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

ОТЧЕТ

по лабораторной работе № 5

по дисциплине «Операционные системы»

Тема: «Сопряжение стандартного и пользовательского обработчика прерываний»

Студент гр. 8304 _____ Сергеев А.Д.

Преподаватель _____ Губкин А.Ф.

Санкт-Петербург
2020

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.

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:

1. Проверяет, установлено ли пользовательское прерывание с вектором 1Ch.
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.

Выгрузка прерывания по соответствующему значению параметра в командной строке /un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождения 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 определённом, 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. Длина кода сигнатуры должна быть достаточной, чтобы сделать случайное совпадение маловероятным.

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

1. Сохранить значение 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 MCB. Полученные результаты поместите в отчёт.

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

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 отчёт.

Необходимые сведения для составления программы.

Клавиатура содержит микропроцессор, который воспринимает каждое нажатие на клавишу и посылает скан-код в порт микросхемы интерфейса с периферией. Когда скан-код 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 клавиш-переключателей.
3. Поместить код клавиши в буфер клавиатуры.

Текущее содержимое буфера клавиатуры определяется указателями на начало и конец записи.

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.

Описание программы.

В результате выполнения лабораторной работы была написана программа, описание функций которой представлено ниже.

- INTERRUPT - резидентный обработчик прерывания, обрабатывает введённые символы;
- LOAD - загрузка резидентного обработчика INTERRUPTION;
- UNLOAD - выгрузка резидентного обработчика INTERRUPTION;
- PRINT_STRING - вывод строки из DX на экран;
- INT_CHECK - проверка того, установлен ли резидентный обработчик INTERRUPTION;
- TAIL_CHECK - проверка того, содержат ли аргументы, с которыми была вызвана программа /un.

Ход работы

Написание исходного кода производилось в редакторе Atom на базе операционной системы Windows 10, сборка и отладка производились в эмуляторе DOSBox.

Написанный обработчик заменяет символы “C”, “O” и “D” на “S”, “A” и “T” соответственно. Следует обратить внимание на то, что при установке этого обработчика набрать при помощи клавиатуры название модуля из ЛР 3 (free.com) становится невозможно из-за того, что в нём содержатся заменяемые символы. Для корректной работы это название необходимо заранее скопировать в буфер обмена, а потом вставить, или выполнить модуль до установки прерывания, а дальше использовать навигацию между введёнными командами при помощи стрелочек.

Рисунок 1 — Вывод программы inter.exe после первого запуска

Для тестирования использовалось слово “COLDBLOODED” и фраза “OCEAN WATERS ARE DEEP AND COLD”

Рисунок 2 — Вывод программы free.com после выполнения inter.exe

Как видно из рисунка, процедура прерывания осталось резидентной в памяти.

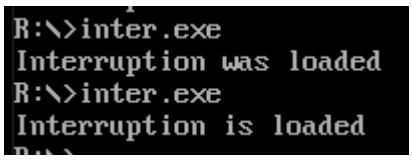

Рисунок 3 — Вывод программы int.exe при повторном запуске

На рисунке 3 показано, что при повторном запуске программа выводит сообщение о том, что резидентный обработчик уже загружен.

