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

ЛАБОРАТОРНАЯ РАБОТА № 4

по дисциплине «Операционные системы»

ТЕМА: Обработка стандартных прерываний.

Студентка гр. 9381 _____ Москаленко Е.М.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

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 данной 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.

Ход работы.

1) Был написан и отлажен программный модуль типа .EXE, который выполняет следующие функции:

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 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 прерывания состоит в восстановлении стандартного вектора прерываний и освобождении памяти, занимаемой резидентом. Затем осуществляется выход по функции 4Ch прерывания int 21h.

2) Далее отлаженная программа была запущена. Можно убедиться, что резидентный обработчик прерывания 1Ch установлен. Работа прерывания

отображается на экране. Для проверки размещения прерывания в памяти была запущена программа лабораторной работы №3, которая отображает карту памяти в виде списка блоков МСВ.

3) Отлаженная программа была запущена еще раз. Убедилась, что она определяет установленный обработчик прерываний.

4) Отлаженная программа была запущена с ключом выгрузки. Убедилась, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также была запущена программа лабораторной работы №3.

Функции и структуры данных.

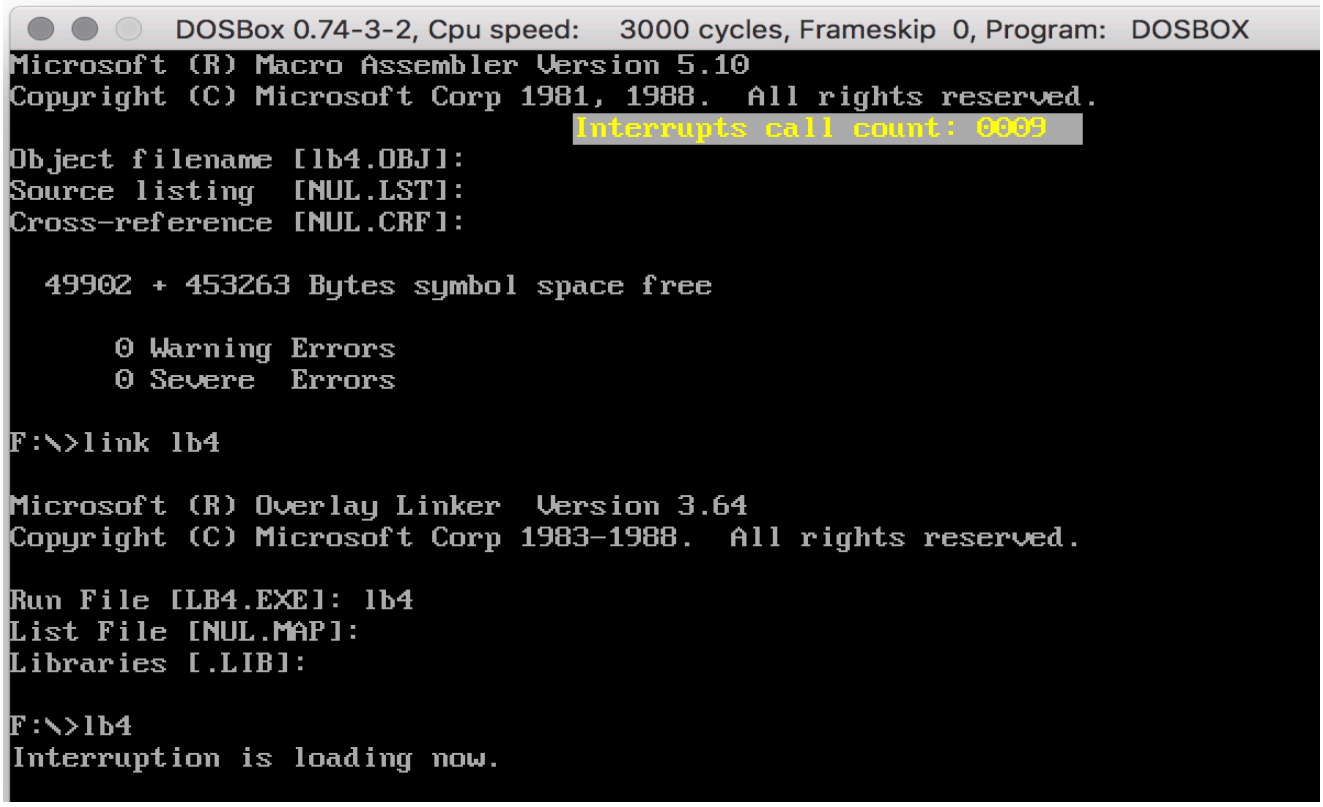
Названия переменных	**Описание**
PSP_1	Хранит старое значение ES до того, как программа была оставлена резидентной в памяти.
KEEP_CS	Переменная для хранения сегмента прерывания
KEEP_IP	Переменная для хранения смещения прерывания
INTERRUPTION_SET	Переменная для хранения количества вызванных прерываний
INT_COUNT	Строка «Interrupts call count:»
Not_loaded	Строка «Interruption not loaded»
Restored	Строка «Interruption was restored»
Loaded	Строка «Interruption is loaded»
Load_process	Строка «Interruption is loading now»

