Посвящается моей жене Памеле и моему сыну Валерио, которые поддерживали меня на протяжении всего процесса создания этой книги.

Лично я убежден, что информационные технологии имеют много общего с физикой. Обе дисциплины занимаются изучением мироздания на самых фундаментальных уровнях. Единственное отличие, безусловно, заключается в том, что в физике задача состоит в изучении мироздания, а в информационных технологиях – в его созидании.

ЛИНУС ТОРВАЛЬДС

Оглавление

Об авторе

Фабио Нелли (Fabio Nelli) получил степень магистра по химии и степень бакалавра по информационным технологиям и автоматизации. В настоящее время он сотрудничает с несколькими научно-исследовательскими институтами и частными компаниями, где ведет образовательные курсы по анализу данных и технологиям визуализации данных. Помимо профессиональной деятельности, он активно участвует в развитии отрасли благодаря публикации статей в интернете, прежде всего на своем веб-сайте *meccanismocomplesso.org*, а также написанию тематических книг с подробным изложением материала.

Технический рецензент

Прашант Рагху (Prashanth Raghu) является страстным поклонником технологий, который уделяет особое внимание изучению влияния технологий с открытым исходным кодом на приложения для крупных систем. Он окончил Технологический институт PES, а затем Национальный университет Сингапура и в настоящее время работает архитектором программного обеспечения в компании Zeta Technologies. Помимо увлечения технологиями, он интересуется искусством, играет на карнатической флейте и занимается спортом.

Благодарности

Хотелось бы поблагодарить нескольких человек за постоянную и непрерывную поддержку, которую они оказывали мне в процессе написания этой книги. Прежде всего я хотел бы поблагодарить свою жену за то постоянное вдохновение, которое она давала мне для написания этой книги; без ее поддержки я бы не смог ее завершить.

Я благодарен команде Orange Education, которая сопровождала меня на всех этапах создания этой книги. Особая благодарность Шали Дираджу (Shali Deeraj) и Шубхе Мурти (Shubha Murthy) за их содействие.

Предисловие

Эта книга познакомит вас с различными методами, доступными в настоящее время для параллельного и многозадачного программирования на языке Python. Существует множество библиотек и методов, позволяющих использовать преимущества различных существующих архитектур для повышения эффективности и производительности вычислений (GPU, процессорные ядра, распределенные сетевые ресурсы и т. д.). В этой книге будут постепенно представлены соответствующие концепции с примерами кода.

Глава 1 начинается с введения в параллелизм с использованием Python, а затем в ней описывается принцип обработки нескольких процессов и потоков на уровне операционной системы. Цель этой главы – познакомить вас с концепцией **параллельного программирования**, рассказав обо всех основных аспектах, которые необходимо учитывать для полного понимания его возможностей и применения. После введения в эти общие концепции вы познакомитесь с особенностями Python в этой области, прежде всего с потоками, а также с GIL (Global Interpreter Lock) и связанными с ним проблемами. Помимо этого, будут рассмотрены модули стандартной библиотеки Python, такие как threading и multiprocessing.

Глава 2 посвящена параллельному программированию на Python с использованием потоков в качестве элементов параллелизма. В частности, в этой главе будет рассмотрен модуль threading, который упрощает реализацию многопоточности и предоставляет множество полезных инструментов для синхронизации потоков.

Глава 3 посвящена параллельному программированию на Python с использованием процессов. Существует два основных подхода к реализации процессов в параллельном программировании: модуль multiprocessing стандартной библиотеки и библиотека mpi4py, которая также обеспечивает поддержку протокола MPI для языка Python. Обе библиотеки будут подробно рассмотрены в серии примеров, демонстрирующих принцип их работы и основные характеристики.

Глава 4 охватывает аспекты асинхронного программирования на Python, в том числе библиотеку AsyncIO в качестве примера.

Глава 5 посвящена распределенным системам, поскольку они также относятся к области параллельного и многопоточного программирования, что может оказаться эффективным подходом. Библиотека Celery в Python является эталоном для реализации распределенных систем. На ряде примеров будет показано, как использовать эту библиотеку для выполнения нескольких параллельных операций, называемых задачами, и как они распараллеливаются и выполняются в системе на базе Celery. Кроме того, далее будут рассмотрены другие альтернативные решения, начиная с похожих, таких как Dramatiq, и заканчивая более простыми, но отличными от них с концептуальной точки зрения, такими как SCOOP.

