МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МОЭВМ

ЛАБОРАТОРНАЯ РАБОТА № 5

по дисциплине «Операционные системы»

ТЕМА: Сопряжение стандартного и пользовательского обработчиков прерываний.

Студентка гр. 9383 Сергиенкова А.А.

Преподаватель Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

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

Ход работы.

Шаг 1. Был написан и отлажен программный модуль типа .EXE, который выполняет такие же функции, как и 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 int21h.
3. Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.

   Прерывание заменяет символы с клавиатуры:
   (g -> %; k-> !; m-> $)

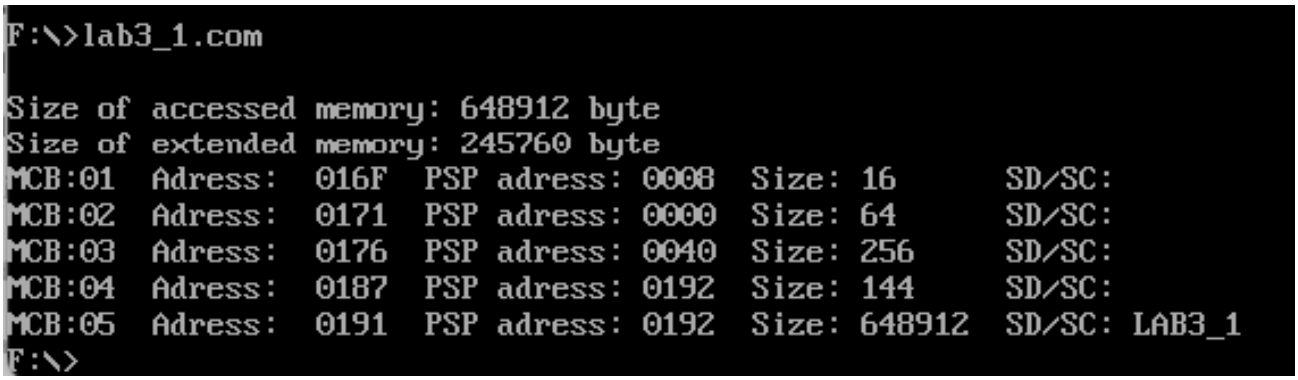

Рисунок 1- Прерывание в памяти не размещено(lab3_1.com).

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.

Рисунок 2 – Проверка установки прерывания 09h(lab5.exe).

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.

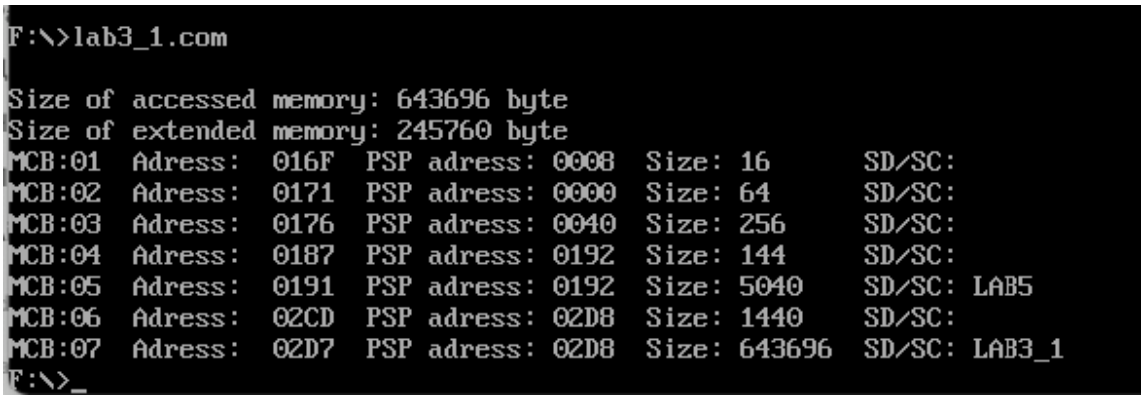

Рисунок 3 – Проверка размещения прерывания в памяти(lab3_1.com).

