**Лабораторная 12:**

**Файловая система -** это инструмент, позволяющий операционной системе и программам обращаться к нужным файлам и работать с ними. При этом программы оперируют только названием файла, его размером и датой созданий. FAT32, NTFS, exFAT.

1. Какие классы содержаться с System.IO?

Множество классов по типу DiskInfo, File, FileInfo, Directory, DirectoryInfo, StreamWriter, StreamReader, FileStream, BinaryReader, BinaryWriter и т.д.

2. Для чего используются классы Directory и DirectoryInfo? В чем отличие?

Парные классы для взаимодействия с директориями. Directory – статический класс, позволяющий через статические методы работать с директориями, а DirectoryInfo позволяет получить информацию о конкретной директории.

3. Для чего используются классы File и FileInfo? Какие методы они содержат.

Парные классы для взаимодействия с файлами. File – статический класс, позволяющий через статические методы работать с файлами, а FileInfo(Create(), Delete(), CopyTo())позволяет получить информацию о конкретном файле.

4. Для чего используются классы StreamReader и StreamWriter?

Классы StreamReader и StreamWriter используются **для работы с потоками символьных данных**. Наибольшее применение классов — работа с текстовыми файлами. Класс StreamReader используется для чтения из потока символьных данных.

5. Для чего используются классы BinaryWriter и BinaryReader?

**Для работы с бинарными файлами** предназначена пара классов BinaryWriter и BinaryReader. Эти классы позволяют читать и записывать данные в двоичном формате.

6. Как можно сжимать и восстанавливать файлы?

Это классы ZipFile, DeflateStream и GZipStream, которые находятся в пространстве имен System.IO.Compression и представляют реализацию одного из алгоритмов сжатия **Deflate или GZip**.

7. Расскажите алгоритм сжатия GZip.

GZIP обеспечивает сжатие без потерь, иными словами, исходные данные можно полностью восстановить при распаковке. Он основан на алгоритме DEFLATE, который использует комбинацию алгоритма LZ77 и алгоритма Хаффмана.

8. Для чего служит класс Path?

Он предоставляет статические методы, которые упрощают выполнение операций с путевыми именами.

9. Что такое тестовый и бинарный файл. Как можно выполнить чтение и запись в эти файлы.

**Текстовый файл** содержит последовательность символов. **Двоичный файл** — последовательность произвольных [байтов](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82).

Метод читает строку символов из текущего потока и возвращает его в виде типа string.

Двоичный файл читается по байтам.

10. Что такое произвольный доступ к файлу? Приведите пример.

Файлы с произвольным доступом — файлы, хранящие информацию в структурированном виде. Поиск в таких файлах осуществляется в области адресов и завершается обращением непосредственно к искомому участку. Дисковое пространство, занимаемое таким файлом, поделено на одинаковые участки, имеющие одинаковую структуру полей.

11. Как применяется конструкция using (не директива) при работе с файловыми потоками? Для чего ее используют.

Конструкция using служит сигнализатором об открытия и закрытии работы с потоком.

**Лабораторная 13:**

1. Что такое сериализация, десериализация?

Сериализация представляет процесс преобразования какого-либо объекта в поток байтов. После преобразования мы можем этот поток байтов или записать на диск или сохранить его временно в памяти. А при необходимости можно выполнить обратный процесс - десериализацию, то есть получить из потока байтов ранее сохраненный объект.

2. Какие существуют форматы сериализации? Поясните структуру для каждого формата. Какие классы для работы с ними существуют в .NET?

Бинарный – BinaryFormatter, сохраняет тип.

SOAP – SoapFormatter, не поддерживается.

xml – XmlSerializer, сериализует только открытые поля.

JSON – DataContractJsonSerializer, JSON сохраняет без информации о точности типов.

3. Какие классы существуют в пространстве имен System.Xml?

XmlNode, XmlDocument, XmlElement, XmlAttribute, XmlText, XmlComment, XmlNodeList

4. Какие атрибуты используются для настройки XML сериализации?

