РЕАЛИЗАЦИЯ КЛИЕНТ-СЕРВЕРНОЙ АРХИТЕКТУРЫ В СИСТЕМЕ УДАЛЕННОГО УПРАВЛЕНИЯ ГРУППОЙ РАБОЧИХ СТАНЦИЙ

Е.И. Максимова
Томский политехнический университет
yelenamaksimova@yandex.ru

Введение

На сегодняшний день для решения различного рода прикладных задач нередко используется клиент-серверная архитектура. Такая архитектура применяется в распределенных базах данных, системах контроля версий, системах удаленного управления и игровых многопользовательских приложениях. Одним из преимуществ подобной организации приложения является независимость скорости выполнения операции на серверной стороне от вычислительных мощностей рабочей станции-клиента. Фактически, все хоть сколько-нибудь требовательные к количеству вычислительных ресурсов операции можно возложить на серверную сторону, избавив от них клиентские рабочие станции.

Клиент-серверная архитектура представляет собой сетевое окружение, в котором управление данными осуществляется на серверном узле – сервере, а другим узлам – клиентам предоставляется доступ к данным. В большинстве случаев клиент и сервер взаимодействуют через компьютерную сеть посредством сетевых протоколов и находятся на разных вычислительных машинах. Основным принципом клиент-серверной архитектуры является возможность разделения функций приложения на функции управления и представления [1].

Как правило, при проектировании клиент-серверного приложения учитывается потенциальная нагрузка на сервер, которая, в свою очередь, зависит от максимального возможного количества клиентов, одновременно осуществляющих взаимодействие с серверным приложением. При разработке систем рассчитанных на большое количество пользователей, приоритет следует отдавать наиболее высокому быстродействию взаимодействия серверного и клиентского приложений. Если же максимальное количество одновременных подключений не столь велико, и существует возможность заранее определить, на какое наибольшее количество клиентов ориентирована система, то первоочередное внимание требуется уделить простоте организации обмена сообщениями между клиентским и серверным приложениями с точки зрения разработки и отладки.

Подобная технология хорошо подходит для организации системы дистанционного управления группой рабочих станций. В таком случае система будет представлять собой совокупность двух приложений: клиентского и серверного. При этом сервер осуществляет мониторинг состояния подключения с клиентом, а также организует диалог между оператором и клиентом. В то время как клиент поддерживает постоянную связь с сервером и выполняет локальное администрирование рабочей станции в соответствие с командами, полученными с сервера. Стоит отметить, что приложение-клиент может быть реализовано в виде службы.

При реализации системы дистанционного управления группой рабочих станций следует обратить внимание на ряд особенностей, которыми должны обладать клиентское и серверное приложения:

- Клиент должен поддерживать постоянную связь с сервером;
- В случае если соединение с сервером в какой-то момент невозможно, клиент должен повторно осуществить попытку подключения через некоторый временной промежуток;
- Сервер должен контролировать состояние подключения множества клиентов и обеспечивать параллельную обработку сообщений от клиентов;
- Сервер должен параллельно отправлять сообщения целой группе клиентов;
- Нештатное поведение одного из клиентов не должно нарушать взаимодействия сервера с оставшимися клиентами.

Для того чтобы учесть перечисленные особенности была предложена следующая модель организации клиент-серверного взаимодействия. Первичное соединение клиента и сервера осуществляется через некоторый мета-порт (порт, заданный по умолчанию на клиенте и сервере). После того как новый клиент подключился к серверу, функция распределения портов на сервере определяет свободный порт, через который будет производиться дальнейшее взаимодействие (рис. 1). Затем на сервере открывается новый поток, использующийся для передачи номера выбранного порта клиенту и дальнейшей работы с ним. После приема первого сообщения, клиент будет продолжать взаимодействие с сервером через полученный порт.

Еще одним преимуществом предварительного использования мета-порта является возможность реализации списка приоритетов портов для отдельных типов клиентов. Так, например, в системе управления группой рабочих станций при соединении через мета-порт клиент может сообщить, под управлением какого пользователя он работает. Это, в свою очередь, может повлиять на выбор порта для взаимодействия с этим клиентом,

в случае если для некоторых типов пользователей ограничен трафик через определенные порты.

