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. 8382 _____ Янкин Д.О.

Преподаватель _____ Ефремов М.А.

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 кан-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.

Ход работы.

Был написан программный модуль типа .EXE, который:

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.
4) Выгрузка прерывания по соответствующему значению параметра в командной строке /un. Выгрузка состоит в восстановлении стандартного вектора прерываний и освобождении памяти, занимаемой резидентом. Затем осуществляется выход в DOS по функции 4Ch прерывания int 21h.

Состояние памяти после загрузки пользовательского обработчика и запуска программы для вывода состояния памяти показано на рисунке 1. Пользовательский обработчик перехватывает нажатия TAB и помещает в буфер клавиатуры символ 03h. Остальные нажатия передаются на стандартный обработчик.

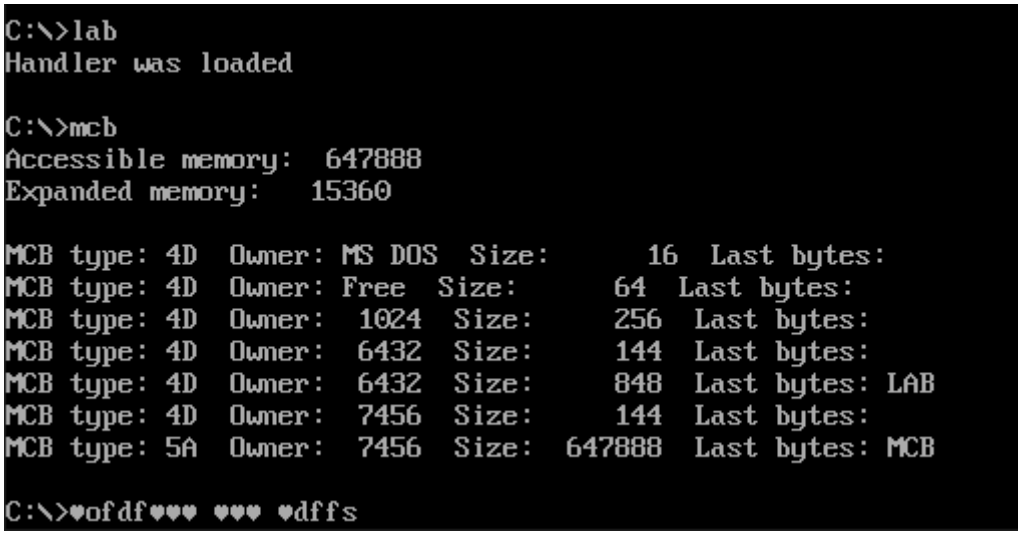

Рисунок 1. Состояние памяти после загрузки резидента

Повторный запуск программы при уже загруженном резиденте показан на рисунке 2.

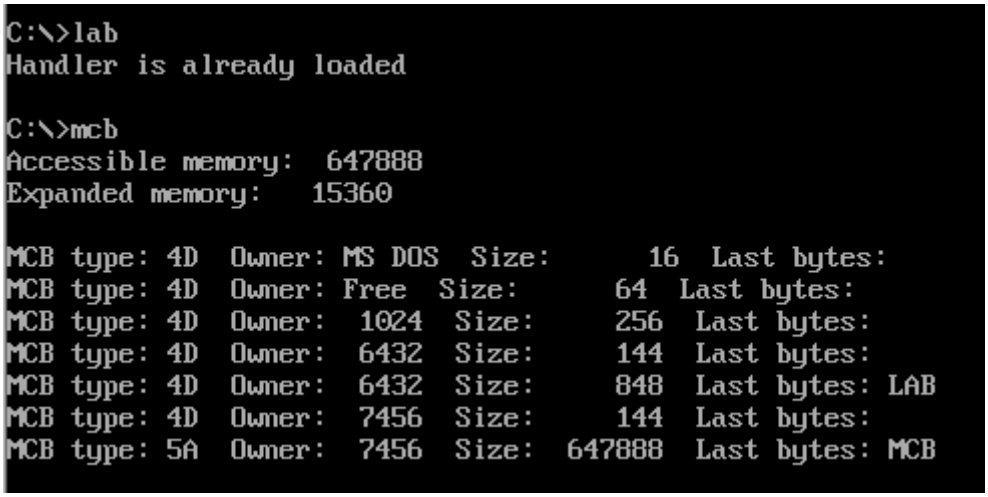

Рисунок 2. Повторный запуск программы

Выгрузка резидента из памяти и состояние памяти после этого показаны на рисунке 3.

