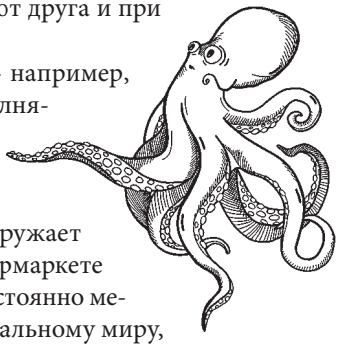
1 Знакомство с конкурентностью

В этой главе

- ✓ Почему конкурентность — важная тема, которую стоит изучить
- ✓ Как измерять производительность систем
- ✓ Какие бывают уровни конкурентности

Выгляните в окно и повнимательнее присмотритесь к окружающему миру. Разве все вокруг происходит по линейной, последовательной схеме? Нет, скорее, вы убедитесь, что мир больше похож на сложное переплетение компонентов, которые все одновременно функционируют независимо друг от друга и при этом взаимодействуют друг с другом.



Хотя люди склонны мыслить последовательно — например, когда они просматривают список текущих дел и выполняют задачи одну за другой, — реальный мир устроен намного сложнее. В нем гораздо больше параллельного, чем последовательного. Взаимосвязанные события случаются одновременно. Конкурентность окружает нас повсюду — от хаотичной толкотни в людном супермаркете до слаженных перемещений футболистов на поле и постоянно меняющегося потока дорожного движения. Подобно реальному миру, компьютеры должны уметь выполнять операции в конкурентном

режиме, чтобы с их помощью можно было моделировать, имитировать и анализировать сложные явления реального мира.

Принцип конкурентности в компьютерных технологиях заключается в том, что система работает сразу с несколькими задачами. Такой системой может быть программа, компьютер или сеть компьютеров. Без конкурентных вычислений наши приложения неизбежно отстали бы от сложности окружающего мира.

Но когда мы начинаем глубже погружаться в тему конкурентности, могут возникнуть определенные вопросы. Первый вопрос (на случай, если я вас еще не убедил) — почему вообще стоит уделять внимание конкурентности?

Почему конкурентность важна

Конкурентность играет чрезвычайно важную роль в разработке программных продуктов. Спрос на высокопроизводительные приложения и конкурентные системы таков, что конкурентное программирование становится критически важным навыком для разработчиков.

Конкурентное программирование — не новое понятие, но за последние годы внимание к нему существенно усилилось. По мере того как в современных компьютерных системах растет количество ядер и процессоров, конкурентное программирование превращается в необходимое требование для разработки ПО. Компании ищут разработчиков, которые уверенно владеют конкурентным программированием, потому что оно часто оказывается единственным выходом в ситуациях, когда нужна высокая производительность, но вычислительные ресурсы ограничены.

Самое значительное преимущество конкурентного программирования (и исторически первая причина для исследований в этой области) состоит в том, что оно помогает повысить производительность систем. Разберемся, почему это важно.

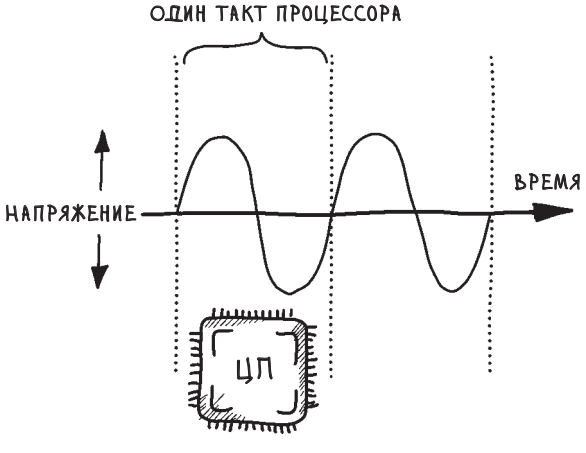
Конкурентность помогает повысить производительность

Если нужно повысить производительность, почему бы просто не купить более быстрые компьютеры? Десятки лет назад так и поступали, но в какой-то момент выяснилось, что быстрые компьютеры уже не решают проблему.

