

Оглавление

1	■ Первое знакомство с asyncio	21
2	■ Основы asyncio	45
3	■ Первое приложение asyncio	74
4	■ Конкурентные веб-запросы	101
5	■ Неблокирующие драйверы баз данных.....	130
6	■ Счетные задачи	157
7	■ Решение проблем блокирования с помощью потоков.....	189
8	■ Потоки данных	223
9	■ Веб-приложения	251
10	■ Микросервисы	279
11	■ Синхронизация	303
12	■ Асинхронные очереди	327
13	■ Управление подпроцессами.....	350
14	■ Продвинутое использование asyncio.....	365

Содержание

Оглавление	6
Предисловие	12
Благодарности.....	14
Об этой книге	15
Об авторе	19
Об иллюстрации на обложке	20

1	Первое знакомство с <i>asuncio</i>	21
1.1	Что такое <i>asuncio</i> ?.....	22
1.2	Что такое ограниченность производительностью ввода-вывода и ограниченность быстродействием процессора	24
1.3	Конкурентность, параллелизм и многозадачность	25
1.3.1	Конкурентность.....	25
1.3.2	Параллелизм.....	26
1.3.3	Различие между конкурентностью и параллелизмом	27
1.3.4	Что такое многозадачность	28
1.3.5	Преимущества кооперативной многозадачности	28
1.4	Процессы, потоки, многопоточность и многопроцессность.....	29
1.4.1	Процесс.....	29
1.4.2	Поток.....	29
1.5	Глобальная блокировка интерпретатора.....	33
1.5.1	Освобождается ли когда-нибудь <i>GIL</i> ?.....	37
1.5.2	<i>Asuncio</i> и <i>GIL</i>	39
1.6	Как работает однопоточная конкурентность	39
1.6.1	Что такое <i>socket</i> ?	39
1.7	Как работает цикл событий	41
	Резюме.....	44

2	Основы <i>asuncio</i>	45
2.1	Знакомство с сопрограммами	46
2.1.1	Создание сопрограмм с помощью ключевого слова <i>asunc</i>	46
2.1.2	Приостановка выполнения с помощью ключевого слова <i>await</i>	48
2.2	Моделирование длительных операций с помощью <i>sleep</i>	49
2.3	Конкурентное выполнение с помощью задач.....	52

2.3.1	Основы создания задач	52
2.3.2	Конкурентное выполнение нескольких задач	53
2.4	Снятие задач и задание тайм-аутов	56
2.4.1	Снятие задач	56
2.4.2	Задание тайм-аута и снятие с помощью <code>wait_for</code>	57
2.5	Задачи, сопрограммы, будущие объекты и объекты, допускающие ожидание	59
2.5.1	Введение в будущие объекты	59
2.5.2	Связь между будущими объектами, задачами и сопрограммами	61
2.6	Измерение времени выполнения сопрограммы с помощью декораторов	62
2.7	Ловушки сопрограмм и задач	65
2.7.1	Выполнение счетного кода	65
2.7.2	Выполнение блокирующих API	67
2.8	Ручное управление циклом событий	68
2.8.1	Создание цикла событий вручную	69
2.8.2	Получение доступа к циклу событий	69
2.9	Отладочный режим	70
2.9.1	Использование <code>asuncio.run</code>	70
2.9.2	Использование аргументов командной строки	71
2.9.3	Использование переменных окружения	71
	Резюме	72

3	Первое приложение <code>asuncio</code>	74
3.1	Работа с блокирующими сокетами	75
3.2	Подключение к серверу с помощью <code>telnet</code>	78
3.2.1	Чтение данных из сокета и запись данных в сокет	79
3.2.2	Разрешение нескольких подключений и опасности блокирования ...	80
3.3	Работа с неблокирующими сокетами	82
3.4	Использование модуля <code>selectors</code> для построения цикла событий сокетов	86
3.5	Эхо-сервер средствами цикла событий <code>asuncio</code>	89
3.5.1	Сопрограммы цикла событий для сокетов	89
3.5.2	Проектирование асинхронного эхо-сервера	90
3.5.3	Обработка ошибок в задачах	92
3.6	Корректная остановка	94
3.6.1	Прослушивание сигналов	95
3.6.2	Ожидание завершения начатых задач	96
	Резюме	99

4	Конкурентные веб-запросы	101
4.1	Введение в <code>aiohhttp</code>	102
4.2	Асинхронные контекстные менеджеры	103
4.2.1	Отправка веб-запроса с помощью <code>aiohhttp</code>	105
4.2.2	Задание тайм-аутов в <code>aiohhttp</code>	107
4.3	И снова о конкурентном выполнении задач	108
4.4	Конкурентное выполнение запросов с помощью <code>gather</code>	111
4.4.1	Обработка исключений при использовании <code>gather</code>	113
4.5	Обработка результатов по мере поступления	115
4.5.1	Тайм-ауты в сочетании с <code>as_completed</code>	117

