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

ОТЧЕТ

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

Студент гр. 8381 _____ Киреев К.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

В архитектуре компьютера существуют стандартные прерывания, за которыми закреплены определённые вектора прерываний. Вектор прерываний хранит адрес подпрограммы обработчика прерываний. При возникновении прерывания, аппаратура компьютера передаёт управление и выполняет соответствующие действия.

В лабораторной работе №4 предлагается построить обработчик прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определённые интервалы времени и, при возникновении такого сигнала, возникает прерывание с определённым значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

Основные теоретические положения.

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.

Обработчик прерывания — это отдельная процедура, имеющая следующую структуру:

```
ROUT PROC FAR
      PUSH AX; сохранение изменяемых регистров
      . . .
      <действия по обработке прерывания>
      . . .
      POP AX; восстановление регистров
      MOV AL, 20H
      OUT 20H, AL
      IRET
 ROUT ENDP
```

Две последние строки необходимы для разрешения обработки прерываний с более низкими уровнями, чем только что обработанное. Для установки написанного прерывания в поле векторов прерываний используется функция 25H прерывания 21H, которая устанавливает вектор прерывания на указанный адрес.

```
PUSH DS
MOV DX, OFFSET ROUT; смещение для процедуры в DX
MOV AX, SEG ROUT; сегмент процедуры
MOV DS, AX; помещаем в DS
MOV AH, 25H; функция установки вектора
MOV AL, 1CH; номер вектора
INT 21H; меняем прерывание
POP DS
```

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. Программа должна содержать следующие инструкции:

```
; -- хранится в обработчике прерываний
      KEEP_CS DW 0; для хранения сегмента
      KEEP_IP DW 0; и смещения прерывания
; -- в программе при загрузке обработчика
      прерывания
      MOV AH, 35H; функция получения вектора
      MOV AL, 1CH; номер вектора
      INT 21H
      MOV KEEP_IP, BX; запоминание смещения
      MOV KEEP_CS, ES; и сегмента
;  --  в  программе  при  выгрузке  обработчика
      прерываний
      CLI
      PUSH DS
      MOV DX, KEEP_IP
      MOV AX, KEEP_CS
      MOV DS, AX
      MOV AH, 25H
      MOV AL, 1CH
      INT 21H; восстанавливаем вектор
      POP DS
      STI
```

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.

Функция 31h прерывания 21h использует следующие параметры:

AH - номер функции 31h;

AL - код завершения программы;

```
DX - размер памяти в параграфах, требуемый резидентной программе.
Пример обращения к функции:
MOV DX, OFFSET LAST_BYTE; размер в байтах от начала сегмента
MOV CL, 4; перевод в параграфы
SHR DX, CL
INC DX; размер в параграфах
mov AH, 31h
int 21h
```

Выполнение работы.

Написан текст исходного EXE модуля, который выполняет некоторые функции. Проверяет, установлено ли пользовательское прерывание с вектором 1Ch. Устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и осуществляется выход о функции 4Ch прерывания int 21h. Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h. Выгрузка прерывания о соответствующему значению параметра в командной строке /un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождении памяти, занимаемой резидентом. Затем осуществляется выход по функции 4Ch прерывания int 21h.

Полученный исходный модуль был отлажен. Результаты выполнения программы представлены на рис. 1.

```
S:\>os4
Resident Interrupt Handler was loaded     No. of ints: 114
```

Рисунок 1 – Результат выполнения OS4.EXE

Работа прерывания отображается справа на экране. Необходимо было проверить размещение прерывания в памяти. Для этого была запущена программа OS3A.COM, которая отображает карту памяти в виде списка блоков MCB. Результат выполнения программы представлен на рис. 2.