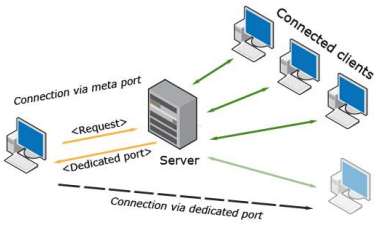

Рис.1. Модель организации клиент-серверного взаимодействия

Для реализации предложенной модели используются классы TcpListener и TcpClient платформы .NET. Класс TcpListener предоставляет разработчику простые методы для ожидания и приема в синхронном режиме запросов на подключение с использованием протокола TCP, в то время как класс TcpClient обеспечивает методы для подключения, а также отправки и получения потоков данных в сети в синхронном режиме [2].

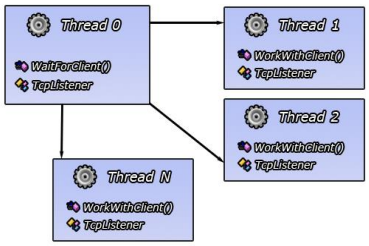

Рис.2. Схема действия многопоточного клиент-серверного приложения

В основном потоке серверного приложения создается объект класса TcpListener, который ожидает подключения к мета-порту. После того как к этому порту подключается клиент, сервер сообщает ему номер порта, через который будет осуществляться дальнейшее взаимодействие между этим клиентом и сервером. Далее для нового клиента создается отдельный поток, в котором так же создается объект класса TcpListener для диалога с этим клиентом (рис. 2). Чтобы определить номер порта, который нужно передать, в некотором заданном диапазоне портов выбирается свободный с наименьшим номером.

Клиентское приложение организовано подобно серверному приложению. При запуске клиента создается объект класса TcpClient, затем через мета-порт принимается сообщение от сервера с номером нового порта. Новое подключение к серверу устанавливается через порт, с принятым номером. Таким образом, взаимодействие клиента и сервера всегда осуществляется с использованием отдельного выделенного порта в новом потоке.

Еще одним преимуществом данной архитектуры является то, что обработку всех входящих сообщений от клиентских приложений можно осуществлять централизованно в основном мета-потоке. Для этого можно воспользоваться общей очередью сообщений, к которой, с использованием средств межпоточного взаимодействия, будут обращаться как потоки работы с клиентами, так и основной мета-поток. Такой подход несколько снизит быстродействие клиент-серверного взаимодействия, но позволит увеличить простоту реализации системы [3] и, что еще более важно, существенно упростит процесс дополнения системы новыми функциональными возможностями.

Описанная выше архитектура позволяет осуществлять непрерывный обмен сообщениями между сервером и клиентом, а так же их параллельную обработку. Стоит отметить, что организация приложения подобным образом позволяет избежать ошибки, возникающей при прослушивании одного и того же порта, ввиду того, что при их распределении выделяются лишь свободные порты из выделенного диапазона. Характеристики данного клиент-серверного приложения позволяют использовать его как инструмент для сетевого взаимодействия в системе дистанционного управления группой рабочих станций.

При реализации системы управления группой рабочих станций также были изучены и использованы основные инструменты для организации взаимодействия отдельных потоков и мета потока. Для того чтобы осуществлять обработку входящих сообщений от клиентов в общем мета-потоке была реализована общая очередь сообщений, одновременное обращение двух или более потоков к которой было исключено использованием мьютексов.

Таким образом, предложенная архитектура клиент-серверного приложения не только позволила удобно распределить обязанности взаимодействия с отдельными клиентами между соответствующими им потоками, но и централизовать обработку входящих сообщений в мета-потоке с использованием разделяемых ресурсов.

Литература

1. Korzhov V. Multi-level systems client-server – «Open systems», 1997. – 217 p;

2. System.Net.Sockets Namespace [Electronic resourse] / URL: http://msdn.microsoft.com/ru-ru/library/System.Net.Sockets(v=vs.110).aspx. Date circulation: 04/12/2014.

3. Delling, D. and Sanders, P. and Schultes, D. and Wagner, D. "Engineering route planning algorithms". Algorithmics of large and complex networks - «Springer», 2009. – 376 p.