Параллельное программирование на C++ в действии

Энтони Уильямс

Практика разработки многопоточных программ

ДМК издательство

MANNING

Энтони Уильямс

Параллельное программирование на C++ в действии

Практика разработки многопоточных программ

C++ Concurrency in Action

Practical Multithreading

ANTHONY WILLIAMS

MANNING
SHELTER ISLAND

Параллельное программирование на C++ в действии

Практика разработки многопоточных программ

ЭНТОНИ УИЛЬЯМС

2-е издание, электронное

ДМК ПРЕСС
издательство
Москва, 2023

УДК 004.438С++11
ББК 32.973.26-018.2
У36

Уильямс, Энтони.
У36 Параллельное программирование на С++ в действии. Практика разработки многопоточных программ / Э. Уильямс ; пер. с англ. А. А. Слинкина. — 2-е изд., эл. — 1 файл pdf : 674 с. — Москва : ДМК Пресс, 2023. — Систем. требования: Adobe Reader XI либо Adobe Digital Editions 4.5 ; экран 10". — Текст : электронный.

ISBN 978-5-89818-319-6

В наши дни компьютеры с несколькими многоядерными процессорами стали нормой. Стандарт С++11 языка С++ предоставляет развитую поддержку многопоточности в приложениях. Поэтому, чтобы сохранять конкурентоспособность, вы должны овладеть принципами и приемами их разработки, а также новыми средствами языка, относящимися к параллелизму.

Книга «Параллельное программирование на С++ в действии» не предполагает предварительных знаний в этой области. Вдумчиво читая ее, вы научитесь писать надежные и элегантные многопоточные программы на С++11. Вы узнаете о том, что такое потоковая модель памяти, и о том, какие средства поддержки многопоточности, в том числе запуска и синхронизации потоков, имеются в стандартной библиотеке. Попутно вы познакомитесь с различными нетривиальными проблемами программирования в условиях параллелизма.

УДК 004.438С++11
ББК 32.973.26-018.2

Электронное издание на основе печатного издания: Параллельное программирование на С++ в действии. Практика разработки многопоточных программ / Э. Уильямс ; пер. с англ. А. А. Слинкина. — Москва : ДМК Пресс, 2014. — 672 с. — ISBN 978-5-94074-537-2. — Текст : непосредственный.

ISBN 978-5-89818-319-6

Ким, Хью и Ирен.

ОГЛАВЛЕНИЕ

ПРИЛОЖЕНИЕ D. Справочник по библиотеке C++ Thread Library 492

ПРЕДИСЛОВИЕ

С идеей многопоточного программирования я столкнулся на своей первой работе после окончания колледжа. Мы занимались приложением, которое должно было помещать входные записи в базу данных. Данных было много, но все они были независимы и требовали значительной предварительной обработки. Чтобы задействовать всю мощь нашего десятипроцессорного компьютера UltraSPARC, мы организовали несколько потоков, каждый из которых обрабатывал свою порцию входных данных. Код был написан на языке C++, с использованием потоков POSIX. Ошибок мы наделали кучу – многопоточность для всех была внове – но до конца все-таки добрались. Именно во время работы над этим проектом я впервые услыхал о комитете по стандартизации C++ и о недавно опубликованном стандарте языка C++.

С тех мой интерес к многопоточному программированию и параллелизму не затухает. Там, где другим видятся трудности и источник разнообразных проблем, я нахожу мощный инструмент, который позволяет программе использовать всё наличное оборудование и в результате работать быстрее. Позднее я научился применять эти идеи и при наличии всего одного процессора или ядра, чтобы улучшить быстроту реакции и повысить производительность, – благодаря тому, что одновременная работа нескольких потоков дает программе возможность не простаивать во время таких длительных операций, как ввод/вывод. Я также узнал, как это устроено на уровне ОС и как в процессорах Intel реализовано контекстное переключение задач.

