УДК 004.94

ИССЛЕДОВАНИЕ И АНАЛИЗ ВОЗМОЖНОСТИ РЕАЛИЗАЦИИ МНОГОПОТОЧНОСТИ ПРИ РАЗРАБОТКЕ ПО НА ЯЗЫКЕ С++

Раздобудов С.А., Сальников И.И.

Пензенский государственный технологический университет, Пенза, e-mail: alexey314@yandex.ru

В статье приводится исследование и анализ возможностей реализации многопоточности при разработке программ на языке С++. Для выбора оптимального метода распараллеливания рассмотрены две модели: параллельного выполнения задач и параллельного использования данных. Для реализации многопоточности в языке С++ имеется обширный набор методов, наиболее известными и часто используемыми среди которых являются: реализация с помощью класса thread, реализация с помощью pthreads в Unix системах; реализация с помощью класса QThread, входящего в состав фреймворка Qt. В заключение отмечено, что многопоточность играет важную роль в современном программировании. В приложениях с параллельным выполнением задач независимые операции, заключенные в функции, переносятся в выполняемые асинхронно потоки. Общим примером реализации параллелизма на уровне данных может служить типичный алгоритм обработки изображений, который применяет фильтр к одной или нескольким точкам изображения для вычисления нового значения для заданной точки.

Ключевые слова: операционная система, многопоточность, язык программирования, многозадачность, процессор, поток, параллелизм, компилятор

RESEARCH AND ANALYSIS OF THE FEASIBILITY OF MULTI-THREADING IN SOFTWARE DEVELOPMENT BY C++

Razdobudov S.A., Salnikov I.I.

Penza State Technological University, Penza, e-mail: alexey314@yandex.ru

The article analyzes and analyzes the potential for multithreading in the development of C ++ programs. To select the optimal parallelization method, two models are considered: parallel execution of tasks and parallel use of data. To implement multithreading in C ++, there is an extensive set of methods, the most known and often used among which are: implementation with the help of class thread, implementation with the help of pthreads on Unix systems; implementation using the QThread class, which is part of the Qt framework. In conclusion, it is noted that multithreading plays an important role in modern programming. In applications with parallel execution of tasks, independent operations enclosed in a function are transferred to asynchronously executing threads. A common example of the implementation of data-level parallelism is a typical image processing algorithm that applies a filter to one or more image points to calculate a new value for a given point.

Keywords: operating system, multithreading, programming language, multitasking, processor, thread, parallelism, compiler

Современные операционные системы (ОС) нацелены на наиболее эффективное использование ресурсов компьютера. В основном эффективность достигается за счет разделения ресурсов компьютера между несколькими процессами (многозадачность). Процессы могут выполняться одновременно за счет переключения центрального процессора между ними. Последние версии ОС предоставляют механизмы, позволяющие приложениям управлять ресурсами компьютера и распределять их с большей степенью детализации, т.е. на уровне потоков.

Сегодня публикаций по языку программирования С++ очень много. Но среди всего многообразия книг нельзя пропустить книгу [1], написанную самим автором языка С++ и считающуюся наиболее каноничным изложением возможностей, предоставляемых языком. Присутствуют многочисленные примеры, демонстрирующие как хороший стиль программирования на С-совместимом ядре, так и современный объектно-ориентированный подход к созданию программных продуктов.

В наши дни компьютеры с несколькими многоядерными процессорами стали нормой. Стандарт С++11 языка С++ предоставляет развитую поддержку многопоточности в приложениях, а книга [2], содержит исчерпывающую информацию о параллельном программировании на С++. Книга [3] посвящена разработке приложений для самых популярных операционных систем, таких как Linux и Windows с использованием библиотеки Qt версии 5.3. В данной книге также очень подробно рассматривается класс QThread, позволяющий работать с потоками. На официальном сайте [4] содержится исчерпывающая информация о процессах и потоках в Qt и конкретно о классе QThread.

Для эффективного распараллеливания приложения необходимо выбрать наиболее подходящий метод распараллеливания. Для выбора оптимального метода распараллеливания следует описать приложение

с точки зрения двух моделей: модели параллельного выполнения задач и модели параллельного использования данных.

Приложения с параллельным выполнением задач. В приложениях с параллельным выполнением задач независимые операции, заключенные в функции, переносятся в выполняемые асинхронно потоки, как показано на рис. 1. Для выражения параллелизма на уровне задач предназначены библиотеки поддержки многопоточности, например, интерфейсы программирования многопоточных приложений Win32 и POSIX*. Администратор личных данных является хорошим примером приложения, реализующего параллелизм на уровне задач. Для выражения параллелизма на уровне задач независимые функции переносятся в потоки, как показано на примере фрагмента псевдокода.

