Научная статья
Original article
УДК 004.032.24

# МНОГОПОТОЧНОСТЬ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C#. ПУЛ ПОТОКОВ

# MULTITHREADING IN THE C# PROGRAMMING LANGUAGE. THREAD POOL

**Опарин Иван Алексеевич**, Студент 2 курса, МИРЭА - Российский технологический университет (РТУ МИРЭА) 119454, Россия, г. Москва, проспект Вернадского, 78, e-mail: ivanopa.rin@yandex.ru

**Oparin Ivan Alekseevich**, 2nd year student, MIREA-Russian technological University (RTU MIREA) 119454, Russia, Moscow, Prospekt Vernadskogo, 78, e-mail: ivanopa.rin@yandex.ru

**Аннотация.** В данной статье раскрываются особенности применения многопоточности и пула потоков при создании современных приложений и параллельной обработки данных, а также обозначаются правила их использования. В качестве языка программирования, на котором раскрываются данные аспекты создания современных программ, был выбран C#.

**Annotation.** This article describes the features of the use of multithreading and thread pool in the creation of modern applications and parallel data processing, as well as the rules of their use. C# was chosen as the programming language in which these aspects of the creation of modern programs are revealed.

**Ключевые слова:** многопоточность, пул потоков, асинхронность, поток

**Keywords:** multithreading, thread pool, asynchrony, thread

**Применение многопоточности в современных приложениях**

Большинству современных приложений приходится выполнять сразу несколько задач одновременно, при этом, чтобы программное обеспечение корректно реагировало на дальнейшие действия пользователя без зависаний, существует механизм многопоточности, позволяющий решить данную проблему. Данный механизм используется во многих задачах, приведём некоторые из них:

1. Создание отзывчивых и способных к параллельной работе пользовательских интерфейсов [2, с. 961].
2. Обеспечение одновременной обработки запросов.
3. Программирование на основе параллельного подхода.
4. Упреждающее выполнение (выполнение некоторых задач заранее, когда ресурсы машины более свободны).

Дадим общие определения в данной предметной области для дальнейшей работы с ними.

Многопоточность – общий механизм, благодаря которому программа может выполнять код одновременно [1, с. 558].

Поток – путь выполнения работы программы, который может проходить в независимости от других таких путей.

Поток также можно представить как некую среду, в которой может выполняться какой-либо процесс, что позволяет использовать его параллельно с другими потоками. Каждый поток запускается с помощью работы операционной системы, которая обеспечивает изолированную среду для выполнения определённой программы. В однопоточном приложении внутри данной изолированной среды работает лишь один поток, что, однако, позволяет ему получить монопольный доступ к всей среде, выдаваемой операционной системой. В многопоточном приложении внутри единственной среды выполнения происходит работа множества потоков, которые разделяют ресурсы данной среды, как, например, память. Это является одной из причин применения

многопоточности, к примеру, один из потоков может получать данные, другой, по мере их поступления, обрабатывать их, что ускоряет работу всего приложения [2, с. 961].

Приведём пример создания потока:

```
Static void Main()
{
Thread t = new Thread(() =>
{
for(int i = 0; i < 100; i++) Console.Write(i);
});
t.Start();
}
```

В данном примере создаётся поток, который принимает метод, выполняющий вывод чисел от 1 до 99 на консоль. Здесь из главного потока программы создаётся новый поток t, в котором запускается метод вывода чисел от 1 до 99 на экран. Данный пример демонстрирует создание потока, его запуск и работу.

В современных приложениях, написанных на языке программирования C#, клиентские программы запускаются в одном единственном главном потоке, который автоматически создаётся операционной системой. В дальнейшем в процессе работы программы он и будет существовать как главный поток, также он будет единственным, если программист не создаст новый поток как в примере выше. Хотя стоит отметить, что сама среда выполнения программы может иногда сама создать новые потоки. В C#, например, среда CLR создаёт новые потоки при сборке мусора и финализации объектов.

На одноядерных компьютерах два потока работаю последовательно, хотя возникает ощущение, что они работаю параллельно. Это связано с тем, что каждому потоку выделяются кванты времени (для Windows это 20 миллисекунд)

поочерёдно, что позволяет работать им якобы параллельно. Но в многоядерных машинах потоки уже на самом деле могут работать параллельно без переключения между собой. Конечно же, сами переключения на одноядерных машинах занимают время и даже память машины.

