## Управление распределенными ресурсами

### Базовые примитивы передачи сообщений в распределенных системах

Единственным по-настоящему важным отличием распределенных систем от централизованных является межпроцессная взаимосвязь. В централизованных системах связь между процессами, как правило, предполагает наличие разделяемой памяти. Типичный пример - проблема "поставщик-потребитель", в этом случае один процесс пишет в разделяемый буфер, а другой - читает из него. Даже наиболее простая форма синхронизации - семафор - требует, чтобы хотя бы одно слово (переменная самого семафора) было разделяемым. В распределенных системах нет какой бы то ни было разделяемой памяти, таким образом вся природа межпроцессных коммуникаций должна быть продумана заново.

Основой этого взаимодействия может служить только передача по сети сообщений. В самом простом случае системные средства обеспечения связи могут быть сведены к двум основным системным вызовам (примитивам), один - для посылки сообщения, другой - для получения сообщения. В дальнейшем на их базе могут быть построены более мощные средства сетевых коммуникаций, такие как распределенная файловая система или вызов удаленных процедур, которые, в свою очередь, также могут служить основой для построения других сетевых сервисов.

Несмотря на концептуальную простоту этих системных вызовов - ПОСЛАТЬ и ПОЛУЧИТЬ - существуют различные варианты их реализации, от правильного выбора которых зависит эффективность работы сети. В частности, эффективность коммуникаций в сети зависит от способа задания адреса, от того, является ли системный вызов блокирующим или неблокирующим, какие выбраны способы буферизации сообщений и насколько надежным является протокол обмена сообщениями.

#### Способы адресации

Для того, чтобы послать сообщение, необходимо указать адрес получателя. В очень простой сети адрес может задаваться в виде константы, но в более сложных сетях нужен и более изощренный способ адресации.

Одним из вариантов адресации на верхнем уровне является использование физических адресов сетевых адаптеров. Если в получающем компьютере выполняется только один процесс, то ядро будет знать, что делать с поступившим сообщением - передать его этому процессу. Однако, если на машине выполняется несколько процессов, то ядру не известно, какому из них предназначено сообщение, поэтому использование сетевого адреса адаптера в качестве адреса получателя приводит к очень серьезному ограничению - на каждой машине должен выполняться только один процесс.

Альтернативная адресная система использует имена назначения, состоящие из двух частей, определяющие номер машины и номер процесса. Однако адресация типа "машина-процесс" далека от идеала, в частности, она не гибка и не прозрачна, так как пользователь должен явно задавать адрес машины-получателя. В этом случае, если в один прекрасный день машина, на которой работает сервер, отказывает, то программа, в которой жестко используется адрес сервера, не сможет работать с другим сервером, установленном на другой машине.

Другим вариантом могло бы быть назначение каждому процессу уникального адреса, который никак не связан с адресом машины. Одним из способов достижения этой цели является использование централизованного механизма распределения адресов процессов, который работает просто, как счетчик. При получении запроса на выделение адреса он просто возвращает текущее значение счетчика, а затем наращивает его на единицу. Недостатком этой схемы является то, что централизованные компоненты, подобные этому, не обеспечивают в достаточной степени расширяемость систем. Еще один метод назначения процессам уникальных идентификаторов заключается в разрешении каждому процессу выбора своего собственного идентификатора из очень большого адресного пространства, такого как пространство 64-х битных целых чисел. Вероятность выбора одного и того же числа двумя процессами является ничтожной, а система хорошо расширяется. Однако здесь имеется одна проблема: как процесс-отправитель может узнать номер машины процесса-получателя. В сети, которая поддерживает широковещательный режим (то есть в ней предусмотрен такой адрес, который принимают все сетевые адаптеры), отправитель может широковещательно передать специальный пакет, который содержит идентификатор процесса назначения. Все ядра получат эти сообщения, проверят адрес процесса и, если он совпадает с идентификатором одного из процессов этой машины, пошлют ответное сообщение "Я здесь", содержащее сетевой адрес машины.

Хотя эта схема и прозрачна, но широковещательные сообщения перегружают систему. Такой перегрузки можно избежать, выделив в сети специальную машину для отображения высокоуровневых символьных имен. При применении такой системы процессы адресуются с помощью символьных строк, и в программы вставляются эти строки, а не номера машин или процессов. Каждый раз перед первой попыткой связаться, процесс должен послать запрос специальному отображающему процессу, обычно называемому сервером имен, запрашивая номер машины, на которой работает процесс-получатель.

Совершенно иной подход - это использование специальной аппаратуры. Пусть процессы выбирают свои адреса случайно, а конструкция сетевых адаптеров позволяет хранить эти адреса. Теперь адреса процессов не обнаруживаются путем широковещательной передачи, а непосредственно указываются в кадрах, заменяя там адреса сетевых адаптеров.

#### Блокирующие и неблокирующие примитивы

Примитивы бывают блокирующими и неблокирующими, иногда они называются соответственно синхронными и асинхронными. При использовании блокирующего примитива, процесс, выдавший запрос на его выполнение, приостанавливается до полного завершения примитива. Например, вызов примитива ПОЛУЧИТЬ приостанавливает вызывающий процесс до получения сообщения.