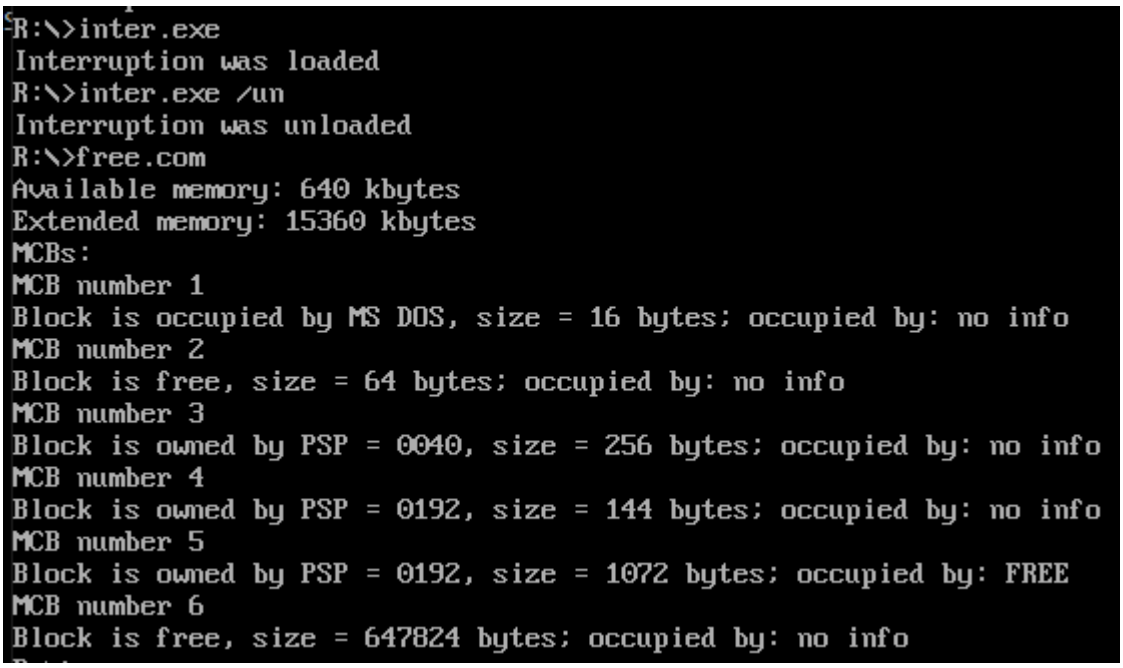

Рисунок 4 - Вывод программы free.com после выполнения int.exe с ключом выгрузки

Из рисунка 4 видно, что после выгрузки резидентного обработчика из памяти вся занятая им память была освобождена.

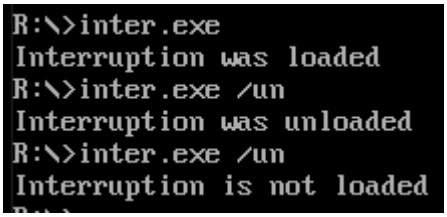

Рисунок 5 — Вывод программы int.exe при повторном запуске с ключом выгрузки

Как видно из рисунка 5, при выгрузке резидентного обработчика было выведено сообщение, а также при запросе повторной выгрузки было показано, что резидентный обработчик не загружен.

Вывод.

В результате выполнения данной лабораторной работы была изучена возможность встраивания пользовательского обработчика прерываний от клавиатуры в стандартный.

Контрольные вопросы.

Какого типа прерывания использовались в программе:

В коде использовались программные прерывания, такие, как int 21h, тогда как само обрабатываемое прерывание от клавиатуры (09h) является аппаратным.

Чем отличается скан-код и ASCII код:

ASCII код — код символа, необходимый для хранения и печати символа. Скан-код — код клавиши на клавиатуре, необходимый для распознавания нажатых клавиш.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. INTER.ASM

```
ASSUME  CS:CODE,    DS:DATA,    ss:ASTACK


CODE    SEGMENT


INTERRUPT    PROC    FAR
        jmp     INT_START
    INT_DATA:
        SUB_STACK             dw  128 dup(0)

        INT_CODE    dw  42025

        KEEP_IP     dw  0
        KEEP_CS     dw  0
        KEEP_SS          dw  0
        KEEP_SP     dw  0
        KEEP_PSP    dw   0

        SYMB        db   0
        TEMP        dw  0

    INT_START:
        mov     KEEP_SS, ss
```

```
mov     KEEP_SP, sp
mov     TEMP, seg INTERRUPT
mov     ss, TEMP
mov     sp, offset SUB_STACK
add     sp, 256

push   ax
push    bx
push    cx
push    dx
push    si
push    es
push    ds

mov    ax, SEG INTERRUPT
mov    ds, ax

in          al, 60h
cmp    al, 2Eh ; replace 'C'
je          INT_PASS_S
cmp    al, 18h ; replace 'O'
je          INT_PASS_A
cmp    al, 20h ; replace 'D'
je          INT_PASS_T
```

```
    pushf
    call   DWORD PTR KEEP_IP
    jmp    INT_END

INT_PASS_S:
    mov         SYMB, 'S'
    jmp         INT_PASS
INT_PASS_A:
    mov         SYMB, 'A'
    jmp         INT_PASS
INT_PASS_T:
    mov         SYMB, 'T'
    jmp         INT_PASS

INT_PASS:
    in          al, 61h
    mov    ah, al
    or          al, 80h
    out    61h, al
    xchg   al, al
    out    61h, al
    mov    al, 20h
    out    20h, al
```

```
INT_PRINT:
    mov    ah, 05h
    mov    cl, SYMB
    mov    ch, 00h
    int    16h
    or          al, al
    jz          INT_END

    mov    ax, 0040h
    mov    es, ax
    mov    ax, es:[1Ah]
    mov    es:[1Ch], ax
    jmp    INT_PRINT

INT_END:
    pop     ds
    pop     es
    pop         si
    pop     dx
    pop     cx
    pop     bx
    pop     ax
```

```
        mov     ss, KEEP_SS
        mov     sp, KEEP_SP

        mov    al, 20h
        out    20h, al
        IRET
INTERRUPT    ENDP
    LAST_BYTE:



LOAD        PROC
        push    ax
          push    bx
          push    cx
          push    dx
          push    es
          push    ds

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

