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

**Процесс** – **объект, который создаётся ОС для каждого приложения в момент его запуска**. Выделяет собственное адресное пространство, которое напрямую недоступно другим процессам. Поддерживает один или несколько потоков выполнения. Процесс содержит один – главный поток. В приложении на C# точкой входа в программу является метод Main. System.Diagnostics

**Поток – используемый внутри процесса путь выполнения**. **System.Threading**

**Среда CLR создает внутри этого процесса логический контейнер** - **домен приложения**. **Внутри его работает запущенное приложение**. В процессе мб несколько доменов приложений. Процесс запускает при старте домен по умолчанию. Класс AppDomain

2. Как получить информацию о процессах?

var Processs = Process.GetProcesses();

3. Как создать и настроить домен?

AppDomain newD = AppDomain.CreateDomain("New");

newD.Load("имя сборки");

AppDomain.Unload(newD);

Один процесс способен обслуживать множество доменов приложений посредством статического метода AppDomain.CreateDomain().

4. Как создать и настроить поток?

var number = new Thread(PrintSimpleNumbers);

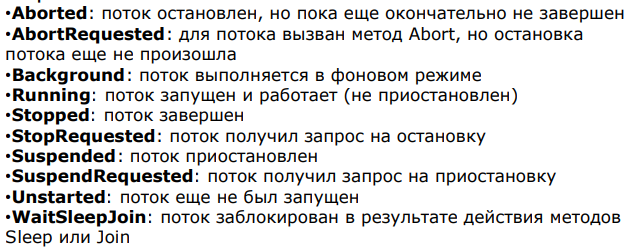
number.Start();

Thread thrd = new Thread((new Point()).Move) { Name = "Point Move", Priority = ThreadPriority.BelowNormal, IsBackground = true, };

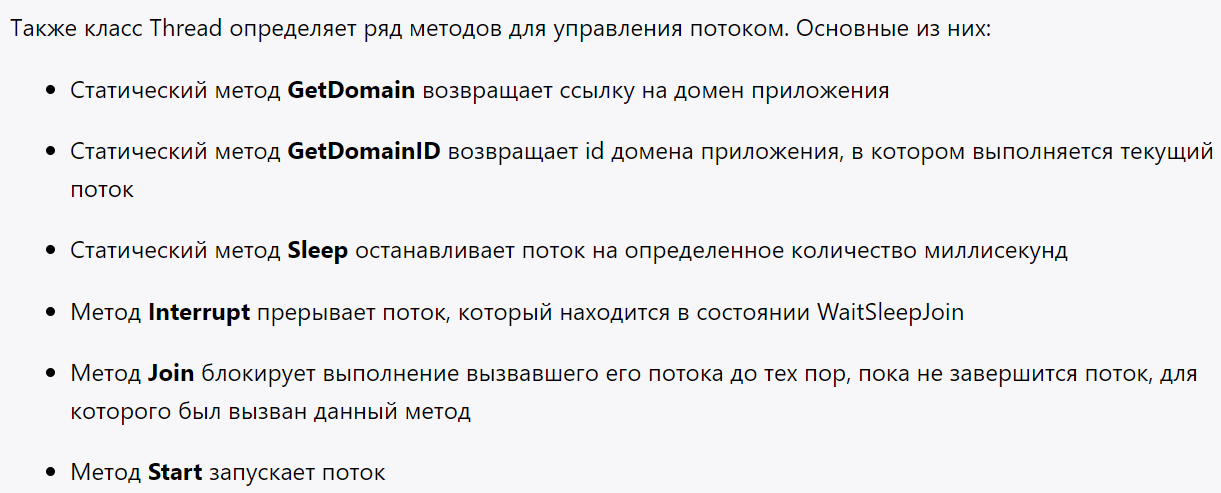
Для создания потока применяется один из конструкторов класса **Thread**:

* **Thread(ThreadStart)**: в качестве параметра принимает объект делегата ThreadStart, который представляет выполняемое в потоке действие
* **Thread(ThreadStart, Int32)**: в дополнение к делегату ThreadStart принимает числовое значение, которое устанавливает размер стека, выделяемого под данный поток
* **Thread(ParameterizedThreadStart)**: в качестве параметра принимает объект делегата ParameterizedThreadStart, который представляет выполняемое в потоке действие
* **Thread(ParameterizedThreadStart, Int32)**: вместе с делегатом ParameterizedThreadStart принимает числовое значение, которое устанавливает размер стека для данного потока.

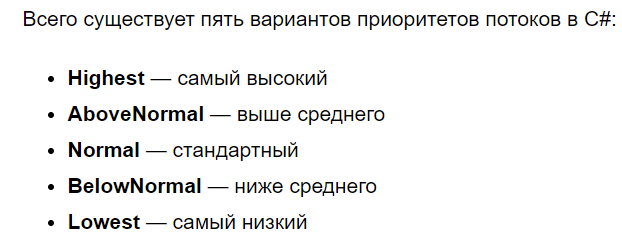
5. В каких состояниях может быть поток?