[XmlAnyAttributeAttribute](https://learn.microsoft.com/ru-ru/dotnet/api/system.xml.serialization.xmlanyattributeattribute), [XmlAnyElementAttribute](https://learn.microsoft.com/ru-ru/dotnet/api/system.xml.serialization.xmlanyelementattribute), [XmlArrayAttribute](https://learn.microsoft.com/ru-ru/dotnet/api/system.xml.serialization.xmlarrayattribute), [XmlArrayItemAttribute](https://learn.microsoft.com/ru-ru/dotnet/api/system.xml.serialization.xmlarrayitemattribute), [XmlAttributeAttribute](https://learn.microsoft.com/ru-ru/dotnet/api/system.xml.serialization.xmlattributeattribute), [XmlChoiceIdentifierAttribute](https://learn.microsoft.com/ru-ru/dotnet/api/system.xml.serialization.xmlchoiceidentifierattribute), [XmlElementAttribute](https://learn.microsoft.com/ru-ru/dotnet/api/system.xml.serialization.xmlelementattribute), [XmlEnumAttribute](https://learn.microsoft.com/ru-ru/dotnet/api/system.xml.serialization.xmlenumattribute), [XmlIgnoreAttribute](https://learn.microsoft.com/ru-ru/dotnet/api/system.xml.serialization.xmlignoreattribute), [XmlIncludeAttribute](https://learn.microsoft.com/ru-ru/dotnet/api/system.xml.serialization.xmlincludeattribute), [XmlRootAttribute](https://learn.microsoft.com/ru-ru/dotnet/api/system.xml.serialization.xmlrootattribute), [XmlTextAttribute](https://learn.microsoft.com/ru-ru/dotnet/api/system.xml.serialization.xmltextattribute), [XmlTypeAttribute](https://learn.microsoft.com/ru-ru/dotnet/api/system.xml.serialization.xmltypeattribute)

5. В чем отличие BinaryFormatter или SoapFormatter?

BinaryFormatter реализует двоичный закрытый метод сериализации, класс SoapFormatter – реализует текстовый и открытый метод сериализации.

6. Что такое сериализация контрактов данных, контракт данных?

*Контракт данных* – это тип (класс или структура), описывающий информационный фрагмент. Если в качестве контракта данных используется обычный класс, информационный фрагмент образуют открытые поля и свойства. Можно пометить тип атрибутом [DataContract].

7. Где и для чего используются атрибуты [OnSerializing], [OnSerialized], [OnDeserializing], [OnDeserialized]?

Если контракт является коллекцией объектов (как класс Group), он маркируется атрибутом [CollectionDataContract]. Кроме этого, для методов контракта данных применимы атрибуты [OnSerializing], [OnSerialized], [OnDeserializing], [OnDeserialized].

8. Что такое XPath? Приведите пример.

XPath представляет язык запросов в XML. Он позволяет выбирать элементы, соответствующие определенному селектору.

9. Какие возможности дает LINQ to Xml. Приведите примеры.

Эта интеграция дает **возможность создавать запросы к загруженному в память XML-документу с целью получения коллекций элементов и атрибутов**.

**Лабораторная 14:**

1. Что такое процесс, домен, поток? Как они связаны между собой?

Процесс – набор действий, выполняемый за промежуток времени. При запуске приложения создаётся отдельный процесс, выделяется адресное пространство в памяти и изолирование от других процессов. Процесс может иметь несколько потоков. Как минимум, процесс содержит один - главный поток. В приложении на C# точкой входа в программу является метод Main. Вызов этого метода автоматически создает главный поток. А из главного потока могут запускаться вторичные потоки.

Домен приложения это изолированная область, в которой выполняется приложение. Один процесс может содержать любое количество доменов приложения. Каждый домен приложения полностью изолирован от других доменов приложения.

Поток – это независимый путь исполнения, способный выполняться одновременно с другими потоками. Для управления домена платформа .NET предоставляет класс AppDomain.

2. Как получить информацию о процессах?

В .NET процесс представлен классом Process из пространства имен System.Diagnostics. Этот класс позволяет управлять уже запущенными процессами, а также запускать новые. В данном классе определено ряд свойств и методов, позволяющих получать информацию о процессах и управлять ими.

3. Как создать и настроить домен?

Для управления домена платформа .NET предоставляет класс AppDomain. Для создания потока применяется один из конструкторов класса Thread:

4. Как создать и настроить поток?

Для создания потока применяется один из конструкторов класса Thread.

5. В каких состояниях может быть поток?

**Новый (New)**. После создания экземпляра потока, он находится в состоянии Новый до тех пор, пока не вызван метод start(). В этом состоянии поток не считается живым.

**Работоспособный (Runnable)**. Поток переходит в состояние Работоспособный, когда вызывается метод start(). Поток может перейти в это состояние также из состояния Работающий или из состояния Блокирован. Когда поток находится в этом состоянии, он считается живым.

