

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

ОТЧЕТ
по лабораторной работе №4
по дисциплине «Операционные системы»
Тема: Обработка стандартных прерываний

Студент гр. 8382

Янкин Д.О.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

В архитектуре компьютера существуют стандартные прерывания, за которыми закреплены определенные вектора прерываний. Вектор прерываний хранит адрес подпрограммы обработчика прерываний. При возникновении прерывания аппаратура компьютера передает управление по соответствующему адресу вектора прерывания. Обработчик прерываний получает управление и выполняет соответствующие действия.

В лабораторной работе предлагается построить обработчик прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определенные интервалы времени и, при возникновении такого сигнала, возникает прерывание с определенным значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

Ход работы.

Был написан программный модуль типа .EXE, который:

- 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 31h прерывания int 21h.
- 3) Если прерывание установлено, выводит соответствующее сообщение и осуществляет выход в DOS 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.

```
C:\>lab
C:\>mcb
Accessible memory: 648240
Expanded memory: 15360
MCB type: 4D Owner: MS DOS Size: 16 Last bytes:
MCB type: 4D Owner: Free Size: 64 Last bytes:
MCB type: 4D Owner: 1024 Size: 256 Last bytes:
MCB type: 4D Owner: 6432 Size: 144 Last bytes:
MCB type: 4D Owner: 6432 Size: 496 Last bytes: LAB
MCB type: 4D Owner: 7104 Size: 144 Last bytes:
MCB type: 5A Owner: 7104 Size: 648240 Last bytes: MCB
```

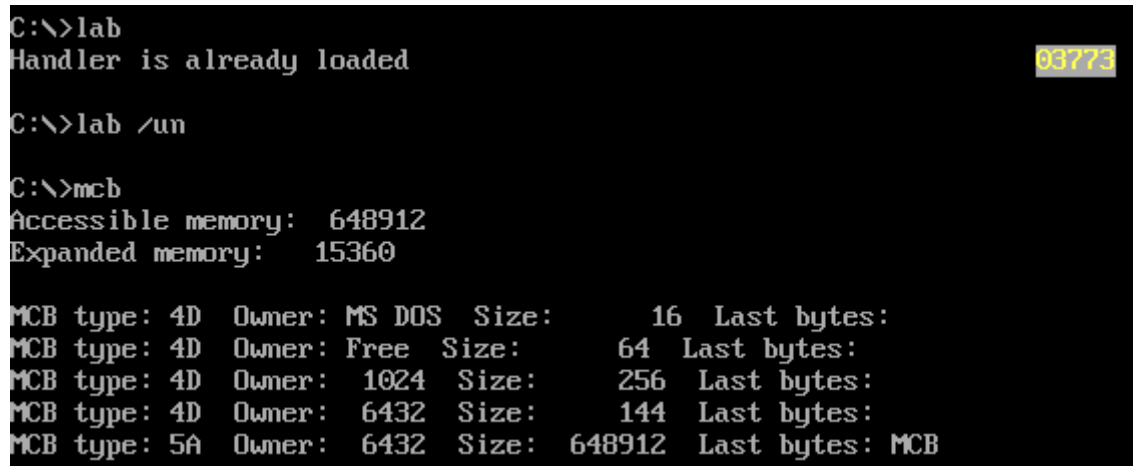
Рисунок 1. Состояние памяти после загрузки резидента

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

```
C:\>lab
C:\>mcb
Accessible memory: 648240
Expanded memory: 15360
MCB type: 4D Owner: MS DOS Size: 16 Last bytes:
MCB type: 4D Owner: Free Size: 64 Last bytes:
MCB type: 4D Owner: 1024 Size: 256 Last bytes:
MCB type: 4D Owner: 6432 Size: 144 Last bytes:
MCB type: 4D Owner: 6432 Size: 496 Last bytes: LAB
MCB type: 4D Owner: 7104 Size: 144 Last bytes:
MCB type: 5A Owner: 7104 Size: 648240 Last bytes: MCB
C:\>lab
Handler is already loaded
```

Рисунок 2. Повторный запуск программы

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



```
C:\>lab
Handler is already loaded

C:\>lab /un

C:\>mcb
Accessible memory: 648912
Expanded memory: 15360

MCB type: 4D Owner: MS DOS Size: 16 Last bytes:
MCB type: 4D Owner: Free Size: 64 Last bytes:
MCB type: 4D Owner: 1024 Size: 256 Last bytes:
MCB type: 4D Owner: 6432 Size: 144 Last bytes:
MCB type: 5A Owner: 6432 Size: 648912 Last bytes: MCB
```

Рисунок 3. Состояние памяти после выгрузки прерывания

Контрольные вопросы.