4.6	Точный контроль с помощью wait	119
4.6.1	Ожидание завершения всех задач	119
4.6.2	Наблюдение за исключениями	122
4.6.3	Обработка результатов по мере завершения	123
4.6.4	Обработка тайм-аутов	126
4.6.5	Зачем оборачивать программы задачами?	127
	Резюме	128
5	Неблокирующие драйверы баз данных	130
5.1	Введение в asynсrg	131
5.2	Подключение к базе данных Postgres	131
5.3	Определение схемы базы данных	133
5.4	Выполнение запросов с помощью asynсrg	135
5.5	Конкурентное выполнение запросов с помощью пулов подключений	138
5.5.1	Вставка случайных SKU в базу данных о товарах	138
5.5.2	Создание пула подключений для конкурентного выполнения запросов	142
5.6	Управление транзакциями в asynсrg	146
5.6.1	Вложенные транзакции	148
5.6.2	Ручное управление транзакциями	149
5.7	Асинхронные генераторы и потоковая обработка результатирующих наборов	151
5.7.1	Введение в асинхронные генераторы	151
5.7.2	Использование асинхронных генераторов и потокового курсора	153
	Резюме	156
6	Счетные задачи	157
6.1	Введение в библиотеку multiprocessing	158
6.2	Использование пулов процессов	160
6.2.1	Асинхронное получение результатов	161
6.3	Использование исполнителей пула процессов в сочетании с asynсio	162
6.3.1	Введение в исполнители пула процессов	162
6.3.2	Исполнители пула процессов в сочетании с циклом событий	164
6.4	Решение задачи с помощью MapReduce и asynсio	166
6.4.1	Простой пример MapReduce	167
6.4.2	Набор данных Google Books Ngram	169
6.4.3	Применение asynсio для отображения и редукции	170
6.5	Разделяемые данные и блокировки	175
6.5.1	Разделение данных и состояние гонки	176
6.5.2	Синхронизация с помощью блокировок	179
6.5.3	Разделение данных в пулах процессов	181
6.6	Несколько процессов и несколько циклов событий	184
	Резюме	188
7	Решение проблем блокирования с помощью потоков	189
7.1	Введение в модуль threading	190

7.2	Совместное использование потоков и <code>asyncio</code>	194
7.2.1	Введение в библиотеку <code>requests</code>	194
7.2.2	Знакомство с исполнителями пула потоков	195
7.2.3	Исполнители пула потоков и <code>asyncio</code>	197
7.2.4	Исполнители по умолчанию	198
7.3	Блокировки, разделяемые данные и взаимоблокировки.....	200
7.3.1	Реентерабельные блокировки.....	201
7.3.2	Взаимоблокировки	204
7.4	Циклы событий в отдельных потоках.....	206
7.4.1	Введение в <code>Tkinter</code>	207
7.4.2	Построение отзывчивого UI с помощью <code>asyncio</code> и потоков	209
7.5	Использование потоков для выполнения счетных задач.....	217
7.5.1	<code>hashlib</code> и многопоточность	217
7.5.2	Многопоточность и <code>NumPy</code>	220
	Резюме.....	222

8	Потоки данных	223
8.1	Введение в потоки данных	224
8.2	Транспортные механизмы и протоколы	224
8.3	Потоковые читатели и писатели	228
8.4	Неблокирующий ввод данных из командной строки.....	231
8.4.1	Режим терминала без обработки и программа <code>read</code>	235
8.5	Создание серверов	242
8.6	Создание чат-сервера и его клиента.....	244
	Резюме.....	249

9	Веб-приложения	251
9.1	Разработка REST API с помощью <code>aiohhttp</code>	252
9.1.1	Что такое REST?.....	252
9.1.2	Основы разработки серверов на базе <code>aiohhttp</code>	253
9.1.3	Подключение к базе данных и получение результатов	255
9.1.4	Сравнение <code>aiohhttp</code> и <code>Flask</code>	260
9.2	Асинхронный интерфейс серверного шлюза.....	263
9.2.1	Сравнение <code>ASGI</code> и <code>WSGI</code>	263
9.3	Реализация <code>ASGI</code> в <code>Starlette</code>	264
9.3.1	Оконечная REST-точка в <code>Starlette</code>	265
9.3.2	<code>WebSockets</code> и <code>Starlette</code>	266
9.4	Асинхронные представления Django.....	269
9.4.1	Выполнение блокирующих работ в асинхронном представлении	275
9.4.2	Использование асинхронного кода в синхронных представлениях.....	277
	Резюме.....	278

10	Микросервисы	279
10.1	Зачем нужны микросервисы?.....	280
10.1.1	Сложность кода.....	281
10.1.2	Масштабируемость	281
10.1.3	Независимость от команды и технологического стека	281
10.1.4	Чем может помочь <code>asyncio</code> ?	281
10.2	Введение в паттерн <code>backend-for-frontend</code>	282
10.3	Реализация API списка товаров.....	283