Глава 6 охватывает тему программирования GPU, поскольку эти процессоры предназначены для чрезвычайно быстрой и эффективной обработки векторных данных для рендеринга изображений, 3D-движков и работы с полигональными примитивами. Язык Python предлагает хорошие решения с различными библиотеками, такими как Numba (CUDA) и PyOpenCL.

Глава 7 познакомит вас с приложениями для параллельных вычислений и продемонстрирует их применение во многих научных и технических дисциплинах.

Глава 8 посвящена параллельным вычислениям, в частности в области науки о данных. Многие библиотеки, широко используемые в науке о данных, такие как NumPy, Pandas и Scikit-learn, могут быть адаптированы для параллельных вычислений благодаря библиотеке Dask. Эта библиотека предоставляет объекты, такие как nparrays, dataframes и machine learning api, предназначенные для работы в параллельном режиме. Существует множество других научных библиотек, которые могут быть интегрированы с Dask.

Глава 9 освещает еще одну очень интересную область применения, которая частично перекликается с материалами главы 8 об аналитике данных. Машинное обучение и глубокое обучение являются подразделами области искусственного интеллекта и являются эффективным инструментом для исследования данных. В последнее время эти технологии получают широкое распространение благодаря использованию параллельных и распределенных вычислений.

Глава 10 посвящена развитию и тенденциям в области программирования, которые в полной мере соответствуют последним инновационным технологиям.

Загрузка пакетов кода и цветных изображений

Загрузка пакетов кода из книги доступна по адресу *https://github.com/OrangeAVA/Parallel-Programming-with-Python*.

Пакеты кода и изображения из книги также можно найти по адресу *https://rebrand.ly/3127a9*.

В случае обновления кода он также будет обновлен в существующем репозитории GitHub.

Исправления

Мы в <Orange Education Pvt Ltd> испытываем огромную гордость за свою работу и следуем лучшим практикам в целях обеспечения точности нашего контента, чтобы предоставить нашим пользователям возможность насладиться чтением. Наши читатели – это наше зеркало, и мы используем их отзывы для анализа и исправления возможных ошибок, которые могли возникнуть в процессе публикации. Мы стремимся поддерживать высокое качество и помогать читателям, которые столкнулись с трудностями из-за непредвиденных ошибок. Пожалуйста, пишите нам по адресу *errata@orangeava.com*.

Мы высоко ценим вашу поддержку, предложения и отзывы.

Глава 1

Введение в параллельное программирование

В первой главе книги представлена идея параллельного программирования с описанием всех основных концепций, которые необходимо знать для полного понимания его особенностей и возможностей. Сначала рассмотрены аппаратные компоненты нового поколения компьютеров, такие как процессоры и ядра, которые позволяют выполнять параллельные вычисления. Затем описаны элементы операционной системы, которые непосредственно обеспечивают параллелизм: процессы и потоки. Далее подробно рассмотрены модели параллельного программирования с введением таких фундаментальных концепций, как параллелизм, синхронность и асинхронность.

После ознакомления с этими общими концепциями мы рассмотрим специфику использования языка Python применительно к этой области, в частности потоки, а также остановимся на Global Interpreter Lock (GIL) и связанных с ним проблемах. Мы упомянем стандартные модули библиотеки Python, такие как threading и multiprocessing, которые будут более подробно рассмотрены в следующих главах. Наконец, мы завершим главу обзором методов оценки эффективности программного обеспечения с параллельной обработкой данных, таких как ускорение и масштабирование, а также обсудим проблемы, которые могут возникнуть при программировании параллельных процессов (состояние конкурентной гонки, тупиковая ситуация и т. д.).

К концу этой главы вы ознакомитесь со всеми основными концепциями и терминологией, на которых основано параллельное программирование. Вы сможете составить в уме общую схему, в которой будут представлены все ключевые элементы процесса параллельного выполнения и их роль в его реализации. После этого вы будете готовы приступить к практической части обучения программированию, которая будет рассмотрена в следующих главах.

Структура

В этой главе будут рассмотрены следующие темы:

- центральные процессоры и ядра;
- процессы и потоки;
- параллельное и многопоточное программирование;

- GIL и потоки в Python;
- ускорение и масштабирование.

Параллельное программирование

Если вы читаете эту книгу, то, безусловно, вам уже понятна необходимость повышения потенциала вашего кода, поскольку вы столкнулись с ограничениями традиционных моделей, которые по историческим причинам (ограничения старых компьютеров) базируются на последовательном принципе.