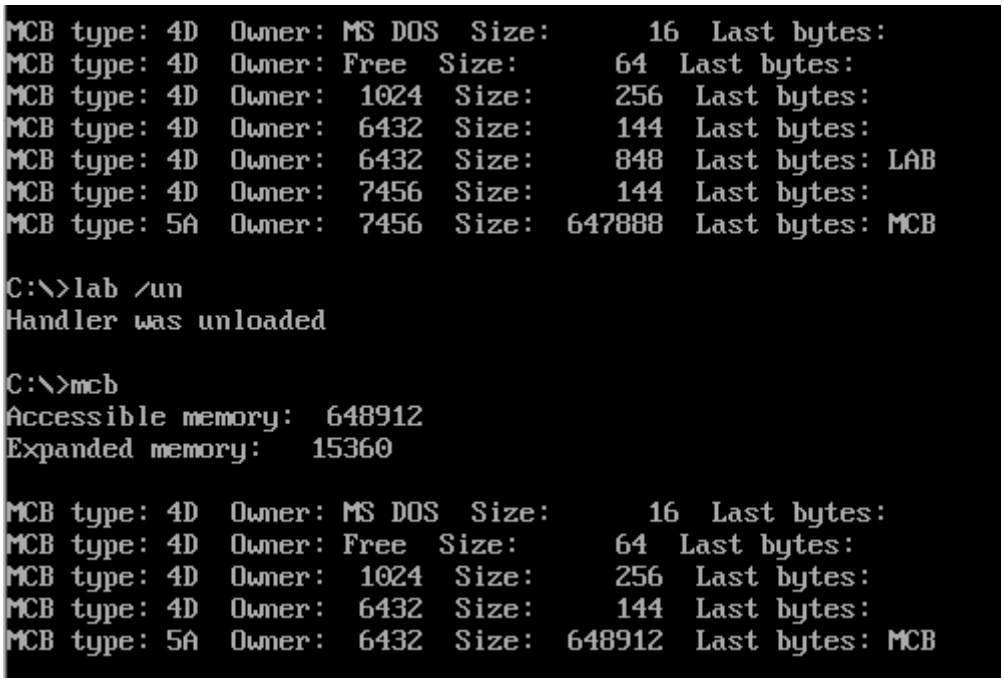

Рисунок 3. Состояние памяти после выгрузки прерывания

Контрольные вопросы.

1) Какого типа прерывания использовались в работе?

   В программе был реализован пользовательский обработчик для аппаратного прерывания от системного клавиатуры 09h. Также использовались программные прерывания 21h и 10h.

2) Чем отличается скан код от кода ASCII?

   Скан код – номер, жестко закрепленный за клавишей, который посылается контроллером клавиатуры в порт 60h. Код ASCII – номер, закрепленный за символом в таблице кодировки.

Выводы.

