**ЛАБОРАТОРНАЯ РАБОТА № 15**

Приложения архитектуры «клиент-сервер»

Компоненты сетевого приложения. Клиент-серверное взаимодействие и роли серверов.

Как правило компьютеры и программы, входящие в состав информационной системы, не являются равноправными. Некоторые из них владеют ресурсами (файловая система, процессор, принтер, база данных и т.д.), другие имеют возможность обращаться к этим ресурсам. Компьютер (или программу), управляющий ресурсом, называют сервером этого ресурса (файл-сервер, сервер базы данных, вычислительный сервер...). Клиент и сервер какого-либо ресурса могут находится как на одном компьютере, так и на различных компьютерах, связанных сетью.

В рамках многоуровневого представления вычислительных систем можно выделить три группы функций, ориентированных на решение различных подзадач:

1. функции ввода и отображения данных (обеспечивают взаимодействие с пользователем);
2. прикладные функции, характерные для данной предметной области;
3. функции управления ресурсами (файловой системой, базой даных и т.д.)



*Рис.1. Компоненты сетевого приложения*

Выполнение этих функций в основном обеспечивается программными средствами, которые можно представить в виде взаимосвязанных компонентов ([рис. 1](http://www.4stud.info/networking/lecture5.html#application-layers)), где:

* *компонент представления* отвечает за пользовательский интерфейс;
* *прикладной компонент* реализует алгоритм решения конкретной задачи;
* *компонент управления* ресурсом обеспечивает доступ к необходимым ресурсам.

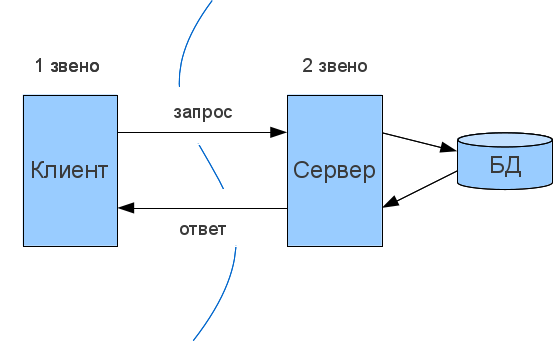
Автономная система (компьютер, не подключенный к сети) представляет все эти компоненты как на различных уровнях (ОС, служебное ПО и утилиты, прикладное ПО), так и на уровне приложений (не характерно для современных программ). Так же и сеть — она представляет все эти компоненты, но, в общем случае, распределенные между узлами. Задача сводится к обеспечению сетевого взаимодействия между этими компонентами.

Архитектура «клиент-сервер» определяет общие принципы организации взаимодействия в сети, где имеются *серверы*, узлы-поставщики некоторых специфичных функций (сервисов) и *клиенты*, потребители этих функций.

Практические реализации такой архитектуры называются клиент-серверными технологиями. Каждая технология определяет собственные или использует имеющиеся правила взаимодейстия между клиентом и сервером, которые называются *протоколом обмена (протоколом взаимодействия)*.

## Двухзвенная архитектура

В любой сети (даже одноранговой), построенной на современных сетевых технологиях, присутствуют элементы клиент-серверного взаимодействия, чаще всего на основе **двухзвенной архитектуры**. Двухзвенной (two-tier, 2-tier) она называется из-за необходимости распределения *трех* базовых *компонентов* между *двумя узлами* (клиентом и сервером).



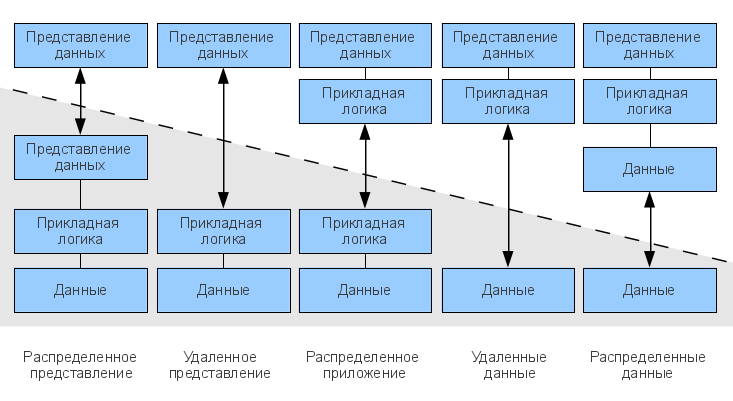
*Рис.2. Двухзвенная клиент-серверная архитектура*