Тем временем интерес к C++ свел меня с членами Ассоциации пользователей C и C++ (ACCU), а затем с членами комиссии по стандартизации C++ при Институте стандартов Великобритании (BSI) и разработчиками библиотек Boost. Я с интересом наблюдал за началом разработки библиотеки многопоточности Boost, а когда автор забросил проект, я воспользовался шансом перехватить инициативу. С тех пор разработка и сопровождение библиотеки Boost Thread Library лежит в основном на мне.

По мере того как в работе комитета по стандартизации C++ наметился сдвиг от исправления дефектов в существующем стандарте в сторону выработки предложений для нового стандарта (получившего условное название C++0x в надежде, что его удастся завершить до 2009 года, и официально названного C++11, так как он наконец был опубликован в 2011 году), я стал принимать более активное участие в деятельности BSI и даже вносить собственные предложения. Когда стало ясно, что многопоточность стоит на повестке дня, я по-настоящему встрепенулся – многие вошедшие в стандарт предложения по многопоточности и параллелизму написаны как мной самим, так и в соавторстве с коллегами. Я считаю большой удачей, что таким образом удалось совместить две основных сферы моих интересов в области программирования – язык C++ и многопоточность.

В этой книге, опирающейся на весь мой опыт работы с C++ и многопоточностью, я ставил целью научить других программистов, как безопасно и эффективно пользоваться библиотекой C++11 Thread Library. Надеюсь, что мне удастся заразить читателей своим энтузиазмом.

БЛАГОДАРНОСТИ

Прежде всего, хочу сказать огромное спасибо своей супруге, Ким, за любовь и поддержку, которую она выказывала на протяжении работы над книгой, отнимавшей изрядную долю моего свободного времени за последние четыре года. Без ее терпения, ободрения и понимания я бы не справился.

Далее я хочу поблагодарить коллектив издательства Manning, благодаря которому эта книга появилась на свет: Марджана Баджи (Marjan Bace), главного редактора; Майкла Стивенса (Michael Stephens), его заместителя; Синтию Кейн (Cynthia Kane), моего редактора-консультанта; Карен Тегтмейер (Karen Tegtmeyer), выпускающего редактора; Линду Ректенвальд (Linda Recktenwald), редактора; Кати Теннант (корректора) и Мэри Пирджис, начальника производства. Без их стараний вы не читали бы сейчас эту книгу. Я хочу также поблагодарить других членов комитета по стандартизации C++, которые подавали на рассмотрение материалы, относящиеся к многопоточности: Андрея Александреску (Andrei Alexandrescu), Пита Беккера (Pete Becker), Боба Блэйнера (Bob Blainer), Ханса Бема (Hans Boehm), Бимана Доуса (Beman Dawes), Лоуренса Кроула (Lawrence Crowl), Петера Димова (Peter Dimov), Джеффа Гарланда (Jeff Garland), Кевлина Хэнни (Kevlin Henney), Ховарда Хиннанта (Howard Hinnant), Бена Хатчингса (Ben Hutchings), Йана Кристоферсона (Jan Kristofferson), Дуга Ли (Doug Lea), Пола Маккинни (Paul McKenney), Ника Макларена (Nick McLaren), Кларка Нельсона (Clark Nelson), Билла Пью (Bill Pugh), Рауля Силвера (Raul Silvera), Герба Саттера (Herb Sutter), Детлефа Вольмана (Detlef Vollmann) и Майкла Вонга (Michael Wong), а также всех тех, кто рецензировал материалы, принимал участие в их обсуждении на заседаниях комитета и иными способами содействовал оформлению поддержки многопоточности и параллелизма в C++11.