Названия функций	Описание
NEW_INTERRUPTION	Собственный обработчик прерывания. Выводит количество прерываний, которые были вызваны.
CHECK_SETTING	Функция проверяет, установлен ли разработанный вектор прерывания
CHECK_LOAD	Загрузка или выгрузка (проверка параметра \un)
LOAD_INTERRUPTION	Функция устанавливает новые обработчики прерывания, используя функцию 25h прерывания int 21h
UNLOAD_INTERRUPTION	Функция восстанавливает сохранённые заранее обработчики прерываний и выгружает резидентную программу
PRINT	Выводит строку на экран

Результат работы программы.

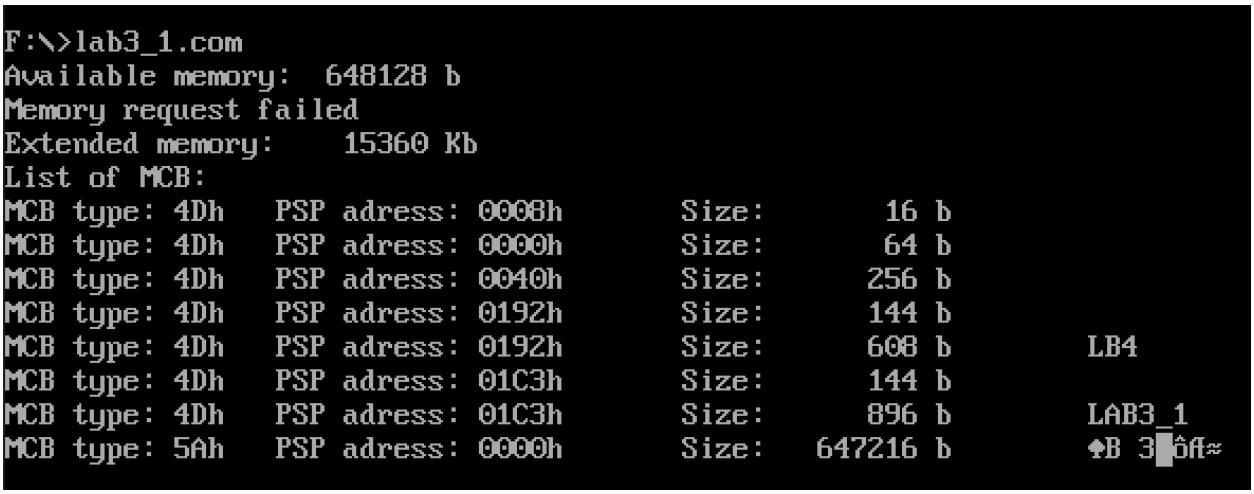
1) Был запущен модуль lb4.exe. На экране появился счётчик, в котором показано, сколько раз было вызвано прерывание. Вывелось сообщение о загрузке прерывания в память.