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

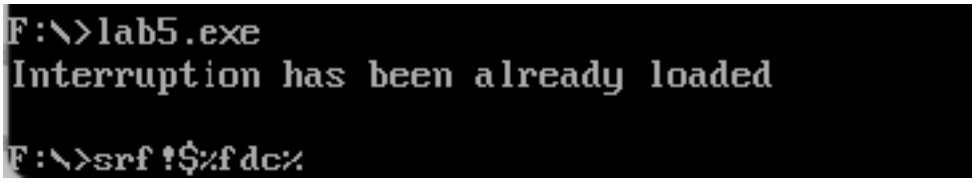

Рисунок 4 – Проверка определения установленного обработчика прерываний(lab5.exe).

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.

Рисунок 5 – Запуск программы с ключом выгрузки(lab5.exe).

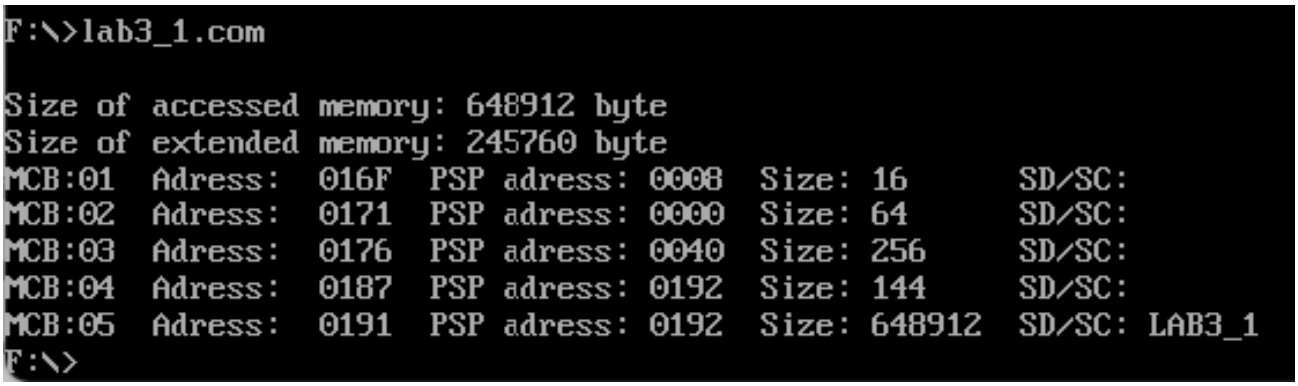

Рисунок 6 – Демонстрация освобождения памяти в lab3_1.com.

Выводы.

Провелось исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Было написано пользовательское прерывание.

Ответы на контрольные вопросы:

1. Какого типа прерывания использовались в работе?

Прерывания функции DOS (int 21h) и аппаратные прерывания (16h, 09h) .

2. Чем отличается скан код от кода ASCII?

Скан-код – код клавиши клавиатуры, с помощью которого определяется, какая клавиша была нажата.