Закон Мура

В 1965 году Гордон Мур, один из основателей Intel, обнаружил любопытную закономерность. Новые модели процессоров появлялись примерно через два года после своих предшественников, и каждый раз количество транзисторов на них приблизительно удваивалось. Мур заключил, что количество транзисторов — и, как следствие, тактовая частота процессоров — удваивается через каждые 24 месяца. Это наблюдение стало известно как *закон Мура*. Для разработчиков это означало, что нужно было подождать всего два года, чтобы приложение стало работать вдвое быстрее.

Проблема в том, что около 2002 года правила изменились. Как сказал знаменитый эксперт по C++ Херб Саттер (Herb Sutter), «бесплатные обеды закончились»[1]. Обнаружилась фундаментальная связь между физическим размером процессора и скоростью обработки (частотой процессора). Время, за которое выполняется операция, зависит от длины цепи и скорости света. Проще говоря, транзисторы (главные структурные блоки компьютерных микросхем) нельзя добавлять в процессор бесконечно. Немаловажную роль играет и повышение температуры. Стало невозможно улучшать производительность только за счет того, чтобы повышать частоту процессора. С этого момента начался так называемый *кризис многоядерности*.



Развитие отдельных процессоров (в смысле повышения тактовой частоты) остановилось из-за физических ограничений, а необходимость повышать производительность систем осталась. Производители переключились на горизонтальное расширение в форме многопроцессорных систем, что заставило разработчиков, архитекторов и создателей языков адаптироваться к архитектурам со множественными вычислительными ресурсами.

Главный вывод из этого исторического обзора таков: важнейшим преимуществом конкурентности (и исторически первой причиной для исследований в этой

[1] Herb Sutter. «The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software», http://www.gotw.ca/publications/concurrency-ddj.htm.

области) стало то, что она позволяет повысить производительность системы так, чтобы при этом эффективно использовать дополнительные вычислительные ресурсы. И здесь возникают два важных вопроса: как измерить эту производительность и как ее улучшить?

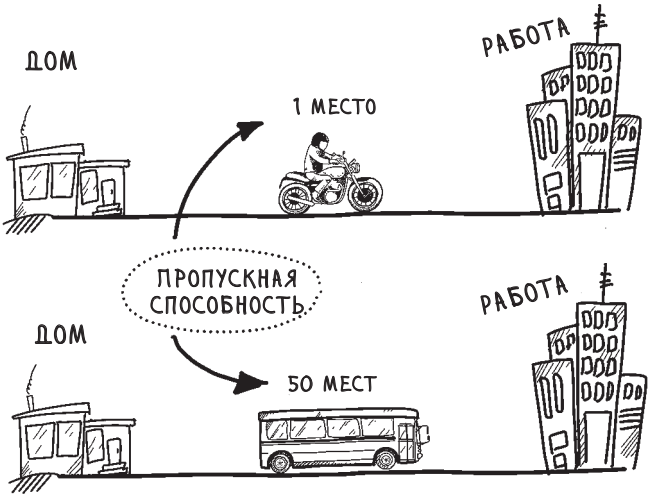
Задержка и пропускная способность

В компьютерных технологиях производительность можно измерять по-разному в зависимости от того, как мы рассматриваем вычислительную систему. Один из способов повысить объем работы — сократить время, за которое выполняются отдельные операции.

Допустим, вы ездите из дома на работу на мотоцикле, и дорога в одну сторону занимает один час. Для вас важно, насколько быстро вы добираетесь до работы, поэтому вы измеряете производительность системы по этой метрике. Если вы будете ехать быстрее, то быстрее окажетесь на работе. С точки зрения компьютерной системы такой эффект называется *задержкой* (latency). Задержка — это метрика, которая измеряет, сколько времени занимает выполнение отдельной задачи от начала до конца.

Теперь представьте, что вы работаете в управлении общественного транспорта и вам поручено повысить производительность автобусной сети. На этот раз ваша задача не в том, чтобы один человек доехал до работы быстрее, а в том, чтобы увеличить количество людей, которые добираются из дома на работу в единицу времени. Такой эффект называется *пропускной способностью* (throughput): эта метрика измеряет количество задач, которые система может выполнить за тот или иной период времени.

