Многопоточная программа состоит из двух иди более частей, выполняемых параллельно. Каждая часть такой программы называется *потоком* и определяет отдельный путь выполнения команд. Таким образом,

многопоточная обработка является особой формой многозадачности.

С выпуском версии 4.0 в среде .NET Framework появились два важных дополнения, имеющих отношение к многопоточным приложениям. Первым из них является TPL (Task Parallel Library — Библиотека распараллеливания задач), а вторым — PLINQ (Parallel LINQ — Параллельный язык интегрированных запросов). Оба дополнения поддерживают параллельное программирование и позволяют использовать преимущества, предоставляемые многопроцессорными (многоядерными) компьютерами в отношении обработки данных. Кроме того, библиотека TPL упрощает создание многопоточных приложений и управление ими.

В силу этого многопоточная обработка, опирающаяся на

TPL, рекомендуется теперь как основной подход к разработке многопоточных приложений.

Тем не менее накопленный опыт создания исходной многопоточной подсистемы

по-прежнему имеет значение по целому ряду причин. Во-первых, уже существует

немалый объем унаследованного кода, в котором применяется первоначальный подход

к многопоточной обработке. Если приходится работать с таким кодом или сопровождать

его, то нужно знать, как работает исходная многопоточная система. Во-вторых,

в коде, опирающемся на TPL, могут по-прежнему использоваться элементы исходной

многопоточной системы, и особенно ее средства синхронизации. И в-третьих, несмотря

на то что сама библиотека TPL основывается на абстракции, называемой *задачей,*

она по-прежнему неявно опирается на потоки и потоковые средства, описываемые

в этой главе. Поэтому для полного усвоения и применения TPL потребуются твердые

знания материала, излагаемого в этой главе.

**Основы многопоточной обработки**

Различают две разновидности многозадачности: на основе процессов и на основе

потоков. В связи с этим важно понимать отличия между ними. ***Процесс***фактически

представляет собой исполняемую программу. Поэтому ***многозадачность на основе процессов***

— это средство, благодаря которому на компьютере могут параллельно выполняться

две программы и более.

***Поток***представляет собой координируемую единицу исполняемого кода. Своим

происхождением этот термин обязан понятию "поток исполнения". При организации

многозадачности на основе потоков у каждого процесса должен быть по крайней мере

один поток, хотя их может быть и больше. Это означает, что в одной программе одновременно

могут решаться две задачи и больше. Например, текст может форматироваться

в редакторе текста одновременно с его выводом на печать, при условии, что оба

эти действия выполняются в двух отдельных потоках.

Отличия в многозадачности на основе процессов и потоков могут быть сведены к

следующему: многозадачность на основе процессов организуется для параллельного

выполнения программ, а многозадачность на основе потоков — для параллельного выполнения

отдельных частей одной программы.

Главное преимущество многопоточной обработки заключается в том, что она позволяет

писать программы, которые работают очень эффективно благодаря возможности

выгодно использовать время простоя, неизбежно возникающее в ходе выполнения

большинства программ. Как известно, большинство устройств ввода-вывода, будь то

устройства, подключенные к сетевым портам, накопители на дисках или клавиатура,

работают намного медленнее, чем центральный процессор (ЦП). Поэтому большую

часть своего времени программе приходится ожидать отправки данных на устройство

ввода-вывода или приема информации из него. А благодаря многопоточной обработке

программа может решать какую-нибудь другую задачу во время вынужденного

простоя. Например, в то время как одна часть программы отправляет файл через

соединение с Интернетом, другая ее часть может выполнять чтение текстовой информации,

вводимой с клавиатуры, а третья — осуществлять буферизацию очередного

блока отправляемых данных.

Поток может находиться в одном из нескольких состояний. В целом, поток может

быть *выполняющимся; готовым к выполнению,* как только он получит время и ресурсы

ЦП; *приостановленным,* т:е. временно не выполняющимся; *возобновленным* в дальнейшем;

*заблокированным* в ожидании ресурсов для своего выполнения; а также *завершенным,*

когда его выполнение окончено и не может быть возобновлено.

В среде .NET Framework определены две разновидности потоков: *приоритетный*

и *фоновый.* По умолчанию создаваемый поток автоматически становится приоритетным,

но его можно сделать фоновым. Единственное отличие приоритетных потоков от

фоновых заключается в том, что фоновый поток автоматически завершается, если в его

процессе остановлены все приоритетные потоки.

В связи с организацией многозадачности на основе потоков возникает потребность

в особого рода режиме, который называется *синхронизацией* и позволяет координировать

выполнение потоков вполне определенным образом. Для такой синхронизации

в C# предусмотрена отдельная подсистема, основные средства которой рассматриваются

в этой главе.

Все процессы состоят хотя бы из одного потока, который обычно называют *основным,*

поскольку именно с него начинается выполнение программы. Следовательно,

в основном потоке выполнялись все приведенные ранее примеры программ. Из основного

потока можно создать другие потоки.

Классы, поддерживающие многопоточное программирование, определены в пространстве

имен System.Threading. Поэтому любая многопоточная программа на C#

включает в себя следующую строку кода.

using System.Threading;