ASCII код – это код из таблицы всех имеющихся символов.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab5.asm

```
ASTACK  segment stack
 dw  256 dup(0)
ASTACK  ends

DATA segment
   IS_LOAD         db                                  0
   IS_UN           db                                  0
   STR_LOAD        db      "Interruption was loaded.",         0dh, 0ah, "$"
   STR_LOADED      db      "Interruption has been already loaded",     0dh, 0ah, "$"
   STR_UNLOAD      db      "Interruption was unloaded.",       0dh, 0ah, "$"
   STR_NOT_LOADED  db      "Interruption is not loaded.",      0dh, 0ah, "$"
DATA ends

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

MY_INTERRUPTION PROC FAR
   jmp  start_iterrupt

Int_Data:
   key_value db 0
   new_stack dw 256 dup(0)
   signature dw 646h
   keep_ip dw 0
   keep_cs dw 0
   keep_psp dw 0
   keep_ax dw 0
   keep_ss dw 0
   keep_sp dw 0
```

```
start_iterrupt:
    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, 22h  ;g
    je key_g
    cmp al, 25h  ;k
    je key_k
    cmp al, 32h   ;m
    je key_m

    pushf
    call dword ptr cs:keep_ip
    jmp end_interruption

key_g:
    mov key_value, '%'
    jmp next_key
```

```
key_k:
   mov key_value, '!'
   jmp next_key
key_m:
   mov key_value, '$'

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_interruption
   mov ax, 40h
   mov es, ax
   mov ax, es:[1ah]
   mov es:[1ch], ax
   jmp print_key


end_interruption:
   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
MY_INTERRUPTION endp
 _end:


INT_LOADED proc
    push ax
    push bx
    push si

    mov  ah, 35h
    mov  al, 09h
    int  21h
    mov  si, offset signature
    sub  si, offset MY_INTERRUPTION
    mov  ax, es:[bx + si]
    cmp  ax, signature
    jne  end_proc
    mov  is_load, 1

end_proc:
    pop  si
    pop  bx
    pop  ax
```

```
    ret
INT_LOADED 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 MY_INTERRUPTION
    mov dx, offset MY_INTERRUPTION
    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


UN_ITERRAPT 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 MY_INTERRUPTION
    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
UN_ITERRAPT endp


IS_UNLOAD proc
    push ax
    push es

    mov ax, keep_psp
    mov es, ax
    cmp byte ptr es:[82h], '/'
    jne end_unload
    cmp byte ptr es:[83h], 'u'
    jne end_unload
    cmp byte ptr es:[84h], 'n'
    jne end_unload
    mov is_un, 1

end_unload:
```

```
    pop es
    pop ax
 ret
IS_UNLOAD endp


PRINT proc near
    push ax
    mov ah, 09h
    int 21h
    pop ax
ret
PRINT endp


BEGIN proc
    push ds
    xor ax, ax
    push ax
    mov ax, DATA
    mov ds, ax
    mov keep_psp, es

    call INT_LOADED
    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
    jmp end_begin

load:
```

```
    mov dx, offset str_load
    call PRINT
    call INT_LOAD
    jmp  end_begin

unload:
    cmp  is_load, 1
    jne  not_loaded
    mov dx, offset str_unload
    call PRINT
    call UN_ITERRAPT
    jmp  end_begin

not_loaded:
    mov  dx, offset str_not_loaded
    call PRINT

end_begin:
    xor al, al
    mov ah, 4ch
    int 21h
BEGIN endp
CODE    ENDS
end begin
```