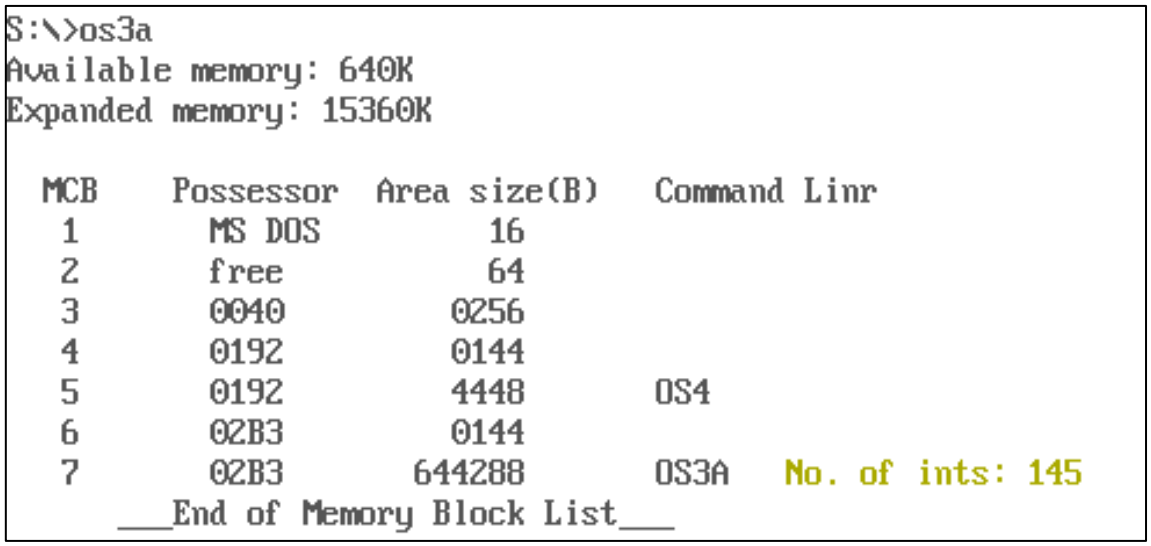

Рисунок 2 – Состояние памяти после загрузки прерывания

После повторного запуска программа определила установленный обработчик прерываний. Результат выполнения программы представлен на рис. 3.

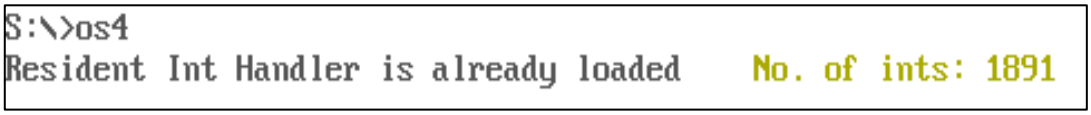

Рисунок 3 – Повторный запуск программы

Далее программа была запущена с ключом выгрузки, чтобы убедиться, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также была запущена программа OS3A.COM. Результаты выполнения программы представлен на рис. 4.

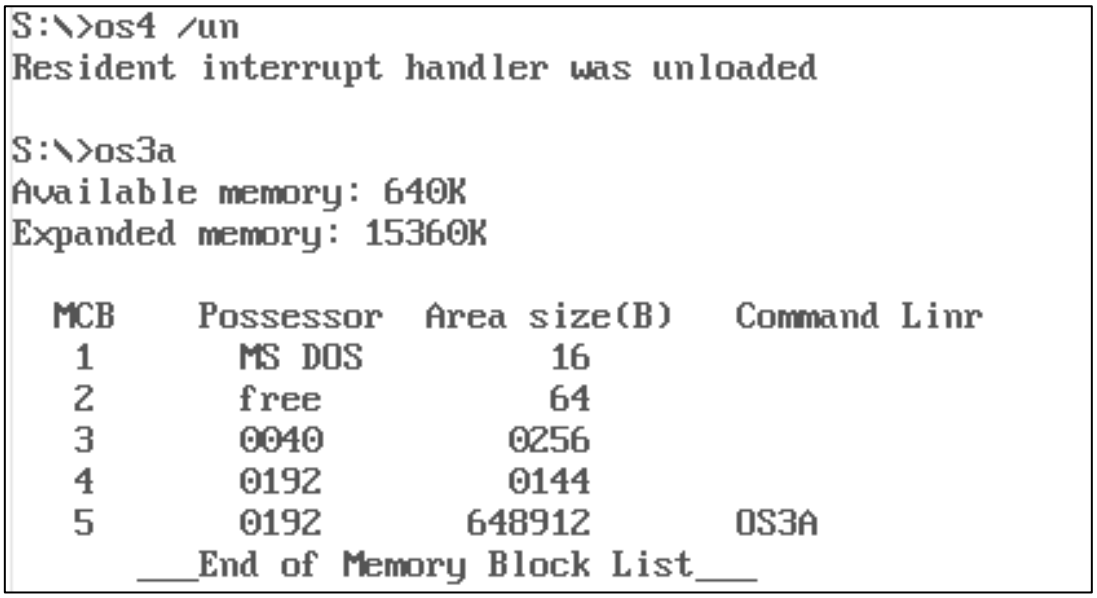

Рисунок 4 – Состояние памяти после выгрузки прерывания

Контрольные вопросы

- **Как реализован механизм прерывания от часов?**

Системный таймер вырабатывает прерывание INT 8h приблизительно 18,2 раза в секунду (точное значение - 1193180/65536 раз в секунду).