Наконец, хочу выразить благодарность людям, чьи предложения позволили заметно улучшить книгу: д-ру Джейми Оллсопу (Jamie Allsop), Петеру Димову, Ховарду Хиннанту, Рику Моллою (Rick

Molloy), Джонатану Уэйкли (Jonathan Wakely) и д-ру Расселу Уиндеру (Russel Winder). Отдельное спасибо Расселу за подробные рецензии и Джонатану, который в качестве технического редактора, тщательно проверил окончательный текст на предмет наличия вопиющих ошибок. (Все оставшиеся ошибки – целиком моя вина.) И напоследок выражаю признательность группе рецензентов: Райану Стивенсу (Ryan Stephens), Нилу Хорлоку (Neil Horlock), Джону Тэйлору младшему (John Taylor Jr.), Эзре Дживану (Ezra Jivan), Джошуа Хейеру (Joshua Heyer), Киту С. Киму (Keith S. Kim), Мишель Галли (Michele Galli) Майку Тянь Чжань Чжану (Mike Tian-Jian Jiang), Дэвиду Стронгу (David Strong), Роджеру Орру (Roger Orr), Вагнеру Рику (Wagner Rick), Майкй Буксасу (Mike Buksas) и Бас Воде (Bas Vodde). Также спасибо всем читателям предварительного издания, которые нашли время указать на ошибки и отметить места, нуждающиеся в уточнении.

ОБ ЭТОЙ КНИГЕ

Эта книга представляет собой углубленное руководство по средствам поддержки многопоточности и параллелизма в новом стандарте C++, от базового использования классов и функций из пространств имен std::thread, std::mutex и std::async до сложных вопросов, связанных с атомарными операциями и моделью памяти.

Структура книги

В первых четырех главах описываются различные библиотечные средства и порядок работы с ними.

Глава 5 посвящена низкоуровневым техническим деталям модели памяти и атомарных операций. В частности, рассматривается вопрос об использовании атомарных операций для задания ограничений на порядок выполнения других частей программы. Вводные главы на этом заканчиваются.

В главах 6 и 7 начинается изучение программирования на более высоком уровне, с примерами использования базовых средств для построения сложных структур данных – с блокировками (глава 6) и без блокировок (глава 7).

В главе 8 эта линия продолжается: даются рекомендации по проектированию многопоточных программ, рассматриваются аспекты, влияющие на производительность, и приводятся примеры реализации различных параллельных алгоритмов.

Глава 9 посвящена средствам управления потоками, рассматриваются пулы потоков, очереди работ и прерывание операций.

Тема главы 10 – тестирование и отладка: типы ошибок, методы их отыскания, способы тестирования и так далее.

В приложениях вы найдете краткое описание некоторых языковых средств, добавленных в новый стандарт и имеющих отношение к многопоточности; детали реализации библиотеки передачи сообщениями, упомянутой в главе 4, и полный справочник по библиотеке C++11 Thread Library.

На кого рассчитана эта книга

Если вы пишете многопоточный код на C++, то эта книга для вас. Если вы пользуетесь средствами многопоточности из стандартной библиотеки C++, то здесь вы найдете руководство по основным вопросам. Если вы работаете с другими библиотеками многопоточности, то описанные рекомендации и приемы все равно могут оказаться полезным подспорьем.

Предполагается владение языком C++ на рабочем уровне, но предварительное знакомство с новыми языковыми средствами необязательно – они описаны в приложении A. Также не требуются знания или опыт работы в области многопоточного программирования, хотя их наличие было бы плюсом.

Как пользоваться этой книгой

Если раньше вы не писали многопоточных программ, то я рекомендую читать книгу последовательно от начала до конца, опустив, быть может, кое-какие детали из главы 5. Глава 7 опирается на материал главы 5, поэтому если вы пропустите главу 5, то отложите также чтение седьмой главы.

Если вам не доводилось использовать новые языковые средства, вошедшие в стандарт C++11, то имеет смысл с самого начала бегло просмотреть приложение A, чтобы понимать приведенные в тексте примеры. Впрочем, в основном тексте упоминания о новых средствах графически выделены, так что, встретив что-то незнакомое, вы всегда можете обратиться к приложению.

Если вы располагаете обширным опытом написания многопоточного кода в других средах, то все-таки стоит просмотреть начальные главы, чтобы понять, как знакомые вам понятия соответствуют средствам из нового стандарта C++. Если вы планируете работать с атомарными переменными на низком уровне, то главу 5 следует изучить обязательно. Полезно также ознакомиться с главой 8, где рассказывается о безопасности исключений в многопоточных программах на C++. Если перед вами стоит конкретная задача, то указатель и оглавление помогут быстро найти соответствующий раздел.