Название файла: lab3_1.asm

```
TESTPC SEGMENT
  ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
  ORG 100H
START: JMP BEGIN
; Данные
ACCESSED_MEMORY db 13,10,'Size of accessed memory:      $'
EXTENDED_MEMORY db 13,10,'Size of extended memory:      $'
STR_BYTE db ' byte $'
```

```
STR_MCB db 13,10,'MCB:0   $'
ADRESS db 'Adress:        $'
ADRESS_PSP db 'PSP adress:       $'
STR_SIZE db 'Size:        $'
MCB_SD_SC db ' SD/SC: $'
STR_ERROR db 13,10,'Memory Error!$'
STR_SUCCECC db 13,10,'Success!$'

; Процедуры
;--------------------------------------------------
TETR_TO_HEX PROC near
  and AL,0Fh
  cmp AL,09
  jbe next
  add AL,07
next:
  add AL,30h
  ret
TETR_TO_HEX ENDP
;--------------------------------
BYTE_TO_HEX PROC near
;байт в AL переводится в два символа шест. числа в AX
  push CX
  mov AH,AL
  call TETR_TO_HEX
  xchg AL,AH
  mov CL,4
  shr AL,CL
  call TETR_TO_HEX ;в AL старшая цифра
  pop CX ;в AH младшая
  ret
BYTE_TO_HEX ENDP
;--------------------------------
WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти разрядного числа
```

```
; в AX - число, DI - адрес последнего символа
  push BX
  mov BH,AH
  call BYTE_TO_HEX
  mov [DI],AH
  dec DI
  mov [DI],AL
  dec DI
  mov AL,BH
  call BYTE_TO_HEX
  mov [DI],AH
  dec DI
  mov [DI],AL
  pop BX
  ret
WRD_TO_HEX ENDP
;--------------------------------------------------
BYTE_TO_DEC PROC near
; перевод в 10с/с, SI - адрес поля младшей цифры
  push CX
  push DX
  xor AH,AH
  xor DX,DX
  mov CX,10
loop_bd:
  div CX
  or DL,30h
  mov [SI],DL
  dec SI
  xor DX,DX
  cmp AX,10
  jae loop_bd
  cmp AL,00h
  je end_l
  or AL,30h
```

```
  mov [SI],AL
end_l:
  pop DX
  pop CX
  ret
BYTE_TO_DEC ENDP
;------------------------------
WRITE_STRING PROC near
  push AX
  mov AH,09h
  int 21h
  pop AX
  ret
WRITE_STRING ENDP
;------------------------------

WRITE_SIZE PROC
  push ax
  push bx
  push cx
  push dx
  push si

mov bx,10h
mul bx
mov bx,0ah
xor cx,cx

separation:
div bx
push dx
inc cx
xor dx,dx
cmp ax,0h
jnz separation
```

```
write_symbol:
pop dx
or dl,30h
mov [si], dl
inc si
loop write_symbol

  pop si
  pop dx
  pop cx
  pop bx
  pop ax
ret
WRITE_SIZE ENDP
;-------------------------------
PRINT_MCB PROC
  push ax
  push bx
  push cx
  push dx
  push si

  mov ah,52h
  int 21h
  mov ax,es:[bx-2]
  mov es,ax
  mov cl,1

pargraph_MCB:
lea si, STR_MCB
add si, 7

 mov al,cl
push cx
```

```
call BYTE_TO_DEC
lea dx, STR_MCB
call WRITE_STRING

mov ax,es
lea di,ADRESS
add di,12
call WRD_TO_HEX
lea dx,ADRESS
call WRITE_STRING

xor ah,ah
mov al,es:[0]
push ax
mov ax,es:[1]
lea di, ADRESS_PSP
add di, 15
call WRD_TO_HEX
lea dx, ADRESS_PSP
call WRITE_STRING
mov ax,es:[3]
lea si,STR_SIZE
add si, 6
call WRITE_SIZE
lea dx,STR_SIZE
call WRITE_STRING
xor dx, dx
lea dx , MCB_SD_SC
call WRITE_STRING
mov cx,8
xor di,di

write_char:
mov dl,es:[di+8]
mov ah,02h
```

```
int 21h
inc di
loop write_char

mov ax,es:[3]
mov bx,es
add bx,ax
inc bx
mov es,bx
pop ax
pop cx
inc cx
cmp al,5Ah ; проверка на не последний ли это сегмент
je exit
cmp al,4Dh
jne exit
jmp pargraph_MCB

exit:
  pop si
  pop dx
  pop cx
  pop bx
  pop ax
 ret
PRINT_MCB ENDP

PRINT_MEM_SIZE proc near

 push ax
 push bx
 push si

 mov  AL,30h
 out 70h,AL
```

```
  in AL,71h
  mov BL,AL
  mov AL,31h
  out 70h,AL
  in AL,71h

  mov bh,al
  mov ax,bx
  lea si,EXTENDED_MEMORY
  add si, 27
  call WRITE_SIZE
  lea dx,EXTENDED_MEMORY
  call WRITE_STRING
  lea dx,STR_BYTE
  call WRITE_STRING

  pop si
  pop bx
  pop ax
  ret

PRINT_MEM_SIZE ENDP

PRINT_AVAILABLE_MEM_SIZE PROC near

  push ax
  push bx
  push si

;доступная память
  mov ah,4ah
  mov bx,0ffffh
  int 21h
  mov ax,bx
  lea si, ACCESSED_MEMORY
```

```
    add si, 27 ;смещение для числа
    call WRITE_SIZE
    lea dx, ACCESSED_MEMORY
    call WRITE_STRING
    lea dx,STR_BYTE
    call WRITE_STRING

    pop si
    pop bx
    pop si
    ret

  PRINT_AVAILABLE_MEM_SIZE ENDP

  ; Код
  BEGIN:
    ;доступная память
    call PRINT_AVAILABLE_MEM_SIZE
    ; расширенная память
    call PRINT_MEM_SIZE
    ;MCB
    call PRINT_MCB

    xor AL,AL
    mov AH,4Ch
    int 21H
  TESTPC ENDS
END START
```