- 1) Как реализован механизм прерывания от часов?

Примерно 18 раз в секунду вызывается аппаратное прерывание от системного таймера. В стеке сохраняется состояние текущей программы: CS:IP и регистр флагов. Устанавливается флаг для запрета внешних прерываний. Из таблицы векторов прерываний берется вектор, соответствующий данному прерыванию, который хранит IP и CS обработчика прерывания. Управление передается по указанному адресу. Обработчик сохраняет состояние регистров, производит инкремент счетчика, переводит число в счетчике строковое представление, выводит его на экран. Значения регистров восстанавливаются, разрешаются внешние прерывания. Управление возвращается прерванной программе.

- 2) Какого типа прерывания использовались в работе?

В программе был реализован пользовательский обработчик для аппаратного прерывания от системного таймера 1Ch. Также использовались программные прерывания 21h и 10h.

Выводы.

В ходе лабораторной работы был реализован пользовательский обработчик прерываний от системного таймера.

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 1488h ; Делать лабки мы
не бросим!

KEEP_CS DW 0

KEEP_IP DW 0

KEEP_PSP DW 0

COUNTER DW 0

COUNTER_STR DB '00000\$'

INT_HANDLER_CODE:

push ax

push bx

push cx

push dx

push si

push ds

mov ax, CODE

mov ds, ax

; Инкремент счетчика и запись в строку

inc COUNTER

mov ax, COUNTER

```
mov dx, 0
mov si, offset COUNTER_STR
add si, 4
call WRD_TO_DEC
```

```
; Сохранение текущей позиции курсора
mov bh, 0
mov ah, 03h
int 10h
push dx
```

```
; Установка новой позиции курсора
mov bh, 0
mov dx, 1640h
mov ah, 02h
int 10h
```

```
; Вывод строки-счетчика
push es
push bp
```

```
mov ax, CODE
mov es, ax
mov bp, offset COUNTER_STR
```

```
mov al, 1
mov bh, 0
mov cx, 5
mov ah, 13h
int 10h
```

```
pop bp
pop es
```

```
; Восстановление изначальной позиции курсора
```

```
pop dx
mov bx, 0
mov ah, 02h
int 10h
```

```
mov al, 20h
out 20h, al
```

```
pop ds
pop si
pop dx
pop cx
pop bx
pop ax
iret
```

```
INT_HANDLER ENDP
```

```
; Число в AX:DX записывается в 10-ой CC в DS:SI
```

```
WRD_TO_DEC PROC near
```

```
    push ax
    push bx
```

```
    mov bx, 10
```

```
div_loop:
```

```
    div bx
    add dl, 30h
    mov [si], dl
    dec si
    mov dx, 0
```



```
        cmp ax, 0
        jne div_loop
```

```
        pop bx
        pop ax
        ret
```

```
WRD_TO_DEC 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 обработчика

```
    mov ah, 35h
    mov al, 1Ch
    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, 1Ch
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, 1Ch
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, 16h
        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_CS
        sub si, offset INT_HANDLER
```

```

; Взятие в ES:BX текущего обработчика
mov ah, 35h
mov al, 1Ch
int 21h

; Загрузка сохраненного дефолтного
cli
push ds
mov dx, es:[bx+si+2]
mov ax, es:[bx+si]
mov ds, ax
mov ah, 25h
mov al, 1Ch
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_LODAED_MESSAGE
call PRINT_STRING
mov ax, 4C00h
int 21h
```

```
UNLOAD_EXIST:
call UNLOAD_HANDLER
mov ax, 4C00h
int 21h
```

```
LOAD:
call CHECK_INT_HANDLER
cmp ax, 1
je LOAD_EXIST
```

```
LOAD_DOESNT_EXIST:
call LOAD_HANDLER
call SET_RESIDENT
mov ax, 4C00h
int 21h
```

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

```
mov ax, 4C00h
int 21h
```

```
MAIN ENDP
```

```
CODE ENDS
```

```
ASTACK    SEGMENT STACK
          DW 64 DUP(0)
ASTACK    ENDS
```

```
DATA SEGMENT
```

```
    HANDLER_ALREADY_LODAED_MESSAGE DB    "Handler is already
loaded", 13, 10, '$'
```

```
    HANDLER_ISNT_LODAED_MESSAGE    DB    "Handler isn't
loaded", 13, 10, '$'
```

```
DATA ENDS
```

```
END MAIN
```