   На рисунке показан результат программы:



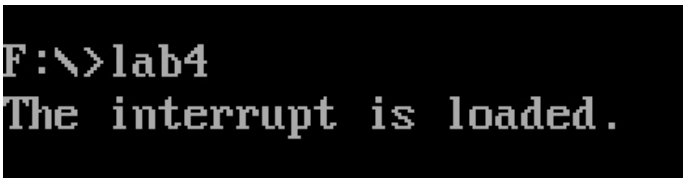
2) Чтобы проверить размещение прерывания в памяти, был запущен модуль 3-ей лабораторной работы lab3_1.com.

Как результат, MCB блок программы расположен 5-ым по порядку:

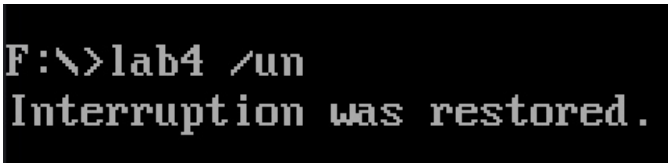


3) Далее вновь запустила lab4.exe. На экран вывелось сообщение о том, что прерывание уже находится в памяти.

Результат на рисунке:

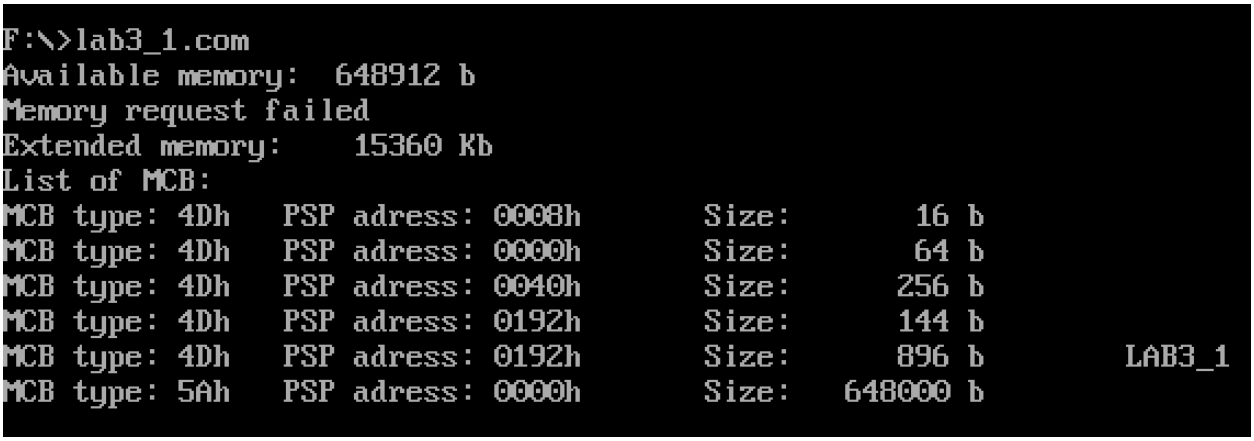


4) Чтобы выгрузить обработчик прерывания из памяти, модуль lab4.exe был запущен с параметром /un:



Модуль lab3_1.com был вновь запущен для проверки того, что обработчик прерывания выгружен и память, занятая резидентом, была освобождена.

На рисунке показан результат программы. Как видно, MCB блока программы lb4 нет в списке:



Выводы.

В ходе лабораторной работы была исследована обработка стандартных прерываний, а также построен обработчик прерываний сигналов таймера, которые генерируются аппаратурой через определённые интервалы времени. Программа загружает и выгружает резидент, а также производится проверка флагов и загрузки прерывание в память.

Ответы на контрольные вопросы:

1. Как реализован механизм прерывания от часов?