При инициализации BIOS устанавливает свой обработчик для прерывания таймера. Этот обработчик каждый раз увеличивает на 1 текущее значение четырехбайтовой переменной, располагающейся в области данных BIOS по адресу 0000:046Ch - счетчик тиков таймера. Если этот счетчик переполняется, в ячейку 0000:0470h заносится 1.

Последнее действие, которое выполняет обработчик прерывания таймера - вызов прерывания INT 1Ch. После инициализации системы вектор INT 1Ch указывает на команду IRET, т.е. ничего не выполняется. Программа может установить собственный обработчик этого прерывания, для того чтобы выполнять какие-либо периодические действия.

Необходимо отметить, что прерывание INT 1Ch вызывается обработчиком прерывания INT 8h до сброса контроллера прерывания, поэтому во время выполнения прерывания INT 1Ch все аппаратные прерывания запрещены. В частности, запрещены прерывания от клавиатуры.

Обработчик прерывания INT 1Ch должен заканчиваться командой IRET, выполняющей определенные действия - извлечение из стека сохраненных там слов и помещение их назад в регистры IP, CS и FLAGS. Это приводит к возврату в основную программу в ту самую точку, где она была прервана.

- **Какого типа прерывания использовались в работе?**
  - Аппаратные прерывания (INT 8H)
  - Прерывания функций BIOS для обслуживания аппаратуры компьютера (INT 10H)
  - Прерывания функций DOS (INT 21H)

Вывод.

