# 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.**

Студент гр. 9382 Русинов Д.А.

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.

Порядок выполнения работы

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.

1. 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.

1. 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 MCB.

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.

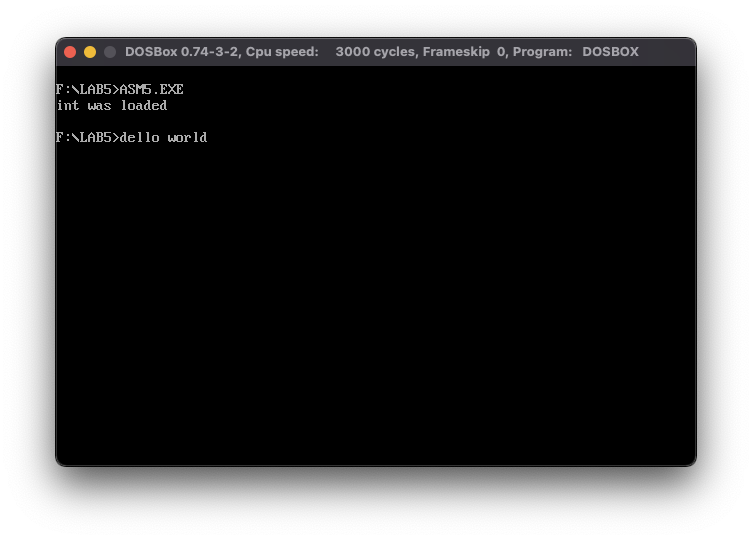
Шаг 5. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Шаг 6. Ответьте на контрольные вопросы.

# Выполнение работы.

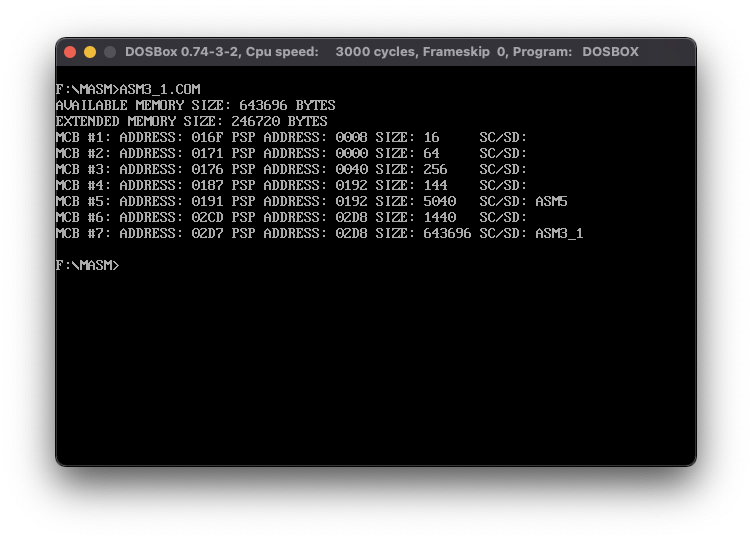
Проверяется наличие аргумента /un при запуске программы. В зависимости от этого будет происходить загрузка или выгрузка резидентной функции. Если прерывание было загружено ранее, то будет выведено сообщение об этом. Если был задан аргумент выгрузки, то будет проверено, загружена ли программа, она будет выгружена только в том случае, если была ранее уже загружена.

Прерывание заменяет определенные символы, введенные с клавиатуры, на заданные в прерывании. Символы “F”, “G”, “H” будут заменены на символы “a”, “s”, “d”.

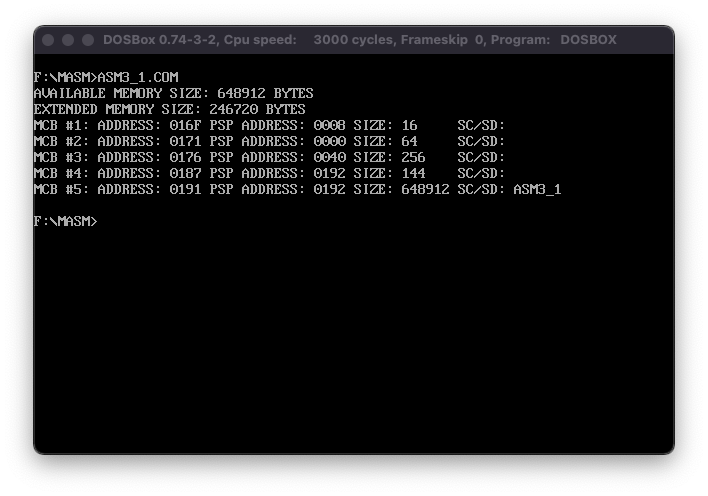


После загрузки модуля, было выведено сообщении о том, что прерывание было установлено. И оно действительно работает – символы, которые должны подменяться – меняются.