**Работающий (Running)**. Поток переходит из состояния Работоспособный в состояние Работающий, когда Планировщик потоков выбирает его как работающий в данный момент.

**Живой, но не работоспособный (Alive, but not runnable)**. Поток может быть живым, но не работоспособным по нескольким причинам:

**Ожидание (Waiting)**. Поток переходит в состояние Ожидания, вызывая метод wait(). Вызов notify() или notifyAll() может перевести поток из состояния Ожидания в состояние Работоспособный.

**Сон (Sleeping)**. Метод sleep() переводит поток в состояние Сна на заданный промежуток времени в миллисекундах.

**Блокировка (Blocked)**. Поток может перейти в это состояние, в ожидании ресурса, такого как ввод/вывод или из-за блокировки другого объекта. В этом случае поток переходит в состояние Работоспособный, когда ресурс становится доступен.

**Мёртвый (Dead)**. Поток считается мёртвым, когда его метод run() полностью выполнен. Мёртвый поток не может перейти ни в какое другое состояние, даже если для него вызван метод start().

6. Какие методы управления потоками вы знаете, для чего и как их использовать?

start() – начало работы потока, sleep() – состояние сна, run() выполнение потока, wait() – состояние ожидания, lock() – блокировка.

Метод Join блокирует выполнение вызвавшего его потока до тех пор, пока не завершится поток, для которого был вызван данный метод

Метод Interrupt прерывает поток, который находится в состоянии WaitSleepJoin

7. Какие приоритеты потока вы знаете?

Lowest = 0,

BelowNormal = 1,

Normal = 2,

AboveNormal = 3,

Highest = 4

8. Что такое пул потоков и для чего он используется?

Пул потоков – это коллекция рабочих потоков, которые эффективно выполняют асинхронные обратные вызовы от имени приложения. Пул потоков в основном используется для уменьшения количества потоков приложения и управления рабочими потоками.

9. Что такое критическая секция? Поясните использование.

Критическая секция – участок исполняемого кода программы, в котором производится доступ к общему ресурсу (данным или устройству), который не должен быть одновременно использован более чем одним потоком выполнения. При нахождении в критической секции двух (или более) потоков возникает состояние «гонки» («состязания»).

10. Что такое мьютекс? Поясните использование

Мьютекс (mutex - mutuallyexclusive), который также называют защелкой – это механизм изоляции, используемый сервером баз данных для синхронизации доступа нескольких потоков к совместно используемым ресурсам.

11. Что такое семафор? Поясните использование

Семафор – примитив синхронизации работы [процессов](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) и [потоков](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F), в основе которого лежит счётчик, над которым можно производить две [атомарные операции](https://ru.wikipedia.org/wiki/%D0%90%D1%82%D0%BE%D0%BC%D0%B0%D1%80%D0%BD%D0%B0%D1%8F_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F): увеличение и уменьшение значения на единицу, при этом операция уменьшения для нулевого значения счётчика является блокирующейся.

12. Что такое неблокирующие средства синхронизации?

Неблокирующая синхронизация – подход в параллельном программировании на симметрично-многопроцессорных системах, в котором отходят от традиционных примитивов блокировки, таких, как семафоры, мьютексы и события.

13. Для чего можно использовать класс Timer?

Класс Timer используется для планирования выполнения задачи. Запланированная к выполнению задача должна быть экземпляром класса TimerTask. Вы сначала создаёте объект класса TimerTask, а затем планируете его запуск с помощью класса Timer.

**Лабораторная 15:**

1. Что такое TPL? Как и для чего используется тип Task

TPL (Task Parallel Library) – библиотека параллельных задач, основной функционал которой располагается в пространстве имен System.Threading.Tasks. Данная библиотека упрощает работу с многопроцессорными, многоядерными система. Кроме того, она упрощает работу по созданию новых потоков. Поэтому обычно рекомендуется использовать именно TPL и ее классы для создания многопоточных приложений, хотя стандартные средства и класс Thread по-прежнему находят широкое применение.

Task – который находится в пространстве имен System.Threading.Tasks. Данный класс описывает отдельную задачу, которая запускается асинхронно в одном из потоков из пула потоков. Хотя ее также можно запускать синхронно в текущем потоке.

2. Почему эффект от распараллеливания наблюдается на большом количестве элементов?

При увеличении размерности решаемой системы уравнений объем вычислительной работы растет пропорционально n3, а объем обменов между процессорами пропорционально n2.

3. В чем основные достоинства работы с задачами по сравнению с потокми?