10.3.1	Сервис избранного.....	284
10.3.2	Реализация базовых сервисов	284
10.3.3	Реализация сервиса <i>backend-for-frontend</i>	289
10.3.4	Повтор неудачных запросов.....	294
10.3.5	Паттерн Прерыватель	297
	Резюме.....	302

11	Синхронизация	303
11.1	Природа ошибок в модели однопоточной конкурентности	304
11.2	Блокировки	309
11.3	Ограничение уровня конкурентности с помощью семафоров	312
11.3.1	Ограниченные семафоры.....	315
11.4	Уведомление задач с помощью событий.....	317
11.5	Условия.....	322
	Резюме.....	326

12	Асинхронные очереди	327
12.1	Основы асинхронных очередей	328
12.1.1	Очереди в веб-приложениях.....	335
12.1.2	Очередь в веб-работе.....	338
12.2	Очереди с приоритетами.....	341
12.3	LIFO-очереди	347
	Резюме.....	349

13	Управление подпроцессами	350
13.1	Создание подпроцесса	351
13.1.1	Управление стандартным выводом	353
13.1.2	Конкурентное выполнение подпроцессов	357
13.2	Взаимодействие с подпроцессами.....	360
	Резюме.....	363

14	Продвинутое использование <i>asynсio</i>	365
14.1	API, допускающие сопрограммы и функции.....	366
14.2	Контекстные переменные	368
14.3	Принудительный запуск итерации цикла событий.....	370
14.4	Использование других реализаций цикла событий	371
14.5	Создание собственного цикла событий.....	373
14.5.1	Сопрограммы и генераторы.....	373
14.5.2	Использовать сопрограммы на основе генераторов не рекомендуется.....	374
14.5.3	Нестандартные объекты, допускающие ожидание.....	376
14.5.4	Сокеты и будущие объекты	378
14.5.5	Реализация задачи	381
14.5.6	Реализация цикла событий	382
14.5.7	Реализация сервера с использованием своего цикла событий	385
	Резюме.....	387
	Предметный указатель.....	388

Предисловие

Почти 20 лет назад я начал профессиональную карьеру в области программной инженерии, написав приложение для управления масс-спектрометрами и другими лабораторными приборами и анализа поступающих от них данных. В приложении использовались языки Matlab, C++ и VB.net. Меня всегда охватывал восторг при виде того, как строчка кода заставляет машину двигаться по моему желанию, и с тех пор я понял, что хочу заниматься только программной инженерией. Шли годы, мои интересы постепенно сместились в сторону разработки API и распределенных систем, в основном на Java и Scala, но попутно я активно изучал Python.

Я впервые столкнулся с Python примерно в 2015 году, работа была связана главным образом с построением конвейера машинного обучения, который принимал данные от датчиков и на их основе делал предсказания относительно действий носителя датчика – например, отслеживание сна, подсчет числа шагов, переходы из положения сидя в положение стоя и т. д. Тогда этот конвейер был медленным настолько, что это стало проблемой для клиентов. Одним из способов, которые я применил для решения проблемы, стало использование конкурентности. Копаясь в доступной информации о конкурентном программировании на Python, я обнаружил, что разобраться в ней труднее, чем в привычном мне мире Java. Почему многопоточность работает не так, как в Java? Есть ли смысл использовать многопроцессную обработку? Что такое глобальная блокировка интерпретатора и зачем она нужна? На тему конкурентности в Python книг было немного, а знания в основном были разбросаны по документации и блогам разного качества. И сегодня ситуация мало чем отличается. Ресурсов стало больше, но ландшафт по-прежнему скудный, разъединенный и не такой дружелюбный к начинающим, каким должен, по идее, быть.

Разумеется, за последние несколько лет многое изменилось. Тогда пакет `asyncio` пребывал в младенчестве, а теперь стал важным модулем в Python. Ныне модели однопоточной конкурентности и сопрограммы являются одним из базовых компонентов Python в дополнение к многопоточности и многопроцессности. Это значит, что ландшафт конкурентности в Python стал шире и сложнее, но так и не обрел исчерпывающих ресурсов, к которым мог бы обратиться желающий изучить его.

Я написал эту книгу, чтобы заполнить этот пробел, конкретно в области `asyncio` и однопоточной конкурентности. Я хотел сделать сложную и плохо документированную тему однопоточной конкурентности более доступной разработчикам всех уровней. Кроме того, я хотел написать книгу, благодаря которой читатель стал бы лучше понимать общие проблемы конкурентности, выходящие за рамки Python. В таких каркасах, как Node.js, и таких языках, как Kotlin, имеются модели однопоточной конкурентности и сопрограммы, поэтому приведенные здесь сведения будут полезны и при работе с этими инструментами. Надеюсь, что книга окажется полезной читателям-разработчикам в повседневной работе, – и не только в Python, но и вообще в области конкурентного программирования.

