

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

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Операционные системы»
Тема: Обработка стандартных прерываний

Студентка гр. 8382

Кузина А.М.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Построение обработчика прерываний сигналов таймера.

Ход выполнения работы.

Была написана программа, исходный код которой приведен в приложении А, выполняющая следующие функции:

- Проверяет, установлено ли пользовательское прерывание с вектором 1Ch
- Устанавливает резидентную функцию для обработки прерываний, настраивает вектор прерываний, если прерывание не установлено, и осуществляет выход по функции 4Ch прерывания int 21h.
- Если прерывание установлено, то выводит сообщение об этом и выходит по функции 4Ch прерывания int 21h.
- Выгрузка прерываний по соответствующему значению параметра в командной строке /un. Выгрузка прерываний состоит в восстановлении стандартного вектора прерываний и освобождении памяти, занимаемой резидентом. Затем осуществляет выход по функции 4Ch прерывания int 21h.

Код пользовательского прерывания должен выполнять следующие функции:

- Сохранять значения регистров в стеке при входе и восстановить их при выходе.
- При выполнении тела процедуры накапливать общее суммарное число прерываний и выводить на экран. Для вывода на экран следует использовать прерывание int 10h, которое позволяет непосредственно выводить информацию на экран.

Состояние памяти после загрузки пользовательского обработчика прерываний и счетчик прерываний показаны на рисунке 1.

Рисунок 1 — Память после загрузки обработчика прерываний.

```

C:\>14
Handler succsefully load
                                     Number of interruptions: 126
C:\>13
Available memory: 647984-B
Extended memory: 15360-KB
MCB:
  Type: 4Dh Owner address: 0008h Size: 16-B Last bytes info:
MCB:
  Type: 4Dh Owner address: 0000h Size: 64-B Last bytes info:
MCB:
  Type: 4Dh Owner address: 0040h Size: 256-B Last bytes info:
MCB:
  Type: 4Dh Owner address: 0192h Size: 144-B Last bytes info:
MCB:
  Type: 4Dh Owner address: 0192h Size: 752-B Last bytes info: L4
MCB:
  Type: 4Dh Owner address: 01CCh Size: 144-B Last bytes info:
MCB:
  Type: 5Ah Owner address: 01CCh Size: 647984-B Last bytes info: L3
                                     Number of interruptions: 1478
C:\>

```

На рисунке 2 показано сообщение, выводимое программой при повторной попытке загрузке обработчика прерывания.

Рисунок 2 — Сообщение при повторной попытке загрузке обработчика.

```

C:\>14
Handler already load, unable to load again
                                     Number of interruptions: 4898

```

На рисунке 3 показано состояние памяти после выгрузки пользовательского обработчика прерывания и сообщение при повторной попытке выгрузки.

Рисунок 3 — Состояние памяти после выгрузки обработчика прерывания.

```

C:\>14 /un
Handler succsefully unload
C:\>13
Available memory: 648912-B
Extended memory: 15360-KB
MCB:
  Type: 4Dh Owner address: 0008h Size: 16-B Last bytes info:
MCB:
  Type: 4Dh Owner address: 0000h Size: 64-B Last bytes info:
MCB:
  Type: 4Dh Owner address: 0040h Size: 256-B Last bytes info:
MCB:
  Type: 4Dh Owner address: 0192h Size: 144-B Last bytes info:
MCB:
  Type: 5Ah Owner address: 0192h Size: 648912-B Last bytes info: L3
C:\>14 /un
Handler not load yet, unable to unload

```

Контрольные вопросы

- Как реализован механизм прерывания от часов?

Примерно каждые 55мс — 18 раз в секунду принимается сигнал прерывания от системного таймера. Текущее состояние программы — содержимое регистров и cs:ip — сохраняется в стек, и также временно запрещается обработка внешних прерываний. Затем управление передается cs:ip и происходит обработка прерывания — в данном случае увеличение счетчика и вывод его на экран. Затем вновь разрешается обработка прерываний от внешних устройств, восстанавливаются значения регистров и управление возвращается прерванной программе.

- Какого типа прерывания использовались в работе?

