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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Операционные системы»

Тема: Обработка стандартных прерываний

Студент гр. 7383 _____ Медведев И.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2019

Цель работы

Построить обработчик прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определенные интервалы времени и, при возникновении такого сигнала, возникает прерывание с определенным значением вектора.

Ход работы

В ходе работы были написаны некоторые функции, которые описаны в табл. 1.

Таблица 1 – Используемые функции

Название функции	Выполняемая задача
setCurs	Установка позиции курсора.
getCurs	Получение позиции и размера курсора.
printSTR	Вывод сообщения на экран.
ROUT	Функция - обработчик прерывания. Функция считает общее число прерываний и выводит на экран полученное значение. Если значение флага равно 1, то функция восстанавливает стандартное прерывание и выгружается из памяти.
SET_INTERRUPT	Установка пользовательского прерывания.
BASE_FUNC	Функция проверяет было ли установлено пользовательское прерывание.

Таблица 2 – Структура данных управляющей программы

Имя	Тип	Назначение
message_1	db	Вывод строки 'Resident was loaded'
message_2	db	Вывод строки 'Resident has already been loaded'
message_3	db	Вывод строки 'Resident was unloaded'

Таблица 3 – Структура данных собственного прерывания.

Имя	Тип	Назаначение
identifier	db	Идентификатор пользовательского прерывания.
KEEP_IP	dw	Переменная для сохранения значения регистра IP.
KEEP_CS	dw	Переменная для сохранения значения регистра CS.
KEEP_PSP	dw	Переменная для сохранения значения регистра PSP
flag	dw	Флаг для определения необходимости выгружать прерывание.
KEEP_SS	dw	Переменная для сохранения значения регистра SS.
KEEP_AX	dw	Переменная для сохранения значения регистра AX.
KEEP_SP	dw	Переменная для сохранения значения регистра SP
COUNT	dw	Число вызовов собственного прерывания.

Также в ходе работы был создан .EXE файл, который 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 по функции 2Ch прерывания int 21h, если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h, также осуществляет выгрузку прерывания по значению параметра в командной строке /un. Примеры работы программы представлены на рис 1-4.

Рисунок 1 – Запуск 4-ой лабораторной работы.

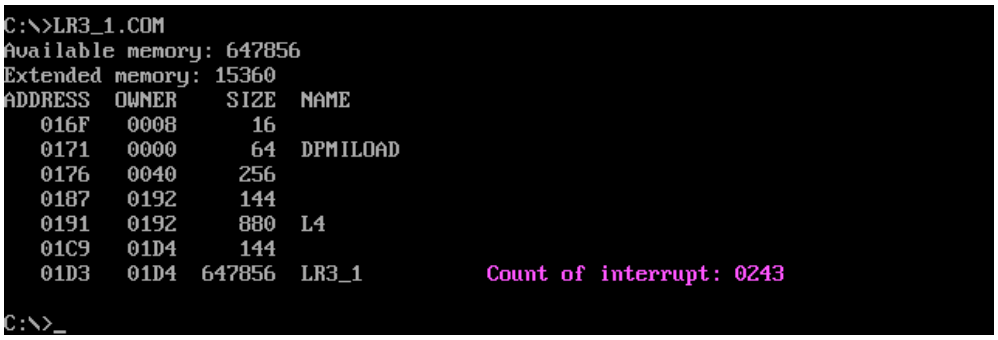

Рисунок 2 – Запуск 3-ей лабораторной работы, после запуска 4-ой.

Рисунок 3 – Повторный запуск 4-ой лабораторной работы без ключа выгрузки.

Рисунок 4 – Запуск 4-ой лабораторной работы с ключом выгрузки.

Вывод

В результате выполнения лабораторной работы был создан обработчик прерываний сигналов таймера, изучены дополнительные функции работы с памятью, такие как установка программы-резидента и его выгрузка из памяти.

Ответы на контрольные вопросы

1) Как реализован механизм прерывания от часов?

Прерывание по таймеру вызывается автоматически по каждому тику аппаратных часов (каждые 55мс). После вызова, сохраняется содержимое регистров, затем определяется источник прерывания, по номеру которого определяется смещение в таблице векторов прерываний. Полученный адрес сохраняется в регистры CS:IP, после чего управление передается по этому адресу, т.е. выполняется запуск обработчика прерываний и происходит его выполнение. После выполнения, происходит возврат управления прерванной программе.

2) Какого типа прерывания использовались в работе?

В данной лабораторной работе использовались аппаратные прерывания (int 1Ch), программные (int 21h, int 10h).

ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ

```
CODE SEGMENT
      ASSUME CS:CODE, DS:DATA, ES:DATA, SS:MY_STACK

setCurs PROC
      push ax
      push bx
      push cx
      mov ah,02h
      mov bh,00h
      int 10h
      pop cx
      pop bx
      pop ax
      ret
setCurs ENDP

getCurs PROC
      push ax
      push bx
      push cx
      mov ah,03h
      mov bh,00h
      int 10h
      pop cx
      pop bx
      pop ax
      ret
getCurs ENDP

printSTR PROC
      push es
      push bp
      mov ax,SEG COUNT
      mov es,ax
      mov ax,offset COUNT
      mov bp,ax
      mov ah,13h
      mov al,00h
      mov cx,25
      mov bh,0
      mov bl, 13
      int 10h
      pop bp
      pop es
      ret
printSTR ENDP

ROUT PROC FAR
      jmp ROUT_START

      ;DATA
      identifier db '0000'
      KEEP_IP dw 0
```

```
      KEEP_CS dw 0
      KEEP_PSP dw 0
      flag db 0
      KEEP_SS dw 0
      KEEP_AX dw 0
      KEEP_SP dw 0
      COUNT db 'Count of interrupt: 0000 $'
      inter_stack dw 64 dup (?)
      end_stack dw 0

ROUT_COUNT:
      push si
      push cx
      push ds
      mov ax,SEG COUNT
      mov ds,ax
      mov si,offset COUNT
      add si, 23
      mov ah,[si]
      add ah,1
      mov [si],ah
      cmp ah,58
      jne END_COUNT
      mov ah,48
      mov [si],ah
      mov bh,[si-1]
      add bh,1
      mov [si-1],bh
      cmp bh,58
      jne END_COUNT
      mov bh,48
      mov [si-1],bh
      mov ch,[si-2]
      add ch,1
      mov [si-2],ch
      cmp ch,58
      jne END_COUNT
      mov ch,48
      mov [si-2],ch
      mov dh,[si-3]
      add dh, 1
      mov [si-3],dh
      cmp dh,58
      jne END_COUNT
      mov dh,48
      mov [si-3],dh

END_COUNT:
    pop ds
    pop cx
      pop si
      call printSTR
      pop dx
      call setCurs
      jmp END_ROUT
```

```
ROUT_START:
      mov KEEP_AX, ax
      mov KEEP_SS, ss
      mov KEEP_SP, sp
      mov ax, cs
      mov ss, ax
      mov sp, offset end_stack
      mov ax, KEEP_AX
      push dx
      push ds
      push es
      cmp flag, 1
      je ROUT_REC
      call getCurs
      push dx
      mov dh,22
      mov dl,39
      call setCurs
      jmp ROUT_COUNT

ROUT_REC:
      CLI
      mov dx,KEEP_IP
      mov ax,KEEP_CS
      mov ds,ax
      mov ah,25h
      mov al,1Ch
      int 21h
      mov es, KEEP_PSP
      mov es, es:[2Ch]
      mov ah, 49h
      int 21h
      mov es, KEEP_PSP
      mov ah, 49h
      int 21h
      STI

END_ROUT:
      pop es
      pop ds
      pop dx
      mov ss, KEEP_SS
      mov sp, KEEP_SP
      mov ax, KEEP_AX
      iret
ROUT ENDP

SET_INTERRUPT PROC
      push dx
      push ds
      mov ah,35h
      mov al,1Ch
      int 21h
      mov KEEP_IP,bx
```

```
      mov KEEP_CS,es
      mov dx,offset ROUT
      mov ax,seg ROUT
      mov ds,ax
      mov ah,25h
      mov al,1Ch
      int 21h
      pop ds
      mov dx,offset message_1
      call PRINT
      pop dx
      ret
SET_INTERRUPT ENDP

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

      mov si, offset identifier
      sub si, offset ROUT

      mov ax,'00'
      cmp ax,es:[bx+si]
      jne NOT_LOADED
      cmp ax,es:[bx+si+2]
      jne NOT_LOADED
      jmp LOADED

NOT_LOADED:
      call SET_INTERRUPT
      mov dx,offset LAST_BYTE
      mov cl,4
      shr dx,cl
      inc dx
      add dx,CODE
      sub dx,KEEP_PSP
      xor al,al
      mov ah,31h
      int 21h

LOADED:
      push es
      push ax
      mov ax,KEEP_PSP
      mov es,ax
      mov al, es:[81h+1]
      cmp al,'/'
      jne NOT_UNLOAD
      mov al, es:[81h+2]
      cmp al,'u'
      jne NOT_UNLOAD
      mov al, es:[81h+3]
      cmp al,'n'
      je UNLOAD
```

```
NOT_UNLOAD:
      pop ax
      pop es
      mov dx,offset message_2
      call PRINT
      ret

UNLOAD:
      pop ax
      pop es
      mov byte ptr es:[bx+si+10],1
      mov dx,offset message_3
      call PRINT
      ret
BASE_FUNC ENDP

PRINT PROC NEAR
      push ax
      mov ah, 09h
      int 21h
      pop ax
      ret
PRINT ENDP

MAIN PROC Far
      mov ax,DATA
      mov ds,ax
      mov KEEP_PSP,es
      call BASE_FUNC
      xor al,al
      mov ah,4Ch
      int 21H
LAST_BYTE:
      MAIN ENDP
      CODE ENDS

MY_STACK SEGMENT STACK
      DW 64 DUP (?)
MY_STACK ENDS

DATA SEGMENT
      message_1 db 'Resident was loaded', 13, 10, '$'
      message_2 db 'Resident has already been loaded', 13, 10, '$'
      message_3 db 'Resident was unloaded', 13, 10, '$'
DATA ENDS

      END MAIN
```