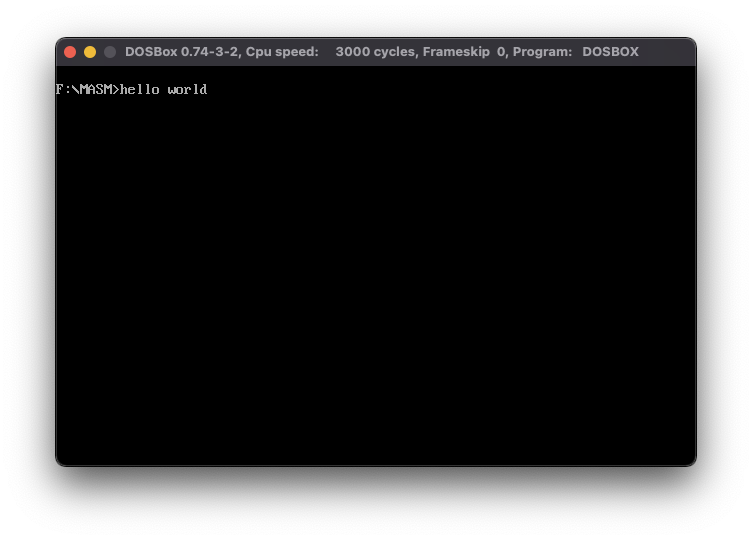
После загрузки модуля прерывания, он находится в памяти.



Теперь попробуем выгрузить модуль с помощью аргумента /un. И проверим с помощью ASM3\_1.COM, загружено ли прерывание в память.



Прерывание было выгружено, в памяти его более нет, теперь нажатия клавиш с клавиатуры обрабатывает стандартный обработчик прерывания.



# Ответы на контрольные вопросы.

1. Какого типа прерывания использовались в работе?

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

1. Чем отличается скан-код от кода ASCII?

Код ASCII – код символы из таблицы ASCII. Скан-код – код, который был присвоен каждой клавише, с помощью которого происходит распознавание нажатия этой клавиши драйвером.

# Выводы.

Были исследованы возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

# ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ

code segment  
assume cs:code**,** ds:data**,** ss:stacks  
  
stacks segment stack  
 dw 256 dup(0)  
stacks ends  
  
  
data segment  
 STR\_LOAD db "int was loaded"**,** 0dh**,** 0ah**,** "$"  
 STR\_LOADED db "int already loaded"**,** 0dh**,** 0ah**,** "$"  
 STR\_UNLOAD db "int was unloaded"**,** 0dh**,** 0ah**,** "$"  
 STR\_NOT\_LOADED db "int is not loaded"**,** 0dh**,** 0ah**,** "$"  
 IS\_LOAD db 0  
 IS\_UN db 0  
data ends  
  
  
interrupt proc far  
 jmp startup  
somedataint:  
 key\_value db 0  
 signature dw 6666h  
 keep\_ip dw 0  
 keep\_cs dw 0  
 keep\_psp dw 0  
 keep\_ax dw 0  
 keep\_ss dw 0  
 keep\_sp dw 0  
 new\_stack dw 256 dup(0)  
startup:  
 mov keep\_ax**,** ax  
 mov keep\_sp**,** sp  
 mov keep\_ss**,** ss  
 mov ax**,** seg new\_stack  
 mov ss**,** ax  
 mov ax**,** offset new\_stack  
 add ax**,** 256  
 mov sp**,** ax   
  
 push ax  
 push bx  
 push cx  
 push dx  
 push si  
 push es  
 push ds  
 mov ax**,** seg key\_value  
 mov ds**,** ax  
   
 in al**,** 60h  
 cmp al**,** 21h  
 je key\_f  
 cmp al**,** 22h  
 je key\_g  
 cmp al**,** 23h  
 je key\_h  
   
 pushf  
 call dword ptr cs:keep\_ip  
 jmp end\_interr  
  
key\_f:  
 mov key\_value**,** 'a'  
 jmp next\_key  
key\_g:  
 mov key\_value**,** 's'  
 jmp next\_key  
key\_h:  
 mov key\_value**,** 'd'  
  