При работе с задачами, можно остановить выполнение одной большой задачи и в промежутке выполнить более мелкую задачу.

4. Приведите три способа создания и/или запуска Task?

Реализация через стрелочную функцию, через метод Run (надо к дополнительному классу обращаться) и через метод Start

5. Как и для чего используют методы Wait(), WaitAll() и WaitAny()?

**Метод Wait(), приостанавливающий исполнение вызывающего потока до тех пор, пока не завершится вызываемая задача.**

**WaitAll() – метод организует ожидание завершения группы задач. Возврата из него не произойдет до тех пор, пока не завершатся все задачи.**

**Иногда требуется организовать ожидание до тех пор, пока не завершится любая из группы задач. Для этой цели служит метод WaitAny().**

6. Приведите пример синхронного запуска Task?

Запуск нескольких задач одновременно можно реализовать через массив задач или использовать методы Run или Factory.StartNew

7. Как создать задачу с возвратом результата?

Во-первых, чтобы получать из задачи не который результат, необходимо типизировать объект Task тем типом, объект которого мы хотим получить из задачи.

Во-вторых, в качестве задачи должен выполняться метод, который возвращает данный тип объекта.

8. Как обработать исключение, если оно произошло при выполнении

Task?

Для обработки исключения в методе Main выражение await помещено в блок try. В итоге при выполнении вызова await PrintAsync("Hi") будет сгенерировано исключение, что привет к генерации исключения. Однако программа не остановит аварийно свою работу, а обработает исключение и продолжит дальнейшие вычисления.

9. Что такое CancellationToken и как с его помощью отменить выполнение задач?

**CancellationToken — собственно сам токен, у него есть свойство IsCancellationRequested, которое показывает состояние отмены, а также есть метод, который выбрасывает исключение (OperationCanceledException) в случае отмены операции.**

10. Как организовать задачу продолжения (continuation task) ?

Задачи продолжения или continuation task позволяют определить задачи, которые выполняются после завершения других задач. Благодаря этому мы можем вызвать после выполнения одной задачи несколько других, определить условия их вызова, передать из предыдущей задачи в следующую некоторые данные.

Метод ContinueWith – который в качестве параметра принимает делегат Action<Task>. То есть метод PrintTask, который передается в вызов ContinueWith, должен принимать параметр типа Task

11. Как и для чего используется объект ожидания при создании задач продолжения?

Объект ожидания предназначен для определения завершённости выполняемой задачи.

12. Поясните назначение класса System.Threading.Tasks.Parallel?

**Этот класс поддерживает набор методов, которые позволяют выполнять итерации по коллекции данных (точнее, по объектам, реализующим IEnumerable<T>) в параллельном режиме.**

16. Как с использованием CancellationToken отменить параллельные операции?

**Чтобы отменить задачу нужно вызвать метод Cancel у экземпляра CancellationTokenSource. Он переведет все выпущенные токены в отмененное состояние. Внутри асинхронного метода сработает проверка токена и работа будет завершена.**

17. Для чего используют BlockingCollection<T>, в чем ее особенность?

**Коллекция, которая осуществляет блокировку и ожидает, пока не появится возможность выполнить действие по добавлению или извлечению элемента. BlockingCollection<T> предлагает интерфейс для добавления и извлечения элементов методами Add() и Take(). Эти методы блокируют поток и затем ожидают, пока не появится возможность выполнить задачу.**

**Метод Add() имеет перегрузку, которой можно также передать CancellationToken. Эта лексема всегда отменяет блокирующий вызов.**

**Если не нужно, чтобы поток ожидал бесконечное время, и не хотите отменять вызов извне, доступны также методы TryAdd() и TryTake(). В них можно указать значение таймаута — максимального периода времени, в течение которого вы готовы блокировать поток и ждать, пока вызов не даст сбой.**

18. Как используя async и await организовать асинхронное выполенение метода?

Ключевое слово async — это метод, который выполняет асинхронные задачи, такие как выборка данных из базы данных, чтение файла и т. д. Они могут быть помечены как «асинхронные». Принимая во внимание, что ключевое слово await делает «ожидание» оператора означает приостановку выполнения асинхронного метода, в котором он находится, до завершения асинхронной задачи.

Асинхронный метод обладает следующими признаками:

В заголовке метода используется модификатор async

Метод содержит одно или несколько выражений await

В качестве возвращаемого типа используется один из следующих:

void

Task

Task<T>

ValueTask<T>