Появление новых аппаратных технологий дало нам возможность запускать на наших компьютерах несколько программ одновременно. В действительности даже самые простые компьютеры оснащены многоядерной системой, которая позволяет программам работать в параллельном режиме. Почему бы тогда не воспользоваться преимуществами этой архитектуры?

Слишком часто нам приходится писать программы на Python для выполнения серии операций. В научной сфере для выполнения трудоемких вычислений зачастую необходимо использовать множество алгоритмов. Но в конце работы, запустив программу на своем компьютере, вы с разочарованием обнаруживаете, что она работает не так быстро, как вы надеялись, и время выполнения только увеличивается по мере возрастания объема обрабатываемой задачи. Но дело не только в скорости. Сегодня все чаще нам приходится иметь дело со все большими объемами данных, и для связанных с ними вычислений программам требуется все больше ресурсов памяти, с которыми, несмотря на свою мощность, наши компьютеры не справляются.

Параллельное программирование позволяет выполнять фрагменты кода одной из наших программ одновременно, что значительно повышает производительность. Таким образом, параллельное программирование позволяет сократить время выполнения программы, более эффективно использовать ресурсы и иметь возможность выполнять более сложные операции, которые ранее считались недостижимыми.

Технологическая эволюция компьютеров и параллелизм

Сегодня для многих программистов параллельное программирование по-прежнему остается чем-то непривычным, поскольку это довольно новая технология. Дело в том, что еще несколько лет назад все компьютеры, доступные разработчикам, были оснащены одним единственным арифметическим логическим устройством (Arithmetic Logic Unit, ALU), и возможным было только последовательное программирование. Инструкции программы выполнялись по одной в последовательном порядке (см. рис. 1.1):

Рис. 1.1. Последовательное выполнение

Многие из вас, безусловно, помнят характеристики компьютера, которые обычно отображались в виде частоты процессора в герцах (Гц), что указывало на количество инструкций, которые могут быть выполнены в секунду. Мощность компьютера в первую очередь измерялась его вычислительной частотой. Чем выше было это значение, тем быстрее работали программы.

Концепция параллелизма формировалась постепенно, по мере развития аппаратного обеспечения компьютеров. До 1980-х годов такие возможности компьютеров были весьма ограниченными: они выполняли одну программу за другой, строго последовательно, команда за командой. Разумеется, в таких технологических условиях концепция параллелизма была совершенно невообразима.

С появлением процессора Intel 80386 возникла возможность прерывать выполнение одной программы для работы с другой. В результате появились такие концепции, как превентивное программирование и чередование процессов. Этот технологический прорыв привел к появлению эффекта псевдопараллелизма, поскольку для пользователя создавалось впечатление одновременной работы нескольких программ. С появлением последующего процессора Intel 80486 ситуация значительно улучшилась благодаря внедрению конвейерной системы, основанной на разделении программ на подзадачи. Они выполнялись независимо, чередуясь друг с другом для различных программ. Кроме того, внутренняя архитектура впервые позволила собирать несколько разных инструкций (даже из разных программ) и выполнять их все вместе одновременно (но не синхронно). И именно на этом этапе произошло реальное становление параллельного программирования. Фрагменты инструкций различных подзадач выполняются по порядку, чтобы быть выполненными как можно скорее (см. рис. 1.2):

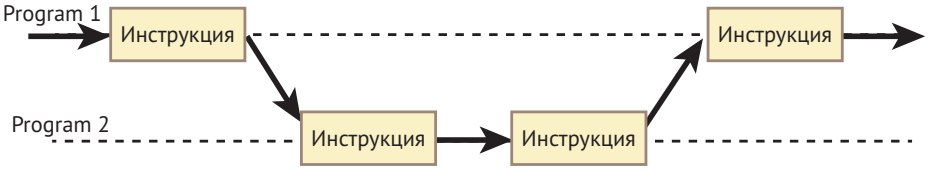

Рис. 1.2. Параллельное выполнение

Это продолжалось более десятилетия, в течение которого выпускались все более мощные модели процессоров, способные работать на более высоких частотах, чем предыдущие. Но вскоре ситуация перешла в кризисную фазу из-за ряда проблем и физических ограничений. Увеличение частоты исполнения приводит к одновременному увеличению тепловыделения и, как следствие, энергопотребления. Стало очевидно, что повышение частоты рано или поздно достигнет границ возможного.

