УДК 004.042

ПРОБЛЕМЫ ПАРАЛЛЕЛИЗМА В ПРОГРАММИРОВАНИИ

Ю.В. ЛАПТЕВ
(Представлено: канд. физ.-мат. наук, доц. Ю.Ф. ПАСТУХОВ)

Рассматриваются некоторые принципиальные проблемы обеспечения параллельной работы программных приложений и способы их решения.

Параллельное выполнение операций – одна из наиболее сложных дисциплин в области разработки программного обеспечения. Проблемы параллелизма возникают каждый раз, когда несколько процессов или потоков вычислений предпринимают попытки манипуляций одними и теми же данными. Также параллельные операции трудно тестировать и невозможно перечислить все возможные сценарии развития событий, способным привести к тем или иным неприятным последствиям [1].

Проблемы параллелизма

Рассмотрим некоторые проблемы обеспечения параллельной работы приложений, на преодоление которых направлены усилия систем управления параллельными заданиями.

Рассмотрим *пример утраты изменений*. Предположим, что разработчик *А* открывает на редактирование файл Customer,с исходным кодом, и правит в нем какой-то метод. В это же время разработчик *Б* обращается к этому же файлу и правит тот же метод. Делает он это быстрее первого разработчика и сохраняет изменения. Разработчик *А* об этом ничего не знает, и как итог, сохраняя свои изменения, перезапишет все изменения второго разработчика. Данная проблема выражается в потере достоверности информации и в некорректном поведении системы, чего можно было бы избежать, если бы два человека не обращались к одним и тем же данным одновременно. Естественно можно упорядочить действия таким образом, чтобы только один субъект имел право обращаться к данным в определенный момент времени, но это снизит возможность параллельной обработки информации.

Также имеется одна из весьма важных и трудных проблем, связанная с взаимоблокировками. Предположим, что разработчик *А* приступает к редактированию файла Customer, а разработчик *В* редактирует файл Order. В какой-то момент времени разработчик *В* понимает, что для завершения работы ему необходимо несколько изменить и файл Customer, но разработчик *А* владеет блокировкой на этот файл и второму разработчику приходится ждать ее освобождения. В то же время первому разработчику приходит мысль, что нужно подправить файл Order, но этому мешает блокировка, удерживаемая разработчиком *В*. Оба разработчика попадают в ситуацию взаимоблокировок, и ни один не может завершить свою работу, пока другой разработчик не завершит свою. В данном случае проблема не выглядит угрожающе и ее несложно предотвратить, но если цепочку взаимозависимостей составляет множество людей или программ, тогда и начинаются серьезные проблемы.

Рассмотрим еще одну проблему с бесконечной отсрочкой, когда одна или несколько задач должны ожидать до тех пор, пока не произойдет какое-то событие или не создадутся определенные условия. Предположим, что если установитьдля электронного банка правила, предписывающие всем задачам снятия денег со счета находиться в состоянии ожидания до тех пор, пока не будут выполнены все задачи вложения денег на счет, то задачи снятия денег рискуют стать бесконечно отсроченными. Если будут без конца поступать запросы на пополнение одного и того же счета, то такая ситуация может вызвать бесконечную отсрочку задач снятия денег [2].

Транзакции

Транзакции являются основным инструментом управления параллельными процессами в приложениях. Транзакция представляет собой ограниченную последовательность действий с явно определенными начальной и завершающей операциями. Например, транзакция, связанная с банкоматом, начинается с размещения магнитной карты в считывающем устройстве и завершается выдачей денег или сообщением о несоответствии запрошенной суммы остатку на счете. Все ресурсы, затрагиваемые транзакцией, пребывают в согласованном состоянии в момент ее начала и остаются в таковом после ее завершения. Кроме этого, транзакция должна либо выполняться целиком, либо не выполняться вовсе. Банковская система не может вычесть сумму остатка на счете до тех пор, пока клиенту не будет выдана соответствующая пачка купюр. Транзакции часто описывают в терминах свойств, которые обозначаются аббревиатурой ACID:

- **atomicity** (атомарность). В контексте транзакции либо выполняются все действия, либо не выполняется ни одно из них. Частичное или избирательное выполнение недопустимо. Например, если клиент банка переводит сумму с одного счета на другой и в момент между завершением расходной и началом приходной операции сервер терпит крах, система должна вести себя так, будто расходной операции

не было вовсе. Система должна либо осуществить обе операции, либо не выполнить ни одной. Фиксация результатов служит свидетельством успешного окончания транзакции, а откат приводит систему в состояние, в котором она пребывала до начала транзакции;

- **consistency** (согласованность). Системные ресурсы должны пребывать в целостном и непротиворечивом состоянии как до начала транзакции, так и после ее окончания;

- **isolation** (изолированность). Промежуточные результаты транзакции должны быть закрыты для доступа со стороны любой другой действующей транзакции до момента их фиксации. Иными словами, транзакция протекает так, будто в тот же период времени других параллельных транзакций не существует;

- **durability** (устойчивость). Результат выполнения завершенной транзакции не должен быть утрачен ни при каких условиях.

В большинстве случаев под транзакциями в приложениях подразумеваются последовательности операций, описывающие процессы взаимодействия с базами данных. Но существует и множество других объектов, управляемых с помощью механизмов поддержки транзакций: например, очереди сообщений или заданий на печать, банкоматы и т.д. Для обеспечения высокого уровня пропускной способности современные системы управления транзакциями проектируются в расчете на максимально короткие транзакции. Обычно из практики исключаются транзакции, которые охватывают действия пообработке нескольких запросов. Если же подобной ситуации избежать не удается, решения реализуются на основе схемы длинных транзакций. Чаще границы транзакции совпадают с моментами начала и завершения обработки одного запроса. Подобная транзакция запроса – весьма удачная модель, и многие среды поддерживают простой и естественный синтаксис ее описания. Альтернативное решение состоит в том, чтобы как можно дольше откладывать процедуру открытия транзакции. При использовании подобной отсроченной транзакции все операции чтения могут выполняться до момента ее начала, который наступает только при необходимости осуществления операций, связанных с внесениемкаких бы то ни было изменений. Это позволяет минимизировать длину транзакции, но лишает возможности на протяжении продолжительных периодов времени применять какие-либо средства управления параллельными операциями. Такая стратегия сопряжена с повышением вероятности возникновения эффекта несогласованного чтения и потому используется сравнительно редко [3].

Применяя транзакции, следует понимать, что именно надлежит блокировать. Во многих случаях диспетчер транзакций СУБД блокирует отдельные записи таблицы базы данных, вовлеченные в операцию, что позволяет сохранить высокий уровень параллелизма при доступе к таблице. Но если транзакция пытается блокировать слишком большое количество записей таблицы, число запросов на блокировку превышает допустимый лимит и система распространяет действие блокировки на таблицу в целом, приостанавливаявыполнение конкурирующих транзакций.

Заключение

Параллелизм в программировании – достаточно обширная и сложная тема. Модель транзакций позволяет избежать массы трудностей, но это не значит, что проблемами управления параллельными заданиями можно полностью пренебречь, так как многие аспекты взаимодействия приложения и системы нельзя свести к контексту единой транзакции. Во многих случаях приходится иметь дело с данными, модифицируемыми несколькими транзакциями. К тому же существуют более сложные системы с поддержкой множества одновременно протекающих потоков вычислений, где приходится применять более сложные подходы к решению проблем параллелизма.

ЛИТЕРАТУРА

1. Параллелизм [Электронный источник]. – 2017. – Режим доступа: https://ru.wikipedia.org/wiki/Параллелизм_ (информатика). – Дата доступа: 27.09.2017.
2. Проблемы параллельного и распределенного программирования [Электронный источник]. – 2017. – Режим доступа: http://www.k2x2.info/kompyutery_i_internet/parallelnoe_i_ raspredelennoe_ programmirovanie_na_s/p4.php . – Дата доступа: 27.09.2017.
3. Фаулер, М. Архитектура корпоративных программных приложений : пер. с англ. / Мартин Фаулер. – М. : Издат. дом «Вильямс», 2006.