Очень важно понимать, чем задержка отличается от пропускной способности. Даже если мотоцикл едет вдвое быстрее автобуса, пропускная способность автобуса в 25 раз больше. (Мотоцикл перевозит одного человека на заданное расстояние за 1 час, тогда как автобус перевозит 50 человек на то же расстояние за 2 часа: если усреднить по времени, получается 25 человек в час.) Иначе говоря, более высокая пропускная способность системы не обязательно означает более низкую задержку. Если оптимизировать производительность, то улучшение по одному показателю (например, по пропускной способности) может привести к ухудшению по другому (например, по задержке).



ОДИНАКОВАЯ ЗАДЕРЖКА, ПОВЫШЕННАЯ ПРОПУСКНАЯ СПОСОБНОСТЬ

Конкурентность помогает снизить задержку. Например, если задача занимает много времени, ее можно разбить на меньшие задачи, которые выполняются параллельно, что сократит общее время выполнения. Конкурентность также помогает повысить пропускную способность, если несколько задач обрабатываются одновременно.

Кроме того, конкурентность может скрывать задержку. Когда вы ждете важного звонка, автобуса до работы и т. д., можно просто ждать, а можно направить вычислительные ресурсы на другую деятельность. Например, в ожидании автобуса можно проверить электронную почту. Таким образом вы фактически выполняете несколько задач одновременно и скрываете задержку благодаря тому, что эффективно используете время ожидания. Скрытие задержки играет ключевую роль при разработке систем с малым временем отклика и применяется в задачах, где фигурирует ожидание.

Итак, конкурентность может улучшить производительность системы по трем основным направлениям:

- сократить задержку (чтобы единица работы выполнялась быстрее);
- скрыть задержку (чтобы система делала что-то еще во время операции с высокой задержкой);
- повысить пропускную способность (чтобы система выполняла больше работы в единицу времени).

Мы увидели, как конкурентность помогает повысить производительность, а теперь рассмотрим, для чего еще она применяется на практике. Ранее в этой главе упоминалось, что конкурентность необходима, чтобы моделировать сложный мир, который нас окружает. Теперь можно конкретнее поговорить о том, как с ее помощью решать большие или сложные задачи вычислительными средствами.

Конкурентность помогает решать сложные и большие задачи

Многие задачи, которые стоят перед разработчиками систем, имеющих отношение к реальному миру, настолько сложны, что решать их в последовательной системе практически нереально. Сложность может быть обусловлена размером задачи или тем, насколько трудно разобраться в отдельной части разрабатываемой системы.

Масштабируемость

Размер задачи связан с *масштабируемостью* (scalability); это характеристика системы, производительность которой можно повысить, если добавить дополнительные ресурсы. Масштабировать систему можно в двух направлениях: по вертикали или по горизонтали.

Вертикальное масштабирование повышает производительность, увеличивая объем памяти (чтобы нарастить существующие вычислительные ресурсы) или заменяя процессор на более мощный. В этом случае масштабируемость ограничена: очень сложно повысить скорость отдельного процессора, зато очень легко упереться в потолок производительности. Кроме того, переход на более мощные вычислительные ресурсы (например, покупка суперкомпьютера) обходится недешево, потому что вам приходится все больше платить за мощные облачные экземпляры или оборудование, получая при этом все меньше выигрыша.

Можно еще немного выгадать, если удастся сократить процессорное время, которое необходимо для той или иной единицы работы, но в конечном счете придется масштабировать систему по горизонтали. *Горизонтальное масштабирование* увеличивает производительность программ или систем за счет того, что нагрузка распределяется между существующими и новыми вычислительными ресурсами. Пока есть возможность добавлять новые ресурсы, можно наращивать производительность системы. В этом случае проблемы с масштабируемостью возникнут не так скоро, как при вертикальном масштабировании.