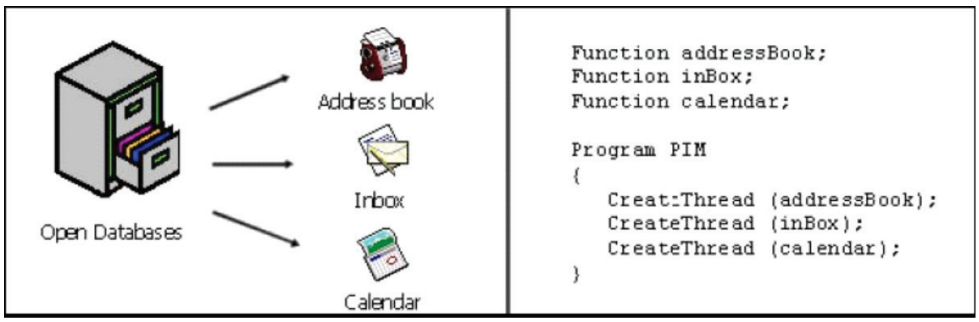

Рис. 1. Администратор личных данных

Приложения с параллельным использованием данных. Под параллелизмом на уровне данных подразумевается многократное применение одних и тех же команд или операций к различным данным. Эта модель показана на рис. 2. Хорошими кандидатами на применение методов параллельного использования данных являются циклы, в которых выполняются интенсивные вычисления.

Общим примером реализации параллелизма на уровне данных может служить типичный алгоритм обработки изображений, который применяет фильтр к одной или нескольким точкам изображения для вычисления нового значения для заданной точки. Поскольку операции, выполняемые над точками изображения, являются независимыми, вычисления новых значений для точек могут выполняться параллельно. Иногда параллелизм на уровне данных может выражаться компилятором автоматически. Можно также описать параллелизм с помощью синтаксиса директив, определенного стандартом OpenMP*. Функции преобразования директив в параллельный код выполняет компилятор. Хорошим примером параллельной работы с данными является программа проверки орфографии. Как видно из фрагмента псевдокода, в этом случае производится многократное выполнение одинаковых независимых операций сравнения слов из файла со словарем.

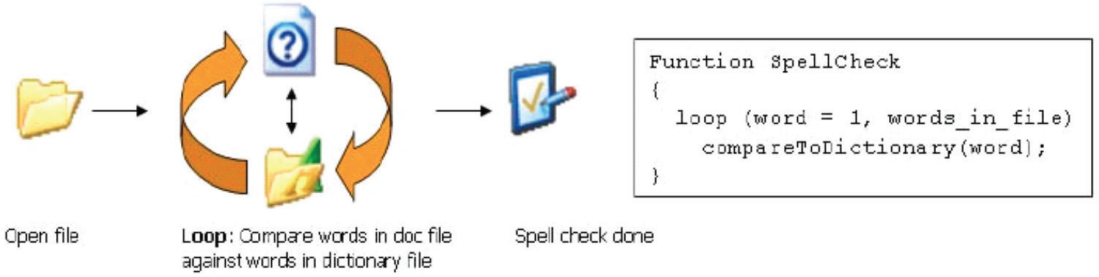

Рис. 2. Пример параллельной работы с данными

Отметим, что в различных частях одного и того же приложения могут применяться разные модели. В качестве примера приложения, использующего обе модели параллелизма, можно привести работу с базой данных. Задача добавления записей в базу данных может быть назначена одному потоку, сортировка данных – другому, индексирование – третьему, а отдельная группа потоков может осуществлять выполнение запросов. При выполнении запроса к различным данным применяются одинаковые операции, что делает такой запрос задачей параллельного использования данных.

Язык C++ является компилируемым строго типизированным языком программирования общего назначения, в котором наибольшее внимание уделено поддержке объектно-ориентированного программирования. C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения невероятно огромна, она включает в себя создание и операционных систем, и разнообразных прикладных программ, и драйверов устройств, и приложений для встраиваемых систем. Для реализации многопоточности в языке C++ также имеется обширный набор методов. Наиболее известными и часто используемыми являются:

• реализация с помощью класса thread появившегося в стандарте C++11;

• реализация с помощью pthreads в Unix системах;

• реализация с помощью класса QThread, входящего в состав фреймворка Qt.

Сутью многопоточности [5, 6, 7] является квазимногозадачность на уровне одного исполняемого процесса, то есть все потоки выполняются в адресном пространстве процесса. Выполняющийся процесс имеет как минимум один поток.

