

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

Студентка гр. 8381

Преподаватель

Ивлева О.А.

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Построение обработчика прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определённые интервалы времени и, при возникновении такого сигнала, возникает прерывание с определённым значением вектора.

Теоретические сведения.

Резидентные обработчики прерываний - это программные модули, которые вызываются при возникновении прерываний определенного типа (сигнал таймера, нажатие клавиши и т.д.), которым соответствуют определенные вектора прерывания. Когда вызывается прерывание, процессор переключается на выполнение кода обработчика, а затем возвращается на выполнение прерванной программы. Адрес возврата в прерванную программу (CS:IP) запоминается в стеке вместе с регистром флагов. Затем в CS:IP загружается адрес точки входа программы обработки прерывания и начинает выполняться его код. Обработчик прерывания должен заканчиваться инструкцией IRET (возврат из прерывания).

Вектор прерывания имеет длину 4 байта. В первом хранится значение IP, во втором - CS. Младшие 1024 байта памяти содержат 256 векторов. Вектор для прерывания 0 начинается с ячейки 0000:0000, для прерывания 1 - с ячейки 0000:0004 и т.д.

Для установки написанного прерывания в поле векторов прерываний используется функция 25H прерывания 21H, которая устанавливает вектор прерывания на указанный адрес.

Программа, выгружающая обработчик прерываний должна восстанавливать оригинальные векторы прерываний. Функция 35 прерывания 21H позволяет восстановить значение вектора прерывания, помещая значение сегмента в ES, а смещение в BX.

Для того, чтобы оставить процедуру прерывания резидентной в памяти, следует воспользоваться функцией DOS 31h прерывания 21h. Эта функция

оставляет память, размер которой указывается в качестве параметра, занятой, а остальную память освобождает и осуществляет выход в DOS.

Выполнение работы.

Сборка и отладка производились на базе эмулятора DOSBox 0.74-3.

Был написан текст исходного .EXE модуля с именем lr4.exe. Описание процедур в программе представлено в табл. 1.

Таблица 1 – Описание процедур программы

Название	Назначение
WRITE	Вывод на экран строки, адрес которой содержится в DX
INTERRUPTION	Процедура обработчика прерывания.
INT_CHECK	Процедура проверки установки резидента INTERRUPTION
INT_LOAD	Процедура загрузки резидентной функции INTERRUPTION
INT_UNLOAD	Процедура загрузки резидентной функции INTERRUPTION (восстановление исходного обработчика прерывания системного таймера)
CL_CHECK	Процедура проверки параметра командной строки

Демонстрация работы.

Вывод программы представлен на рис. 1.

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Fram...
D:\>
D:\>cd
D:\>
D:\>cd
D:\>
D:\>cd
D:\>
D:\>cd
D:\>
D:\>lr4.exe
D:\>cd
D:\>
D:\>cd
D:\>
D:\>lr4.exe /un
D:\>_
```

Рисунок 1 – Выполнение lr4.exe

На рис. 2 видно, что процедура прерывания осталась резидентной в памяти и располагается в блоке 5.

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Fram...
D:\>lr31.com
Avl mem: 640 kb
Ext mem: 15360 kb
MCB num: 1
Block is MS DOS Area size: 16
MCB num: 2
Block is free Area size: 64
MCB num: 3
Block is 0040 Area size: 256
MCB num: 4
Block is 0192 Area size: 144
MCB num: 5
Block is 0192 Area size: 4480
LR4
MCB num: 6
Block is 02B5 Area size: 144
MCB num: 7
Block is 02B5 Area size: 644256
LR31
D:\>
```

Рисунок 2 – Выполнение lr31.com после запуска lr4.exe

При повторном запуске программы lr4.exe, программа вывела сообщение о том, что уже определен установленный обработчик прерывания, что показано на рис. 3.

```
DOS FOR DOSBox 0.74-3, Cpu speed: 3000 cycles, Fram...
MCB num: 1
Block is MS DOS Area size: 16

MCB num: 2
Block is free Area size: 64

MCB num: 3
Block is 0040 Area size: 256 437 interruptions

MCB num: 4
Block is 0192 Area size: 144 462 interruptions

MCB num: 5
Block is 0192 Area size: 4480
LR4
MCB num: 6
Block is 02B5 Area size: 144

MCB num: 7
Block is 02B5 Area size: 644256
LR31
D:\>lr4.exe
Int was already loaded
D:\>_
```