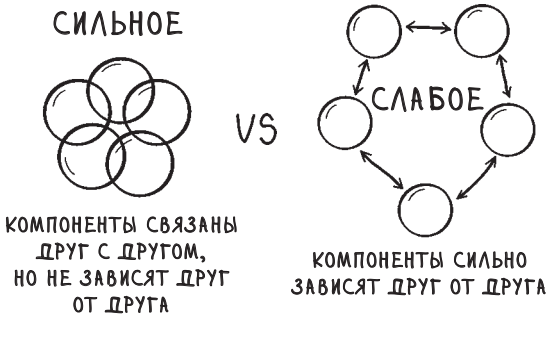
Компьютерная отрасль в основном тяготеет к горизонтальному масштабированию. Основные причины этого — потребность в системах реального времени, большие объемы данных, популярный подход «надежность через избыточность», а также оптимизированное потребление ресурсов за счет их совместного использования при переходе на облачные и SaaS-среды.

Горизонтальное масштабирование требует конкурентности на уровне системы, и одного компьютера может оказаться недостаточно. Комплексы соединенных машин — так называемые *вычислительные кластеры* — решают задачи обработки данных за разумное время.

Ослабление сцепления

У больших задач есть еще один важный аспект — сложность. К сожалению, со временем системы не становятся менее сложными, если только разработчики не прилагают специальных усилий. Компании хотят, чтобы их продукты были более мощными и функциональными, а от этого неизбежно усложняется и кодовая база, и инфраструктура, и работа по сопровождению. Специалистам приходится искать и реализовывать разные архитектурные приемы, чтобы упрощать системы и делить их на более простые независимые единицы, которые взаимодействуют друг с другом.

В разработке ПО практически всегда приветствуется разделение обязанностей. Создавать слабо сцепленные системы позволяет основной инженерный принцип — «разделяй и властвуй». Если группировать логически и функционально связанные фрагменты кода (компоненты с *сильным сцеплением*, tightly coupled) и разделять не связанные фрагменты (компоненты со *слабым сцеплением*, loosely coupled), то приложения становится проще понимать и тестировать, а количество ошибок сокращается — по крайней мере теоретически.



Так что конкурентность можно рассматривать и как стратегию ослабления сцепления. Если разделять функциональность между модулями или единицами конкурентности, то отдельные части смогут лучше сосредоточиться на конкретной функциональности, их будет проще сопровождать, а общая сложность системы уменьшится.

Разработчики отделяют то, *что* нужно сделать, от того, *когда* это будет сделано. Это кардинально улучшает производительность, масштабируемость, надежность и внутреннюю структуру приложения.

Конкурентность играет важную роль и широко используется в современных вычислительных системах, операционных системах и больших распределенных кластерах. Она помогает моделировать реальный мир, максимизирует эффективность систем с точки зрения пользователей и разработчиков, а также позволяет разработчикам решать большие и сложные задачи.

Путешествие по миру конкурентности изменит ваши представления о компьютерных системах и их возможностях. Посмотрим на то, как устроен этот мир с точки зрения разных уровней конкурентности.

Уровни конкурентности

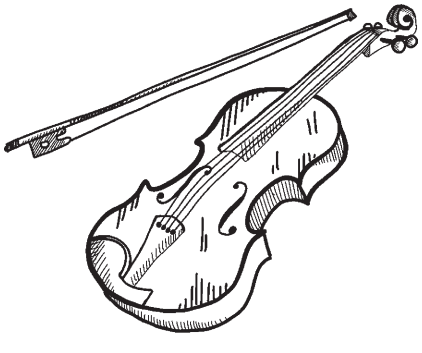
Как и многие сложные концепции проектирования, конкурентность строится на нескольких уровнях. В многоуровневой архитектуре важно понимать, что структуры, которые на первый взгляд противоречат друг другу или взаимно исключают друг друга, могут *конкурентно* сосуществовать на разных уровнях. Например, ничто не мешает организовать конкурентное выполнение на последовательной машине.