Даже после того как вы освоите библиотеку C++ Thread Library, приложение D все равно останется полезным, потому что в нем легко найти детали использования каждого класса и функции. Время от времени вы, наверное, будет заглядывать и в основные главы, чтобы освежить в памяти порядок работы с той или иной конструкцией или взглянуть на пример кода.

Графические выделения и загрузка исходного кода

Исходный код в листингах и основном тексте набран моноширинным шрифтом. Многие листинги сопровождаются аннотациями, в которых излагаются важные концепции. В некоторых случаях в листингах присутствуют нумерованные маркеры, с которыми соотносятся последующие пояснения.

Исходный код всех примеров можно скачать с сайта издательства по адресу www.manning.com/CPlusPlusConcurrencyinAction.

Требования к программному обеспечению

Чтобы приведенный в этой книге код работал без модификаций, понадобится версия компилятора C++ с поддержкой тех вошедших в стандарт C++11 средств, которые перечислены в приложении A. Кроме того, нужна стандартная библиотека многопоточности C++ (Standard Thread Library).

На момент написания этой книги единственный известный мне компилятор, поставляемый с библиотекой Standard Thread Library, – это g++, хотя в предварительную версию Microsoft Visual Studio 2011 она также входит. Что касается g++, то первая реализация основных возможностей библиотеки многопоточности была включена в версию g++ 4.3, а впоследствии добавлялись улучшения и расширения. Кроме того, в g++ 4.3 впервые появилась поддержка некоторых новых языковых средств C++11, и в каждой новой версии она расширяется. Дополнительные сведения см. на странице текущего состояния реализации C++11 в g++[1].

В составе Microsoft Visual Studio 2010 также имеются некоторые новые средства из стандарта C++11, например лямбда-функции и ссылки на r-значения, но реализация библиотеки Thread Library отсутствует.

Моя компания, Just Software Solutions Ltd, продает полную реализацию стандартной библиотеки C++11 Standard Thread Library для Microsoft Visual Studio 2005, Microsoft Visual Studio 2008, Microsoft Visual Studio 2010 различных версий g++[2]. Именно эта реализация применялась для тестирования примеров из этой книги.

---

[1] Страница состояния компилятора GNU C++0x/C++11 http://gcc.gnu.org/projects/cxx0x.html.

[2] Реализация just::thread библиотеки C++ Standard Thread Library, http://www.stdthread.co.uk.

В библиотеке Boost Thread Library[3], протестированной на многих платформах, реализован API, основанный на предложениях, поданных в комитет по стандартизации C++. Большинство приведенных в книге примеров будут работать с Boost Thread Library, если заменить `std::` на `boost::` и включить подходящие директивы `#include`. Но некоторые возможности в библиотеке Boost Thread Library либо не поддерживаются вовсе (например, `std::async`), либо называются по-другому (например, `boost::unique_future`).

Автор в Сети

Приобретение книги «Параллелизм на C++ в действии» открывает бесплатный доступ к закрытому форуму, организованному издательством Manning Publications, где вы можете оставить свои комментарии к книге, задать технические вопросы и получить помощь от автора и других пользователей. Получить доступ к форуму и подписаться на список рассылки можно на странице www.manning.com/CPlusPlusConcurrencyinAction. Там же написано, как зайти на форум после регистрации, на какую помощь можно рассчитывать, и изложены правила поведения в форуме.

Издательство Manning обязуется предоставлять читателям площадку для общения с другими читателями и автором. Однако это не означает, что автор обязан как-то участвовать в обсуждениях; его присутствие на форуме остается чисто добровольным (и не оплачивается). Мы советуем задавать автору хитроумные вопросы, чтобы его интерес к форуму не угасал!

Форум автора в сети и архивы будут доступны на сайте издательства до тех пор, пока книга не перестанет печататься.