Аппаратное прерывание от таймера происходит каждые 55 мс.

Порядок действий:

1. Сохраняется содержимое регистров
2. Определяется источник прерывания (по номеру источника прерывания определяется смещение в таблице векторов прерываний)
3. Первые 2 байта помещаются в IP, второе 2 байта в CS
4. Передаётся управление по адресу CS:IP
5. Происходит обработка прерывания
6. возврат управления прерванной программе

2 .Какого типа прерывания использовались в работе?

Были использованы **аппаратные** (возникают как реакция микропроцессора на физический сигнал от некоторого устройства) прерывания: **int 1Ch**

А также **пользовательские** (вызываются искусственно с помощью соответствующей команды из программы (int), предназначены для выполнения некоторых действий операционной системы): **int 21h** (для работы с клавиатурой), **int 10h** (для работы с экраном)

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

Файл lb4.asm

```
ASSUME CS: CODE, DS: DATA, SS: SSTACK

SSTACK SEGMENT STACK
     DW 64 DUP(?)
SSTACK ENDS

DATA SEGMENT
     Not_loaded db "Interruption not loaded.", 0Dh, 0Ah, '$'
     Restored db "Interruption was restored.", 0Dh, 0Ah, '$'
     Loaded db "The interrupt is loaded.", 0Dh, 0Ah, '$'
     Load_process db "Interruption is loading now.", 0Dh, 0Ah, '$'
DATA ENDS

CODE SEGMENT

NEW_INTERRUPTION PROC FAR
     jmp START
     PSP_1 dw 0                                          ; 3
     PSP_2 dw 0                                          ; 5
     KEEP_CS dw 0                                  ; 7 segment storage
     KEEP_IP dw 0                                     ; 9  storing the
interrupt offset
     INTERRUPTION_SET dw 0FEDCh                             ; 11
     INT_COUNT db 'Interrupts call count: 0000  $' ; 13

START:
     push ax
     push bx
     push cx
     push dx

     mov ah, 3h            ; read the position of cursor
     mov bh, 0h
     int 10h                          ;  print  the  information  about
interrupt
     push dx

     mov ah, 2h
     mov bh, 0h
     mov dx, 220h
     int 10h

     push si
     push cx
     push ds
     mov ax, SEG INT_COUNT
     mov ds, ax
     lea si, INT_COUNT
     add si, 1Ah
```

```
      mov ah,[si]
      inc ah
      mov [si], ah
      cmp ah, 3Ah
      jne END_CALC
      mov ah, 30h
      mov [si], ah

      mov bh, [si-1]
      inc bh
      mov [si-1], bh
      cmp bh, 3Ah
      jne END_CALC
      mov bh, 30h
      mov [si-1], bh

      mov ch, [si-2]
      inc ch
      mov [si-2], ch
      cmp ch, 3Ah
      jne END_CALC
      mov ch, 30h
      mov [si-2], ch

      mov dh, [si-3]
      inc dh
      mov [si-3], dh
      cmp dh, 3Ah
      jne END_CALC
      mov dh, 30h
      mov [si-3],dh

END_CALC:
  pop ds
  pop cx
      pop si

      push es
      push bp
      mov ax, SEG INT_COUNT
      mov es, ax
      lea ax, INT_COUNT
      mov bp, ax
      mov ah, 13h
      mov al, 0h
      mov cx, 1Dh
      mov bh, 0
      int 10h

      pop bp
      pop es
      pop dx
      mov ah, 2h
      mov bh, 0h
      int 10h

      pop dx
```

```
      pop cx
      pop bx
      pop ax
      iret
NEW_INTERRUPTION ENDP

NEED_MEM_AREA PROC
NEED_MEM_AREA ENDP

; check whether the interrupt vector is set
CHECK_SETTING PROC NEAR
      push bx