Двухзвенная архитектура используется в клиент-серверных системах, где сервер отвечает на клиентские запросы напрямую и в полном объеме, при этом используя только собственные ресурсы. Т.е. сервер не вызывает сторонние сетевые приложения и не обращается к сторонним ресурсам для выполнения какой-либо части запроса ([рис. 2](http://www.4stud.info/networking/lecture5.html#2-tier))

Расположение компонентов на стороне клиента или сервера определяет следующие основные модели их взаимодействия в рамках двухзвенной архитектуры:

* сервер терминалов — распределенное представление данных;
* файл-сервер — доступ к удаленной базе данных и файловым ресурсам;
* сервер БД — удаленное представление данных;
* сервер приложений — удаленное приложение.

Перечисленные модели с вариациями представлены на [рис. 3](http://www.4stud.info/networking/lecture5.html#client-servers-models).



*Рис.3. Модели клиент-серверного взаимодействия*

Исторически первой появилась модель распределенного представления данных (модель сервер терминалов). Она реализовывалась на универсальной ЭВМ (мэйнфрейме), выступавшей в роли сервера, с подключенными к ней алфавитно-цифровыми терминалами. Пользователи выполняли ввод данных с клавиатуры терминала, которые затем передавались на мэйнфрейм и там выполнялась их обработка, включая формирование «картинки» с результатами. Эта «картинка» и возвращалась пользователю на экран терминала.

С появлением персональных компьютеров и локальных сетей, была реализована модель файлового сервера, представлявшего доступ файловым ресурсам, в т.ч и к удаленной базе данных. В этом случае выделенный узел сети является файловым сервером, на котором размещены файлы базы данных. На клиентах выполняются приложения, в которых совмещены компонент представления и прикладной компонент (СУБД и прикладная программма), использующие подключенную удаленную базу как локальный файл. Протоколы обмена при этом представляют набор низкоуровневых вызовов операций файловой системы.

Такая модель показала свою неэффективность ввиду того, что при активной работе с таблицами БД возникает большая нагрузка на сеть. Частичным решением является поддержка тиражирования (репликации) таблиц и запросов. В этом случае, например при изменении данных, обновляется не вся таблица, а только модифицированная ее часть.

С появлением специализированных СУБД появилась возможность реализации другой модели доступа к удаленной базе данных — модели сервера баз данных. В этом случае ядро СУБД функционирует на сервере, прикладная программа на клиенте, а протокол обмена обеспечивается с помощью языка SQL. Такой подход по сравнению с файловым сервером ведет к уменьшению загрузки сети и унификации интерфейса «клиент-сервер». Однако, сетевой трафик остается достаточно высоким, кроме того, по прежнему невозможно удовлетворительное администрирование приложений, поскольку в одной программе совмещаются различные функции.

С разработкой и внедрением на уровне серверов баз данных механизма хранимых процедур появилась концепция *активного сервера БД*. В этом случае часть функций прикладного компонента реализованы в виде хранимых процедур, выполняемых на стороне сервера. Остальная прикладная логика выполняется на клиентской стороне. Протокол взаимодействия — соответствующий диалект языка SQL.