Мне нравится представлять многоуровневую архитектуру конкурентности как симфонический оркестр, который играет, допустим, Чайковского:

- На самом верху располагается концептуальный уровень, или уровень проектирования (*прикладной уровень*). Его можно сравнить с нотами, которые композитор написал для оркестра. В компьютерной системе роль такой партитуры играют алгоритмы, которые указывают компонентам системы, что им делать.

- Далее идет механизм многозадачности в среде выполнения (*уровень среды выполнения*). Он подобен музыкантам, которые совместно играют разные партии одного и того же произведения на разных инструментах. Музыкальное исполнение переходит от одной группы инструментов к другой в соответствии с жестами дирижера. В компьютерной системе этот уровень представлен разными процессами, каждый из которых выполняет свою часть работы в рамках общей цели.

- Наконец, мы приходим к низкоуровневому исполнению кода (*аппаратный уровень*). Здесь мы детализируем исполнение до уровня конкретных инструментов, например скрипки. Каждая нота, которую играет скрипач, возникает в результате того, что от одной до четырех струн колеблются с определенной частотой, которая зависит от длины, диаметра, натяжения и плотности струны. В компьютерной системе каждый процесс выполняет задачи в соответствии с инструкциями, которые относятся к этому конкретному процессу.

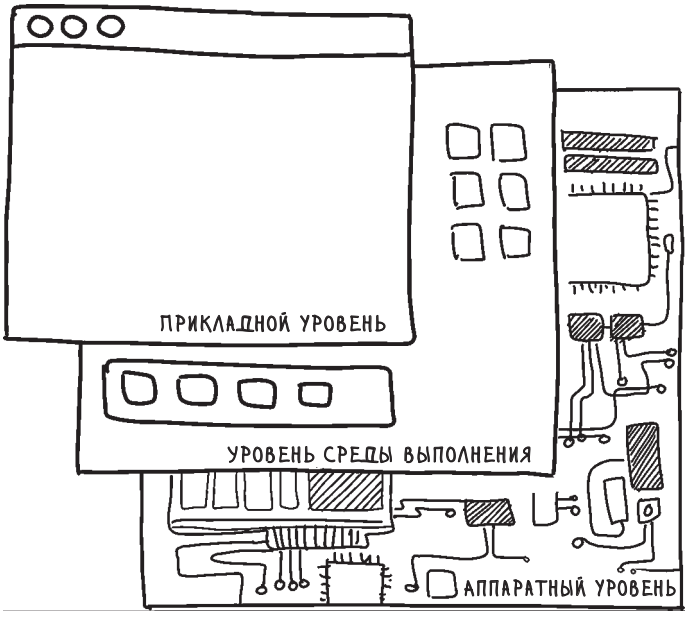


Каждый уровень описывает один и тот же процесс с разной степенью детализации, но подробности описания различаются, а иногда и противоречат друг другу.

То же самое происходит с конкурентностью:

- На *аппаратном уровне* мы имеем дело непосредственно с машинными командами: их выполняют вычислительные ресурсы, которые с помощью сигналов обращаются к периферийным устройствам. Современные архитектуры становятся все сложнее, и поэтому, чтобы оптимизировать производительность приложений в таких архитектурах, нужно глубоко разбираться в том, как приложения взаимодействуют с аппаратными компонентами.

- При переходе на *уровень среды выполнения* многие высокоуровневые сущности, связанные с программными абстракциями, скрываются за таинственными системными вызовами, драйверами устройств и алгоритмами планирования, которые существенно влияют на конкурентные системы, — а значит, в них тоже нужно глубоко разбираться. Этот уровень часто представлен операционной системой, и мы поговорим об этом подробнее в главе 3.
- Наконец, на прикладном уровне становятся доступными абстракции, которые по духу ближе к тому, как устроен физический мир. Программисты пишут исходный код, который реализует сложные алгоритмы и отражает бизнес-логику. Кроме того, этот код может влиять на порядок выполнения при помощи специальных языковых средств; в общем случае он представляет очень абстрактные сущности, которые понятны только разработчикам.



Далее мы будем широко использовать эти уровни как ориентиры в нашем восхождении к вершинам мастерства в области конкурентности.