Однако работа с потоками не ограничивается их созданием и запуском. Существует много других операций с ними, которые позволяют решать множество практических задач. Например, потоки можно заблокировать, переименовать (что полезно для отладки) и другое. Например:

```
Static void Main()
{
Thread t = new Thread(() => {for(int i = 0; i < 20; i++) Console.Write(i);});
t.Start();
t.Join();
}
```

В данном примере создаётся новый поток t, который в процессе своей работы выводит числа от 0 до 19, при этом основной поток «ждёт», пока данный поток t не завершит свою работу.

Данный пример демонстрирует блокировку основного потока. Поток считается заблокированным, если выполнение какой-либо задачи потока приостановлено. При этом поток немедленно выделяет свои кванты времени другим потокам, а также не будет требовать их от среды в дальнейшем, пока он не будет разблокирован. При блокировке или разблокировке потока происходит переключение контекста операционной системой, с чем связано появление некоторых накладных расходов (в C# составляющие одну или две микросекунды в зависимости от процессора).

Стоит отметить, что блокировка потоков не является удачным вариантом, так как каждый поток в период всего своего существования связывает около 1

Мбайт памяти и служит причиной других накладных расходов со стороны системы.

Для каждого потока создаётся собственный стек в памяти компьютера, это означает использование локальных данных полностью независимо от других потоков. Однако если поток ссылается на общие данные с другими потоками, это может служить причиной некорректной работы программы, если эти данные изменяются в одном потоке, то они изменяются и в других. Данная ситуация показывает ситуацию разделяемого состояния, в которых может произойти одна из разновидностей несистематических ошибок, связанных с многопоточностью. Одно из решений данной проблемы – избегать применения разделяемых состояний, хотя это не единственный вариант решения проблемы, но по возможности лучше использовать именно его.

Один из других способов – использование блокировки другого потока при его попытке взаимодействовать с разделяемыми данными для монопольного их использования лишь одним потоком, если в этом есть необходимость. Код, где применяется данный подход, называет безопасным в отношении потоков. Хотя данное решение не является абсолютно правильным, так как довольно легко забыть заблокировать доступ к полю, после чего могут произойти довольно существенные проблемы, например, взаимоблокировка потоков.

Все потоки, создаваемые явно, являются потоками переднего плана. Данные потоки удерживают приложение в процессе работы до того момента, как завершит работу последний из них, даже если основной поток закончил свою работу. Однако существуют фоновые потоки, которые не имеют такого свойства. При завершении работы всеми потоками переднего плана фоновые потоки принудительно завершаются. Это является ещё одним источником ошибок в современных многопоточных приложениях, так как фоновые потоки могут не завершить некоторые задачи, требуемые для корректной работы программы. Иначе говоря, назначение потоку фонового состояния требует мер предосторожности, если данный поток несёт важные задачи. Также иногда

данные потоки не смогут завершиться самостоятельно, одно из решений данный проблемы является использование временной задержки для завершения всех потоков программы перед её завершением.

Всё вышесказанное иллюстрирует необходимость использования многопоточности в большинстве современных приложений, однако существуют немалые проблемы, связанные с производительностью данного механизма в целом и в написании программ, использующих данный механизм. Одно из решений данных проблем предполагает использование пула потоков. Пул потоков – это механизм многопоточности, предлагающий накопитель для заранее созданных и применяемых потоков [1, с. 572].

Каждый раз при создании нового потока тратится примерно несколько сотен микросекунд на настройку нового потока средой выполнения, например, на создание стека локальных переменных. Для современных компьютеров это очень большая задержка, хотя, конечно, для разных процессоров это число будет различно. Для снижения этих накладных расходов и существует пул потоков. Организация данного механизма очень важно для эффективного параллельного программирования и реализации мелкомодульного параллелизма [1, с. 572]. Пул потоков позволяет использовать потоки без их настройки, что увеличивают скорость работы приложений.

Хотя применительно к языку программирования C# существуют некоторые ограничения использования потоков из пула:

- Потоки из пула сложнее отлаживать, так как им нельзя задать имя.
- Данные потоки - фоновые.
- Блокировка потоков из пула может повлечь за собой снижение производительности.

