**Вопросы к лабораторной работе №16:**

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

Данная библиотека позволяет распараллелить задачи и выполнять их сразу на нескольких процессорах, если на целевом компьютере имеется несколько ядер.

Класс Task упрощает написание кода для создания новых потоков.

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

Распараллеливание позволяет использовать полностью вычислительные возможности многопроцессорных или многоядерных систем. Когда количество элементов велико, каждая задача может быть выполнена на отдельном ядре или процессоре, что позволяет одновременно выполнять несколько операций над разными элементами.

1. В чем основные достоинства работы с задачами по сравнению с потоками?

• Потоки могут легко разрушить адресное пространство процесса.

• Потоки необходимо синхронизировать при параллельном доступе (для чтения или записи) к памяти.

• Один поток может ликвидировать целый процесс или программу.

• Потоки существуют только в рамках единого процесса и, следовательно, не являются многократно используемыми.

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

1) Task task = new Task(() = > Console.WriteLine(“Hello!”)); //3

task.Start()

2) Task task = Task.Factory.StartNew(() => Console.WriteLine(“Hello!”); //1

3) Task task = Task.Run(() => Console.WriteLine(“Hello!”); //2

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

Wait() – приостановка тек. поток до завершения задачи

WaitAll() – статический, приостановка тек. поток до завершения всех указ. задач

WaitAny() – статический, приостановка тек. поток до завершения любой из указ. задач

1. Приведите пример синхронного запуска Task?

Action <object> method = x => Console.WriteLine(“yo”);

var task4 = new Task( method, TaskCreationOptions.LongRunnig);

task4. RunSynchronously();

1. Как создать задачу с возвратом результата?

int sum(int a,int b,int c){ return a + b + c;}

Task<int> result = new Task<int>(() => sum(rand\_num1.Result, rand\_num2.Result, rand\_num3.Result));

result.Start();

WriteLine($"Результат вычислений задачи {result.Id}:{result.Result} ");

1. Как обработать исключение, если оно произошло при выполнении Task?

Для обработки ошибок выражение **await** помещается в блок try.

При возникновении ошибки у объекта **Task**, представляющего асинхронную задачу, в которой произошла ошибка, свойство **IsFaulted** имеет значение true. Кроме того, свойство **Exception** объекта **Task** содержит всю информацию об ошибке.

Можно воспользоваться свойством **task.Exception.InnerException.Message**.

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

CancellationToken (токен отмены) - это механизм в .NET, который используется для отмены выполнения задачи или операции. Он представлен в виде структуры CancellationToken и используется совместно с классами CancellationTokenSource и Task.

При вызове метода Cancel() у CancellationTokenSource все задачи, которые используют связанный токен отмены, будут получать сигнал об отмене.

10.Как организовать задачу продолжения (continuation task)?

Задачи продолжения (continuation tasks) позволяют выполнить определенные действия после завершения другой задачи. Задача продолжения задается с помощью метода **ContinueWith**, который в качестве параметра принимает делегат **Action<Task>.**

Task<int> task1 = new Task<int>(() => Sum(4, 5));

Task task2 = task1.ContinueWith(sum => Display(sum.Result));

task1.Start();

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

* Объект ожидания (TaskAwaiter) используется при создании задач продолжения для выполнения дополнительных действий после завершения предыдущей задачи.
* Объект ожидания возвращается при использовании ключевого слова await или при вызове метода GetAwaiter() на объекте задачи.

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

Предоставляет удобные средства для параллельного выполнения итераций и операций в .NET. Он предназначен для упрощения разработки параллельных программ и повышения производительности путем распределения работы между несколькими ядрами процессора или потоками.

Одним из методов, позволяющих параллельное выполнение задач, является метод **Invoke.**

Метод **Parallel.For** позволяет выполнять итерации цикла параллельно.

Метод **Parallel.ForEach** осуществляет итерацию по коллекции, реализующей интерфейс IEnumerable, подобно циклу foreach, только осуществляет параллельное выполнение перебора.

Методы **Parallel.ForEach** и **Parallel.For** возвращают объект **ParallelLoopResult**, наиболее значимыми свойствами которого являются два следующих:

**IsCompleted**: определяет, завершилось ли полное выполнение параллельного цикла

**LowestBreakIteration**: возвращает индекс, на котором произошло прерывание работы цикла.

13.Приведите пример задачи с Parallel.For(int, int, Action<int>)

Parallel.For (1, 10, z => //1,10 – нач и кон знач счетчика,

{ int r=1; //z – тело цикла в виде объекта делегата

for (int y=1; y<=10; y++) // 3 4 6 8 9 2 5 1 7

r \*= z;

}

Paralle.For (1, 10, (int z, ParallelLoopState pd) => {

Console.WriteLine(z);

int r=1;

for (int y=1; y<=10; y++) // 3 4 6 8 9 1 2 7 5

r\*=z;

14.Приведите пример задачи с Parallel.ForEach

decimal factorial(int x)

{

decimal result = 1;

for (decimal i = 1; i <= x; i++)

{

result \*= i;

}

return result;} Parallel.ForEach(list, (a) => .WriteLine("Факториал числа " + a + " равен " + factorial(a)));

15.Приведите пример с Parallel.Invoke()