Об этой книге

Эта книга написана для тех, кто хочет научиться использовать средства конкурентности в Python, чтобы повысить производительность, пропускную способность и отзывчивость приложений. Сначала мы рассмотрим базовые вопросы конкурентности, объясним, как работает модель однопоточной конкурентности в `asyncio`, а также расскажем о принципах работы сопрограмм и синтаксисе `async/await`. Затем перейдем к практическим приложениям конкурентности, например: как отправить несколько конкурентных веб-запросов или запросов к базе данных, как управлять потоками и процессами, строить веб-приложения и решать вопросы синхронизации.

Кому стоит прочитать эту книгу?

Книга адресована разработчикам средней и высокой квалификации, которые хотят разобраться в конкурентности и использовать ее в уже написанных или новых приложениях. Одна из целей книги – объяснить сложные вопросы простым и понятным языком. Предварительного знакомства с конкурентностью не требуется, хотя оно, конечно, не помешает. Мы рассмотрим широкий круг применений: от API на основе веба до командных приложений, так что книга будет полезна для решения многих задач, с которыми сталкиваются разработчики.

Структура книги

В этой книге 14 глав, в которых рассматриваются постепенно усложняющиеся темы. Последующие главы основаны на материале предыдущих.

- **Глава 1** посвящена базовым вопросам конкурентности в Python. Мы узнаем о задачах, ограниченных быстродействием процессора (счетных) и производительностью ввода-вывода, а также

начнем рассказ о том, как работает модель однопоточной конкурентности.

- **Глава 2** посвящена основам сопрограмм `asyncio` и использованию синтаксиса `async/await` в конкурентных приложениях.
- В **главе 3** рассматриваются неблокирующие сокеты и селекторы, а также описывается построение эхо-сервера с применением `asyncio`.
- **Глава 4** посвящена отправке нескольких конкурентных веб-запросов. Попутно мы больше узнаем о том, как использовать `asyncio API` для конкурентного выполнения сопрограмм.
- Тема **главы 5** – конкурентное выполнение запросов к базе данных с использованием пула подключений. Также мы узнаем об асинхронных контекстных менеджерах и асинхронных генераторах в контексте баз данных.
- В **главе 6** обсуждается многопроцессная обработка, в частности использование `asyncio` для выполнения счетных задач. Для демонстрации будет построено сообщение типа `map-reduce`.
- **Глава 7** посвящена многопоточной обработке, а особенно ее использованию в сочетании с `asyncio` для обработки блокирующего ввода-вывода. Это полезно при работе с библиотеками, в которых поддержка `asyncio` не встроена, что не мешает им получать выгоды от конкурентного выполнения.
- **Глава 8** посвящена потоковой обработке и протоколам. Мы напишем сервер и клиент чата, способные конкурентно обрабатывать несколько пользователей.
- В **главе 9** рассматриваются веб-приложения на основе `asyncio` и асинхронный интерфейс серверного шлюза, `ASGI`. Мы изучим несколько каркасов, поддерживающих `ASGI`, и обсудим, как строить в них веб-API. Также поговорим о технологии `WebSockets`.
- В **главе 10** описывается, как с помощью веб-API на основе `asyncio` построить гипотетическую микросервисную архитектуру.
- **Глава 11** посвящена проблемам синхронизации в модели однопоточной конкурентности и методам их решения. Мы рассмотрим блокировки, семафоры, события и условия.
- **Глава 12** посвящена асинхронным очередям. С их помощью мы построим веб-приложение, мгновенно отвечающее на клиентские запросы, хотя в фоновом режиме производятся длительные операции.
- В **главе 13** обсуждается создание подпроцессов и управление ими. Мы покажем, как читать и передавать данные подпроцессу.
- В **главе 14** рассматриваются дополнительные темы, в том числе принудительный переход к новой итерации цикла событий, контекстные переменные и создание собственного цикла событий. Эта информация полезна прежде всего проектировщикам `asyncio API` и читателям, интересующимся внутренним устройством цикла событий в `asyncio`.

Как минимум следует прочитать первые четыре главы, чтобы понять, как работает `asyncio`, как написать свое первое реальное приложение и как использовать базовые функции `asyncio API` для конкурентного выполнения сопрограмм (рассматриваются в главе 4). После этого вы вольны читать книгу в любом порядке, отвечающем вашим интересам.

