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

Библиотека параллельных задач TPL (Task Parallel Library). Позволяет распараллелить задачи и выполнять их сразу на нескольких процессорах(для создания многопоточных приложений).

**Задача (task) –абстракция более высокого уровня чем поток.**

класс Task

- описывает отдельную продолжительную операцию, которая запускается асинхронно в одном из потоков из пула потоков(можно запускать синхронно в текущем потоке)–подобна потокам, но абстракция более высокого уровня

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

Эффект от распараллеливания наблюдается на большом количестве элементов, потому что в этом случае преимущества от распределения работы между несколькими процессорами или потоками перевешивают накладные расходы на синхронизацию, коммуникацию и координацию. Когда количество элементов мало, то распараллеливание может даже ухудшить производительность, так как время, затрачиваемое на создание, запуск и ожидание параллельных задач, может быть больше, чем время, необходимое для выполнения задачи последовательно.

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

в Thread:  
1) нет механизма продолжений  
2) затруднено получение значения результата из потока  
3) повышение расхода памяти и замедление работы приложения

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

- For(), ForEach(), Invoke() – Являются параллельными аналогами циклов for и foreach, могут принимать аргумент типа ParallelOptions.

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

**Wait()** – приостанавливает текущий поток до завершения задачи  
**WaitAll()** – приостанавливает текущий поток до завершения всех указ. задач  
**WaitAny()** – приостанавливает текущий поток до завершения любой из указ. задач

# 6. Приведите пример синхронного запуска *Task*?

Action <object> method = x => Console.WriteLine(“yo”);  
var task4 = new Task( method, TaskCreationOptions.LongRunnig);  
task4. **RunSynchronously**();

# 7. Как создать задачу с возвратом результата?

Task<TResult> - описывает задачу, возвращает значение типа TResult

принимает аргументы типа:  
Func<TResult>  
Func<object, TResult>

# 8. Как обработать исключение, если оно произошло при выполнении *Task*?

**System.AggregateException**

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

CancellationTokenSource tokenSource = new CancellationTokenSource();  
//использует токен в двух задачах  
new Task(method, tokenSource.Token).Start();  
new Task(method, tokenSource.Token).Start();

//отменяем задачи  
tokenSource.Cancel()

# 10. Как организовать задачу продолжения (continuation task)?

сообщает задаче, что после ее завершения она д. продолжить делать что-то другое

1) Task task6 = Task.Run(() => Console.Write(“Doing..”));  
Task task7 = task6.ContinueWith (t => Console.Write(“Continuation”);

2) Task task8 = Task.Run(() => Console.Write(“One…”);  
Task task9 = Task.Run(() => Console.Write(“Two…”);  
Task continuation = Task.WhenAll(task8, task9).  
 ContinueWith ( t => Console.WriteLine(“Three..”);

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

Объект ожидания –это любой объект, имеющий методы OnCompleted() и GetResult() и свойство IsCompleted.

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

позволяет распараллеливать циклы и последовательность блоков кода  
For(), ForEach() //параллельные аналоги циклов for, foreach  
..Invoke() – шаблоны (на задачах, поддерживает исключения и токен отмены)

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

Parallel.For (1, 10, z => //1,10 – начальное и конечное значение 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;

Поддерживает императивность – оп-р, след. за вызовом метода вызван после завершения всех задач

# 14. Приведите пример задачи с *Parallel.ForEach*

ParallelLoopResult listFact = Parallel.ForEach<int>  
 (new List<int>() { 1, 3, 5, 8 }, //коллекция  
 Factorial); //делегат, вып. раз за итерацию  
 //перебир. эл-та коллекции

# 15. Приведите пример с *Parallel.Invoke()*

Позволяет распараллелить использование блоков операторов – набор задач, которые выполняются в одном потоке.  
Их могут запускать одновременно

Parallel.Invoke(() => new WebClient().DownloadFile(“http…”),  
 () => new WebClient().DownloadFile(“http…”));

# 16. Как с использованием *CancellationToken* отменить параллельные операции?

CancellationTokenSource ctks = new CancellationTokenSource();  
CancellationToken token = ctks.Token;

Parallel.ForEach<int>(  
 new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8},  
 new ParallelOptions { CancellationToken = token},  
 Factorial);

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

Используется в качестве оболочки для IProducerConsumerCollection<**T**> экземпляра и позволяет выполнять попытки удаления из коллекции, пока данные не будут доступны для удаления.

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

Обычно метод включает в себя хотя бы одно выражение await, которое отмечает точку, в которой метод не может продолжаться до тех пор, пока ожидаемая асинхронная операция не будет завершена.