Преимущества такого подхода очевидны:

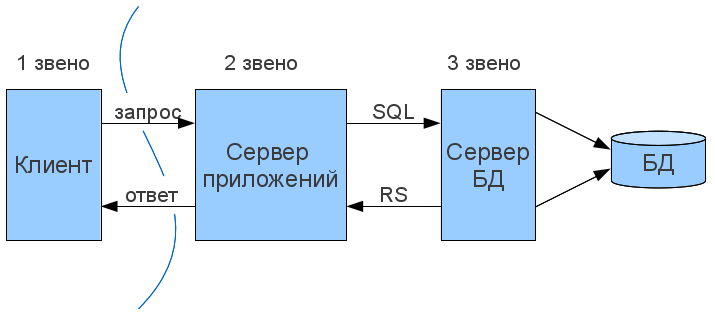
* возможно централизованное администрирование прикладных функций;
* снижение стоимости владения системой (TOC, total cost of ownership) за счет [аренды сервера](http://www.di-net.ru/collocation/dedicated/), а не его покупки;
* значительное снижение сетевого трафика (т.к. передаются не SQL-запросы, а вызовы хранимых процедур).

Основной недостаток — ограниченность средств разработки хранимых процедур по сравнению с языками высокого уровня.

Реализация прикладного компонента на стороне сервера представляет следующую модель — сервер приложений. Перенос функций прикладного компонента на сервер снижает требования к конфигурации клиентов и упрощает администрирование, но представляет повышенные требования к производительности, безопасности и надежности сервера.

В настоящее время намечается тенденция возврата к тому, с чего начиналась клиент-серверная архитектура — к централизации вычислений на основе модели терминал-сервера. В современной реинкарнации терминалы отличаются от своих алфавитно-цифровых предков тем, что имея минимум программных и аппаратных средств, представляют мультимедийные возможности (в т.ч. *графический пользовательский интерфейс*). Работу терминалов обеспечивает высокопроизводительный сервер, куда вынесено все, вплоть до виртуальных драйверов устройств, включая драйверы видеоподсистемы.

## Трехзвенная архитектура

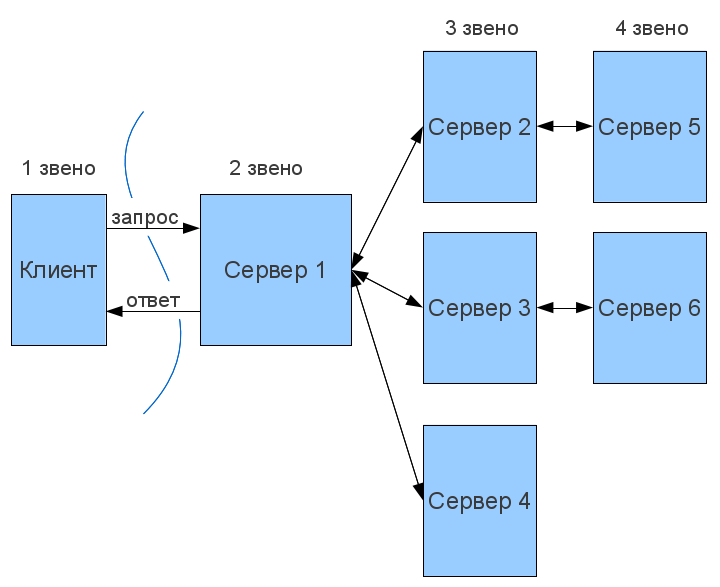


*Рис.4. Трехзвенная клиент-серверная архитектура*

Еще одна тенденция в клиент-серверных технологиях связана со все большим использованием распределенных вычислений. Они реализуются на основе модели сервера приложений, где сетевое приложение разделено на две и более частей, каждая из которых может выполняться на отдельном компьютере. Выделенные части приложения взаимодействуют друг с другом, обмениваясь сообщениями в заранее согласованном формате. В этом случае двухзвенная клиент-серверная архитектура становится трехзвенной (three-tier, 3-tier).

Как правило, третьим звеном в трехзвенной архитектуре становится сервер приложений, т.е. компоненты распределяются следующим образом ([рис. 4](http://www.4stud.info/networking/lecture5.html#three-tiers-model)):

1. Представление данных — на стороне клиента.
2. Прикладной компонент — на выделенном сервере приложений (как вариант, выполняющем функции [промежуточного ПО](http://www.4stud.info/networking/lecture6.html)).
3. Управление ресурсами — на сервере БД, который и представляет запрашиваемые данные.