Был реализован пользовательский обработчик аппаратного прерывания с вектором 1Ch. Также были использованы программные прерывания int 10h — видео сервис BIOS и int 21h — сервисы DOS.

Выводы

В ходе лабораторной работы была исследована обработка стандартных прерываний, а также написан пользовательский обработчик прерываний сигналов таймера, которые аппаратно генерируются каждые 55мс. Программа загружает или выгружает обработчик прерывания в память, в зависимости от параметра в командной строке.

ПРИЛОЖЕНИЕ А

Исходный код программы l4.asm

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

NewInt PROC FAR
    jmp IntCode

IntData:
    Signature DW 6666h ;сигнатура для проверки загружено ли наше прерывание
    KeepCS   DW 0
    KeepIP   DW 0
    KeepPSP  DW 0
    KeepSS   DW 0
    KeepSP   DW 0
    KeepAX   DW 0
    Count    DW 0 ;счетчик, который будет добавлен в строку
    SCount   DB 'Number of interruptions:      $'
    ExtraStack DW 100 dup(0)

IntCode:
    mov KeepSS, ss
    mov KeepSP, sp
    mov KeepAX, ax

    mov ax, seg ExtraStack
    mov ss, ax
    mov sp, offset IntCode

    push bx
    push cx
    push dx
    push si
    push ds

    mov ax, seg IntData
    mov ds, ax

    inc Count ;увеличение счетчика
    mov ax, Count
    mov dx, 0
    mov si, offset SCount
    add si, 30

    push ax
    push bx
    mov bx, 10
div_loop: ;запись нового значения счетчика в строку
    div bx
    add dl, 30h
    mov [si], dl
    dec si
    mov dx, 0
    cmp ax, 0
    jne div_loop

    pop bx
    pop ax

    mov bh, 0 ;сохранение позиции курсора - откуда должен был быть продолжен вывод
```

```

mov ah, 03h
int 10h
push dx

mov bh, 0
mov dx, 1720h ;установка курсора в dl - столбец dh - строка куда мы выведем нашу
строку
mov ah, 02h
int 10h

push es
push bp
mov ax, seg SCount
mov es, ax
mov bp, offset SCount
mov al, 1 ; режим вывода строки
mov bh, 0 ; страница
mov cx, 31 ;длина выводимой строки
mov bl, 13 ; цвет текста
mov ah, 13h ;вывод строки туда, куда установлен курсор
int 10h

pop bp
pop es

pop dx ;возвращаем курсор в его изначальное положение, сохраненное ранее
mov bx, 0
mov ah, 02h
int 10h

pop ds
pop si
pop dx
pop cx
pop bx

mov sp, KeepSP
mov ax, KeepSS
mov ss, ax
mov ax, KeepAX ;восстанавливаем регистры

mov al, 20h ;разрешение прерываний более низкого уровня
out 20h, al

iret
NewInt ENDP

LastIntByte:

Residency PROC
; Установка резидентности
mov dx, offset LastIntByte; размер в байтах от начала сегмента
mov cl, 4 ; перевод в параграфы
shr dx, cl
add dx, 16h
inc dx ;размер в параграфах

mov ah, 31h ;Резидентная программа активизируется каждый раз при возникновении
прерывания,
int 21h ; вектор которого эта программа изменила на адрес одной из своих процедур.
Residency ENDP

```

```

IsTailUnLoad PROC
; провер, есть ли /un, результат в ax
    cmp byte ptr es:[82h], '/'
        jne TailFalse
    cmp byte ptr es:[83h], 'u'
        jne TailFalse
    cmp byte ptr es:[84h], 'n'
        jne TailFalse

;проверка, нет ли каких-то еще букв после /un, ищем пробел или перевод строки
    cmp byte ptr es:[85h], 13
        je TailTrue
    cmp byte ptr es:[85h], ' '
        je TailTrue