В ходе лабораторной работы был встроен пользовательский обработчик прерывания в стандартное прерывание от клавиатуры.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ LAB.ASM

```
CODE SEGMENT
	ASSUME  CS:CODE, DS:DATA, ES:NOTHING, SS:ASTACK

	; Обработчик, ради которого вся программа
	INT_HANDLER PROC FAR
		jmp		INT_HANDLER_CODE

		INT_HANDLER_DATA:
			INT_HANDLER_SIGNATURE	DW	1825h
			KEEP_IP			DW	0
			KEEP_CS			DW	0
			KEEP_PSP		DW	0

			KEEP_SS				DW	0
			KEEP_SP				DW	0
			KEEP_AX				DW	0

			KEY_CODE		DB	0Fh

			HANDLER_STACK		DW	100	DUP(0)
			HANDLER_STACK_TOP:

		INT_HANDLER_CODE:
		mov	KEEP_SS, ss
		mov	KEEP_SP, sp
		mov	KEEP_AX, ax

		mov	ax, seg HANDLER_STACK
		mov	ss, ax
		mov	sp, offset HANDLER_STACK_TOP

		push	cx
```

```
push  si
push  es

in          al, 60h
cmp   al, KEY_CODE
je          INT_HANDLER_PROCESS

pushf
call  dword ptr CS:KEEP_IP
jmp   INT_HANDLER_RET


INT_HANDLER_PROCESS:
; Сигнал подтверждения
in          al, 61h
mov   ah, al
or          al, 80h
out   61h, al
xchg  ah, al
out   61h, al
mov   al, 20h
out   20h, al

; Добавление в буфер
INT_HANDLER_BUFFER:
mov   ah, 05h
mov   cl, 03h
mov   ch, 00h
int   16h
or          al, al
jz          INT_HANDLER_RET

mov   ax, 0040h
mov   es, ax
```

```
	mov	si, 001ah
	mov	ax, es:[si]
	mov	si, 001ch
	mov	es:[si], ax
	jmp	INT_HANDLER_BUFFER


	INT_HANDLER_RET:
	pop	es
	pop		si
	pop		cx

	mov		sp, KEEP_SP
	mov		ax, KEEP_SS
	mov		ss, ax
	mov		ax, KEEP_AX

	mov		al, 20h
	out		20h, al

	iret
INT_HANDLER ENDP
INT_HANDLER_END:


; Результат в AX. 1 – сигнатура совпала; 0 – не совпала
CHECK_INT_HANDLER PROC
	push	bx
	push	si
	push	es

	; Взятие смещения от начала обработчика до его сигнатуры
	mov		si, offset INT_HANDLER_SIGNATURE
	sub		si, offset INT_HANDLER
```

```
        ; Взятие адреса установленного обработчка
        mov     ah, 35h
        mov     al, 09h
        int     21h

        ; В AX кладется предполагаемая сигнатура из
установленного обработчика
        ; В BX кладется правильная сигнатура
        mov     ax, es:[bx+si]
        mov     bx, INT_HANDLER_SIGNATURE

        ; Сравнение предполагаемой с эталонной
        ; Не совпали – 0
        ; Совпали – 1
        cmp     ax, bx
        je      CHECK_INT_HANDLER_TRUE

        CHECK_INT_HANDLER_FALSE:
        mov     ax, 0
        jmp     CHECK_INT_HANDLER_END

        CHECK_INT_HANDLER_TRUE:
        mov     ax, 1

        CHECK_INT_HANDLER_END:
        pop     es
        pop     si
        pop     bx
        ret
    CHECK_INT_HANDLER ENDP


    ; Результат в AX. 1 – хвост /un; 0 – не /un
```

```
CHECK_CML_TAIL PROC
      ; Проверка на непосредственно /un
      cmp        byte ptr es:[82h], '/'
      jne        CHECK_CML_TAIL_FALSE
      cmp        byte ptr es:[83h], 'u'
      jne        CHECK_CML_TAIL_FALSE
      cmp        byte ptr es:[84h], 'n'
      jne        CHECK_CML_TAIL_FALSE

      ; Проверка на перевод строки или пробел после /un
      cmp        byte ptr es:[85h], 13
      je         CHECK_CML_TAIL_TRUE
      cmp        byte ptr es:[85h], ' '
      je         CHECK_CML_TAIL_TRUE

      CHECK_CML_TAIL_FALSE:
      mov        ax, 0
      ret
      CHECK_CML_TAIL_TRUE:
      mov        ax, 1
      ret
