



Параллелизм против многопоточности против асинхронного программирования: разъяснение

Хочу представить вашему вниманию перевод статьи [Concurrency vs Multi-threading vs Asynchronous Programming: Explained](#). В последнее время, я выступал на мероприя...

 By Aliaxandr

 Sep 09, 2017 11:57 AM ·  4 мин. на чтение ·  [Посмотреть оригинал](#)

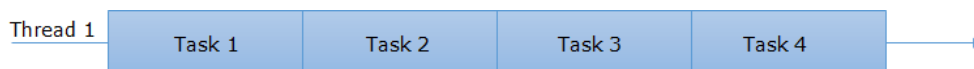
Хочу представить вашему вниманию перевод статьи [Concurrency vs Multi-threading vs Asynchronous Programming: Explained](#).

В последнее время, я выступал на мероприятиях и отвечал на вопрос аудитории между моими выступлениями о Асинхронном программировании, я обнаружил что некоторые люди путали многопоточное и асинхронное программирование, а некоторые говорили, что это одно и то же. Итак, я решил разъяснить эти термины и добавить еще одно понятие Параллелизм. Здесь есть две концепции и обе они совер-

шенно разные, первая синхронное и асинхронное программирование и вторая – однопоточные и многопоточные приложения. Каждая программная модель (синхронная или асинхронная) может работать в однопоточной и многопоточной среде. Давайте обсудим их подробно.

Синхронная программная модель – это программная модель, когда потоку назначается одна задача и начинается выполнение. Когда завершено выполнение задачи тогда появляется возможность заняться другой задачей. В этой модели невозможно останавливать выполнение задачи чтобы в промежутке выполнить другую задачу. Давайте обсудим как эта модель работает в одно и многопоточном сценарии.

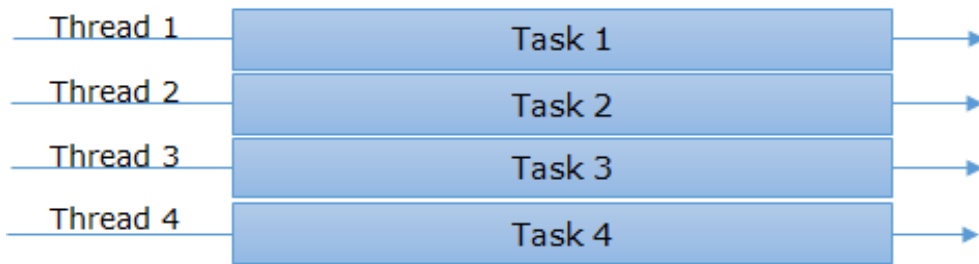
Однопоточность – если мы имеем несколько задач, которые надлежит выполнить, и текущая система предоставляет один поток, который может работать со всеми задачами, то он берет поочередно одну за другой и процесс выглядит так:



Здесь мы видим, что мы имеем поток (Поток 1) и 4 задачи, которые необходимо выполнить. Поток начинает выполнять поочередно одну за одной и выполняет их все. (Порядок, в котором задачи выполняются не влияет на общее выполнение, у нас может быть другой алгоритм, который может определять приоритеты задач.

Многопоточность – в этом сценарии, мы использовали много потоков, которые могут брать задачи и приступать к работе с ними. У нас есть пулы потоков (новые потоки также создаются, основываясь на по-

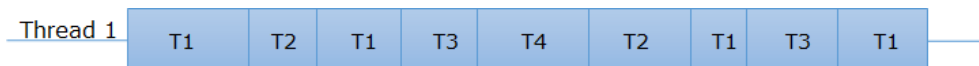
требности и доступности ресурсов) и множество задач. Итак, поток может работать вот так:



Здесь мы можем видеть, что у нас есть 4 потока и столько же задач для выполнения, и каждый поток начинает работать с ними. Это идеальный сценарий, но в обычных условиях мы используем большее количество задач чем количество доступных потоков, таким образом освободившийся поток получает другое задание. Как уже говорилось создание нового потока не происходит каждый раз потому что для этого требуются системные ресурсы такие как процессор, память и начальное количество потоков должно быть определенным.

Теперь давайте поговорим о Асинхронной модели и как она ведет себя в одно и многопоточной среде.

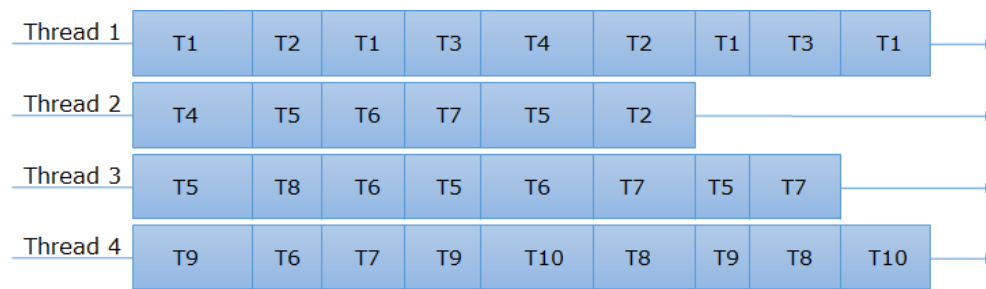
Асинхронная модель программирования – в отличие от синхронной программной модели, здесь поток однажды начав выполнение задачи может приостановить выполнение сохранив текущее состояние и между тем начать выполнение другой задачи.



