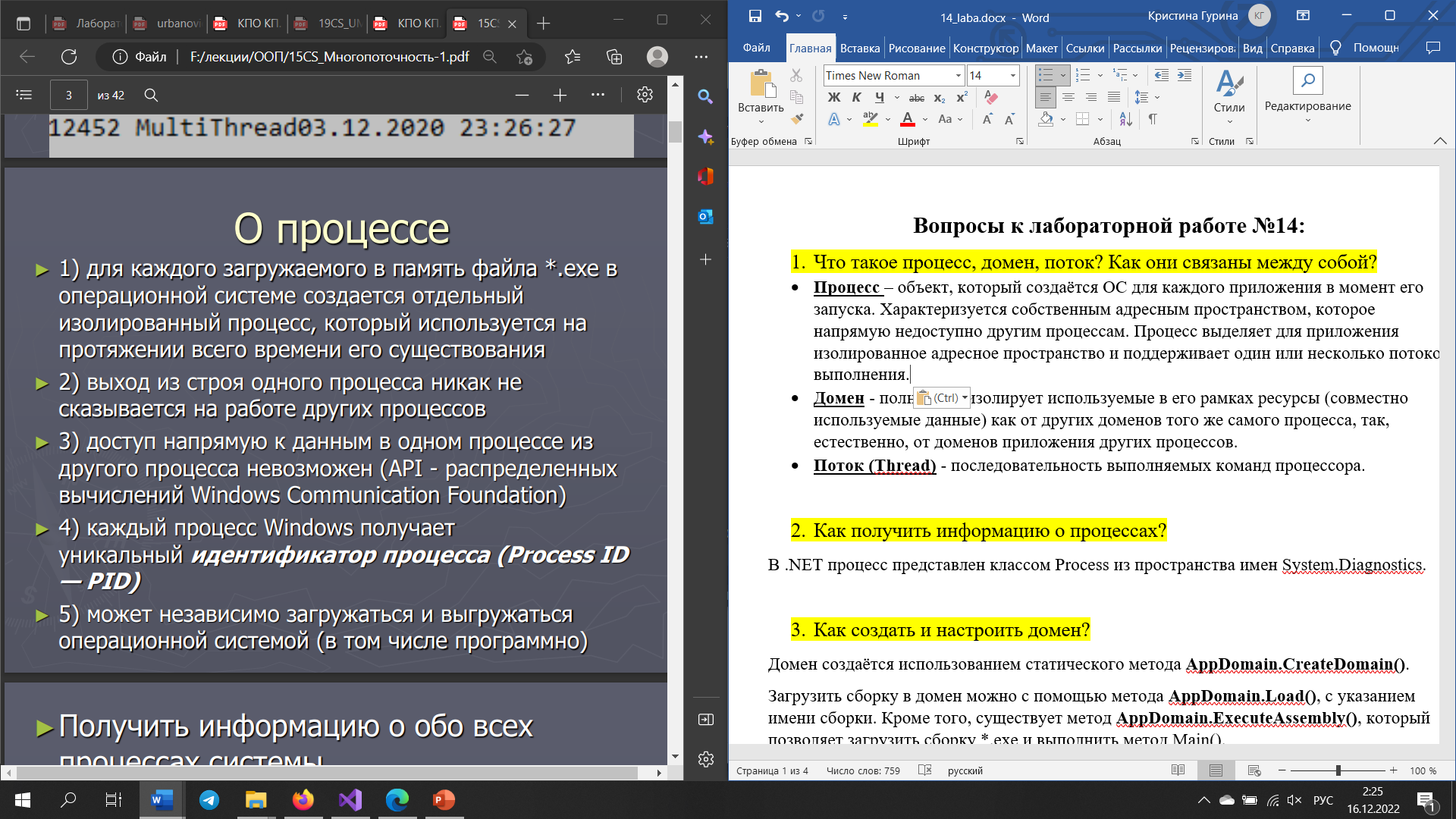
**Вопросы к лабораторной работе №14:**

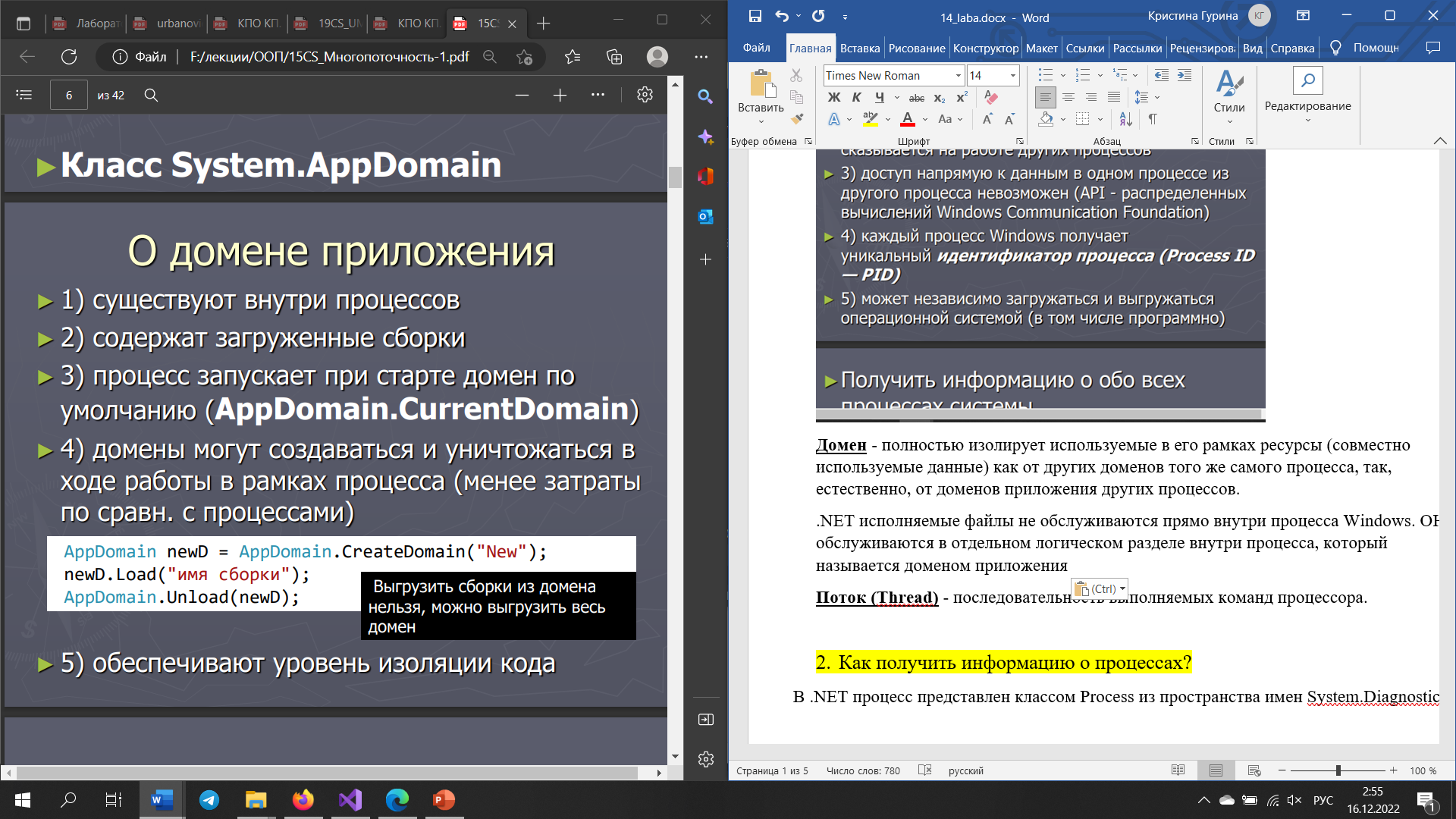
1. Что такое процесс, домен, поток? Как они связаны между собой?

**Процесс** – объект, который создаётся ОС для каждого приложения в момент его запуска. Процесс выделяет для приложения изолированное адресное пространство и поддерживает один или несколько потоков выполнения.