```
mov     dx, offset INTERRUPT
mov     ax, seg INTERRUPT
  mov     ds, ax
  mov     ah, 25h
  mov     al, 09h
  int     21h
  pop       ds

mov     dx, offset LAST_BYTE
add     dx, 100h
  mov     cl, 4h
  shr     dx, cl
  inc     dx
  mov     ah, 31h
  int     21h

pop     es
  pop     dx
  pop     cx
  pop     bx
  pop     ax

 ret
```

```
LOAD        ENDP



UNLOAD      PROC
          push    ax
          push    bx
          push    dx
          push    ds
          push    es
          push    si

          mov     ah, 35h
          mov     al, 09h
          int     21h

         mov    si, offset KEEP_IP
         sub     si, offset INTERRUPT
          mov  dx, es:[bx + si]
         mov    si, offset KEEP_CS
         sub     si, offset INTERRUPT
          mov  ax, es:[bx + si]

          push      ds
```

```
  mov     ds, ax
  mov     ah, 25h
  mov     al, 09h
  int     21h
  pop  ds

mov    si, offset KEEP_PSP
sub     si, offset INTERRUPT
mov    ax, es:[bx + si]
mov     es, ax
push    es
mov    ax, es:[2Ch]
  mov  es, ax
  mov  ah, 49h
  int  21h

pop     es
mov    ah, 49h
  int  21h

pop     si
  pop     es
  pop     ds
  pop     dx
```

```
          pop     bx
          pop     ax

          sti
         ret
UNLOAD      ENDP



INT_CHECK       PROC
          push    ax
          push    bx
          push    si

          mov     ah, 35h
          mov     al, 09h
          int     21h

         mov     si, offset INT_CODE
         sub     si, offset INTERRUPT
          mov     ax, es:[bx + si]
          cmp      ax, INT_CODE
          jne     INT_CHECK_END
```

```
          mov     INT_LOADED, 1

     INT_CHECK_END:
          pop     si
          pop     bx
          pop     ax

         ret
INT_CHECK       ENDP



TAIL_CHECK        PROC
          cmp     byte ptr es:[82h], '/'
          jne     CL_CHECK_END
          cmp     byte ptr es:[83h], 'u'
          jne     CL_CHECK_END
          cmp     byte ptr es:[84h], 'n'
          jne     CL_CHECK_END
          mov     TAIL_UN, 1

     CL_CHECK_END:
          ret
TAIL_CHECK        ENDP
```

```
PRINT_STRING    PROC    NEAR
        push    ax
        mov     ah, 09h
        int     21h
        pop     ax

        ret
PRINT_STRING    ENDP



MAIN PROC
          push    ds
          xor     ax, ax
          push    ax
          mov     ax, DATA
          mov     ds, ax

        mov     KEEP_PSP, es

        call    TAIL_CHECK
```

```
    call    INT_CHECK

    cmp     TAIL_UN, 1
    je      INT_UNLOAD
    cmp     INT_LOADED, 1
    jne     INT_LOAD

    mov     dx, offset IS_LOADED_INFO
    call    PRINT_STRING
    jmp     MAIN_END

INT_LOAD:
   mov     dx, offset LOADED_INFO
   call    PRINT_STRING
    call    LOAD
    jmp     MAIN_END

INT_UNLOAD:
    cmp     INT_LOADED, 1
    jne     NOT_EXIST

    call    UNLOAD
   mov     dx, offset NOT_LOADED_INFO
    call    PRINT_STRING
```

```
        jmp     MAIN_END

    NOT_EXIST:
        mov     dx, offset IS_NOT_LOADED_INFO
        call    PRINT_STRING

    MAIN_END:
        xor  al, al
        mov  ah, 4Ch

        int  21h
    MAIN ENDP

CODE    ENDS

ASTACK  SEGMENT STACK
    dw  128 dup(0)
ASTACK  ENDS

DATA    SEGMENT
    LOADED_INFO         db  "Interruption was loaded$"
    IS_LOADED_INFO      db  "Interruption is loaded$"
    NOT_LOADED_INFO     db  "Interruption was unloaded$"
    IS_NOT_LOADED_INFO  db  "Interruption is not loaded$"
```

```
    INT_LOADED          db  0

    TAIL_UN               db  0

DATA    ENDS


END  MAIN
```