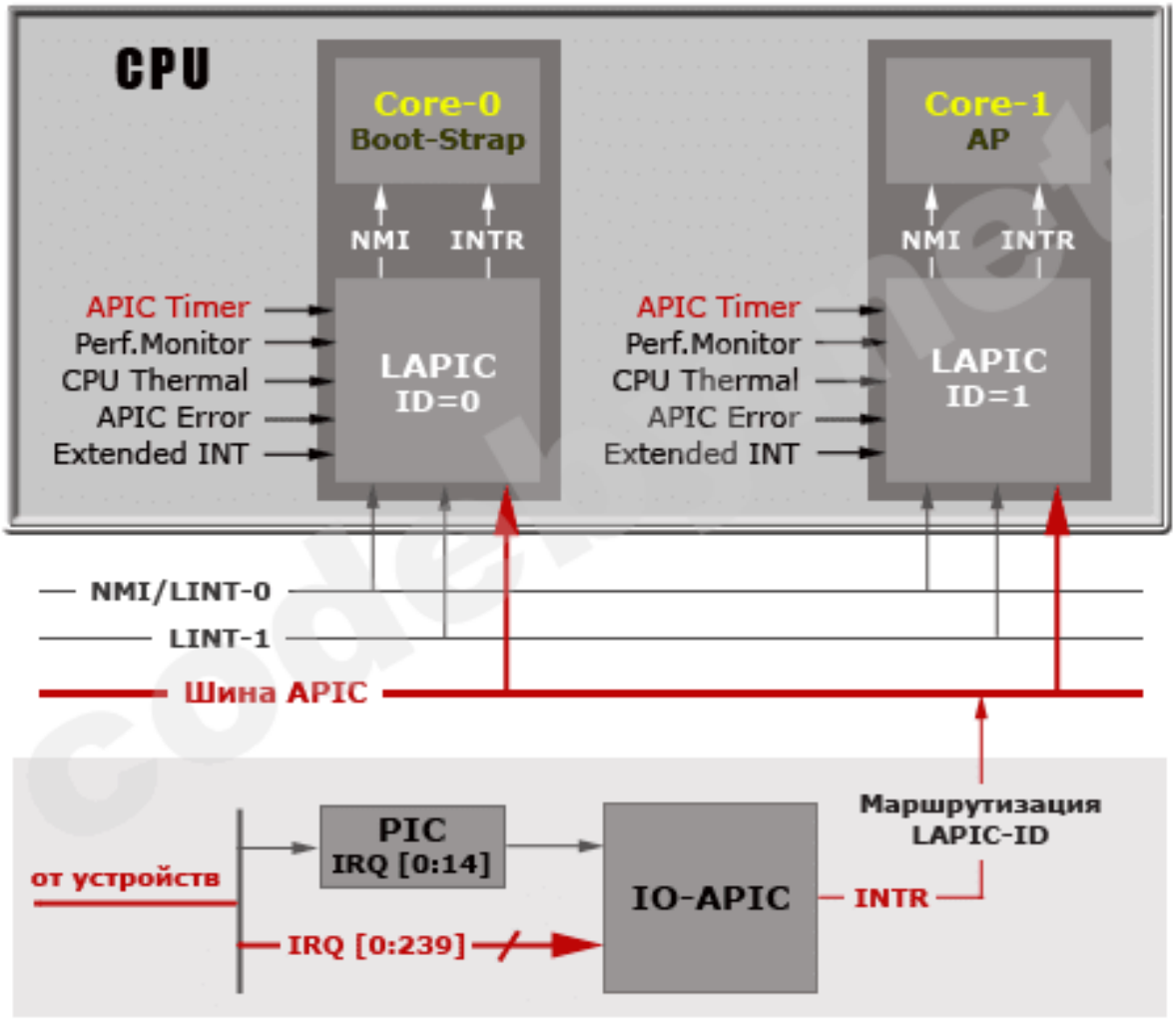
Работа с таймерами

Архитектурные основы

Аппаратный таймер

1. Генерация сигнала
2. Обработка прерывания
3. Работа планировщика
4. Переключение контекста

Архитектура контролёра прерываний *„APIC"*



Проблема точности

Windows — не ОС реального времени, и вызов таймера может быть отложен из-за высокой нагрузки на систему или низкого приоритета сообщения.

Многие разработчики ожидают, что Sleep(10) проснется ровно через 10 мс, но в Windows это почти никогда не так

Проблема точности

1. Дискретность системного таймера
2. Отсутствие Real-Time гарантий
3. Очередь сообщений

Виды таймеров

Счетчик

Инкрементируется при каждом прерывании таймера. Вызов этой функции не приостанавливает программу, возвращает время прошедшее со старта. (логирование, профилирование).

Пример

```
using System.Diagnostics;

// Запуск

Stopwatch sw = Stopwatch.StartNew();

// Выполняем какой-то код

DoSomeHeavyWork();

sw.Stop();

// Выводим результат

Console.WriteLine($"Прошло времени: {sw.ElapsedMilliseconds} мс");

Console.WriteLine($"Точный замер в тиках: {sw.ElapsedTicks}");
```

Объекты ожидания

Это инструменты синхронизации. При вызове Sleep(100), планировщик убирает поток из очереди на выполнение и не возвращает его туда, пока не пройдет 100 мс. Поток засыпает и не потребляет ресурсы CPU.