Так процессоры достигли нового уровня инноваций с внедрением ядер в свою систему. Эти ядра, также известные как логические процессоры, позволяют имитировать наличие нескольких процессоров в одном центральном про-

цессоре, в результате чего получаются многоядерные процессоры. На практике это привело к появлению многопроцессорных компьютеров, способных одновременно и параллельно выполнять инструкции из разных программ. Именно поэтому в начале 2000-х годов появилось параллельное программирование, дающее разработчикам возможность одновременно выполнять различные фрагменты одной и той же программы.

Процессоры, ядра, потоки и процессы

Чтобы понять концепции, рассматриваемые в этой книге, необходимо сначала разобраться, что такое потоки (threads) и процессы (processes) и как они взаимосвязаны с режимами обработки данных центральным процессором и ядрами.

Это не какие-то абстрактные понятия, а реальные объекты, существующие в нашей операционной системе. Познакомиться с ними можно непосредственно в нашей операционной системе. Например, если вы работаете в Windows, откройте **Task Manager** (Диспетчер задач) и перейдите на вкладку **Performance** (Производительность).

Вы увидите окно, очень похожее на то, что показано на рис. 1.3, где можно в режиме реального времени отслеживать потребление различных ресурсов, таких как процессор, память и сеть Wi-Fi:

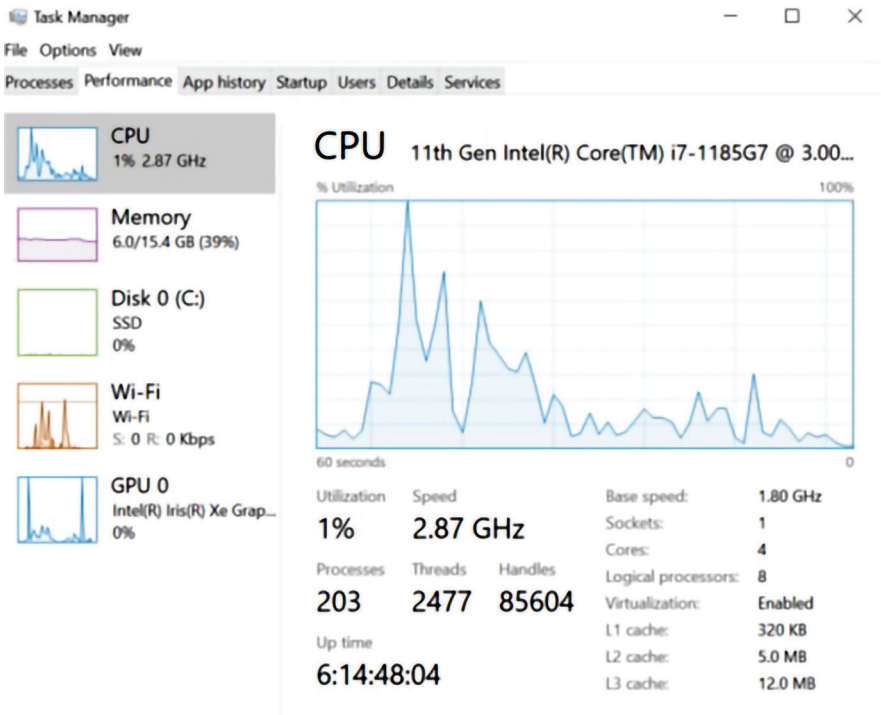

Рис. 1.3. Диспетчер задач в Windows

Кроме того, здесь также отображается различная информация, такая как количество процессов и запущенных в данный момент потоков. Справа перечислены некоторые характеристики системы, на которой мы работаем, например количество ядер.

Если же вы работаете в системах Linux, таких как Ubuntu, вы можете запустить соответствующее приложение, введя в терминале:

```
$ top
```

Появится экран, очень похожий на представленный на рис. 1.4:

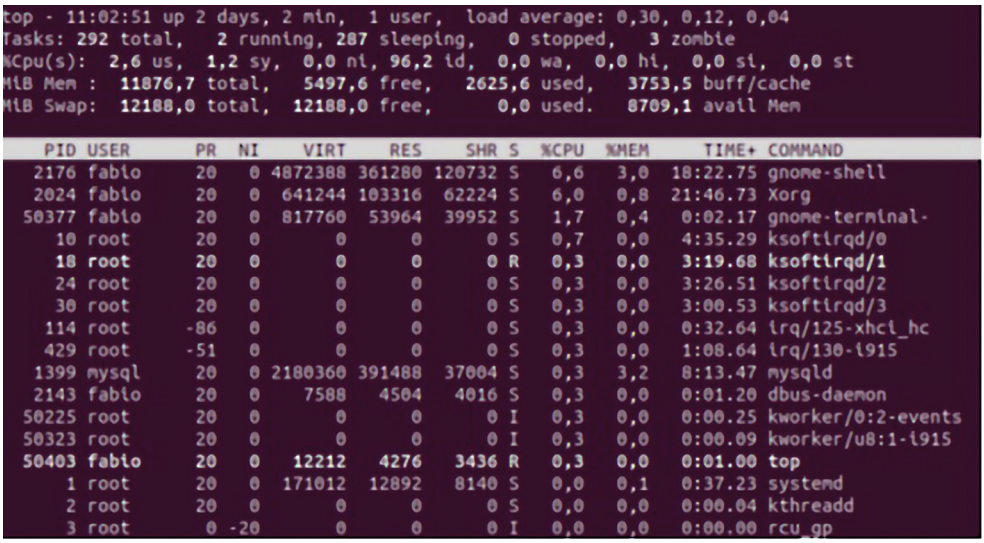

Рис. 1.4. top в терминале Ubuntu

Как можно видеть, в верхней части отображаются все используемые ресурсы с их значениями, которые обновляются в режиме реального времени. В нижней части находится список всех активных процессов в операционной системе. Каждый процесс идентифицируется уникальным номером – идентификационным номером процесса (process identification number, PID).

Поскольку системы Linux гораздо более гибкие и мощные прежде всего благодаря многочисленным командам оболочки, мы также можем отслеживать все потоки, связанные с каждой отдельной командой. Для этой цели воспользуемся более специфической командой для мониторинга процессов: `pid`.

```
$ pid -T <PID>
```

Опция `-T` используется для отображения потоков, которые являются вполне реалистичными для процесса. Затем `pid` процесса, который вы хотите отслеживать в деталях, передается команде `pid`. В данном случае, при выборе, например, процесса с `pid 2176`, будет получен результат, показанный на рис. 1.5, на котором все потоки предыдущего процесса отображаются с их идентификационным номером, `SPID`:

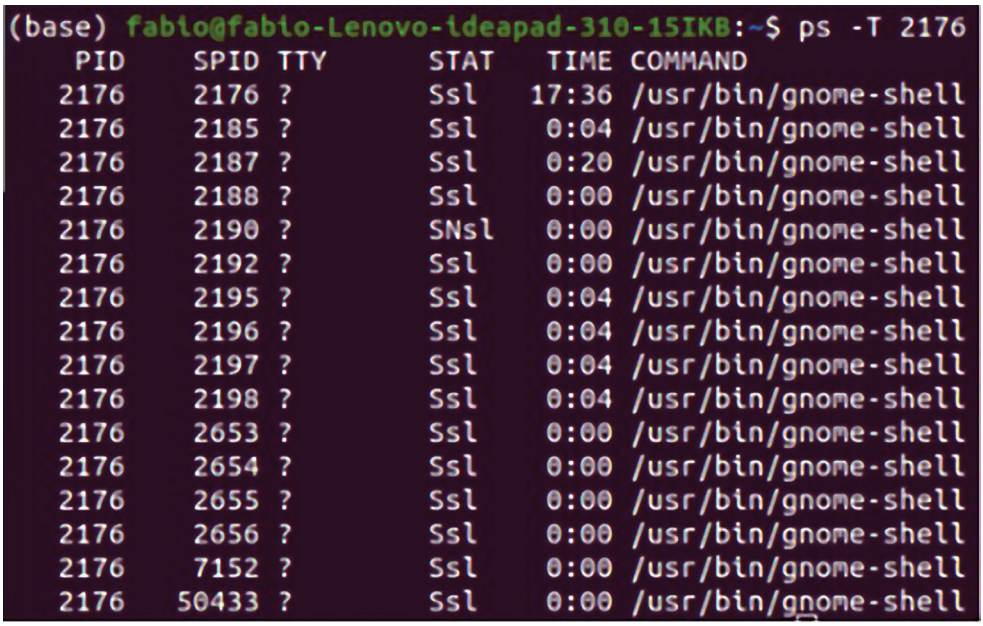

Рис. 1.5. Результаты выполнения команды pid в терминале Ubuntu

Центральный процессор (Central Processing Unit, CPU) – это истинный «мозг» нашего компьютера, где в основном и происходит обработка нашего кода. Производительность CPU характеризуется циклами, или единицами времени, которые он использует для выполнения операции. Обычно мы указываем мощность центрального процессора как частоту циклов в секунду (см. значение скорости 2,87 ГГц на рис. 1.3).

