Тема 9.
Работа с таймером

Таймер

Таймер в Windows является устройством ввода информации, которое периодически извещает приложение о том, что истек заданный интервал времени.

Таймер в Windows является относительно простым расширением таймерной логики, встроенной в аппаратуру PC и ROM BIOS. ROM BIOS компьютера инициализирует микросхему таймера так, чтобы она генерировала аппаратное прерывание. Это прерывание иногда называют "тиком таймера". Эти прерывания генерируются каждые 54,925 миллисекунды или примерно 18,2 раза в секунду.

Таймер

В программах, сделанных для Windows, Windows сама обрабатывает аппаратные прерывания и приложения их не получают.

Для каждой программы, где в данный момент установлен таймер, Windows обрабатывает таймерное прерывание путем уменьшения на 1 значения счетчика.

Когда это значение становится равным 0, Windows помещает сообщение WM_TIMER в очередь сообщений соответствующего приложения и восстанавливает начальное значение счетчика.

Приложение Windows при использовании простого таймера не сможет получать сообщения WM_TIMER в темпе, превышающем 18,2 раза в секунду.

Таймерные сообщения не являются асинхронными!

Основы использования таймера

- Многозадачность.
- Поддержка обновления информации о состоянии.
- Реализация "автосохранения».
- Завершение демонстрационных версий программ.
- Задание темпа изменения в играх или обучающих программах.
- Мультимедиа.

Функции Win32 API для работы с таймером

1) Создание таймера, посылающего сообщения с заданным интервалом:

```
UINT SetTimer(
    HWND hWnd,      // дескриптор окна, которому будут
                   //посылаться сообщения таймера
    UINT nIDEvent, // идентификатор таймера
    UINT uElapse, // временной интервал
    // ( от 1 до 4 294 967 295 миллисекунд)
    TIMERPROC lpTimerFunc // указатель на функцию,
                          //которая обрабатывает
                          //сообщение WM_TIMER
);
```

Функции Win32 API для работы с таймером

2) Удаление таймера:

```
BOOL KillTimer(
    HWND hWnd,      // дескриптор окна,
                    // ассоциированного с таймером,
                    //совпадающий по значению с
                    // соответствующим параметром
                    // функции  SetTimer
    UINT nIDEvent, // идентификатор таймера
);
```

Использование таймера

Если нужен таймер для измерения продолжительности работы программы, то вызывается **SetTimer** из функции **WinMain** или при обработке сообщения **WM_CREATE**, а **KillTimer** - в ответ на сообщение **WM_DESTROY**.

Установка таймера в функции **WinMain** обеспечивает простейшую обработку ошибки, если таймер недоступен:

```
if (!SetTimer(hwnd, 1, 1000, NULL))
{
        MessageBox(hwnd,
                "Too many clocks or timers!",
                "Program Name",
                MB_ICONEXCLAMATION | MB_OK);
        return FALSE;
}
```

Использование таймера

Если оконная процедура получает сообщение **WM_TIMER**, значение **wParam** равно значению идентификатора таймера (который равен 1 в приведенном примере), а **lParam** равно 0.

Если нужно более одного таймера, то для каждого таймера используется свой идентификатор:

```
#define TIMER_SEC 1
#define TIMER_MIN 2
```

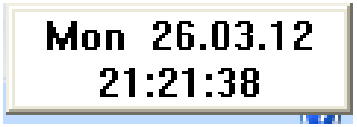
Использование таймера

Значение параметра **wParam** позволит различать передаваемые в оконную процедуру сообщения **WM_TIMER**.

Логика обработки сообщения:

```
case WM_TIMER:
      switch (wParam)
      {
      case TIMER_SEC:
            [обработка одного сообщения в секунду]
            break;
      case TIMER_MIN:
            [обработка одного сообщения в минуту]
            break;
      }
      return 0;
```

Использование таймера для часов




```
struct tm
{
  int    tm_sec;    // секунды
  int    tm_min;    // минуты
  int    tm_hour;   // часы (0...23)
  int    tm_mday;   // день месяца (1...31)
  int    tm_mon;    // месяц (0...11)
  int    tm_year;   // год (календарный год минус 1900)
  int    tm_wday;   // номер дня недели
                    //   (0...6, 0 - воскресенье)
  int    tm_yday;   // день года (0...365)
  int    tm_isdst;  // флаг летнего времени (0 - летнее
                    // время не используется)
};
```

Использование таймера для часов

```
#include <time.h>

void WndPaint (HWND hwnd, HDC hdc)
{
...
 struct tm  *datetime ;
 time_t      lTime ;

 time (&lTime) ;
 datetime = localtime (&lTime);
...
}
```

Использование таймера для часов

WndProc

```
switch (iMsg)
     {
          case WM_CREATE :
               SetInternational () ;
               return 0 ;

          case WM_TIMER :
               InvalidateRect (hwnd, NULL, FALSE) ;
               return 0 ;
```

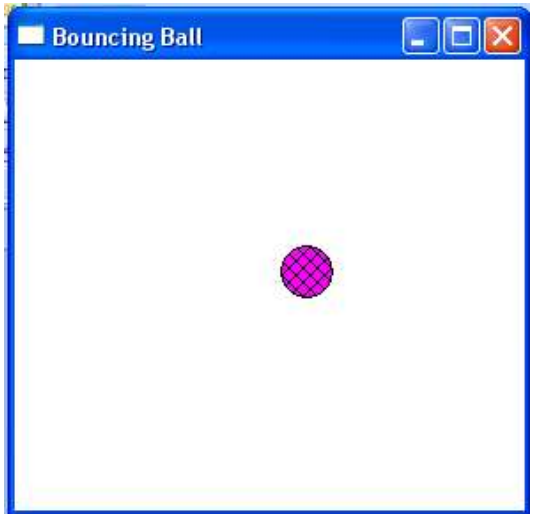
Использование таймера для часов

```
        case WM_PAINT :
             hdc = BeginPaint (hwnd, &ps) ;
             WndPaint (hwnd, hdc) ;
             EndPaint (hwnd, &ps) ;
             return 0 ;

        case WM_WININICHANGE :
             SetInternational () ;
             InvalidateRect (hwnd, NULL, TRUE) ;
             return 0 ;

        case WM_DESTROY :
             KillTimer (hwnd, ID_TIMER) ;
             PostQuitMessage (0) ;
             return 0 ;
}
```

Программирование анимации



1) Создание таймера в **`WinMain`**

2) Обработка сообщения **`WM_TIMER`**:
Мяч является битовым образом.
Сначала создается мяч путем
создания битового образа,
который выбирается в контекст памяти,
а затем вызываются простые функции GDI.

Растровый мяч рисуется на экране путем передачи
блока битов из другого контекста памяти.

3) Уничтожение таймера в **`WM_DESTROY`**

Программирование анимации

```
case WM_LBUTTONDOWN:
      if (bTimer)
            KillTimer(hwnd, TIMER_ID);
      else
            SetTimer(hwnd, TIMER_ID, TIMER_SPEED, NULL);
      bTimer=!bTimer;
      SayCaption(hwnd);
      break;
```


```
case WM_TIMER:
      if (bTimer) {
            // пересчет координат
            MoveKitten(drawDC);
      }
      break;
```

Программирование анимации

```
case WM_PAINT:
      DrawKitten(drawDC);
      return DefWindowProc (hwnd, message,
                            wParam, lParam);

case WM_DESTROY:
      if (bTimer)
            KillTimer(hwnd, TIMER_ID);
      ...
      PostQuitMessage (0);
      break;
```

Спасибо за внимание!