TailFalse:
    mov ax, 0
    ret

TailTrue:
    mov ax, 1
    ret

IsTailUnLoad ENDP

```

```

IsIntLoad PROC
    push bx
    push si
    push es
    push dx
    mov si, offset Signature
    sub si, offset NewInt ;смещение до сигнатуры нашего обработчика

    mov ah, 35h
    mov al, 1Ch
    int 21h ;es:bx - адрес обработчика прерывания
    mov ax, es:[bx+si] ;сигнатура из установленного сейчас обработчика
    mov dx, Signature ; сигнатура нашего обработчика
    cmp ax, dx ;если совпали, значит загружен наш обработчик
        je IntIsLoad

IntIsntLoad:
    mov ax, 0
    jmp reter

IntIsLoad:
    mov ax, 1

reter:
    ;ax = 1 - сигнатуры совпали, иначе ax = 0
    pop dx
    pop es
    pop si
    pop bx

    ret
IsIntLoad ENDP

```

LoadInt PROC

```
    push ax
    push dx
    push bx
    push es

    mov ah, 35h ; получение вектора предыдущего прерывания
    mov al, 1Ch
    int 21h ;сохранение адреса обработчика прерывания
    mov KeepIP, bx ; запоминаем смещение
    mov KeepCS, es ; и сегмент

    push ds
    mov dx, offset NewInt ;ds:dx - адрес нашего обработчика прерывания
    mov ax, seg NewInt
    mov ds, ax
    mov ah, 25h ; установка нашего прерывания
    mov al, 1Ch ;прерывание от таймера
    int 21h
    pop ds

    pop es
    pop bx
    pop dx
    pop ax
    ret
```

LoadInt ENDP

UnLoadInt PROC

```
    push ax
    push bx
    push dx
    push si
    push es
    push ds
    ; Взятие смещения до сохраненных данных
    mov si, offset KeepCS
    sub si, offset NewInt ;смещение

    mov ah, 35h ;получение, сохранение адреса обработчика текущего прерывания
    mov al, 1Ch
    int 21h

    cli
    mov ax, es:[bx+si] ;KeepCS
    mov dx, es:[bx+si+2]; KeepIP
    mov ds, ax
    mov ah, 25h
    mov al, 1Ch
    int 21h ; воостанавливаем исходный вектор прерывания
    sti

    pop ds
    mov ax, es:[bx+si+4] ;KeepPSP
    mov es, ax
    push es
    mov ax, es:[2Ch]
    mov es, ax
```



```

        mov ah, 49h ;освобождаем память, занятую обработчиком, в es сег.адрес осв. блока
памяти
        int 21h

        pop es
        mov ah, 49h
        int 21h

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

MAIN PROC
        mov ax, DATA
        mov ds, ax
        mov KeepPSP, es
        call IsTailUnLoad ; проверка есть ли /un
        cmp ax, 1
            je UnLoad

Load: ;ключа нет - или загрузить обработчик или вывести сообщение
        call IsIntLoad
        cmp ax, 1
            je CantLoad

CanLoad:
        call LoadInt
        mov dx, offset InLoad
        mov ah, 09h
        int 21h
        call Residency

            jmp ext

CantLoad:
        mov dx, offset HLoad
        mov ah, 09h
        int 21h
            jmp ext

UnLoad: ;ключ есть - или выгрузить обработчик или вывести сообщение
        call IsIntLoad
        cmp ax, 0
            jne CanUnLoad

CantUnLoad:
        mov dx, offset HnLoad
        mov ah, 09h
        int 21h
            jmp ext

CanUnLoad:
        call UnLoadInt
        mov dx, offset OutLoad
        mov ah, 09h
        int 21h

```

```

        jmp ext

ext:
    mov ah, 4Ch
    int 21h

    MAIN ENDP
CODE ENDS

;важно, что сегменты стека и данных после сегмента кода
;тк иначе обработчик займет очень много памяти
ASTACK SEGMENT STACK
    DW 64 DUP(0)
ASTACK ENDS

DATA SEGMENT
    HLoad db 'Handler already load, unable to load again', 13, 10, '$'
    InLoad db 'Handler succsefully load', 13, 10, '$'
    OutLoad db 'Handler succsefully unload', 13, 10, '$'
    HnLoad db 'Handler not load yet, unable to unload', 13, 10, '$'
DATA ENDS

END MAIN

```