асинхронность в UI

- Асинхронная операция: не происходит блокировка текущего потока до завершения операции
- Асинхронность важна:
  - для приложений с пользовательским интерфейсом (UI)
  - при использовании операций ввода/вывода
- Модели асинхронности:
  - Begin_, End
  - EAM = event-asynchronous model
  - async/await

Асинхронные вызовы делегатов

- Делегаты, связанные с методом, предоставляют возможность асинхронного вызова
- **BeginInvoke** – поставить в делегат в очередь на выполнение (добавить в пул потоков)
- **EndInvoke** – дождаться завершения асинхронного вызова и получить результат
- Объект **IAsyncResult** – связь с асинхронным методом (текущее состояние)
- **AsyncCallBack** – делегат обратного вызова, запускается после завершения асинхронного вызова (**EndInvoke**)

```
public static void SomeFunction()
{
    ..
}
public delegate void Working();
static void Main()
{
    Working w = new Working(SomeFunction);
    w();
    IAsyncResult res = w.BeginInvoke(null, null);
    SomeOtherWork();
    w.EndInvoke(res);
}
```

Асинхронность в UI

- Вычислительно-емкие процессы не должны запускаться в основном потоке, который **обслуживает** взаимодействие пользователя с приложением

- Обновление пользовательского интерфейса может выполняться только в одном потоке
- Для реализации корректной работы в UI можно использовать:
  - ручная реализация с помощью Thread и метода Control.Invoke
  - компонент BackgroundWorker
  - компонент SynchronizationContext

Thread + Invoke

Для любого элемента Control (в т.ч. и формы) определен метод Invoke гарантирующий запуск обработчика в UI-потоке

```
class MyForm : Form
{
    public MyForm() {
        new Thread(() => LongComputation).Start();
    }
    public void LongComputation() {
        ..
        ..
        this.Invoke(() => this.Title = “Finish”);
    }
}
```

BackgroundWorker

- Компонент предназначен для выполнения длительной операции в фоновом потоке
- Позволяет отслеживать прогресс обработки
- Реализует событийную модель: для использования необходимо выполнить привязку обработчиков к событиям компонента.
- Обработчик события DoWork – длительная фоновая операция
- Обработчик события RunWorkerCompleted предназначен для выполнения каких-либо действий после завершения фоновой работы;
- Обработчик события ProgressChanged используется для отслеживания прогресса в ходе обработки и обновления UI-интерфейса

```
// Сценарий работы с BackgroundWorker
worker = new BackgroundWorker();
worker.WorkerReportsProgress = true;
worker.DoWork += VeryLongComputation;
worker.RunWorkerCompleted += (sender, args) =>
      {          Title = "work is finished";        };

worker.ProgressChanged += (sender, args) =>
      {
            Title = "Current value: " + args.UserState +
                              " , progress: " + args.ProgressPercentage;
       };

worker.RunWorkerAsync();
```

Компонент SynchronizationContext

- Позволяет организовать взаимосвязь основного потока, обновляющего UI-интерфейс, и дополнительного потока, выполняющего длительную операцию
- Необходимо сохранить текущий контекст синхронизации через свойство SynchronizationContext**.Current**
- Фоновый поток отправляет сообщения основному потоку через сохраненный контекст:

```
// отправка сообщения в процессе обработки
context.Send( .. )
// отправка сообщения в конце обработки
context.Post( .. )
```

- Сообщение – это обработчик, который будет вызван в сохраненном контексте