[3] Библиотеки Boost для C++, http://www.boost.org.

ОБ ИЛЛЮСТРАЦИИ НА ОБЛОЖКЕ

Рисунок на обложке книги «Параллельное программирование на C++ в действии» называется «Традиционный костюм японской девушки». Репродукция взята из четырехтомного «Собрания костюмов разных народов», напечатанного в Лондоне между 1757 и 1772 годом. Это издание, включающее изумительные раскрашенные вручную гравюры на меди с изображениями одежды народов мира, оказало большое влияние на дизайн театральных костюмов. Разнообразие рисунков позволяет составить наглядное представление о великолепии костюма на Лондонской сцене свыше 200 лет назад. Костюмы, исторические и того времени, позволяли познакомиться с традиционной одеждой людей, живших в разное время в разных странах, и тем самым сделать их ближе и понятнее лондонской театральной публике.

Манера одеваться за последние 100 лет сильно изменилась, и различия между областями, когда-то столь разительные, сгладились. Теперь трудно отличить друг от друга даже выходцев с разных континентов. Но можно взглянуть на это и с оптимизмом – мы обменяли культурное и визуальное разнообразие на иное устройство личной жизни – основанное на многостороннем и стремительном технологическом и интеллектуальном развитии.

Издательство Manning откликается на новации, инициативы и курьезы в компьютерной отрасли обложками своих книг, на которых представлено широкое разнообразие местных укладов и театральной жизни в позапрошлом веке. Мы возвращаем его в виде иллюстраций из этой коллекции.

ГЛАВА 1.
Здравствуй, параллельный мир!

В этой главе:

- Что понимается под параллелизмом и многопоточностью.
- Зачем использовать параллелизм и многопоточность в своих приложениях.
- Замечания об истории поддержки параллелизма в C++.
- Структура простой многопоточной программы на C++.

Для программистов на языке C++ настали радостные дни. Спустя тринадцать лет после публикации первой версии стандарта C++ в 1998 году комитет по стандартизации C++ решил основательно пересмотреть как сам язык, так и поставляемую вместе с ним библиотеку. Новый стандарт C++ (обозначаемый C++11 или C++0x), опубликованный в 2010 году, несет многочисленные изменения, призванные упростить программирование на C++ и сделать его более продуктивным.

К числу наиболее существенных новшеств в стандарте C++11 следует отнести поддержку многопоточных программ. Впервые комитет официально признал существование многопоточных приложений, написанных на C++, и включил в библиотеку компоненты для их разработки. Это позволит писать на C++ многопоточные программы с гарантированным поведением, не полагаясь на зависящие от платформы расширения. И как раз вовремя, потому что разработчики,

стремясь повысить производительность приложений, все чаще посматривают в сторону параллелизма вообще и многопоточного программирования в особенности.

Эта книга о том, как писать на C++ параллельные программы с несколькими потоками и о тех средствах самого языка и библиотеки времени выполнения, благодаря которым это стало возможно. Я начну с объяснения того, что понимаю под параллелизмом и многопоточностью и для чего это может пригодиться в приложениях. После краткого отвлечения на тему о том, когда программу *не* следует делать многопоточной, я в общих чертах расскажу о поддержке параллелизма в C++ и закончу главу примером простой параллельной программы. Читатели, имеющие опыт разработки многопоточных приложений, могут пропустить начальные разделы. В последующих главах мы рассмотрим более сложные примеры и детально изучим библиотечные средства. В конце книги приведено подробное справочное руководство по всем включенным в стандартную библиотеку C++ средствам поддержки многопоточности и параллелизма.

Итак, что же я понимаю под *параллелизмом* и *многопоточностью*?

1.1. Что такое параллелизм?

Если упростить до предела, то параллелизм – это одновременное выполнение двух или более операций. В жизни он встречается на каждом шагу: мы можем одновременно идти и разговаривать или одной рукой делать одно, а второй – другое. Ну и, разумеется, каждый из нас живет своей жизнью независимо от других – вы смотрите футбол, я в это время плаваю и т. д.