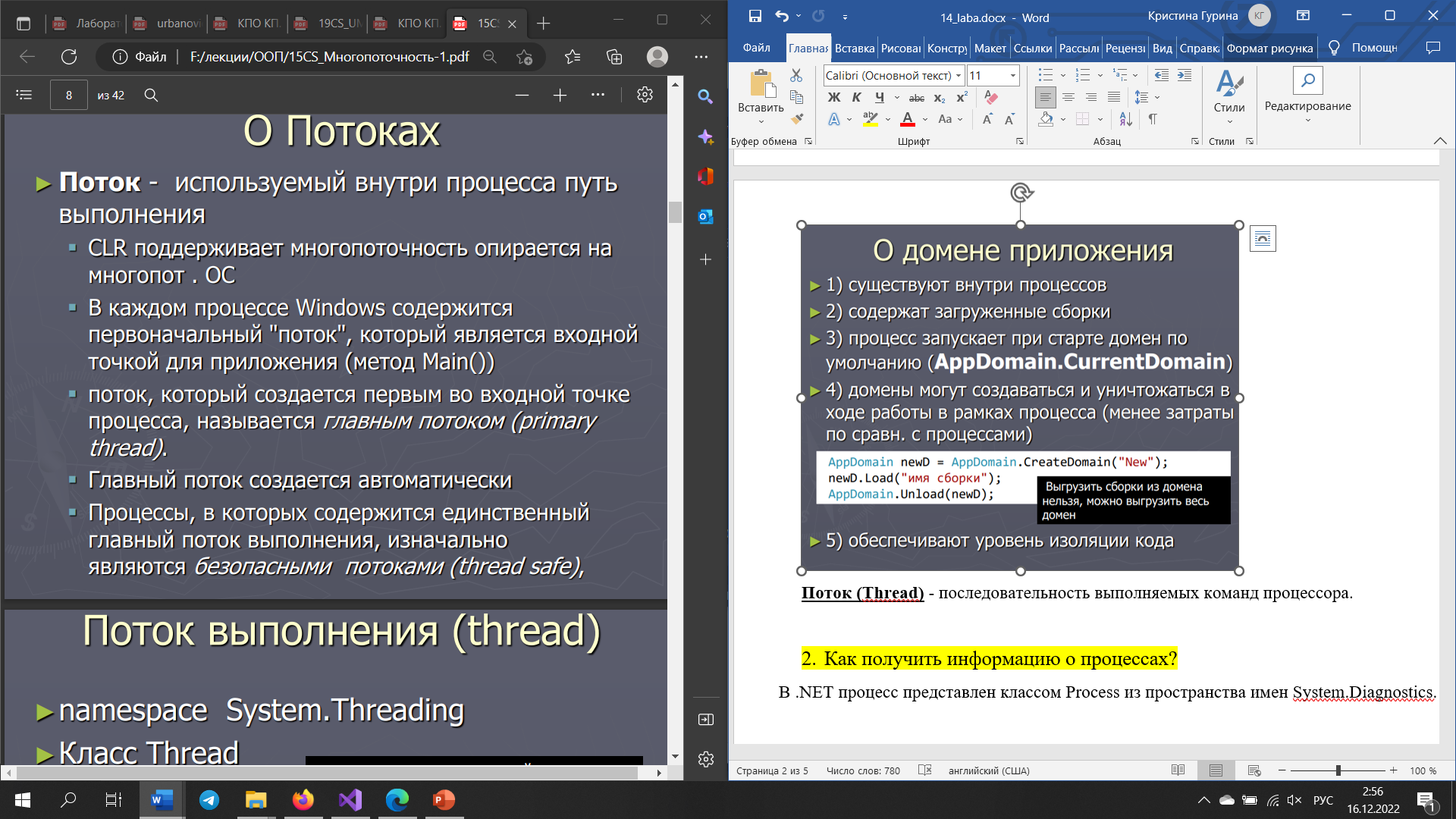


**Домен** - полностью изолирует используемые в его рамках ресурсы (совместно используемые данные) как от других доменов того же самого процесса, так, естественно, от доменов приложения других процессов.

.NET исполняемые файлы не обслуживаются прямо внутри процесса Windows. ОНИ обслуживаются в отдельном **логическом разделе внутри процесса**, который называется доменом приложения



**Поток (Thread)** - последовательность выполняемых команд процессора.



1. Как получить информацию о процессах?

В .NET процесс представлен классом Process из пространства имен System.Diagnostics.

1. Как создать и настроить домен?

Домен создаётся использованием статического метода **AppDomain.CreateDomain()**.

Загрузить сборку в домен можно с помощью метода **AppDomain.Load()**, с указанием имени сборки. Кроме того, существует метод **AppDomain.ExecuteAssembly()**, который позволяет загрузить сборку \*.ехе и выполнить метод Main().

С помощью метода **AppDomain.Unload()** можно производить избирательную выгрузку определенного домена приложения из обслуживающего процесса. В этом случае вместе с доменом приложения будут выгружаться и все содержащиеся в нем сборки.

1. Как создать и настроить поток?

Для создания нового потока используется делегат **ThreadStart**, который получает в качестве параметра метод. Чтобы запустить поток, вызывается метод **Start**. Метод **Sleep** позволяет приостановить поток на промежуток времени, указанный как параметр данного метода. Поток можно сделать фоновым, изменив значение свойства **IsBackground**, изменить его приоритет **Priority**, изменить его имя **Name**.