При использовании неблокирующего примитива управление возвращается вызывающему процессу немедленно, еще до того, как требуемая работа будет выполнена. Преимуществом этой схемы является параллельное выполнение вызывающего процесса и процесса передачи сообщения. Обычно в ОС имеется один из двух видов примитивов и очень редко - оба. Однако выигрыш в производительности при использовании неблокирующих примитивов компенсируется серьезным недостатком: отправитель не может модифицировать буфер сообщения, пока сообщение не отправлено, а узнать, отправлено ли сообщение, отправитель не может. Отсюда сложности в построении программ, которые передают последовательность сообщений с помощью неблокирующих примитивов.

Имеется два возможных выхода. Первое решение - это заставить ядро копировать сообщение в свой внутренний буфер, а затем разрешить процессу продолжить выполнение. С точки зрения процесса эта схема ничем не отличается от схемы блокирующего вызова: как только процесс снова получает управление, он может повторно использовать буфер.

Второе решение заключается в прерывании процесса-отправителя после отправки сообщения, чтобы проинформировать его, что буфер снова доступен. Здесь не требуется копирование, что экономит время, но прерывание пользовательского уровня делает программирование запутанным, сложным, может привести к возникновению гонок.

Вопросом, тесно связанным с блокирующими и неблокирующими вызовами, является вопрос тайм-аутов. В системе с блокирующим вызовом ПОСЛАТЬ при отсутствии ответа вызывающий процесс может заблокироваться навсегда. Для предотвращения такой ситуации в некоторых системах вызывающий процесс может задать временной интервал, в течение которого он ждет ответ. Если за это время сообщение не поступает, вызов ПОСЛАТЬ завершается с кодом ошибки.

#### Буферизуемые и небуферизуемые примитивы

Примитивы, которые были описаны, являются небуферизуемыми примитивами. Это означает, что вызов ПОЛУЧИТЬ сообщает ядру машины, на которой он выполняется, адрес буфера, в который следует поместить пребывающее для него сообщение.

Эта схема работает прекрасно при условии, что получатель выполняет вызов ПОЛУЧИТЬ раньше, чем отправитель выполняет вызов ПОСЛАТЬ. Вызов ПОЛУЧИТЬ сообщает ядру машины, на которой выполняется, по какому адресу должно поступить ожидаемое сообщение, и в какую область памяти необходимо его поместить. Проблема возникает тогда, когда вызов ПОСЛАТЬ сделан раньше вызова ПОЛУЧИТЬ. Каким образом сможет узнать ядро на машине получателя, какому процессу адресовано вновь поступившее сообщение, если их несколько? И как оно узнает, куда его скопировать?

Один из вариантов - просто отказаться от сообщения, позволить отправителю взять тайм-аут и надеяться, что получатель все-таки выполнит вызов ПОЛУЧИТЬ перед повторной передачей сообщения. Этот подход не сложен в реализации, но, к сожалению, отправитель (или скорее ядро его машины) может сделать несколько таких безуспешных попыток. Еще хуже то, что после достаточно большого числа безуспешных попыток ядро отправителя может сделать неправильный вывод об аварии на машине получателя или о неправильности использованного адреса.

Второй подход к этой проблеме заключается в том, чтобы хранить хотя бы некоторое время, поступающие сообщения в ядре получателя на тот случай, что вскоре будет выполнен соответствующий вызов ПОЛУЧИТЬ. Каждый раз, когда поступает такое "неожидаемое" сообщение, включается таймер. Если заданный временной интервал истекает раньше, чем происходит соответствующий вызов ПОЛУЧИТЬ, то сообщение теряется.

Хотя этот метод и уменьшает вероятность потери сообщений, он порождает проблему хранения и управления преждевременно поступившими сообщениями. Необходимы буферы, которые следует где-то размещать, освобождать, в общем, которыми нужно управлять. Концептуально простым способом управления буферами является определение новой структуры данных, называемой почтовым ящиком.

Процесс, который заинтересован в получении сообщений, обращается к ядру с запросом о создании для него почтового ящика и сообщает адрес, по которому ему могут поступать сетевые пакеты, после чего все сообщения с данным адресом будут помещены в его почтовый ящик. Такой способ часто называют буферизуемым примитивом.

#### Надежные и ненадежные примитивы

Ранее подразумевалось, что когда отправитель посылает сообщение, адресат его обязательно получает. Но реально сообщения могут теряться. Предположим, что используются блокирующие примитивы. Когда отправитель посылает сообщение, то он приостанавливает свою работу до тех пор, пока сообщение не будет послано. Однако нет никаких гарантий, что после того, как он возобновит свою работу, сообщение будет доставлено адресату.

Для решения этой проблемы существует три подхода. Первый заключается в том, что система не берет на себя никаких обязательств по поводу доставки сообщений. Реализация надежного взаимодействия становится целиком заботой пользователя.

Второй подход заключается в том, что ядро принимающей машины посылает квитанцию-подтверждение ядру отправляющей машины на каждое сообщение. Посылающее ядро разблокирует пользовательский процесс только после получения этого подтверждения. Подтверждение передается от ядра к ядру. Ни отправитель, ни получатель его не видят.

Третий подход заключается в использовании ответа в качестве подтверждения в тех системах, в которых запрос всегда сопровождается ответом. Отправитель остается заблокированным до получения ответа. Если ответа нет слишком долго, то посылающее ядро может переслать запрос специальной службе предотвращения потери сообщений.