1.1.1. Параллелизм в вычислительных системах

Говоря о параллелизме в контексте компьютеров, мы имеем в виду, что одна и та же система выполняет несколько независимых операций параллельно, а не последовательно. Идея не нова: многозадачные операционные системы, позволяющие одновременно запускать на одном компьютере несколько приложений с помощью переключения между задачами уже много лет как стали привычными, а дорогие серверы с несколькими процессорами, обеспечивающие истинный параллелизм, появились еще раньше. *Новым* же является широкое распространение компьютеров, которые не просто создают иллюзию

одновременного выполнения задач, а действительно исполняют их параллельно.

Исторически компьютеры, как правило, оснащались одним процессором с одним блоком обработки, или ядром, и это остается справедливым для многих настольных машин и по сей день. Такая машина в действительности способна исполнять только одну задачу в каждый момент времени, но может переключаться между задачами много раз в секунду. Таким образом, сначала одна задача немножко поработает, потом другая, а в итоге складывается впечатление, будто все происходит одновременно. Это называется *переключением задач*. Тем не менее, и для таких систем мы можем говорить о *параллелизме*: задачи сменяются очень часто и заранее нельзя сказать, в какой момент процессор приостановит одну и переключится на другую. Переключение задач создает иллюзию параллелизма не только у пользователя, но и у самого приложения. Но так как это всего лишь *иллюзия*, то между поведением приложения в однопроцессорной и истинно параллельной среде могут существовать тонкие различия. В частности, неверные допущения о модели памяти (см. главу 5) в однопроцессорной среде могут не проявляться. Подробнее эта тема рассматривается в главе 10.

Компьютеры с несколькими процессорами применяются для организации серверов и выполнения высокопроизводительных вычислений уже много лет, а теперь машины с несколькими ядрами на одном кристалле (многоядерные процессоры) все чаще используются в качестве настольных компьютеров. И неважно, оснащена машина несколькими процессорами или одним процессором с несколькими ядрами (или комбинацией того и другого), она все равно может исполнять более одной задачи в каждый момент времени. Это называется *аппаратным параллелизмом*.

На рис. 1.1 показан идеализированный случай: компьютер, исполняющий ровно две задачи, каждая из которых разбита на десять одинаковых этапов. На двухъядерной машине каждая задача может исполняться в своем ядре. На одноядерной машине с переключением задач этапы той и другой задачи чередуются. Однако между ними существует крохотный промежуток времени (на рисунке эти промежутки изображены в виде серых полосок, разделяющих более широкие этапы выполнения) – чтобы обеспечить чередование, система должна произвести *контекстное переключение* при каждом переходе от одной задачи к другой, а на это требуется время. Чтобы переключить контекст, ОС должна сохранить состояние процессора и счетчик ко-

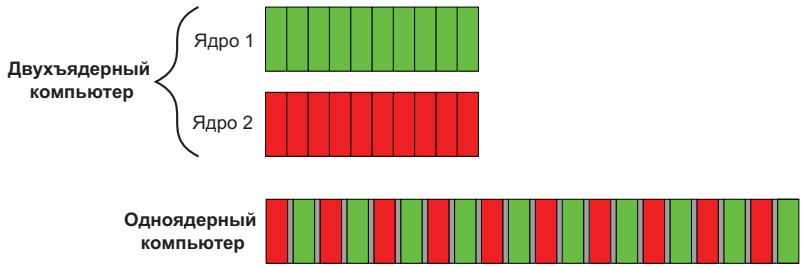

Рис. 1.1. Два подхода к параллелизму: параллельное выполнение на двухъядерном компьютере и переключение задач на одноядерном

манд для текущей задачи, определить, какая задача будет выполняться следующей, и загрузить в процессор состояние новой задачи. Не исключено, что затем процессору потребуется загрузить команды и данные новой задачи в кэш-память; в течение этой операции никакие команды не выполняются, что вносит дополнительные задержки.