Построен обработчик прерывания от сигналов таймера. Изучены дополнительные функции работы с памятью: установка программы-резидента и его выгрузка из памяти.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. OS4.ASM

```
Astack segment stack
    dw 12 dup(?)
Astack ends

data segment
    load_msg db 'Resident Interrupt Handler was loaded', 13,
10, '$'
     alrd_load_msg  db  'Resident  Int  Handler  is  already
loaded', 13, 10, '$'
    unload_msg db 'Resident interrupt handler was unloaded',
13, 10, '$'
data ends

code segment
    assume cs:code, ds:data, ss:Astack
rout proc far
    jmp rout_start
     signature  dw  4321h  ;сигнатура,  которая  идентифицирует
резидент
     keep_psp dw ?
    keep_ip dw ?
    keep_cs dw ?
    num dw 0
     mesto db 16 dup (0)
     int_count_msg db 'No. of ints:     $'
     rout_start:
    push ax
    push bx
    push cx
    push dx
    push si
    push di
    push es
    push ds
     mov ax, cs
```

```
     mov ds, ax ;для чисел
     mov es, ax ;для строки
     mov ax, num
     inc ax
     mov num, ax
     lea si, int_count_msg + 13
     mov bx, 10
     call word_to_str
     lea bp, int_count_msg
     call outputBP
     pop ds
     pop es
    pop di
     pop  si
     pop dx
     pop cx
     pop bx
     pop ax
    mov al, 20h
    out 20h, al
    iret
rout endp
last_byte:

word_to_str proc near
     ;на входе ax число 16 бит
     ;si указатель на строку
     ;bx разрядность результата
    push ax
     push bx
     push cx
     push dx
     push di
     push si
    cmp bx, 16
    ja end_wts
    cmp ax, 7FFFh
    jna plus
```

```
mov byte ptr [si], '-'
inc si
not ax
inc ax
plus:
 xor cx, cx
 jmp manipulation
manipulation:
 xor dx, dx
      div bx
      mov di, ax
      mov al, dl
      cmp al, 10
      sbb al, 69h
      das
      push di
      lea di, mesto
      add di, cx
      mov byte ptr [di], al
      pop di
      mov ax, di
inc cx
test ax, ax
jz endrep
jmp manipulation
endrep:
 lea di, mesto
      add di, cx
copyrep:
      dec di
      mov dl, byte ptr [di]
      mov byte ptr [si], dl
      inc si
      loop copyrep
end_wts:
pop si
 pop di
 pop dx
```

```
     pop cx
     pop bx
     pop ax
    ret
word_to_str endp

outputBP proc near ;вывод строки по адресу es:bp на экран
     push ax
     push bx
     push cx
     push dx
     mov ah, 13h ;функция вывода строки в bp
     mov al, 0 ;использовать атрибут в bl и не трогать курсор
     mov bl, 09h ;цвет
     ;0 = Черный      8 = Серый
    ;1 = Синий       9 = Светло-синий
    ;2 = Зеленый     A = Светло-зеленый
    ;3 = Голубой     B = Светло-голубой
    ;4 = Красный     C = Светло-красный
    ;5 = Лиловый     D = Светло-лиловый
    ;6 = Желтый      E = Светло-желтый
    ;7 = Белый       F = Ярко-белый
     mov bh, 0 ;номер страницы
     mov dh, 22 ;строка начала вывода
     mov dl, 42 ;колонка начала вывода
     mov cx, 17 ;длина строки
     int 10h
     pop dx
     pop cx
     pop bx
     pop ax
     ret
outputBP ENDP

print proc near
     push ax
     mov ah, 09h
     int 21h
```

```
     pop ax
     ret
print endp

rout_load proc near
    push ax
     push bx
     push cx
     push dx
    push es
     push ds
     mov ah, 62h
     int 21h
     push bx
     pop es ;в es PSP
     Interrupt_handler_load: ;загрузка обработчика прерывания
     mov ah, 35h ;функция получения вектора
     mov al, 1Ch ;номер вектора
     int 21h ;es:bx - адрес обработчика прерывания
     mov keep_cs, es ;запоминание сегмента
    mov keep_ip, bx ;запоминание смещения
     lea  dx, load_msg
     call print
    ;для функции 25h прерывания 21h
    ;al - номер прерывания
    ;ds:dx - адрес программы обработки прерывания
     lea dx, rout ;смещение процедуры
     mov ax, seg rout ;сегмент процедуры
     mov ds, ax
     mov ah, 25h ;функция установки вектора
     mov al, 1Ch ;номер вектора
     int 21h ;замена прерывания
     pop ds
    ;для функции 31h прерывания 21h
    ;al - код выхода
    ;dx - объем памяти, оставляемой резидентной, в параграфах
    ;выходит в родительский процесс, сохраняя код выхода в al
    ;DOS устанавливает начальное распределение памяти
```

```
    ;далее возвращает управление родительскому процессу, оставляя указанную память резидентной
    lea dx, last_byte ;размер в байтах от начала сегмента
     mov cl, 4 ;перевод в параграфы
     shr dx, cl
    add dx, 10Fh
     inc dx
     xor al, al ;0 - нормальное завершение
     mov ah, 31h
     int 21h
     mov ah, 4Ch
     int 21h
    pop es
    pop dx
     pop cx
     pop bx
     pop ax
     ret
rout_load endp

rout_unload proc near
    cli
     push ax
     push bx
     push cx
     push dx
     push ds
     push es
     push si
     push di
     mov ah, 62h
     int 21h
     push bx
     pop es ;в es PSP
     ;командная строка при запуске программы находится по адресу es:[80h]
     cmp byte ptr es:[82h], '/'
     jne alrd_load_rout
```

```
 cmp byte ptr es:[83h], 'u'
 jne alrd_load_rout
 cmp byte ptr es:[84h], 'n'
 jne alrd_load_rout
 lea  dx, unload_msg
 call print
 mov ah, 35h ;функция получения вектора
 mov al, 1Ch ;номер вектора
 int 21h ;es:bx - адрес обработчика прерывания
 push ds
 mov si, offset keep_ip
 sub si, offset rout ;si - смещение ip
 mov dx, es:[bx+si] ;адрес ip
 mov ax, es:[bx+si+2] ;адрес cs
 mov ds, ax
 mov ah, 25h
 mov al, 1ch
 int 21h
 pop ds
mov ax, es:[bx+si-2] ;адрес psp
 mov es, ax
 push es
 mov ax, es:[2ch] ;сегментный адрес среды
 mov es, ax
 ;DOS Function 49H: Освободить распределенный блок памяти
 mov ah, 49h
 int 21h
 pop es ;адрес psp
 mov ah, 49h
 int 21h
 jmp unload_ending
 alrd_load_rout:
 mov dx, offset alrd_load_msg
 call print
 unload_ending:
sti
 pop di
pop si
```

```
     pop es
     pop ds
     pop dx
     pop cx
     pop bx
     pop ax
     ret
rout_unload endp

main proc far
    push ds
    xor ax, ax
    push ax
    mov ax, DATA
    mov ds, ax
     mov  keep_psp, es
    mov ah, 35h ;функция получения вектора
     mov al, 1Ch ;номер вектора
     int 21h ;es:bx - адрес обработчика прерывания
     lea di, signature ;адрес, записанный в векторе прерывания
     sub di, offset rout ;di - смещение сигнатуры
     cmp ES:[bx+di], 4321h ;сравнение значения сигнатуры с
реальным кодом
     je rl ;если совпадают, то резидент установлен
     call rout_load ;иначе не установлен
     rl:  call rout_unload
     mov ax, 4C00h
     int 21h
     ret
main endp
code ends
end main
```