1. В каких состояниях может быть поток?

* Выполнение;
* Ожидание - ожидания конца ввода-вывода;
* Готовность - готов работать, но компьютер занят другим.
* Aborted: поток остановлен, но пока еще окончательно не завершен
* AbortRequested: для потока вызван метод Abort, но остановка потока еще не произошла
* Background: поток выполняется в фоновом режиме
* Running: поток запущен и работает (не приостановлен)
* Stopped: поток завершен
* StopRequested: поток получил запрос на остановку
* Suspended: поток приостановлен
* SuspendRequested: поток получил запрос на приостановку
* Unstarted: поток еще не был запущен
* WaitSleepJoin: поток заблокирован в результате действия методов Sleep или Join

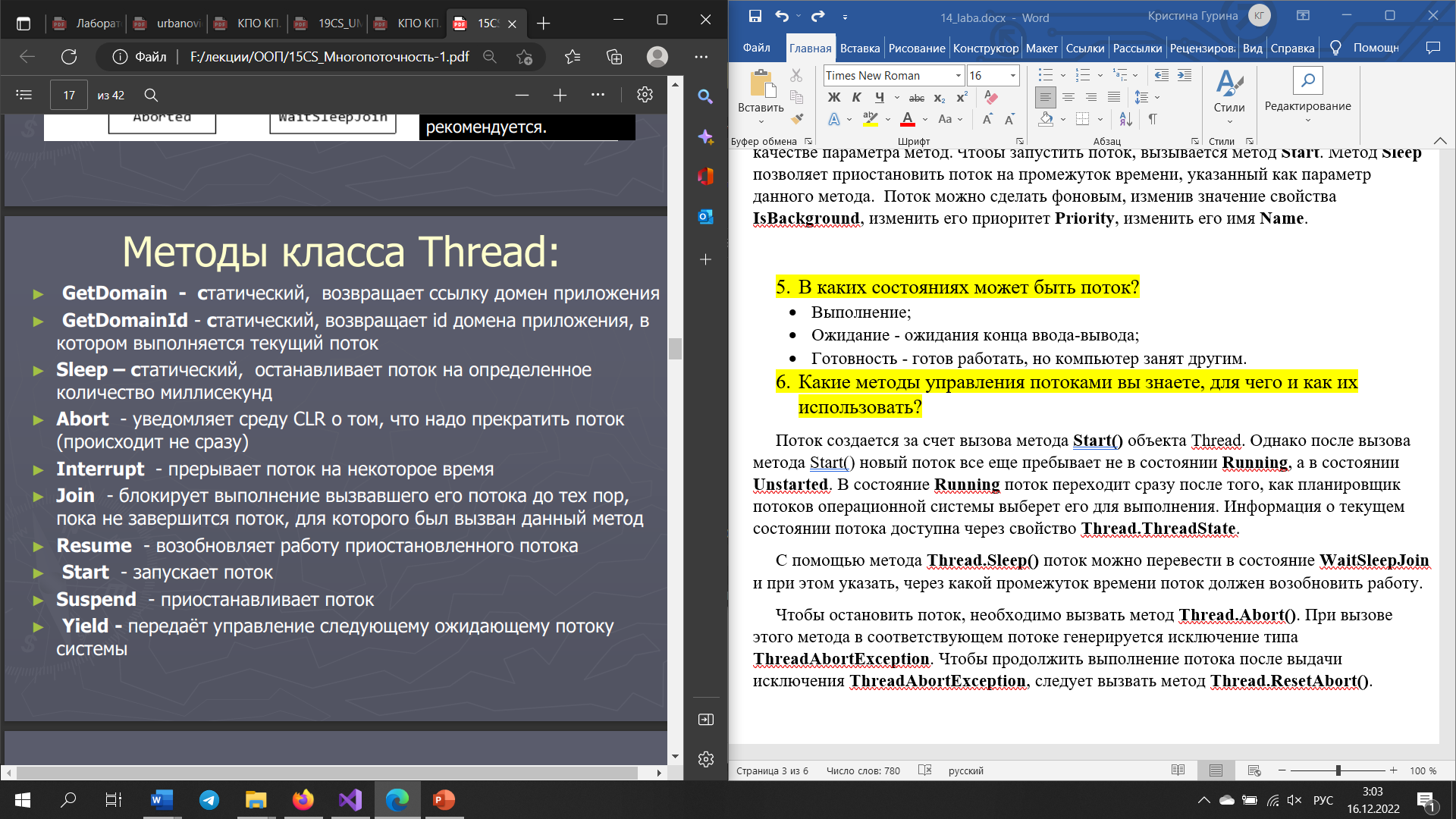
1. Какие методы управления потоками вы знаете, для чего и как их использовать?