Хотя аппаратная реализация параллелизма наиболее наглядно проявляется в многопроцессорных и многоядерных компьютерах, существуют процессоры, способные выполнять несколько потоков на одном ядре. В действительности существенным фактором является количество *аппаратных потоков* – характеристика числа независимых задач, исполняемых оборудованием по-настоящему одновременно. И наоборот, в системе с истинным параллелизмом количество задач может превышать число ядер, тогда будет применяться механизм переключения задач. Например, в типичном настольном компьютере может быть запущено несколько сотен задач, исполняемых в фоновом режиме даже тогда, когда компьютер по видимости ничего не делает. Именно за счет переключения эти задачи могут работать параллельно, что и позволяет одновременно открывать текстовый процессор, компилятор, редактор и веб-браузер (да и вообще любую комбинацию приложений). На рис. 1.2 показано переключение четырех задач на двухъядерной машине, опять-таки в идеализированном случае, когда задачи разбиты на этапы одинаковой продолжительности. На практике существует много причин, из-за которых разбиение неравномерно и планировщик выделяет процессор каждой задаче не столь регулярно. Некоторые из них будут рассмотрены в главе 8 при обсуждении факторов, влияющих на производительность параллельных программ.

Все рассматриваемые в этой книге приемы, функции и классы применимы вне зависимости от того, исполняется приложение на ма-

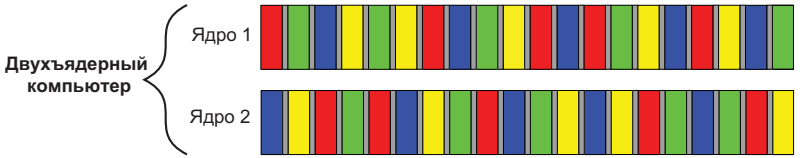

Рис. 1.2. Переключение задач на двухъядерном компьютере

шине с одноядерным процессором или с несколькими многоядерными процессорами. Не имеет значения, как реализован параллелизм: с помощью переключения задач или аппаратно. Однако же понятно, что способ использования параллелизма в приложении вполне может зависеть от располагаемого оборудования. Эта тема обсуждается в главе 8 при рассмотрении вопросов проектирования параллельного кода на C++.

1.1.2. Подходы к организации параллелизма

Представьте себе пару программистов, работающих над одним проектом. Если они сидят в разных кабинетах, то могут мирно трудиться, не мешая друг другу, причем у каждого имеется свой комплект документации. Но общение при этом затруднено – вместо того чтобы просто обернуться и обменяться парой слов, приходится звонить по телефону, писать письма или даже встать и дойти до коллеги. К тому же, содержание двух кабинетов сопряжено с издержками, да и на несколько комплектов документации надо будет потратиться.

А теперь представьте, что всех разработчиков собрали в одной комнате. У них появилась возможность обсуждать между собой проект приложения, рисовать на бумаге или на доске диаграммы, обмениваться мыслями. Содержать придется только один офис и одного комплекта документации вполне хватит. Но есть и минусы – теперь им труднее сконцентрироваться и могут возникать проблемы с общим доступом к ресурсам («Ну куда опять запропастилось это справочное руководство?»).

Эти два способа организации труда разработчиков иллюстрируют два основных подхода к параллелизму. Разработчик – это модель потока, а кабинет – модель процесса. В первом случае имеется несколько однопоточных процессов (у каждого разработчика свой кабинет), во втором – несколько потоков в одном процессе (два разработчика в одном кабинете). Разумеется, возможны произвольные комбинации:

может быть несколько процессов, многопоточных и однопоточных, но принцип остается неизменным. А теперь поговорим немного о том, как эти два подхода к параллелизму применяются в приложениях.