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



7. Какие приоритеты потока вы знаете?

По умолчанию поток с приоритетом Normal.

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. System.Treading. Класс **ThreadPool**

9. Что такое критическая секция? Поясните использование.

**Критическая секция**— участок исполняемого кода программы, в котором производится доступ к общему ресурсу, который не должен быть одновременно использован более чем одним потоком. Реализуется с использованием блокировки ключевого слова lock, чтобы обеспечить доступ только одному потоку к критической части кода одновременно.

Wait() – освобождает блокировку объекта и переводит поток в очередь ожидания.

Pulse() – уведомляет поток из очереди ожидания, что текущий поток освободил объект obj

10. Что такое мьютекс? Поясните использование

Инструмент управления синхронизацией потоков представляет класс **Mutex**. System.Threading.

Метод mutexObj.**WaitOne**() приостанавливает выполнение потока до тех пор, пока не будет получен мьютекс mutexObj. После выполнения всех действий, когда мьютекс больше не нужен, поток освобождает его с помощью метода mutexObj.**ReleaseMutex**(). А мьютекс получает один из ожидающих потоков.

Таким образом, когда выполнение дойдет до вызова mutexObj.WaitOne(), поток будет ожидать, пока не освободится мьютекс. И после его получения продолжит выполнять свою работу.

11. Что такое семафор? Поясните использование

Инструмент управления синхронизацией потоков, позволяющий войти в заданный участок кода не более чем N потокам (N –ёмкость семафора).

* получение и снятие блокировки может выполняться из разных потоков!! Такого нет у мьютекса и монитора
* классы System.Threading.Semaphore(между процессами) и SemaphoreSlim(в рамках одного процесса)
* **WaitOne()**: ожидает получения свободного места в семафоре
* **Release()**: освобождает место в семафоре

Для создания семафора применяется один из конструкторов класса Semaphore:

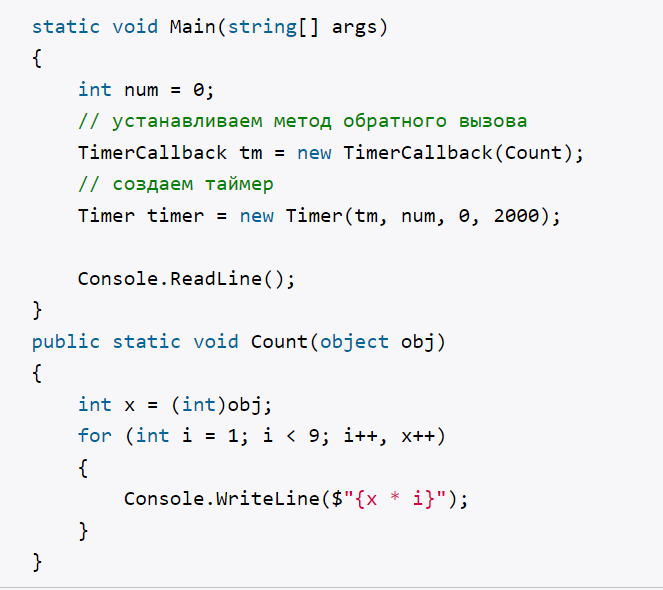
* **Semaphore (int initialCount, int maximumCount)**: параметр initialCount задает начальное количество потоков, а maximumCount - максимальное количество потоков, которые имеют доступ к общим ресурсам
* **Semaphore (int initialCount, int maximumCount, string? name)**: в дополнение задает имя семафора
* **Semaphore (int initialCount, int maximumCount, string? name, out bool createdNew)**: последний параметр - createdNew при значении true указывает, что новый семафор был успешно создан. Если этот параметр равен false, то семафор с указанным именем уже существует

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. Они позволяют осуществлять совместный доступ к простым ресурсам нескольких потоков без блокировки, паузы или ожидания.

13. Для чего можно использовать класс Timer?

Одним из важнейших классов, находящихся в пространстве имени System.Threading, является класс **Timer**. Данный класс позволяет запускать определенные действия по истечению некоторого периода времени.

Первым делом создается объект делегата **TimerCallback**, который в качестве параметра принимает метод. Причем данный метод должен в качестве параметра принимать объект типа object.

И затем создается таймер. Данная перегрузка конструктора таймера принимает четыре параметра:

* объект делегата TimerCallback
* объект, передаваемый в качестве параметра в метод Count
* количество миллисекунд, через которое таймер будет запускаться. В данном случае таймер будет запускать немедленно после создания, так как в качестве значения используется 0
* интервал между вызовами метода Count

И, таким образом, после запуска программы каждые две секунды будет срабатывать метод Count.

Если бы нам не надо было бы использовать параметр obj у метода Count, то при создании таймера мы могли бы указывать в качестве соответствующего параметра значение null: Timer timer = new Timer(tm, null, 0, 2000);