CHECK_CML_TAIL ENDP


; Загрузка местного обработчика
LOAD_HANDLER PROC
      push  ax
      push  bx
      push  dx
      push  es

      ; Сохранение предыдущего обработчика
      mov        ah, 35h
      mov        al, 09h
```

```
	int		21h
	mov		KEEP_IP, bx
	mov		KEEP_CS, es

	; Загрузка нашего, домашнего
	push  ds
	mov		dx, offset INT_HANDLER
	mov		ax, seg INT_HANDLER
	mov		ds, ax
	mov		ah, 25h
	mov		al, 09h
	int		21h
	pop		ds

	pop		es
	pop		dx
	pop		bx
	pop		ax
	ret
LOAD_HANDLER ENDP


; Установка резидентности
SET_RESIDENT PROC
	mov		dx, offset INT_HANDLER_END
	mov		cl, 4
	shr		dx, cl

	add		dx, 20h
	inc		dx

	mov		ax, 3100h
	int		21h
SET_RESIDENT ENDP
```

```
; Возвращение короля
UNLOAD_HANDLER PROC
	push  ax
	push  bx
	push  dx
	push  es
	push  si

	; Взятие смещения до сохраненных данных
	mov	si, offset KEEP_IP
	sub	si, offset INT_HANDLER

	; Взятие в ES:BX текущего обработчика
	mov	ah, 35h
	mov	al, 09h
	int	21h

	; Загрузка сохраненного дефолтного
	cli
	push  ds
	mov	dx, es:[bx+si]
	mov	ax, es:[bx+si+2]
	mov	ds, ax
	mov	ah, 25h
	mov	al, 09h
	int	21h
	pop	ds
	sti

	; Освобождение той памяти, что занимал наш обработчик и
переменные среды
	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

        pop     si
        pop     es
        pop     dx
        pop     bx
        pop     ax
        ret
    UNLOAD_HANDLER ENDP


    ; Просто выводит строку с уже указанным в dx смещением, очень
сложная функция
    PRINT_STRING PROC
        push  ax
        mov     ah, 09h
        int     21h
        pop     ax
        ret
    PRINT_STRING ENDP


    MAIN PROC
        mov     ax, DATA
        mov     ds, ax
```

```
mov        KEEP_PSP, es

call  CHECK_CML_TAIL
cmp        ax, 1
jne        LOAD

UNLOAD:
     call  CHECK_INT_HANDLER
     cmp        ax, 1
     je         UNLOAD_EXIST

     UNLOAD_DOESNT_EXIST:
     mov        dx, offset HANDLER_ISNT_LOADED_MESSAGE
     call  PRINT_STRING
     mov        ax, 4C00h
     int        21h

     UNLOAD_EXIST:
     call  UNLOAD_HANDLER
     mov        dx, offset HANDLER_UNLOADED_MESSAGE
     call  PRINT_STRING
     mov        ax, 4C00h
     int        21h


LOAD:
     call  CHECK_INT_HANDLER
     cmp        ax, 1
     je         LOAD_EXIST

     LOAD_DOESNT_EXIST:
     call  LOAD_HANDLER
     mov        dx, offset HANDLER_LOADED_MESSAGE
```

```
                call  PRINT_STRING
                call  SET_RESIDENT
                mov         ax, 4C00h
                int         21h

                LOAD_EXIST:
                mov         dx, offset
HANDLER_ALREADY_LOADED_MESSAGE
                call  PRINT_STRING
                mov         ax, 4C00h
                int         21h

           mov         ax, 4C00h
           int         21h
     MAIN ENDP
CODE ENDS

ASTACK     SEGMENT  STACK
     DW 100 DUP(0)
ASTACK     ENDS

DATA SEGMENT
     HANDLER_LOADED_MESSAGE                DB    "Handler was
loaded", 13, 10, '$'
     HANDLER_ALREADY_LOADED_MESSAGE   DB    "Handler is already
loaded", 13, 10, '$'
     HANDLER_UNLOADED_MESSAGE         DB    "Handler was unloaded",
13, 10, '$'
     HANDLER_ISNT_LOADED_MESSAGE           DB    "Handler isn't
loaded", 13, 10, '$'
DATA ENDS

END MAIN
```