Пример

```
using System.Threading;


Console.WriteLine("Начало паузы...");

Thread.Sleep(1000); // Поток замирает на 1 секунду и ничего не делает

Console.WriteLine("Проснулся!");
```

Пример

```
using System.Threading.Tasks;


public async Task MyMethodAsync()
{
    Console.WriteLine("Начало ожидания...");
    await Task.Delay(1000); // Поток освобождается для других дел на 1 секунду
    Console.WriteLine("Время вышло!");
}
```

Регулярное выполнение

Можно постоянннно вызывать какой-то метод на выполнение по истечении времени.

Пример

```
// Колбэк, который будет вызываться
TimerCallback tm = new TimerCallback(DoWork);

// Создаем таймер
Timer timer = new Timer(tm, null, 0, 5000);

void DoWork(object? obj)
{
    Console.WriteLine("Таймер сработал!");
}
```

Таймеры в .NET C#

System.Windows.Forms.Timer

Однопоточный, работает в UI-потоке. Безопасен для изменения контролов, но тормозит при тяжелых вычислениях. Имеет низкую точность.

System.Threading.Timer

Самый легковесный. Выполняется в потоке из пула ThreadPool. Не блокирует основной поток приложения — подходит для фоновых задач. Нельзя напрямую менять UI-элементы из него. Нужна синхронизация через Invoke.

Пример

```
static void Main(string[] args) {
    var timer = new Timer(OnTimerElapsed, null, 0, 1000); // таймер на 1 секунду
    Console.WriteLine("Press any key to stop the timer.");
    Console.ReadKey();
    timer.Dispose();
}
static void OnTimerElapsed(object state) {
    Console.WriteLine("Timer elapsed at {0}", DateTime.Now);
}
```

Пример

```
public MainWindow(){
        InitializeComponent();
        _timer = new System.Threading.Timer(Tick, null, 0, 1000);
    }
    private void Tick(object state){
        string time = DateTime.Now.ToLongTimeString();
        // Используем диспетчер текущего окна
        this.Dispatcher.Invoke(() =>
        {
            myTextBlock.Text = $"Обновлено из фона: {time}";
        });
    }
```

System.Timers.Timer

Оболочка над Threading.Timer с серверным уклоном. Удобнее для использования в компонентах. Имеет свойство SynchronizingObject. Если его установить (например, передать туда форму), таймер будет сам «пробрасывать» вызовы в UI-поток. Если оставить null — работает как обычный потоковый таймер. Вместо колбэков здесь используется привычное событие Elapsed.

Пример

```
static void Main(string[] args)
{
    var timer = new System.Timers.Timer(1000); // таймер на 1 секунду
    timer.Elapsed += OnTimerElapsed;
    timer.AutoReset = true; // таймер будет срабатывать каждую секунду
    timer.Enabled = true; // запустить таймер
    Console.WriteLine("Press any key to stop the timer.");
    Console.ReadKey();
    timer.Stop();
}
static void OnTimerElapsed(object sender, ElapsedEventArgs e)
{
    Console.WriteLine("Timer elapsed at {0}", e.SignalTime);
}
```

System.Windows.Threading.DispatcherTimer

Аналог для WPF, интегрированный в очередь диспетчера.

PeriodicTimer (C# 10+)

Современный подход для async/await циклов, решающий проблему «наползания» итераций друг на друга. Следующий «тик» не наступит, пока вы не завершите обработку текущего. Это делает код чистым, избавляет от callback-hell и автоматизирует управление ресурсами.

Пример

```
// Создаем таймер на 1 секунду
using var timer = new PeriodicTimer(TimeSpan.FromSeconds(1));

// Просто цикл. Читается как книга: "Пока ждем следующего тика..."
while (await timer.WaitForNextTickAsync())
{
    // ТУТ ВАШ КОД имитация раюоты
    await DoSomethingHeavyAsync(); // Даже если это займет 5 секунд,
                                   // итерации НЕ наслоятся друг на друга.
}
```

Что выбрать?

Нужно просто обновить текст на форме? — Forms.Timer или DispatcherTimer.

Нужна высокая производительность в фоне? — Threading.Timer.

Пишете современный асинхронный сервис? — PeriodicTimer.

Высокоточное измерение времени

C++ QueryPerformanceCounter

В отличие от GetTickCount, который просто считывает значение из переменной в памяти (обновляемой по прерыванию таймера), QueryPerformanceCounter обращается напрямую к аппаратному счетчику с высокой частотой. Позволяет измерять интервалы с точностью до микросекунд (10^{-6}) и даже выше.

C# Класс Stopwatch

DateTime.Now и DateTime.UtcNow берут данные из системных тиков. Если ваша функция выполняется за 2 мс, DateTime может показать, что прошло 0 мс или сразу 15 мс. Stopwatch же покажет честные 2.000 мс.

Пример - ссылка