# Тезаурусный словарь

Веб-сервер — сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными. Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер (см.: Сервер (аппаратное обеспечение)), на котором это программное обеспечение работает.

Оперативная память (англ. Random Access Memory, RAM, память с произвольным доступом) или операти́вное запомина́ющее устро́йство (ОЗУ) — энергозависимая часть системы компьютерной памяти, в которой во время работы компьютера хранится выполняемый машинный код (программы), а также входные, выходные и промежуточные данные, обрабатываемые процессором. Обмен данными между процессором и оперативной памятью производится:

* непосредственно;
* через сверхбыструю память 0-го уровня — регистры в АЛУ, либо при наличии аппаратного кэша процессора — через кэш.

**Ментальная карта**

**Применение потоков**

Многопоточный веб-сервер

Текстовый процессор, использующий три потока

# Применение потоков

Зачем нам нужна какая-то разновидность процесса внутри самого процесса? Необходимость в подобных мини-процессах, называемых потоками, обусловливается целым рядом причин. Рассмотрим некоторые из них. Основная причина использования потоков заключается в том, что во многих приложениях одновременно происходит несколько действий, часть которых может периодически быть заблокированной. Модель программирования упрощается за счет разделения такого приложения на несколько последовательных потоков, выполняемых в квазипараллельном режиме.

Мы уже сталкивались с подобными аргументами. Именно они использовались в поддержку создания процессов. Вместо того чтобы думать о прерываниях, таймерах и контекстных переключателях, мы можем думать о параллельных процессах. Но только теперь, рассматривая потоки, мы добавляем новый элемент: возможность использования параллельными процессами единого адресного пространства и всех имеющихся данных. Эта возможность играет весьма важную роль для тех приложений, которым не подходит использование нескольких процессов (с их раздельными адресными пространствами).

Вторым аргументом в пользу потоков является легкость (то есть быстрота) их создания и ликвидации по сравнению с более «тяжеловесными» процессами. Во многих системах создание потоков осуществляется в 10-100 раз быстрее, чем создание процессов. Это свойство особенно пригодится, когда потребуется быстро и динамично изменять количество потоков.

Третий аргумент в пользу потоков также касается производительности. Когда потоки работают в рамках одного центрального процессора, они не приносят никакого прироста производительности, но когда выполняются значительные вычисления, а также значительная часть времени тратится на ожидание ввода-вывода, наличие потоков позволяет этим действиям перекрываться по времени, ускоряя работу приложения.

И наконец, потоки весьма полезны для систем, имеющих несколько центральных процессоров, где есть реальная возможность параллельных вычислений. К этому вопросу мы вернемся в главе 8.

Понять, в чем состоит польза от применения потоков, проще всего на конкретных примерах. Рассмотрим в качестве первого примера текстовый процессор. Обычно эти программы отображают создаваемый документ на экране в том виде, в каком он будет выводиться на печать. В частности, все концы строк и концы страниц находятся именно там, где они в результате и появятся на бумаге, чтобы пользователь мог при необходимости их проверить и подправить (например, убрать на странице начальные и конечные висячие строки, имеющие неэстетичный вид).

Предположим, что пользователь пишет какую-то книгу. С авторской точки зрения проще всего всю книгу иметь в одном файле, облегчая поиск тем, выполнение глобальных замен и т. д. С другой точки зрения каждая глава могла бы быть отдельным файлом. Но если каждый раздел и подраздел будут размещаться в отдельных файлах, это принесет массу неудобств, когда понадобится вносить во всю книгу глобальные изменения, поскольку тогда придется отдельно и поочередно редактировать сотни файлов. Например, если предложенный стандарт xxxx одобрен непосредственно перед выходом книги в печать, то в последнюю минуту все вхождения «Проект стандарта xxxx» нужно заменить на «Стандарт xxxx».

Если вся книга представлена одним файлом, то, как правило, все замены могут быть произведены с помощью одной команды. А если книга разбита на более чем 300 файлов, редактированию должен быть подвергнут каждый из них.

Теперь представим себе, что происходит, когда пользователь вдруг удаляет одно предложение на первой странице 800-страничного документа. Теперь, проверив внесенные изменения, он хочет внести еще одну поправку на 600-й странице и набирает команду, предписывающую текстовому процессору перейти на эту страницу (возможно, посредством поиска фразы, которая только там и встречается). Теперь текстовый процессор вынужден немедленно переформатировать всю книгу вплоть до 600-й страницы, поскольку он не знает, какой будет первая строка на 600-й странице, пока не обработает всех предыдущие страницы. Перед отображением 600-й страницы может произойти существенная задержка, вызывающая недовольство пользователя.