О примерах кода

В этой книге много примеров кода как в пронумерованных листингах, так и в виде небольших фрагментов. Некоторые листинги используются повторно путем импорта в той же главе, а иногда и в нескольких главах. Предполагается, что код, используемый в нескольких главах, помещен в модуль `util`; мы создадим его в главе 2. Предполагается также, что для каждого отдельного листинга будет создан модуль с именем `chapter_{номер_главы}`, а код будет помещен в файл с именем `listing_{номер_главы}_{номер_листинга}.py`, принадлежащий этому модулю. Например, код из листинга 2.2 в главе 2 должен находиться в файле `listing_2_2.py`, принадлежащем модулю `chapter_2`.

В нескольких местах приводятся сведения о производительности, например время работы программы или количество веб-запросов в секунду. Примеры кода выполнялись на компьютере 2019 MacBook Pro с 8-ядерным процессором Intel Core i9 с тактовой частотой 2,4 ГГц и 32 Гб памяти DDR4, работающей на частоте 2667 МГц; использовалось также гигабитное беспроводное подключение к интернету. На вашей машине цифры могут получиться другими, как и коэффициенты ускорения или иного улучшения производительности.

Исполняемые фрагменты кода можно найти в онлайн-версии книги на сайте <https://livebook.manning.com/book/python-concurrency-with-asyncio>. Полный исходный код можно скачать бесплатно с сайта издательства Manning по адресу <https://www.manning.com/books/python-concurrency-with-asyncio>, также он доступен на Github по адресу <https://github.com/concurrency-in-python-with-asyncio>.

Форум на сайте liveBook

Приобретение этой книги открывает бесплатный доступ к платформе liveBook онлайн-чтения, созданной издательством Manning. Средства обсуждения на liveBook позволяют присоединять комментарии как к книге в целом, так и к отдельным разделам или абзацам. Совсем несложно добавить примечания для себя, задать или ответить на технический вопрос и получить помощь от автора и других пользователей. Для доступа к форуму перейдите по адресу <https://livebook.manning.com/#!/book/python-concurrency-with-asyncio/discussion>. Узнать о форумах Manning и правилах поведения на них можно по адресу <https://livebook.manning.com/#!/discussion>.

Издательство Manning обязуется предоставлять площадку для содержательного диалога между читателями, а также между читателями

и автором. Но это обязательство не подразумевает какого-то конкретного количества часов присутствия автора, участие которого в работе форума остается добровольным (и не оплачивается). Мы рекомендуем задавать автору трудные вопросы, чтобы его интерес не угасал! Форум и архивы прошлых обсуждений остаются доступны на сайте издательства до тех пор, пока книга продолжает допечатываться.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning Publications очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторе



Мэттью Фаулер почти 20 лет занимается программной инженерией на различных должностях: от архитектора ПО до директора по инженерии. Он начинал с написания научных программ, затем перешел на сквозную разработку веб-приложений и распределенных систем и в итоге возглавил несколько команд разработчиков и менеджеров, которые занимались созданием интернет-магазина, обслуживающего десятки миллионов пользователей. Проживает в Лексингтоне, штат Массачусетс, с женой Кэти.

Первое знакомство с *asuncio*

Краткое содержание главы

- Что такое библиотека *asuncio* и какие преимущества она дает.
- Конкурентность, параллелизм, потоки и процессы.
- Глобальная блокировка интерпретатора и создаваемые ей проблемы для конкурентности.
- Как неблокирующие сокеты позволяют добиться конкурентного выполнения в одном потоке.
- Начала конкурентности на основе цикла событий.

Многие программы, особенно в нынешнем мире, где правят бал веб-приложения, сильно зависят от операций ввода-вывода. Это скачивание веб-страницы из интернета, взаимодействие по сети с группой микросервисов или отправка нескольких запросов такой базе данных, как MySQL или Postgres. Выполнение веб-запроса или взаимодействие с микросервисом может занять несколько сотен миллисекунд или даже секунды, если сеть медленная. Запрос к базе данных может занимать много времени, особенно если нагрузка на базу высока или запрос сложный. Веб-серверу иногда приходится обрабатывать одновременно сотни или тысячи запросов.

Попытка выполнить эти запросы ввода-вывода сразу может привести к серьезным проблемам с производительностью. Если отправлять запросы один за другим, как в последовательном приложении, то за-

медление будет нарастать. Например, если мы пишем приложение, которое должно скачать 100 веб-страниц или выполнить 100 запросов к базе данных, и каждое взаимодействие продолжается 1 с, то для завершения приложения потребуется 100 с. Но если воспользоваться конкурентностью и начать все скачивания в одно и то же время, после чего дождаться результатов, то теоретически все операции можно было бы завершить за 1 с.

Библиотека *asyncio* впервые появилась в версии Python 3.4 как еще один способ справляться с высокими конкурентными нагрузками, не прибегая к нескольким потокам или процессам. При правильном использовании эта библиотека может значительно повысить производительность и уменьшить потребление ресурсов в приложениях, выполняющих много операций ввода-вывода, поскольку позволяет запускать сразу много таких долго работающих задач.