Что вы узнаете из этой книги

Конкурентность заслужила репутацию непростой дисциплины. Ее сложность отчасти связана с тем, что по этой теме не хватает литературы, которую написали бы опытные практикующие специалисты. Сфера конкурентности больше опирается на устные традиции, чем на знания, закрепленные в письменном виде, и поэтому она

до сих пор окутана тайнами. Я написал эту книгу, чтобы хоть немного приподнять завесу над этими тайнами.

Не стоит ожидать, что эта книга научит всему, что вам когда-либо понадобится знать о конкурентности. Она всего лишь поможет сделать первые шаги и понять, какие темы придется изучать подробнее. Мы рассмотрим некоторые задачи из области конкурентного программирования и опишем передовые методы, которые пригодятся, чтобы создавать конкурентные и масштабируемые приложения.

Программисты начального и среднего уровня получат базовые представления о том, как разрабатывать конкурентные системы. Чтобы извлечь максимум пользы из материала, желательно иметь некоторый опыт программирования, но вовсе не обязательно быть экспертом. Мы будем сперва объяснять ключевые понятия в общих чертах на конкретных примерах, а затем демонстрировать их в действии на языке программирования Python.

Книга делится на три части, в которых рассматриваются разные уровни конкурентности. Первая часть посвящена фундаментальным понятиям и примитивам написания конкурентных программ; она охватывает все уровни — от аппаратного до прикладного.

Тема второй части — проектирование конкурентных приложений и популярные паттерны конкурентности. В этой части также рассказывается о том, как преодолевать распространенные проблемы, которые возникают при разработке конкурентных систем.

Третья часть книги расширит ваши знания в области конкурентности: мы не ограничимся одной машиной, а будем масштабировать приложения на несколько машин, которые связаны по сети. Мы рассмотрим асинхронную коммуникацию между задачами — тему, которая чрезвычайно важна в этом контексте. Кроме того,

здесь вы найдете пошаговое руководство о том, как писать конкурентные приложения.

К концу книги вы возьмете уверенный курс к вершинам мастерства в области конкурентности и будете ориентироваться в современных методах асинхронного и конкурентного программирования. Мы пройдем путь от низкоуровневых аппаратных операций до высокоуровневого проектирования приложений и переведем теорию на язык практической реализации.

Весь код в книге написан на языке Python 3.9 и протестирован в операционных системах macOS и Linux. Материал не привязан ни к какому конкретному языку программирования, хотя опирается на подсистему ядра Linux. Весь исходный код примеров доступен в репозитории GitHub (https://github.com/luminousmen/grokking_concurrency) и на сайте книги (www.manning.com/books/grokking-concurrency).

Итоги

- Конкурентная система — это система, которая способна работать сразу с несколькими задачами.
- В реальном мире многие события происходят одновременно. Если вы хотите моделировать реальный мир, вам понадобится конкурентное программирование.
- Конкурентность радикально улучшает производительность и пропускную способность системы, потому что при этом задержка сокращается или скрывается, а существующие ресурсы эксплуатируются эффективнее.
- В книге используются понятия *масштабируемости* и *ослабления сцепления*:
  - Масштабируемость бывает вертикальной или горизонтальной. Вертикальное масштабирование повышает производительность за счет обновления существующих вычислительных мощностей. Горизонтальное масштабирование повышает производительность за счет того, что нагрузка распределяется между существующими и новыми вычислительными ресурсами. Архитектуры компьютерных систем в основном тяготеют к горизонтальному масштабированию, при котором обязательно задействуется конкурентность.
  - Сложные задачи можно разделить на простые компоненты, связанные друг с другом. В определенном смысле конкурентность — это стратегия ослабления сцепления, которая помогает решать сложные и большие задачи.
- Отправляясь в незнакомое место, обычно стоит захватить с собой карту, которая поможет вам не заблудиться. В этой книге нашим ориентиром станут уровни конкурентности: *прикладной уровень*, *уровень системы выполнения* и *аппаратный уровень*.