Параллелизм за счет нескольких процессов

Первый способ распараллелить приложение – разбить его на несколько однопоточных одновременно исполняемых процессов. Именно так вы и поступаете, запуская вместе браузер и текстовый процессор. Затем эти отдельные процессы могут обмениваться сообщениями, применяя стандартные каналы межпроцессной коммуникации (сигналы, сокеты, файлы, конвейеры и т. д.), как показано на рис. 1.3. Недостаток такой организации связи между процессами в его сложности, медленности, а иногда том и другом вместе. Дело в том, что операционная система должна обеспечить защиту процессов, так чтобы ни один не мог случайно изменить данные, принадлежащие другому. Есть и еще один недостаток – неустранимые накладные расходы на запуск нескольких процессов: для запуска процесса требуется время, ОС должна выделить внутренние ресурсы для управления процессом и т. д.

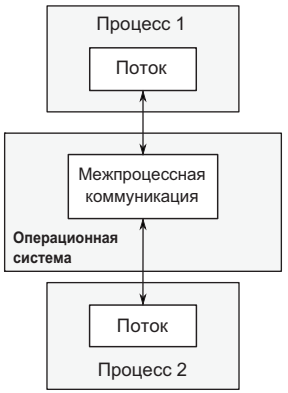

Рис. 1.3. Коммуникация между двумя параллельно работающими процессами

Конечно, есть и плюсы. Благодаря надежной защите процессов, обеспечиваемой операционной системой, и высокоуровневым механизмам коммуникации написать *безопасный* параллельный код проще, когда имеешь дело с процессами, а не с потоками. Например, в среде исполнения, создаваемой языком программирования Erlang, в качестве фундаментального механизма параллелизма используются процессы, и это дает отличный эффект.

У применения процессов для реализации параллелизма есть и еще одно достоинство – процессы можно запускать на разных машинах, объединенных сетью. Хотя затраты на коммуникацию при этом возрастают, но в хорошо спроектированной системе такой способ повышения степени параллелизма может оказаться очень эффективным, и общая производительность увеличится.

Параллелизм за счет нескольких потоков

Альтернативный подход к организации параллелизма – запуск нескольких потоков в одном процессе. Потоки можно считать облегченными процессами – каждый поток работает независимо от всех остальных, и все потоки могут выполнять разные последовательности команд. Однако все принадлежащие процессу потоки разделяют общее адресное пространство и имеют прямой доступ к большей части данных – глобальные переменные остаются глобальными, указатели и ссылки на объекты можно передавать из одного потока в другой. Для процессов тоже можно организовать доступ к разделяемой памяти, но это и сделать сложнее, и управлять не так просто, потому что адреса одного и того же элемента данных в разных процессах могут оказаться разными. На рис. 1.4 показано, как два потока в одном процессе обмениваются данными через разделяемую память.

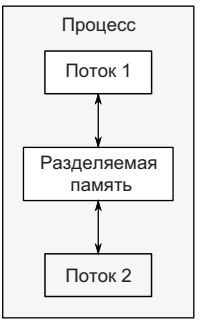

Рис. 1.4. Коммуникация между двумя параллельно исполняемыми потоками в одном процессе

Благодаря общему адресному пространству и отсутствию защиты данных от доступа со стороны разных потоков накладные расходы, связанные с наличием нескольких потоков, существенно меньше, так как на долю операционной системы выпадает гораздо меньше учетной работы, чем в случае нескольких процессов. Однако же за гибкость разделяемой памяти приходится расплачиваться – если к некоторому элементу данных обращаются несколько потоков, то программист должен обеспечить согласованность представления этого элемента во всех потоках. Возникающие при этом проблемы, а также средства и рекомендации по их разрешению рассматриваются на протяжении всей книги, а особенно в главах 3, 4, 5 и 8. Эти проблемы не являются непреодолимыми, надо лишь соблюдать осторожность при написании кода. Но само их наличие означает, что коммуникацию между потоками необходимо тщательно продумывать.

Низкие накладные расходы на запуск потоков внутри процесса и коммуникацию между ними стали причиной популярности этого подхода во всех распространенных языках программирования, включая C++, даже несмотря на потенциальные проблемы, связанные с разделением памяти. Кроме того, в стандарте C++ не оговаривается встроенная поддержка межпроцессной коммуникации, а, значит, при-