В этой главе мы познакомимся с основами конкурентности, чтобы лучше понять, как она достигается в Python и библиотеке *asyncio*. Мы рассмотрим различия между задачами, ограниченными быстродействием процессора и производительностью ввода-вывода, чтобы вы могли понять, какая модель конкурентности лучше отвечает вашим потребностям. Мы также поговорим об основах процессов и потоков и о специфических проблемах, связанных с наличием в Python глобальной блокировки интерпретатора (GIL). Наконец, мы поймем, как использовать *неблокирующий ввод-вывод* совместно с циклом событий и таким образом добиться конкурентности всего в одном процессе и потоке. Это основная модель конкурентности в *asyncio*.

1.1 Что такое *asyncio*?

В синхронном приложении код выполняется последовательно. Следующая строка кода выполняется после завершения предыдущей, и в каждый момент времени происходит что-то одно. Эта модель хорошо работает для многих, если не для большинства приложений. Но что, если какая-то одна строка кода занимает слишком много времени? В таком случае весь последующий код должен будет замереть, пока эта строка не соблаговолит завершиться. Многие из нас раньше встречали плохо написанные пользовательские интерфейсы, в которых все сначала шло хорошо, а потом приложение внезапно зависало, оставляя нас созерцать крутящееся колесико или ждать хоть какого-нибудь ответа. Такие блокировки в приложении оставляют тягостное впечатление у пользователя.

Любая достаточно длительная операция может блокировать приложение, но особенно часто это бывает, когда приложение ждет завершения ввода-вывода. Ввод-вывод выполняют такие устройства, как клавиатура, жесткий диск и, конечно же, сетевая карта. Такие операции ждут ввода от пользователя или получения содержимого

от веб-API. В синхронном приложении мы будем ждать завершения операции и до тех пор ничего не сможем делать. Это ведет к проблемам с производительностью и отзывчивостью, поскольку в каждый момент времени может выполняться только одна длительная операция, а она не дает приложению больше ничего делать.

Один из способов решения этой проблемы – ввести в программу конкурентность. Говоря по-простому, *конкурентность* позволяет одновременно выполнять более одной задачи. Примерами конкурентного ввода-вывода могут служить одновременная отправка нескольких веб-запросов или создание одновременных подключений к веб-серверу.

В Python есть несколько способов организовать такую конкурентность. Одним из последних добавлений в экосистему Python является библиотека асинхронного ввода-вывода *asyncio*. Она позволяет исполнять код в рамках модели асинхронного программирования, т. е. производить сразу несколько операций ввода-вывода, не жертвуя отзывчивостью приложения.

Так что же означают слова «асинхронное программирование»? Что длительную задачу можно выполнять в фоновом режиме отдельно от главного приложения. И система не блокируется в ожидании завершения этой задачи, а может заниматься другими вещами, не зависящими от ее исхода. Затем, по завершении задачи, мы получим уведомление о том, что она все сделала, и сможем обработать результат.

В Python версии 3.4, когда состоялся дебют библиотеки *asyncio*, она включала декораторы и синтаксис генератора `yield from` для определения сопрограмм. Сопрограмма – это метод, который можно приостановить, если имеется потенциально длительная задача, а затем возобновить, когда она завершится. В Python 3.5 в самом языке была реализована полноценная поддержка сопрограмм и асинхронного программирования, для чего были добавлены ключевые слова `async` и `await`. Этот синтаксис, общий с другими языками программирования, например C и JavaScript, позволяет писать асинхронный код так, что он выглядит как синхронный. Такой асинхронный код проще читать и понимать, поскольку он похож на последовательный код, с которым большинство программистов хорошо знакомо. Библиотека *asyncio* исполняет сопрограммы асинхронно, пользуясь моделью конкурентности, получившей название *однопоточный цикл событий*.

Название *asyncio* может навести на мысль, будто библиотека годится только для операций ввода-вывода. Но на самом деле она способна выполнять и операции других типов благодаря взаимодействию с механизмами многопроцессности и многопоточности. Поэтому синтаксис `async` и `await` можно использовать в сочетании с процессами и потоками, что делает соответствующий код понятнее. Следовательно, библиотека пригодна не только для организации конкурентного ввода-вывода, но и для выполнения счетных задач, активно использующих процессор. Чтобы лучше разобраться в том, с какими рабочими нагрузками позволяет справляться *asyncio* и какая модель

лучше всего подходит для каждого типа конкурентности, рассмотрим различия между операциями, ограниченными производительностью ввода-вывода и быстродействием процессора.

1.2 Что такое ограниченность производительностью ввода-вывода и ограниченность быстродействием процессора

Говоря, что операция ограничена производительностью ввода-вывода или быстродействием процессора, мы имеем в виду фактор, препятствующий более быстрой работе. Это значит, что если увеличить производительность того, что ограничивает операцию, то для ее завершения понадобится меньше времени.

