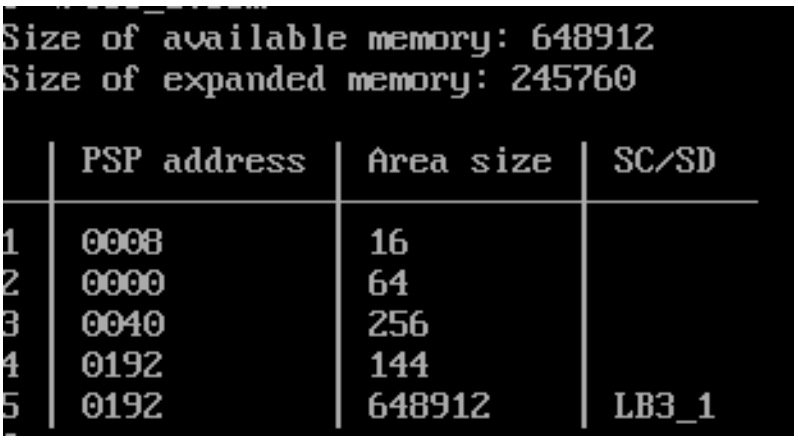

Рисунок 3: Выполнение ЛР3 после выгрузки прерывания.

Контрольные вопросы:

1) Какого типа прерывания использовались в работе?

Прерывания функций BIOS, например 16h, 09h, и прерывания функций DOS(21h).

2) Чем отличается скан код от кода ASCII?

Тем, что символы ASCII берутся из таблицы ASCII, а скан-код изначальо присвоен каждой клавише на клавиатуре, с помощью него драйвер клавиатуры может распознавать, какая из клавиш была нажата.

Выводы.

Были исследованы возможности встраивания пользовательского обработчика в стандартный обработчик от клавиатуры. Пользовательский обработчик прерывания получал управление по прерыванию(int 09h) при нажатии разных клавиш на клавиатуре(заданных). Обрабатывая скан-код, он осуществлял замену символов на заданные.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: lb5.asm

```
AStack      SEGMENT STACK
      DB 256 DUP(?)
AStack ENDS

DATA  SEGMENT

    count DB '0000'
      INT_LOAD DB 'My interrupt has been unloaded!', 13,  10, '$'
      INT_UNLOAD DB 'My interrupt has been unloaded!', 13,  10, '$'
      INT_NOT_LOAD DB 'My interrupt has not been loaded!', 13,  10, '$'
      INT_ALREADY_LOAD DB 'My interrupt has already been loaded!', 13,  10, '$'

DATA  ENDS

CODE  SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:AStack

MY_INTERRUPT      PROC far
      jmp start_interrupt

      PSP DW ?
      KEEP_IP DW 0
      KEEP_CS DW 0
      KEEP_SS DW 0
      KEEP_SP DW 0
      KEEP_AX DW 0
      KEY_CODE DB 01h
      KEY_VALUE DB 0

      INT_STACK DW 128 DUP (?)
END_INT_STACK:

start_interrupt:
      mov KEEP_AX, AX
      mov KEEP_SS, SS
      mov KEEP_SP, SP
      mov AX, CS
      mov SS, AX
      mov SP, OFFSET END_INT_STACK
      mov AX, KEEP_AX
      push AX
      push DX
      push DS
      push ES


      in AL, 60h
      cmp AL, KEY_CODE
      je key_esc
      cmp AL, 1Eh
      je key_a
      cmp AL, 30h
      je key_b
      cmp AL, 2Eh
      je key_c
```

```
    pushf
    call dword ptr CS:KEEP_IP
    jmp end_interrupt

key_esc:
        mov KEY_VALUE, '*'
        jmp print_key

key_a:
        mov KEY_VALUE, '1'
        jmp print_key

key_b:
        mov KEY_VALUE, '2'
        jmp print_key

key_c:
        mov KEY_VALUE, '3'
        jmp print_key

next_key:
      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

print_key:
      mov AH, 05h
      mov CL, KEY_VALUE
      mov ch, 00h
      int 16h
      or AL, AL
      jz end_interrupt
      mov AX, 0040h
      mov ES, AX
      mov AX, ES:[1Ah]
      mov ES:[1Ch], AX
      jmp print_key

end_interrupt:
      pop ES
      pop DS
      pop DX
      pop AX
      mov SS, KEEP_SS
      mov SP, KEEP_SP
      mov AX, KEEP_AX
      mov AL, 20h
      out 20h, AL
      iret
ending:
MY_INTERRUPT ENDP


SET_INTERRUPT PROC
      push AX
      push DX
```

```
      push DS
      mov AH, 35h
      mov AL, 09h
      int 21h
      mov KEEP_IP, BX
      mov KEEP_CS, ES
      mov DX, OFFSET MY_INTERRUPT
      mov AX, SEG MY_INTERRUPT
      mov DS, AX
      mov AH, 25h
      mov AL, 09h
      int 21h
      pop DS
      mov DX, OFFSET INT_LOAD
      call PRINT_STRING
      pop DX
      pop AX
      ret
SET_INTERRUPT ENDP

DELETE_INTERRUPT PROC
      push AX
      push DS
      CLI
      mov AH, 35h
      mov AL, 09h
      int 21h
      mov SI, OFFSET KEEP_IP
      sub SI, OFFSET MY_INTERRUPT
      mov DX, ES:[BX+SI]
      mov AX, ES:[BX+SI+2]
      mov DS, AX
      mov AH, 25h
      mov AL, 09h
      int 21h
      pop DS
      mov AX, ES:[BX+SI-2]
      mov ES, AX
      mov AX, ES:[2Ch]
      push ES
      mov ES, AX
      mov AH, 49h
      int 21h
      pop ES
      mov AH, 49h
      int 21h
      STI
      pop AX
      ret
DELETE_INTERRUPT ENDP

MY_FUNC PROC
      mov AH, 35h
      mov AL, 09h
      int 21h

      mov SI, OFFSET count
      sub SI, OFFSET MY_INTERRUPT

      mov AX, '00'
      cmp AX, ES:[BX+SI]
      jne NOT_LOADED
      cmp AX, ES:[BX+SI+2]
```

```
      jne NOT_LOADED
      jmp LOADED

NOT_LOADED:
      call SET_INTERRUPT
      mov DX, OFFSET ending
      mov CL, 4
      shr DX, CL
      inc DX
      add DX, CODE
      sub DX, PSP
      xor AL, AL
      mov AH, 31h
      int 21h

LOADED:
      push ES
      push AX
      mov AX, PSP
      mov ES, AX
      mov AL, ES:[81h+1]
      cmp AL, '/'
      jne NOT_UNLOAD
      mov AL, ES:[81h+2]
      cmp AL, 'u'
      jne NOT_UNLOAD
      mov AL, ES:[81h+3]
      cmp AL, 'n'
      je UNLOAD

NOT_UNLOAD:
      pop AX
      pop ES
      mov DX, OFFSET INT_ALREADY_LOAD
      call PRINT_STRING
      ret

UNLOAD:
      pop AX
      pop ES
      call DELETE_INTERRUPT
      mov DX, OFFSET INT_UNLOAD
      call PRINT_STRING
      ret
MY_FUNC ENDP

PRINT_STRING PROC NEAR
      push AX
      mov AH, 09h
      int 21h
      pop AX
      ret
PRINT_STRING ENDP

MAIN PROC Far
      mov AX, DATA
      mov DS, AX
      mov PSP, ES
      call MY_FUNC
      xor AL, AL
      mov AH, 4Ch
      int 21H
MAIN ENDP
```