Рисунок 3 – Повторный запуск lr4.exe

Далее программа lr4.exe была запущена с параметром “/un” для выгрузки резидентного обработчика прерываний, а также была запущена программа lr31.com для вывода блоков MCB. Результат выполнения программы представлен на рис. 4. Видно, что память для резидентного обработчика была освобождена (ранее он занимал блок 5) и обработчик прерывания прекратил работу.

```
DOS FOR DOSBox 0.74-3, Cpu speed: 3000 cycles, Fram...
D:\>lr4.exe /un

D:\>cd
D:\>

D:\>lr31.com
Avl mem: 640 kb
Ext mem: 15360 kb
MCB num: 1
Block is MS DOS Area size: 16

MCB num: 2
Block is free Area size: 64

MCB num: 3
Block is 0040 Area size: 256

MCB num: 4
Block is 0192 Area size: 144

MCB num: 5
Block is 0192 Area size: 648912
LR31
D:\>_
```

Рисунок 4 – Выгрузка обработчика

Контрольные вопросы.

1. Как реализован механизм прерывания от часов?

Аппаратное прерывание int 8h от системного таймера срабатывает 18 раз в секунду. Изначально в системе еще при инициализации устанавливается свой обработчик для прерывания таймера, который каждый раз увеличивает на 1 текущее значение счетчика тиков таймера.

В конце этот обработчик прерывания вызывает прерывание int 1Ch – пользовательское прерывание по таймеру. После инициализации системы вектор INT 1Ch указывает на команду IRET. В программе вектор меняется на пользовательский обработчик, в котором на экран выводится счетчик вызовов прерывания системного таймера.

После выполнения обработчиков происходит возврат к коду, выполнение которого было прервано.

2. Какого типа прерывания использовались в программе?

1. Были использованы программные прерывания, например, int 21h.
2. Был реализован обработчик прерывания от таймера, котор является аппаратным.

Выводы.

В ходе выполнения лабораторной работы была реализована программа, загружающая и выгружающая пользовательское прерывание от системного таймера в память.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. LR4.ASM

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

INTERRUPTION    PROC    FAR
    jmp    START
INT_DATA:
    STR_COUNTER    DB    "000 interruptions"
    INT_CODE        DW    3158h

    KEEP_IP    DW    0
    KEEP_CS    DW    0
    KEEP_PSP    DW    0

START:
    push    AX
    push    BX
    push    CX
    push    DX
    push    SI
    push    ES
    push    DS
SET_STACK:
    mov     AX, seg STR_COUNTER
    mov     DS, AX

SET_CURSOR:
    mov     AH, 03h
    mov     BH, 0h
    int     10h
    push    DX

    mov     AH, 02h
    mov     BH, 0h
    mov     DX, 0920h
    int     10h

INCR:
    mov     AX, SEG STR_COUNTER
    push    DS
    mov     DS, AX
    mov     SI, offset STR_COUNTER
    add     SI, 2
    mov     CX, 3
```

CYCLE:

```
    mov     AH, [SI]
    inc     AH
    mov     [SI], AH
    cmp     AH, ':'
    jne     END_CYCLE
    mov     AH, '0'
    mov     [SI], AH
    dec     SI
    loop    CYCLE
```

END_CYCLE:

```
    pop     DS
```

PRINT:

```
    push    ES
    push    BP
    mov     AX, SEG STR_COUNTER
    mov     ES, AX
    mov     BP, offset STR_COUNTER
    mov     AH, 13h
    mov     AL, 1h
    mov     BL, 3h
    mov     BH, 0
    mov     CX, 17
    int     10h

    pop     BP
    pop     ES
```

```
    pop     DX
    mov     AH, 02h
    mov     BH, 0h
    int     10h

    pop     DS
    pop     ES
    pop     SI
    pop     DX
    pop     CX
    pop     BX
    pop     AX

    mov     AL, 20h
    out     20h, AL
    iret
```

INTERRUPTION ENDP

LAST_BYTE:


```

INT_CHECK      PROC
    push    AX
    push    BX
    push    SI
    mov     AH, 35h
    mov     AL, 1Ch
    int     21h
    mov     SI, offset INT_CODE
    sub     SI, offset INTERRUPTION
    mov     AX, ES:[BX + SI]
    cmp     AX, INT_CODE
    jne     INT_CHECK_END
    mov     INT_LOADED, 1
INT_CHECK_END:
    pop     SI
    pop     BX
    pop     AX
    ret
INT_CHECK      ENDP

INT_LOAD       PROC
    push    AX
    push    BX
    push    CX
    push    DX
    push    ES
    push    DS
    mov     AH, 35h
    mov     AL, 1Ch
    int     21h
    mov     KEEP_CS, ES
    mov     KEEP_IP, BX
    mov     AX, seg INTERRUPTION
    mov     DX, offset INTERRUPTION
    mov     DS, AX
    mov     AH, 25h
    mov     AL, 1Ch
    int     21h
    pop     DS
    mov     DX, offset LAST_BYTE
    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

INT_UNLOAD    PROC
    CLI
    push     AX
    push     BX
    push     DX
    push     DS
    push     ES
    push     SI
    mov      AH, 35h
    mov      AL, 1Ch
    int      21h
    mov      SI, offset KEEP_IP
    sub      SI, offset INTERRUPTION
    mov      DX, ES:[BX + SI]
    mov      AX, ES:[BX + SI + 2]
    push     DS
    mov      DS, AX
    mov      AH, 25h
    mov      AL, 1Ch
    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
INT_UNLOAD      ENDP

CL_CHECK      PROC
        push    AX
        push    ES
        mov     AX, KEEP_PSP
        mov     ES, AX
        cmp     byte ptr ES:[82h], '/'
        jne     CL_CHECK_END
        cmp     byte ptr ES:[83h], 'u'
        jne     CL_CHECK_END
        cmp     byte ptr ES:[84h], 'n'
        jne     CL_CHECK_END
        mov     UNLOAD_CL, 1
CL_CHECK_END:
        pop     ES
        pop     AX
        ret
CL_CHECK      ENDP

WRITE      PROC      NEAR
        push    AX
        mov     AH, 09h
        int     21h
        pop     AX
        ret
WRITE      ENDP

MAIN      PROC
        push    DS
        xor     AX, AX
        push    AX
        mov     AX, DATA
        mov     DS, AX
        mov     KEEP_PSP, ES
        call    INT_CHECK
        call    CL_CHECK
        cmp     UNLOAD_CL, 1
        je      UNLOAD
        mov     AL, INT_LOADED
        cmp     AL, 1
        jne     LOAD
        mov     DX, offset STR_WAS_LOADED
        call    WRITE
        jmp     MAIN_END
LOAD:

```

```

        call    INT_LOAD
        jmp     MAIN_END
UNLOAD:
        cmp     INT_LOADED, 1
        jne     NOT_EXIST
        call    INT_UNLOAD
        jmp     MAIN_END
NOT_EXIST:
        mov     DX, offset STR_NOT_LOADED
        call    WRITE
MAIN_END:
        xor     AL, AL
        mov     AH, 4Ch
        int     21h
MAIN    ENDP

CODE    ENDS

ASTACK  SEGMENT STACK
        DW 128 dup(0)
ASTACK  ENDS

DATA    SEGMENT
        STR_WAS_LOADED    DB  "Int was already loaded", 10, 13, "$"
        STR_NOT_LOADED    DB  "Int is not loaded", 10, 13, "$"
        INT_LOADED        DB  0
        UNLOAD_CL         DB  0
DATA    ENDS

END     MAIN

```