next\_key:  
 in al**,** 61h  
 mov ah**,** al  
 or al**,** 80h  
 out 61h**,** al  
 xchg al**,** al  
 out 61h**,** al  
 mov al**,** 20h  
 out 20h**,** al  
  
  
print\_key:  
 mov ah**,** 05h  
 mov cl**,** key\_value  
 mov ch**,** 00h  
 int 16h  
 or al**,** al  
 jz end\_interr  
 mov ax**,** 0040h  
 mov es**,** ax  
 mov ax**,** es:[1ah]  
 mov es:[1ch]**,** ax  
 jmp print\_key  
  
  
end\_interr:  
 pop ds  
 pop es  
 pop si  
 pop dx  
 pop cx  
 pop bx  
 pop ax  
  
 mov sp**,** keep\_sp  
 mov ax**,** keep\_ss  
 mov ss**,** ax  
 mov ax**,** keep\_ax  
  
 mov al**,** 20h  
 out 20h**,** al  
iret  
interrupt endp  
 \_end:  
  
  
is\_interr\_load proc  
 push ax  
 push bx  
 push si  
   
 mov ah**,** 35h  
 mov al**,** 09h  
 int 21h  
 mov si**,** offset signature  
 sub si**,** offset interrupt  
 mov ax**,** es:[bx + si]  
 cmp ax**,** signature  
 jne eeendis\_l  
 mov is\_load**,** 1  
   
eeendis\_l:  
 pop si  
 pop bx  
 pop ax  
 ret  
 is\_interr\_load endp  
  
 int\_load proc  
 push ax  
 push bx  
 push cx  
 push dx  
 push es  
 push ds  
  
 mov ah**,** 35h  
 mov al**,** 09h  
 int 21h  
 mov keep\_cs**,** es  
 mov keep\_ip**,** bx  
 mov ax**,** seg interrupt  
 mov dx**,** offset interrupt  
 mov ds**,** ax  
 mov ah**,** 25h  
 mov al**,** 09h  
 int 21h  
 pop ds  
  
 mov dx**,** offset \_end  
 mov cl**,** 4h  
 shr dx**,** cl  
 add dx**,** 10fh  
 inc dx  
 xor ax**,** ax  
 mov ah**,** 31h  
 int 21h  
  
 pop es  
 pop dx  
 pop cx  
 pop bx  
 pop ax  
ret  
int\_load endp  
  
  
unload\_interrupt proc  
 cli  
 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 ax**,** es:[bx + si + 2]  
   
 push ds  
 mov ds**,** ax  
 mov ah**,** 25h  
 mov al**,** 09h  
 int 21h  
 pop ds  
   
 mov ax**,** es:[bx + si + 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 si  
 pop es  
 pop ds  
 pop dx  
 pop bx  
 pop ax  
   
ret  
unload\_interrupt endp  
  
  
is\_unload\_ proc  
 push ax  
 push es  
  
 mov ax**,** keep\_psp  
 mov es**,** ax  
 cmp byte ptr es:[82h]**,** '/'  
 jne eeendun  
 cmp byte ptr es:[83h]**,** 'u'  
 jne eeendun  
 cmp byte ptr es:[84h]**,** 'n'  
 jne eeendun  
 mov is\_un**,** 1  
   
eeendun:  
 pop es  
 pop ax  
 ret  
is\_unload\_ endp  
  
  
print\_str proc near  
 push ax  
 mov ah**,** 09h  
 int 21h  
 pop ax  
ret  
print\_str endp  
  
  
begin proc  
 push ds  
 xor ax**,** ax  
 push ax  
 mov ax**,** data  
 mov ds**,** ax  
 mov keep\_psp**,** es  
   
 call is\_interr\_load  
 call is\_unload\_  
 cmp is\_un**,** 1  
 je unload  
 mov al**,** is\_load  
 cmp al**,** 1  
 jne load  
 mov dx**,** offset str\_loaded  
 call print\_str  
 jmp eeend  
  
load:  
 mov dx**,** offset str\_load  
 call print\_str  
 call int\_load  
 jmp eeend  
  
unload:  
 cmp is\_load**,** 1  
 jne not\_loaded  
 mov dx**,** offset str\_unload  
 call print\_str  
 call unload\_interrupt  
 jmp eeend  
  
not\_loaded:  
 mov dx**,** offset str\_not\_loaded  
 call print\_str  
  
eeend:  
 xor al**,** al  
 mov ah**,** 4ch  
 int 21h  
  
  
begin endp  
code ends  
end begin