      push dx
      push es

      mov ah, 35h
      mov al, 1Ch
      int 21h

      mov dx, es:[bx + 11]
      cmp dx, 0FEDCh
      je INT_IS_SET
      mov al, 0h
      jmp POP_REG

INT_IS_SET:
      mov al, 01h
      jmp POP_REG

POP_REG:
      pop es
      pop dx
      pop bx

      ret
CHECK_SETTING ENDP

; load or unload (checking  \un)
CHECK_LOAD PROC NEAR
      push es

      mov ax, PSP_1
      mov es, ax

      mov bx, 82h

      mov al, es:[bx]
      inc bx
      cmp al, '/'
      jne NULL_CMD

      mov al, es:[bx]
      inc bx
      cmp al, 'u'
      jne NULL_CMD

      mov al, es:[bx]
```

```
      inc bx
      cmp al, 'n'
      jne NULL_CMD

      mov al, 0001h
NULL_CMD:
      pop es

      ret
CHECK_LOAD ENDP

LOAD_INTERRUPTION PROC NEAR  ;loading new interrupt handlers
  push ax
      push bx
      push dx
      push es

      mov ah, 35h
      mov al, 1Ch
      int 21h

      mov KEEP_IP, bx
      mov KEEP_CS, es

      push ds
      lea dx, NEW_INTERRUPTION     ;offset for the procedure in dx
      mov ax, seg NEW_INTERRUPTION ; segment of the procedure
      mov ds, ax

      mov ah, 25h                                                    ;
setting the vector
      mov al, 1Ch                                                    ;
number of the vector
      int 21h                                                              ;
changing the interrupt
      pop ds

      lea dx, Load_process
      call PRINT

      pop es
      pop dx
      pop bx
      pop ax

      ret
LOAD_INTERRUPTION ENDP

UNLOAD_INTERRUPTION PROC NEAR
      push ax
      push bx
      push dx
      push es

      mov ah, 35h
      mov al, 1Ch
      int 21h
```

```
      cli
      push ds
      mov dx, es:[bx + 9]
      mov ax, es:[bx + 7]
      mov ds, ax
      mov ah, 25h
      mov al, 1Ch
      int 21h
      pop ds
      sti
      lea dx, Restored
      call PRINT

      push es
      mov cx, es:[bx + 3]
      mov es, cx
      mov ah, 49h
      int 21h
      pop es

      mov cx, es:[bx + 5]
      mov es, cx
      int 21h

      pop es
      pop dx
      pop bx
      pop ax

      ret
UNLOAD_INTERRUPTION ENDP

PRINT PROC NEAR            ;write string
      push ax
      mov ah, 9h
      int  21h
      pop ax
      ret
PRINT ENDP

MAIN_PR PROC FAR
      mov bx, 2Ch
      mov ax, [bx]
      mov PSP_2, ax
      mov PSP_1, ds
      sub ax, ax
      xor bx, bx

      mov ax, DATA
      mov ds, ax

      call CHECK_LOAD   ;load or unload function (checking parameter)
      cmp al, 1h
      je UNLOAD_START

      call CHECK_SETTING   ;checking vector
```

```
      cmp al, 1h
      jne INT_NOT_LOADED

      lea dx, Loaded  ; vector is set
      call PRINT
      jmp EXIT

      mov ah,4Ch
      int 21h

INT_NOT_LOADED:
      call LOAD_INTERRUPTION

      lea dx, NEED_MEM_AREA
      mov cl, 4h            ; to the paragraphs
      shr dx, cl
      add dx, 1Bh

      mov ax, 3100h              ; leave the function resident in memory
      int 21h

UNLOAD_START:
      call CHECK_SETTING
      cmp al, 0h
      je NOT_SET
      call UNLOAD_INTERRUPTION
      jmp EXIT

NOT_SET:
      lea dx, Not_loaded
      call PRINT
  jmp EXIT

EXIT:
      mov ah, 4Ch
      int 21h
MAIN_PR ENDP

CODE ENDS

END MAIN_PR
```