И здесь на помощь могут прийти потоки. Предположим, что текстовый процессор написан как двухпоточная программа. Один из потоков взаимодействует с пользователем, а другой занимается переформатированием в фоновом режиме. Как только предложение с первой страницы будет удалено, поток, отвечающий за взаимодействие с пользователем, приказывает потоку, отвечающему за формат, переформатировать всю книгу. Пока взаимодействующий поток продолжает отслеживать события клавиатуры и мыши, реагируя на простые команды вроде прокрутки первой страницы, второй поток с большой скоростью выполняет вычисления. Если немного повезет, то переформатирование закончится как раз перед тем, как пользователь запросит просмотр 600-й страницы, которая тут же сможет быть отображена.

Ну раз уж начали, то почему бы не добавить и третий поток? Многие текстовые процессоры обладают свойством автоматического сохранения всего файла на диск каждые несколько минут, чтобы уберечь пользователя от утраты его дневной работы в случае программных или системных сбоев или отключения электропитания. Третий поток может заниматься созданием резервных копий на диске, не мешая первым двум. Ситуация, связанная с применением трех потоков, показана на рис. 2.5.

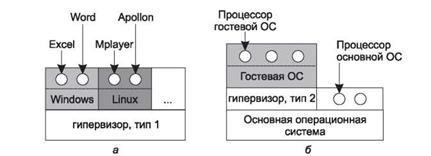


Рис. 2.5. Текстовый процессор, использующий три потока

Если бы программа была рассчитана на работу только одного потока, то с начала создания резервной копии на диске и до его завершения игнорировались бы команды

с клавиатуры или мыши. Пользователь ощущал бы это как слабую производительность. Можно было бы сделать так, чтобы события клавиатуры или мыши прерывали создание резервной копии на диске, позволяя достичь более высокой производительности, но это привело бы к сложной модели программирования, основанной на применении прерываний. Программная модель, использующая три потока, гораздо проще. Первый поток занят только взаимодействием с пользователем. Второй поток по необходимости занимается переформатированием документа. А третий поток периодически сбрасывает содержимое ОЗУ на диск.

Вполне очевидно, что три отдельных процесса так работать не будут, поскольку с документом необходимо работать всем трем потокам. Три потока вместо трех процессов используют общую память, таким образом, все они имеют доступ к редактируемому документу. При использовании трех процессов такое было бы невозможно.

Аналогичная ситуация складывается во многих других интерактивных программах. Например, электронная таблица является программой, позволяющей поддерживать матрицу, данные элементов которой предоставляются пользователем. Остальные элементы вычисляют исходя из введенных данных с использованием потенциально сложных формул. Когда пользователь изменяет значение одного элемента, нужно пересчитывать значения многих других элементов. При использовании потоков пересчета, работающих в фоновом режиме, поток, взаимодействующий с пользователем, может позволить последнему, пока идут вычисления, вносить дополнительные изменения. Подобным образом третий поток может сам по себе периодически сбрасывать на диск резервные копии.

Рассмотрим еще один пример, где могут пригодиться потоки: сервер для веб-сайта. Поступают запросы на веб-страницы, и запрошенные страницы отправляются обратно клиентам. На большинстве веб-сайтов некоторые страницы запрашиваются чаще других. Например, главная страница веб-сайта Sony запрашивается намного чаще, чем страница, находящаяся глубже, в ответвлении дерева, содержащем техническое описание какой-нибудь конкретной видеокамеры. Веб-службы используют это обстоятельство для повышения производительности за счет размещения содержания часто используемых страниц в основной памяти, чтобы исключить необходимость обращаться за ними к диску. Такие подборки называются кэшем и используются также во многих других случаях. Кэши центрального процессора уже рассматривались в главе 1.

Один из способов организации веб-сервера показан на рис. 2.6. Один из потоков — диспетчер — читает входящие рабочие запросы из сети. После анализа запроса он выбирает простаивающий (то есть заблокированный) рабочий поток и передает ему запрос, возможно, путем записи указателя на сообщение в специальное слово, связанное с каждым потоком. Затем диспетчер пробуждает спящий рабочий поток, переводя его из заблокированного состояния в состояние готовности.

При пробуждении рабочий поток проверяет, может ли запрос быть удовлетворен из кэша веб-страниц, к которому имеют доступ все потоки. Если нет, то он, чтобы получить веб-страницу, приступает к операции чтения с диска и блокируется до тех пор, пока не завершится дисковая операция. Когда поток блокируется на дисковой операции, выбирается выполнение другого потока, возможно, диспетчера, с целью получения следующей задачи или, возможно, другого рабочего потока, который находится в готовности к выполнению.