Реализация многопоточности с помощью класса thread. С выходом C++11 жить стало проще. Теперь для создания своего треда не надо использовать сложные API Майкрософт или вызывать устаревшую _beginthread. В новом стандарте появилась нативная поддержка работы с потоками. В частности, сейчас нас интересует класс std::thread, который является не чем иным, как STL представлением потоков. Конструктор std::thread принимает первым аргументом функцию исполнения, т.е. функцию, код которой будет исполнен в отдельном потоке. Остальные аргументы – аргументы исполняемой функции. Количество аргументов ограничено лишь реализацией variadic templates в вашем компиляторе. Важно помнить, что аргументы будут использованы в другом потоке исполнения, а следовательно нельзя передавать ссылки и указатели на объекты, время жизни которых не больше, чем время жизни потока. Также thread всегда копирует аргументы, и только потом передаёт их исполняемой функции. Функция начинает свое исполнения сразу по окончании работы конструктора std::thread. Завершение потока происходит по завершении работы исполняемой функции.

Реализация многопоточности с помощью pthreads. Необходимость написания многопоточных приложений возникает весьма и весьма часто. В наше время многоядерные процессоры проникли даже в настольные ПК и смартфоны, так что у большинства машин есть низкоуровневая аппаратная поддержка, позволяющая им одновременно выполнять несколько потоков. В былые времена одновременное исполнение потоков на одноядерных ЦПУ было лишь впечатляюще изобретательной, но очень эффективной иллюзией.

Перейдем к рассмотрению реализации многопоточности посредством pthread. В начале создается потоковая функция. Затем новый поток создается функцией pthread_create(), объявленной в заголовочном файле pthread.h. Далее, вызывающая сторона продолжает выполнять какие-то свои действия параллельно потоковой функции. При удачном завершении pthread_create() возвращает код 0, ненулевое значение сигнализирует об ошибке.

Функция pthread_join() ожидает завершения потока обозначенного THREAD_ID. Если этот поток к тому времени был уже завершен, то функция немедленно возвращает значение. Смысл функции в том, чтобы синхронизировать потоки. Важно понимать, что несмотря на то, что pthread_cancel() возвращается сразу и может завершить поток досрочно, ее нельзя назвать средством принудительного завершения потоков. Дело в том, что поток не только может самостоятельно выбрать момент завершения в ответ на вызов pthread_cancel(), но и вовсе его игнорировать. Вызов функции pthread_cancel() следует рассматривать как запрос на выполнение досрочного завершения потока. Поэтому, если для вас важно, чтобы поток был удален, нужно дождаться его завершения функцией pthread_join().

Любому потоку по умолчанию можно присоединиться вызовом pthread_join()

и ожидать его завершения. Однако в некоторых случаях статус завершения потока и возврат значения нам не интересны. Все, что нам надо, это завершить поток и автоматически выгрузить ресурсы обратно в распоряжение ОС. В таких случаях мы обозначаем поток отсоединившимся и используем вызов pthread_detach().

Реализация многопоточности с помощью класса QThread. QThread допускает различную реализацию многопоточности при разработке ПО. Одним из распространённых способов создания отдельных параллельных потоков в приложении на Qt и выполнения полезных действий в них является наследование от класса QThread и переопределение метода run(), в котором и будет выполняться полезный код приложения.

Данный метод является самым низкоуровневым и используется в первую очередь для кастомизации нативных потоков. Что несколько противоречит обычной необходимости выполнения задачи в отдельном потоке. То есть, как было сказано выше, подобный подход в первую очередь необходим для того, чтобы расширить функционал класса.

В заключение можно отметить, что многопоточность играет огромную роль в современном программировании. Без многопоточности невозможно представить большинство клиент-серверных приложений. Вследствие этого важным становится получение представления о многопоточности и умение разрабатывать программное обеспечение с поддержкой работы с несколькими потоками. При всем этом важно не только знать теорию, но и уметь применять эффективные подходы для решения конкретных задач так как существуют различные способы реализации многопоточности.

Список литературы

1. Страуструп Б. Язык программирования C++ / Бином, 2015. – 1136 с.

2. Уильямс Э. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ / ДМК Пресс, 2014. – 672 с.

3. Шлее М. Qt 5.3. Профессиональное программирование на C++ / БХВ-Петербург, 2015. – 928с.

4. Qt [Электронный ресурс] – Official Qt WebSite. – Режим доступа: https://doc.qt.io/

5. Стивенс У.Р. UNIX: разработка сетевых приложений / Питер, 2007. – 1039 с.

6. Стивенс У.Р. UNIX: взаимодействие процессов / Питер, 2003. – 576 с.

7. Бикташев Р.А., Мартышкин А.И. Комплекс программ для измерения производительности функций операционных систем // XXI век: итоги прошлого и проблемы настоящего плюс. – 2013. – № 10 (14). – С. 190–197.