Поток создается за счет вызова метода **Start()** объекта Thread. Однако после вызова метода Start() новый поток все еще пребывает не в состоянии **Running**, а в состоянии **Unstarted**. В состояние **Running** поток переходит сразу после того, как планировщик потоков операционной системы выберет его для выполнения. Информация о текущем состоянии потока доступна через свойство **Thread.ThreadState**.

С помощью метода **Thread.Sleep()** поток можно перевести в состояние **WaitSleepJoin** и при этом указать, через какой промежуток времени поток должен возобновить работу.

Чтобы остановить поток, необходимо вызвать метод **Thread.Abort()**. При вызове этого метода в соответствующем потоке генерируется исключение типа **ThreadAbortException**. Чтобы продолжить выполнение потока после выдачи исключения **ThreadAbortException**, следует вызвать метод **Thread.ResetAbort()**.

Если необходимо дожидаться завершения работы потока, можно вызвать метод **Thread.Join()**. Этот метод блокирует текущий поток и переводит его в состояние **WaitSleepJoin** до тех пор, пока не будет завершен присоединенный к нему поток.



1. Какие приоритеты потока вы знаете?

Перечисление ThreadPriority:

* Lowest;
* BelowNormal;
* Normal (по умолч.);
* AboveNormal;
* Highest;
* Real time.

Каждый процесс принадлежит одному из следующих классов приоритета:

\_класс приоритета простоя \_  
НИЖЕ \_ обычного \_ класса приоритета \_  
\_класс обычного приоритета \_  
ВЫШЕ \_ класса с нормальным \_ приоритетом \_  
класс с высоким \_ приоритетом \_  
\_класс приоритета в режиме реального времени \_

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

* **Пул потоков** – автоматизированный ссылочный тип-контейнер ссылок на рабочие потоки на языке C#. Определен в пространстве имен библиотеки типов **System.Treading**. По сути выполняет роль посредника между планировщиком задач операционной системы и потоками, реализованными в рамках.

**Для уменьшения издержек, связанных с созданием потоков**, платформа .NET поддерживает специальный механизм, называемый **пул потоков**. Пул состоит из двух основных элементов:

1) очереди методов

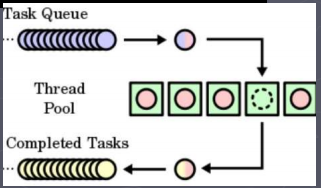
2) рабочих потоков.

1. 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

Mutex является одним из классов в .NET Framework, позволяющих обеспечить синхронизацию среди множества процессов.

**WaitOne()** – вход в критическую секцию

**ReleaseMutex()** – выход из нее (выход может быть произведен только в том же потоке выполнения, что и вход)

11.Что такое семафор? Поясните использование

объект синхронизации, позволяющий войти в заданный участок кода не более чем N потокам (N – ёмкость семафора)

получение и снятие блокировки в случае семафора может выполняться из разных потоках

\* класс System.Threading.Semaphore – между процессами

\* класс SemaphoreSlim – в рамках одного процесса

\* Wait() – получ. блокировки

\* Release() – снятие блокировки.

12.Что такое неблокирующие средства синхронизации?

**Неблокирующая синхронизация** — Неблокирующие средства синхронизации позволяют осуществлять совместный доступ к простым ресурсам нескольких потоков без блокировки, паузы или ожидания. Разделение доступа между потоками идёт за счёт атомарных операций и специальных, разработанных под конкретную задачу, механизмов блокировки.

подход в параллельном программировании в котором принят отказ от традиционных примитивов блокировки, таких, как семафоры, мьютексы и события. Разделение доступа между потоками идёт за счёт атомарных операций и специальных, разработанных под конкретную задачу, механизмов блокировки.

13.Для чего можно использовать класс Timer?

Данный класс позволяет запускать определенные действия по истечению некоторого периода времени.

Одна из перегрузок конструктора таймера принимает четыре параметра:

* объект делегата TimerCallback;
* объект, передаваемый в качестве параметра в метод делегата;
* количество миллисекунд, через которое таймер будет запускаться;
* интервал между вызовами метода делегата.

// устанавливаем метод обратного вызова

        TimerCallback tm = new TimerCallback(Count);

// создаем таймер

        Timer timer = new Timer(tm, num, 0, 2000);

