Промышленное программирование

Лекция 9

1. Многопоточность
2. Task Parallel Library
3. Асинхронность

1. **Многопоточность**
2. Task Parallel Library
3. Асинхронность

Потоки

```
using System.Globalization;

Console.WriteLine(Environment.CurrentManagedThreadId);
Console.WriteLine(Thread.CurrentThread.ManagedThreadId);
Thread.Sleep(1000);

Thread.CurrentThread.CurrentCulture = CultureInfo.GetCultureInfo("en-US");
Console.WriteLine(3.14);

var t = new Thread(() => Console.WriteLine("Hello!"));
t.Start();

var failedThread = new Thread(() => throw new Exception());
failedThread.Start();
failedThread.Join();

Console.WriteLine("End");
```

Синхронизация: Monitor и lock

```
var lockObject = new object();

new Thread(Do).Start();
new Thread(Do).Start();

void Do() {
    for (int i = 0; i < 2; ++i)
        try {
            Monitor.Enter(lockObject);

            Console.WriteLine($"[{Environment.CurrentManagedThreadId}] Enter");
            Thread.Sleep(1000);
            Console.WriteLine($"[{Environment.CurrentManagedThreadId}] Exit");
        }
        finally {
            Monitor.Exit(lockObject);
        }

        // lock (lockObject) {
        //     ...
        // }
}
```

Синхронизация: AutoResetEvent

```
using System.Collections.Concurrent;

using var t = new ReusableThread();
t.Post(() => /* ... */);
t.Post(() => /* ... */);
Thread.Sleep(1000);
```

```
sealed class ReusableThread : IDisposable {
    private readonly ConcurrentQueue<Action> _queue = new();
    private readonly AutoResetEvent _event = new(false);
    private volatile bool _exit;

    public ReusableThread() => new Thread(Run).Start();

    public void Post(Action action) {
        _queue.Enqueue(action);
        _event.Set();
    }

    public void Dispose() {
        _exit = true;
        _event.Set();
    }

    private void Run() {
        while (!_exit) {
            if (!_queue.TryDequeue(out var action)) {
                _event.WaitOne();
                continue;
            }
            action();
        }
        _event.Dispose();
    }
}
```

Отмена выполнения

```
using var cancellationTokenSource = new CancellationTokenSource();
var token = cancellationTokenSource.Token;

var t = new Thread(Run);
t.Start();
Thread.Sleep(200);
cancellationTokenSource.Cancel();
t.Join();

void Run() {
    try {
        const int count = 10;
        for (int i = 0; i < 10; ++i) {
            Console.WriteLine($"{i + 1} / {count}");
            Thread.Sleep(100);
            token.ThrowIfCancellationRequested();
            // if (token.IsCancellationRequested)
            //     return;
        }
    }
    catch (OperationCanceledException) {
    }
}
```

1. Многопоточность
2. **Task Parallel Library**
3. Асинхронность

Пул потоков

1. Создание потока – очень дорогое удовольствие
2. Выгоднее заранее создать пул потоков
   и переиспользовать их по мере необходимости

```
ThreadPool.QueueUserWorkItem(o => Run());
ThreadPool.QueueUserWorkItem(o => Run());
ThreadPool.QueueUserWorkItem(o => Run());

static void Run()
{
    Console.WriteLine(Environment.CurrentManagedThreadId);
}
```

Task

```
public class Task : IAsyncResult, IDisposable
{
    public bool IsCompleted { get; }
    public WaitHandle AsyncWaitHandle { get; }
    public object? AsyncState { get; }
    public bool CompletedSynchronously { get; }

    public AggregateException? Exception { get; }

    public Task(Action action);
    public Task(Action action, CancellationToken cancellationToken);

    public void Start();

    public Task ContinueWith(Action<Task> continuationAction);
    public Task<T> ContinueWith(Func<Task, T> continuationFunction);

    public void Dispose();
}
```

Обработка исключений в Task

```
using var t = new Task(() => throw new InvalidOperationException("Oops"));
t.Start();
while (!t.IsCompleted)
{
}
Console.WriteLine(t.Exception);
```

```
using var t = new Task(() => throw new InvalidOperationException("Oops"));
t.Start();
try
{
    t.Wait();
}
catch (AggregateException e)
{
    Console.WriteLine(e);
}
```

Task<T>

```
public class Task<TResult> : Task {
    public Task(Func<TResult> function)
    public Task(Func<TResult> function, CancellationToken cancellationToken)

    public TResult Result { get; }

    ...
}
```

```
using var t1 = new Task(() => Console.WriteLine("#1"));
using var t2 = t1.ContinueWith(t => Console.WriteLine("#2"));
using var t3 = t2.ContinueWith(t => 3);

t1.Start();
Console.WriteLine(t3.Result);
```

Статические члены Task

```
Task.Run(() => Console.WriteLine("#1"));

Task.Delay(1000).Wait();

Console.WriteLine(Task.FromResult(5).Result);
Console.WriteLine(Task.FromException<int>(new InvalidOperationException()).Result);
```

Пример

```
using var firstAsset = new StreamReader("path/to/first/asset");
using var secondAsset = new StreamReader("path/to/second/asset");

using var firstTask = firstAsset.ReadToEndAsync();
using var secondTask = secondAsset.ReadToEndAsync();

Console.WriteLine(firstTask.Result);
Console.WriteLine(secondTask.Result);
```

1. Многопоточность
2. Task Parallel Library
3. **Асинхронность**

Синхронность и асинхронность

Синхронное выполнение метода
- Метод выполняется незамедлительно
- Пока выполняется метод, вызвавший метод ждёт
- Метод возвращает готовый результат

Асинхронное выполнение метода
- Метод выполняется отложенно и, может быть, в отдельном потоке
- Управление вызвавшему методу возвращается до потенциальной готовности результата
- Метод возвращает прокси-объект

Асинхронный метод

```
async [Task | Task<T> ] MethodName(/*in, out, ref запрещены*/)
{
    // Тело содержит одно или несколько выражений await
}
```

Пример: синхронная версия

```
Console.WriteLine(Read());

static string Read()
{
    var intermediate = Read(string.Empty);
    return Read(intermediate);
}

static string Read(string src)
{
    Task.Delay(1000).Wait();
    return src + "a";
}
```

Пример: асинхронная версия

```
static void Main() {
    var t = ReadAsync();
    Console.WriteLine("Some other work...");
    Console.WriteLine(t.Result);
}

static Task<string> ReadAsync() {
    var intermediateTask = ReadAsync(string.Empty);
    return intermediateTask.ContinueWith(t => ReadAsync(t.Result).Result);
}

static Task<string> ReadAsync(string src) {
    return Task.Run(() =>
    {
        Task.Delay(1000).Wait();
        return src + "a";
    });
}
```

Пример: асинхронная версия

```
static async Task Main() {
    var r = ReadAsync();
    Console.WriteLine("Some other work...");
    Console.WriteLine(await r);
}

static async Task<string> ReadAsync() {
    var intermediate = await ReadAsync(string.Empty);
    return await ReadAsync(intermediate);
}

static async Task<string> ReadAsync(string src) {
    await Task.Delay(1000);
    return src + "a";
}
```