Операция, ограниченная быстродействием процессора (счетная операция), завершилась бы быстрее, будь процессор более мощным, например с частотой 3, а не 2 ГГц. Операция, ограниченная производительностью ввода-вывода, работала бы быстрее, если бы устройство могло обработать больше данных за меньшее время. Для этого можно было бы увеличить пропускную способность сети, заплатив больше денег интернет-провайдеру или поставив более шустрю сетевую карту.

Счетные операции в Python – это обычно вычисления и обработка данных. Примерами могут служить вычисления цифр числа π или применение бизнес-логики к каждому элементу словаря. В случае операции, ограниченной вводом-выводом, мы тратим большую часть времени на ожидание ответа от сети или другого устройства. Примерами могут служить запрос к веб-серверу или чтение файла с жесткого диска.

Листинг 1.1 Операции, ограниченные производительностью ввода-вывода и быстродействием процессора

```
import requests
response = requests.get('https://www.example.com')
items = response.headers.items()
headers = [f'{key}: {header}' for key, header in items]
formatted_headers = '\n'.join(headers)
with open('headers.txt', 'w') as file:
    file.write(formatted_headers)
```

Веб-запрос ограничен производительностью ввода-вывода

Обработка ответа ограничена быстродействием процессора

Конкатенация строк ограничена быстродействием процессора

Запись на диск ограничена производительностью ввода-вывода

Операции, ограниченные производительностью ввода-вывода и быстродействием процессора, обычно сосуществуют бок о бок. Сначала мы выполняем ограниченный производительностью ввода-вывода запрос, чтобы скачать содержимое страницы <https://www.example.com>. Получив ответ, мы выполняем ограниченный быстродействием процессора цикл форматирования заголовков ответа, в котором они преобразуются в строку и разделяются символами новой строки. Затем мы открываем файл и записываем в него строку – обе операции ограничены производительностью ввода-вывода.

Асинхронный ввод-вывод позволяет приостановить выполнение метода, встретив операцию ввода-вывода; ожидая завершения этой операции, работающей в фоновом режиме, мы можем выполнять какой-нибудь другой код. Это позволяет выполнять одновременно много операций ввода-вывода и тем самым ускорить работу приложения.

1.3 Конкурентность, параллелизм и многозадачность

Чтобы лучше понять, как конкурентность может повысить производительность приложения, важно для начала разобраться в терминологии конкурентного программирования. Мы узнаем, что означает слово «конкурентность» и как в `asyncio` используется концепция многозадачности для ее достижения. Конкурентность и параллелизм – два понятия, помогающие понять, как осуществляется планирование программы и какие задачи, методы и процедуры приводят весь механизм в действие.

1.3.1 Конкурентность

Говоря, что две задачи выполняются *конкурентно*, мы имеем в виду, что они работают в одно и то же время. Взять, к примеру, пекаря, выпекающего два разных торта. Чтобы их испечь, нужно сначала разогреть духовку. Разогрев может занимать десятки минут – это зависит от духовки и требуемой температуры, но нам необязательно ждать, пока духовка нагреется, поскольку можно в это время заняться другими делами, например смешать муку с сахаром и вбить в тесто яйца. Это можно делать, пока духовка звуковым сигналом не известит нас о том, что нагрелась.

Мы также не хотим связывать себя ограничением – приступать ко второму тарту только после готовности первого. Ничто не мешает замесить тесто для одного торта, положить его в миксер и начать готовить вторую порцию, пока первая взбивается. В этой модели мы переключаемся между разными задачами конкурентно. Такое переключение (делать что-то, пока духовка разогревается, переключаться с одного торта на другой) – пример *конкурентного* поведения.

1.3.2 Параллелизм

Хотя конкурентность подразумевает, что несколько задач выполняется одновременно, это еще не значит, что они работают параллельно. Говоря о *параллельной* работе, мы имеем в виду, что две задачи или более не просто чередуются, а выполняются строго в одно и то же время. Возвращаясь к примеру с тортами, представьте, что мы призвали на помощь второго пекаря. В этом случае мы можем трудиться над первым тортом, а помощник будет заниматься вторым. Два человека, готовящих тесто, работают параллельно, потому что одновременно делают два разных дела (рис. 1.1).