Parallel.Invoke

(

() => { Thread.Sleep(1000); WriteLine($"Выполняется задача {Task.CurrentId}"); Thread.Sleep(3000); },

() => { Thread.Sleep(1000); WriteLine($"Выполняется задача {Task.CurrentId}"); Thread.Sleep(3000); },

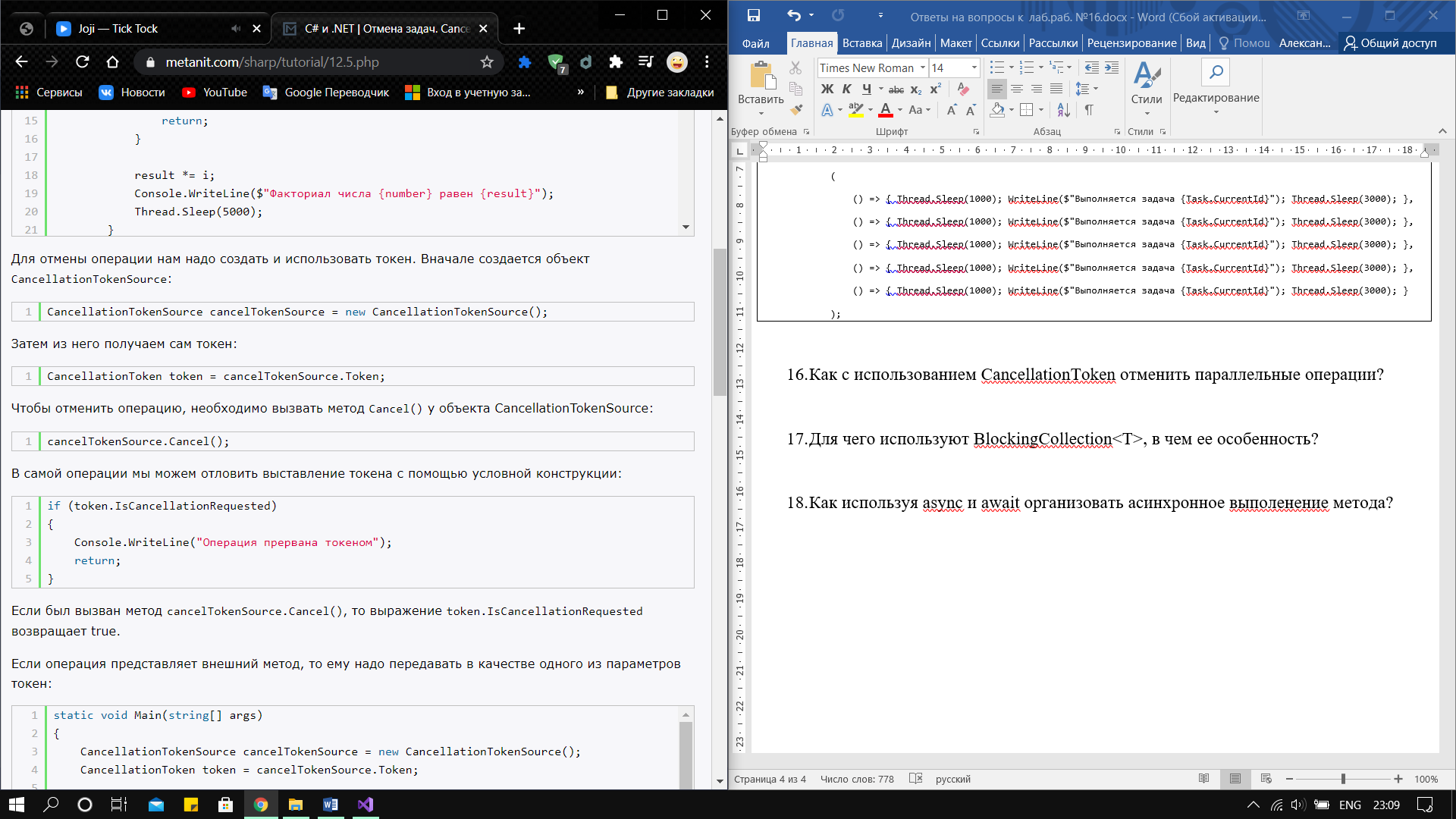
() => { Thread.Sleep(1000); WriteLine($"Выполняется задача {Task.CurrentId}"); Thread.Sleep(3000); },

() => { Thread.Sleep(1000); WriteLine($"Выполняется задача {Task.CurrentId}"); Thread.Sleep(3000); },

() => { Thread.Sleep(1000); WriteLine($"Выполняется задача {Task.CurrentId}"); Thread.Sleep(3000); }

);

16.Как с использованием CancellationToken отменить параллельные операции?



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.

Основная особенность BlockingCollection<T> заключается в том, что она предоставляет потокобезопасную коллекцию с автоматической блокировкой потоков, когда операции добавления или извлечения не могут быть выполнены немедленно.

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

**Асинхонный 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>

Асинхронный метод, как и обычный, может использовать любое количество параметров или не использовать их вообще. Однако асинхронный метод не может определять параметры с модификаторами **out** и **ref**.

Также стоит отметить, что слово **async**, которое указывается в определении метода, не делает автоматически метод асинхронным. Оно лишь указывает, что данный метод может содержать одно или несколько выражений **await**.