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

* **Процесс** — это выполняющаяся программа, имеющая свое собственное пространство памяти, набор ресурсов (например, файловые дескрипторы) и права доступа. Каждый процесс может запускать несколько потоков. В операционной системе процессы изолированы друг от друга.
* **Домен приложения (AppDomain)** — это изолированная среда выполнения для приложения в рамках одного процесса. Он позволяет разделить выполнение разных частей приложения в одном процессе, предоставляя контроль над сборками, безопасность и управление памятью. Разные домены могут быть использованы для безопасного выполнения кода, так как они изолированы друг от друга.
* **Поток** — это наименьшая единица выполнения в рамках процесса. Потоки в одном процессе могут разделять память, данные и другие ресурсы. Поток выполняет инструкцию, в отличие от процесса, который может содержать несколько потоков. Каждый поток в процессе имеет свое собственное состояние исполнения, но использует общую память.

**Связь:** Потоки запускаются внутри процессов. Процесс может содержать несколько потоков, и эти потоки могут обмениваться данными между собой. Домен приложения работает в рамках одного процесса и может включать несколько потоков.

**2. Как получить информацию о процессах?**

В .NET можно использовать класс System.Diagnostics.Process. Пример:

using System;

using System.Diagnostics;

class Program

{

static void Main()

{

var processes = Process.GetProcesses();

foreach (var process in processes)

{

Console.WriteLine($"Имя: {process.ProcessName}, ID: {process.Id}, Статус: {(process.Responding ? "Отвечает" : "Не отвечает")}");

}

}

}

**3. Как создать и настроить домен?**

Для создания домена приложения в .NET используется класс AppDomain. Пример создания нового домена:

AppDomain newDomain = AppDomain.CreateDomain("NewDomain");

Console.WriteLine($"Новый домен: {newDomain.FriendlyName}");

**4. Как создать и настроить поток?**

Потоки создаются с помощью класса Thread. Пример:

using System;

using System.Threading;

class Program

{

static void Main()

{

Thread thread = new Thread(PrintMessage);

thread.Start();

thread.Join(); // Ожидаем завершения потока

}

static void PrintMessage()

{

Console.WriteLine("Привет из потока!");

}

}

**5. В каких состояниях может быть поток?**

Поток может находиться в нескольких состояниях:

* **New** (Новый) — поток был создан, но еще не запущен.
* **Runnable** (Запущенный) — поток выполняется или готов к выполнению.
* **Blocked** (Заблокирован) — поток ожидает освобождения ресурса или синхронизации.
* **Waiting** (Ожидание) — поток ожидает выполнения другой задачи или события.
* **Timed Waiting** (Ожидание с тайм-аутом) — поток ждет, но только до истечения заданного времени.
* **Terminated** (Завершен) — поток завершил выполнение.

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

* **Start()** — запускает поток.
* **Join()** — блокирует текущий поток, пока не завершится указанный поток.
* **Sleep()** — приостанавливает выполнение потока на указанное время.
* **Interrupt()** — прерывает поток.
* **Abort()** (устаревший) — завершает выполнение потока (не рекомендуется к использованию).

**7. Какие приоритеты потока вы знаете?**

Потоки могут иметь различные приоритеты:

* **ThreadPriority.Lowest** — низкий приоритет.
* **ThreadPriority.BelowNormal** — ниже нормального.
* **ThreadPriority.Normal** — нормальный приоритет.
* **ThreadPriority.AboveNormal** — выше нормального.
* **ThreadPriority.Highest** — высокий приоритет.

Пример установки приоритета потока:

Thread thread = new Thread(PrintMessage);

thread.Priority = ThreadPriority.AboveNormal;

thread.Start();

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

**Пул потоков** — это коллекция потоков, которые управляются системой и могут быть переиспользованы для выполнения задач. Он используется для уменьшения накладных расходов на создание и уничтожение потоков, что улучшает производительность при выполнении большого числа задач.

В .NET для работы с пулом потоков используется класс ThreadPool. Пример использования:

ThreadPool.QueueUserWorkItem(Task);

**9. Что такое критическая секция? Поясните использование.**

**Критическая секция** — это участок кода, который не должен выполняться одновременно несколькими потоками. Для синхронизации доступа к критической секции используется объект блокировки, например, lock в C#.

Пример:

private static readonly object lockObject = new object();

static void WriteToFile()

{

lock (lockObject)

{

// Здесь код, который должен выполняться только одним потоком

}

}

**10. Что такое мьютекс? Поясните использование.**

**Мьютекс** (Mutex) — это объект синхронизации, который используется для взаимного исключения между потоками, чтобы только один поток мог выполнить код в критической секции. В отличие от lock, мьютекс может работать между разными процессами.

Пример:

using (Mutex mutex = new Mutex())

{

mutex.WaitOne(); // захватываем мьютекс

// Критическая секция

mutex.ReleaseMutex(); // освобождаем мьютекс

}

**11. Что такое семафор? Поясните использование.**

**Семафор** — это объект синхронизации, который ограничивает количество потоков, которые могут одновременно получить доступ к ресурсу. Семафор может быть использован для управления доступом к ограниченным ресурсам.

Пример:

Semaphore semaphore = new Semaphore(2, 2); // Максимум 2 потока могут одновременно получить доступ

semaphore.WaitOne(); // Захват семафора

// Доступ к ресурсу

semaphore.Release(); // Освобождение семафора

**12. Что такое неблокирующие средства синхронизации?**

Неблокирующие средства синхронизации — это объекты или конструкции, которые позволяют потокам работать параллельно без необходимости блокировать друг друга. Например:

* **Monitor.TryEnter** — попытка захвата блокировки без блокировки потока.
* **Interlocked** — атомарные операции для безопасного обновления значений переменных.
* **SemaphoreSlim** — более легковесный семафор, оптимизированный для работы в пределах одного процесса.

Пример использования Interlocked:

int counter = 0;

Interlocked.Increment(ref counter); // Атомарное увеличение переменной

**13. Для чего можно использовать класс Timer?**

**Класс Timer** используется для выполнения периодических задач или одноразовых задач через определенный промежуток времени. Он полезен для тайм-аутов, периодических опросов или фоновых задач.

Пример:

Timer timer = new Timer(TimerCallback, null, 0, 2000); // Запуск задачи через 2 секунды

TimerCallback — это метод, который будет вызываться каждое указанное время (2000 миллисекунд).