Также стоит отметить, что при изменении свойств потоков из пула изменения не будут сохранены, то есть при возвращении потока обратно в пул изменения будут аннулированы. Применительно к C#, существует свойство для

того, чтобы узнать, является ли данный поток потоком из пула, это свойство – Thread.CurrentThread.IsThreadPoolThread.

Приведём пример запуска потоков из пула. Данный процесс отличается от запуска обычного потока:

Task.Run(() => Console.WriteLine(“Это поток из пула”));

В данном примере берётся поток из пула и сразу же запускается с задачей вывода строки на консоль.

Пул потоков используется во многих различных технологиях, основанных на C#, таких как:

- WCF (Windows Communication Foundation) – технология для обмена данными между различными приложениями.
- ASP.NET – технология для создания веб-приложений.
- Remoting – компонент для реализации межпроцессного взаимодействия.
- ASMX – технология для создания SOAP веб-сервисов.

Пул потоков во многом необходим для более производительной работы всего компьютера в целом, так как данный механизм многопоточности гарантирует, что излишки вычислительных работ процессора не приведут к превышению лимита процессора. Данное состояние характеризуется тем фактом, что активных (то есть работающих) потоков становится больше, чем ядер процессора, что в итоге приводит к выделению отдельным потокам квантов времени. Это в свою очередь наносит ущерб производительности, так как выделение определенного количества времени приводит к переключению контекста выполнения, а также может приводить к меньшему использованию кэшей процессора.

Для предотвращения данной ситуации среда выполнения сама производит настройку потоков для их запуска. При этом можно сказать, что данная среда использует такое число задач, которое соответствует числу ядер процессора, что

даёт наилучшую скорость и стабильность работы всего компьютера. Далее в процессе появления новых задач и потоков среда лишь регулирует степень параллелизма по алгоритмам поиска экстремумов, настраивая тем самым рабочую нагрузку на процессор. Всё это даёт хороший прирост производительности даже при использовании соперничающих задач.

Потоки, взятые из пула, будут наиболее эффективны при соблюдении следующих условий:

- Метод, передаваемый потоку из пула, должен выполняться менее 250 миллисекунд, так как среда выполнения при этом сможет выполнять расширенные изменения его использования и корректировки данного потока.
- Задания потока не будут заблокированными, блокировка отрицательно влияет на производительность пула потоков, так как среда полагает, что поток выполняет важные вычисления, хотя поток по факту бездействует. Это приводит к большей возможности превышения лимита процессора [1, с. 574].

Отсутствие чистоты пула потоков приводит к меньшей производительности всего компьютера, поддержание чистоты пула важно при требовании к большой производительности системы.

В заключение можно добавить, что понимание принципов многопоточности при создании современных приложений – ключ к высокой скорости приложений, корректности их работы. Без многопоточности не может обойтись ни одно большое программное обеспечение, поэтому знание данных принципов может иметь большие преимущества при разработке программных продуктов.

**Список литературы**

1.Албахари Д., Албахари Б. C# 7.0 Справочник. Полное описание языка. – Издательство «Диалектика», 2019. – 1024 с.

2. Мэтью М. Windows Presentation Foundation в .NET 4.5 с примерами на C# 5.0. – Издательство «Вильямс», 2013. – 1018 с.

3. Рихтер Д. CLR via C#. – Издательство «Питер», 2017. – 896 с.
4. Вагнер Б. Наиболее эффективное программирование на C#. – Издательство «Вильямс», 2017. – 240 с.

**References**

1. Albahari D., Albahari B. C# 7.0 in a nutshell. – Publishing «Dialektika», 2019. – 1024 p.
2. Matthew M. Pro WPF 4.5 in C#. – Publishing «Williams», 2013. – 1018 p.
3. Richter D. CLR via C#. – Publishing «Piter», 2017. – 896 p.
4. Wagner B. Effective C#. – Publishing «Williams», 2017. – 240 p.

*© Опарин И.А., 2022 Научно-образовательный журнал для студентов и преподавателей «StudNet» №6/2022.*

**Для цитирования:** Опарин И.А. МНОГОПОТОЧНОСТЬ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C#. ПУЛ ПОТОКОВ // Научно-образовательный журнал для студентов и преподавателей «StudNet» 6/2022