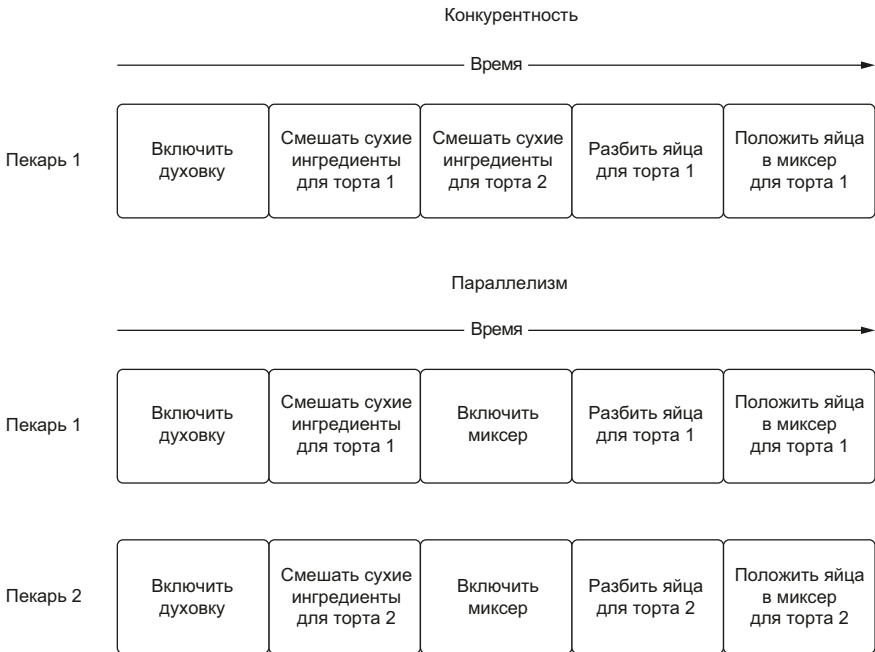


Рис. 1.1 В случае *конкурентности* несколько задач работает в течение одного промежутка времени, но только одна активна в каждый момент. В случае *параллелизма* несколько задач активно одновременно

Теперь переведем это на язык приложений, исполняемых операционной системой. Пусть работают два приложения. В конкурентной системе мы можем переключаться между ними, дав немного поработать сначала одному, а потом другому. Если делать это достаточно быстро, создается впечатление, что два дела делаются одновременно. В параллельной системе два приложения работают действительно одновременно, т. е. оба активны в одно и то же время.

Понятия конкурентности и параллелизма похожи (рис. 1.2), и различить их бывает затруднительно, но важно хорошо понимать, чем они отличаются.

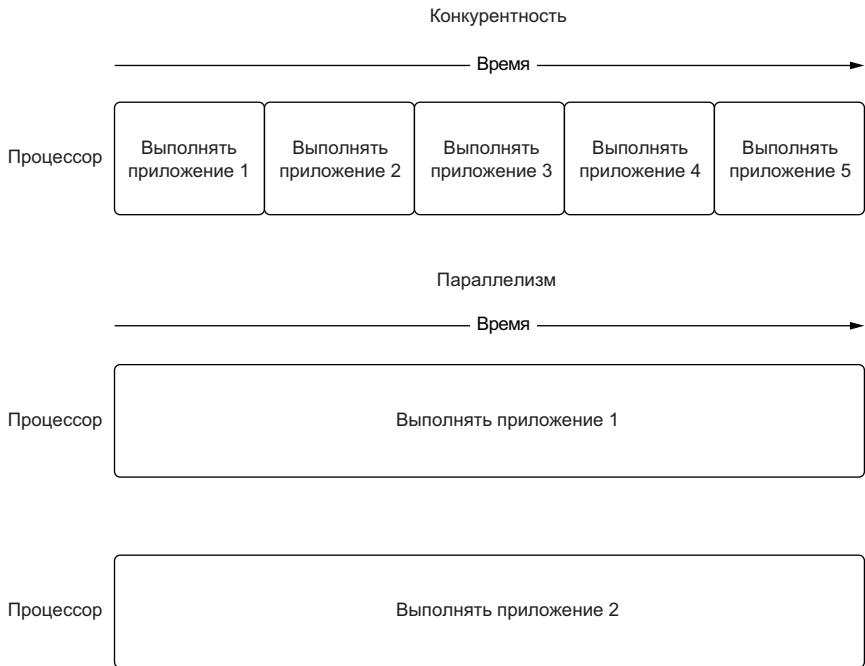


Рис. 1.2 В случае конкурентности мы переключаемся между двумя приложениями. В случае параллелизма мы активно выполняем два приложения одновременно

1.3.3 Различие между конкурентностью и параллелизмом

Конкурентность возможна, когда несколько задач может работать независимо друг от друга. Конкурентность можно организовать, имея процессор всего с одним ядром, применив *вытесняющую многозадачность* (определяется в следующем разделе) для переключения между задачами. С другой стороны, параллелизм означает, что мы должны выполнять две задачи или более строго одновременно. На машине с одним ядром это невозможно, необходимо иметь процессор с несколькими ядрами.

Параллелизм подразумевает конкурентность, но обратное верно не всегда. Многопоточное приложение, работающее на многоядерной машине, является и конкурентным, и параллельным. В этом случае имеется несколько задач, работающих одновременно, и два ядра, независимо исполняющих код этих задач. Но многозадачность допускает также наличие нескольких задач, работающих конкурентно, но так, что в каждый момент времени выполняется только одна.