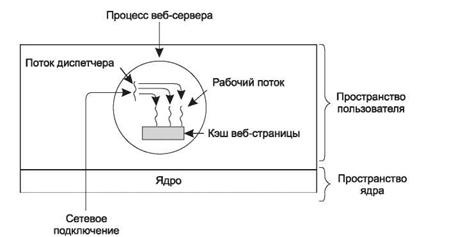


Рис. 2.6. Многопоточный веб-сервер

Эта модель позволяет запрограммировать сервер в виде коллекции последовательных потоков. Программа диспетчера состоит из бесконечного цикла для получения рабочего запроса и перепоручения его рабочему потоку. Код каждого рабочего потока состоит из бесконечного цикла, в котором принимается запрос от диспетчера и веб-кэш проверяется на присутствие в нем страницы. Если страница в кэше, она возвращается клиенту. Если нет, поток получает страницу с диска, возвращает ее клиенту и блокируется в ожидании нового запроса.

Приблизительный набросок кода показан на рис. 2.7. Здесь, как и во всей книге, константа TRUE предполагается равной 1. Также buf и page являются структурами, предназначенными для хранения рабочего запроса и веб-страницы соответственно.

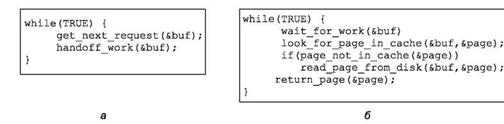


Рис. 2.7. Приблизительный набросок кода для модели, изображенной на рис. 2.6: а — для потока-диспетчера; б — для рабочего потока

Рассмотрим, как можно было бы написать код веб-сервера в отсутствие потоков. Можно заставить его работать в виде единого потока. Основной цикл веб-сервера получает запрос, анализирует его и завершает обработку до получения следующего запроса. Ожидая завершения дисковой операции, сервер простаивает и не обрабатывает никаких других входящих запросов. Если веб-сервер запущен на специально выделенной машине, что чаще всего и бывает, то центральный процессор, ожидая завершения дисковой операции, остается без дела. В итоге происходит значительное сокращение количества запросов, обрабатываемых в секунду. Таким образом, потоки существенно повышают производительность, но каждый из них программируется последовательно, то есть обычным способом.

До сих пор мы видели две возможные конструкции: многопоточный и однопоточный веб-серверы. Представьте, что потоки недоступны, а системные программисты считают, что потери производительности при использовании одного потока недопустимы. Если доступна неблокирующая версия системного вызова read, то возможен и третий подход. При поступлении запроса его анализирует один-единственный поток. Если запрос может быть удовлетворен из кэша, то все в порядке, но если нет, то стартует неблокирующая дисковая операция.

Сервер записывает состояние текущего запроса в таблицу, а затем приступает к получению следующего события. Этим событием может быть либо запрос на новую задачу, либо ответ от диска, связанный с предыдущей операцией. Если это новая задача, то процесс приступает к ее выполнению. Если это ответ от диска, то из таблицы выбирается соответствующая информация и происходит обработка ответа. При использовании неблокирующего ввода-вывода ответ, наверное, должен принять форму сигнала или прерывания.

При такой конструкции модель «последовательного процесса», присутствующая в первых двух случаях, уже не работает. Состояние вычисления должно быть явным образом сохранено и восстановлено из таблицы при каждом переключении сервера с обработки одного запроса на обработку другого. В результате потоки и их стеки имитируются более сложным образом. Подобная конструкция, в которой у каждого вычисления есть сохраняемое состояние и имеется некоторый набор событий, которые могут происходить с целью изменения состояния, называются машиной с конечным числом состояний (finite-state machine), или конечным автоматом. Это понятие получило в вычислительной технике весьма широкое распространение.

Теперь, наверное, уже понятно, чем должны быть полезны потоки. Они дают возможность сохранить идею последовательных процессов, которые осуществляют блокирующие системные вызовы (например, для операций дискового ввода-вывода), но при этом позволяют все же добиться распараллеливания работы. Блокирующие системные вызовы упрощают программирование, а параллельная работа повышает производительность. Однопоточные серверы сохраняют простоту блокирующих системных вызовов, но уступают им в производительности. Третий подход позволяет добиться высокой производительности за счет параллельной работы, но использует неблокирующие вызовы и прерывания, усложняя процесс программирования. Сводка моделей приведена в табл. 2.3.