Центральный процессор может иметь одно (одноядерный процессор) или несколько ядер (многоядерный процессор). Ядра (cores) представляют собой блоки выполнения данных внутри центрального процессора. Каждое ядро способно выполнять несколько процессов. Процесс (process) – это, по сути, программа, которая запускается на компьютере и для которой зарезервирован участок памяти. Кроме того, каждый процесс сам может запускать другие процессы (подпроцессы) либо запускать один (`MainThread`) или несколько потоков (threads) внутри себя. Графическое представление всего этого показано на рис. 1.6.

Потоки, в свою очередь, можно рассматривать как подпроцессы, которые выполняются одновременно в рамках одного процессора. Как и процессы, потоки также имеют ряд схожих механизмов, которые управляют их синхронизацией, обменом данными и переходами состояний во время их выполнения: готово (ready), выполняется (running) и заблокировано (blocked).

Это общая структура, которую следует принимать во внимание, чтобы лучше понять принципы работы процессов и потоков в наших машинах и, следовательно, наилучшим образом реализовать программирование в параллельном режиме.

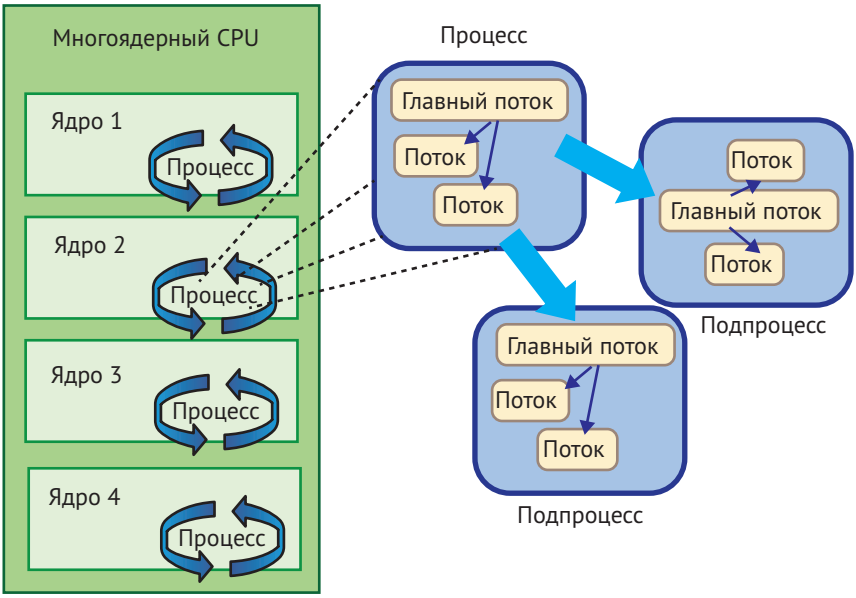

Рис. 1.6. Центральный процессор, ядро, процесс и потоки

Многозадачное и параллельное программирование

Очень часто возникает путаница между концепциями многозадачности (concurrency) и параллелизма (parallelism), поэтому эти два термина иногда используются как синонимы, но это нельзя считать корректным. Эти две концепции, хотя и взаимосвязаны, все же отличаются в контексте параллельного программирования, и очень важно понимать эти различия.

Начнем с того, что общего у этих двух концепций. И многозадачность, и параллелизм имеются в программах, которые должны выполнять несколько задач одновременно. Но именно в этом и заключается смысл многозадачности.

Многозадачность означает одновременное управление (а не выполнение) несколькими задачами, но они не обязательно будут выполняться одновременно.

Таким образом, даже программа, которая должна выполнять несколько задач одновременно, может делать это, только обрабатывая одну задачу за раз. Как только программа завершит выполнение инструкций, относящихся к какой-либо задаче или ее части (подзадаче), она перейдет к следующей задаче и т. д. Одна задача будет сменяться другой, и программа завершит свою работу. Если это поможет, можно представить себе, что задачи конкурируют друг с другом за право на выполнение.

Таким образом, даже если наш компьютер оснащен одноядерным процессором, конкурирующая программа без проблем запустится (см. рис. 1.7):