Здесь мы можем видеть, что один поток отвечает за выполнение всех задач и задачи чередуются друг за другом.

Если наша система способно иметь много потоков

тогда все потоки могут работать в асинхронной модели как показано ниже:



Здесь мы можем видеть, что одна и та же задача скажем T4, T5, T6 ... обрабатывается несколькими потоками. Это красота этого сценария. Как мы можем видеть, что задача T4 начала выполнение первой Потоком 1 и завершен Потоком 2. Подобным образом задача T6 выполнена Потоком 2, Потоком 3 и Потоком 4. Это демонстрирует максимальное использование потоков.

Итак, до сих пор мы обсудили 4 сценария:

- Синхронный однопоточный
- Синхронный многопоточный
- Асинхронный однопоточный
- Асинхронный многопоточный

Давайте обсудим еще один термин – параллелизм.

Параллелизм

Проще говоря параллелизм способ обработки множества запросов одновременно. Так как мы обсуждали два сценария когда обрабатывались множественные запросы, многопоточное программирование и асинхронная модель (одно и многопоточная). В случае асинхронной модели будь она однопоточной или многопоточной, в то время, когда выполняются множество задач, некоторые из них приостанавливаются, а некоторые выполняются. Суще-

ствуется много особенностей, но это выходит за рамки этой публикации.

Как обсуждалось ранее новая эпоха за асинхронным программированием. Почему это так важно?

Преимущества асинхронного программирования

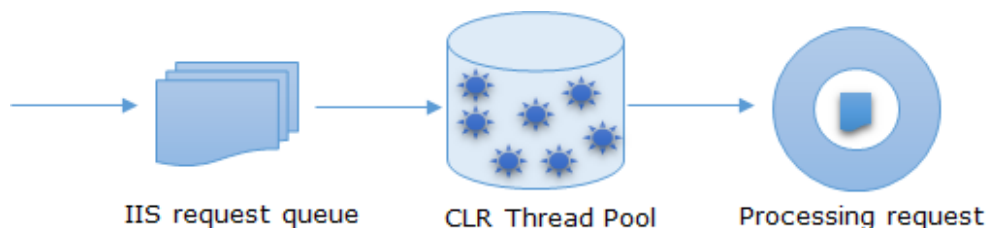
Существует две вещи очень важные для каждого приложения – удобство использования и производительность. Удобство использования потому что пользователь нажав кнопку чтобы сохранить некоторые данные что в свою очередь требует выполнения множества задач таких как чтение и заполнение данных во внутреннем объекте, установление соединения с SQL и сохранения его там. В свою очередь SQL запускается на другой машине в сети и работает под другим процессом, это может потребовать много времени. Таким образом если запрос обрабатывается одним процессом экран будет находиться в зависшем состоянии до тех пор, пока процесс не завершится. Вот почему сегодня многие приложения и фреймворки полностью полагаются на асинхронную модель.

Производительность приложения и системы также очень важны. Было замечено в то время как выполняется запрос, около 70-80% из них попадают в ожидания зависимых задач. Таким образом, это может быть максимально использовано в асинхронном программировании, где, как только задача передается другому потоку (например, SQL), текущий поток сохраняет состояние и доступен для выполнения другого процесса, а когда задача sql завершается,

любой поток, который является свободным, может заняться этой задачей.

Асинхронность в ASP.NET

Асинхронность в ASP.NET может стать большим стимулом для повышения производительности вашего приложения. Вот, как IIS обрабатывает запрос:



Когда запрос получен IIS, он берет поток из пула потоков CLR (IIS не имеет какого-либо пула потоков, а сам вместо этого использует пул потоков CLR) и назначает его ему, который далее обрабатывает запрос. Поскольку количество потоков ограничено, и новые могут быть созданы с определенным пределом, тогда если поток будет находится большую часть времени в состоянии ожидания, то это сильно ударит по вашему серверу, вы можете предположить, что это реальность. Но если вы пишете асинхронный код (который теперь становится очень простым и может быть написан почти аналогично синхронному при использовании новых ключевых слов `async / await`), то он будет работать намного быстрее, и пропускная способность вашего сервера значительно возрастет, потому что вместо ожидания какого-нибудь завершения, он будет доступен пулу потоков, для нового запроса. Если приложение имеет множество зависимостей и длительный процесс выполнения, то для этого приложения асинхронное программирование будет не меньшим благом.

Итак, теперь мы поняли разницу многопоточного, асинхронного программирования и преимущества, которые мы можем получить, используя асинхронную модель программирования.