*Рис.5. Многозвенная (N-tier) клиент-серверная архитектура*

Трехзвенная архитектура может быть расширена до многозвенной (N-tier, Multi-tier) путем выделения дополнительных серверов, каждый из которых будет представлять собственные сервисы и пользоваться услугами прочих серверов разного уровня. Абстрактный пример многозвенной модели приведен на [рис. 5](http://www.4stud.info/networking/lecture5.html#n-tiers-model).

## Сравнение архитектур

Двухзвенная архитектура проще, так как все запросы обслуживаются одним сервером, но именно из-за этого она менее надежна и предъявляет повышенные требования к производительности сервера.

Трехзвенная архитектура сложнее, но благодаря тому, что функции распределены между серверами второго и третьего уровня, эта архитектура представляет:

1. Высокую степень гибкости и масштабируемости.
2. Высокую безопасность (т.к. защиту можно определить для каждого сервиса или уровня).
3. Высокую производительность (т.к. задачи распределены между серверами).

## Клиент-серверные технологии

Архитектура клиент-сервер применяется в большом числе сетевых технологий, используемых для доступа к различным сетевым сервисам. Кратко рассмотрим некоторые типы таких сервисов (и серверов).

**Web-серверы**

Изначально представляли доступ к гипертекстовым документам по протоколу HTTP (Huper Text Transfer Protocol). Сейчас поддерживают расширенные возможности, в частности работу с бинарными файлами (изображения, мультимедиа и т.п.).

**Серверы приложений**

Предназначены для централизованного решения прикладных задач в некоторой предметной области. Для этого пользователи имеют право *запускать серверные программы* на исполнение. Использование серверов приложений позволяет снизить требования к конфигурации клиентов и упрощает общее управление сетью.

**Серверы баз данных**

Серверы баз данных используются для обработки пользовательских запросов на языке SQL. При этом СУБД находится на сервере, к которому и подключаются клиентские приложения.

**Файл-серверы**

Файл-сервер *хранит* информацию в виде файлов и представляет пользователям доступ к ней. Как правило файл-сервер обеспечивает и определенный уровень защиты от несакционированного доступа.

**Прокси-сервер**

Во-первых, действует как посредник, помогая пользователям получить информацию из Интернета и при этом обеспечивая защиту сети.

Во-вторых, сохраняет часто запрашиваемую информацию в кэш-памяти на локальном диске, быстро доставляя ее пользователям без повторного обращения к Интернету.

**Файрволы** (брандмауэры)

Межсетевые экраны, анализирующие и фильтрующие проходящий сетевой трафик, с целью обеспечения безопасности сети.

**Почтовые серверы**

Представляют услуги по отправке и получению электронных почтовых сообщений.

Серверы удаленного доступа (RAS)

Эти системы обеспечивают связь с сетью по коммутируемым линиям. Удаленный сотрудник может использовать ресурсы корпоративной ЛВС, подключившись к ней с помощью обычного модема.

Это лишь несколько типов из всего многообразия клиент-серверных технологий, используемых как в локальных, так и в глобальных сетях.

Для доступа к тем или иным сетевам сервисам используются клиенты, возможности которых характеризуются понятием «толщины». Оно определяет конфигурацию оборудования и программное обеспечение, имеющиеся у клиента. Рассмотрим возможные граничные значения:

«Тонкий» клиент

Этот термин определяет клиента, вычислительных ресурсов которого достаточно лишь для запуска необходимого сетевого приложения через web-интерфейс. Пользовательский интерфейс такого приложения формируется средствами *статического* HTML (выполнение JavaScript не предусматривается), вся прикладная логика выполняется на сервере.  
Для работы тонкого клиента достаточно лишь обеспечить возможность запуска web-браузера, в окне которого и осуществляются все действия. По этой причине web-браузер часто называют "универсальным клиентом".

«Толстый» клиент

Таковым является рабочая станция или персональный компьютер, работающие под управлением собственной дисковой операционной системы и имеющие необходимый набор программного обеспечения. К сетевым серверам «толстые» клиенты обращаются в основном за дополнительными услугами (например, доступ к web-серверу или корпоративной базе данных).  
Так же под «толстым» клиентом подразумевается и клиентское сетевое приложение, запущенное под управлением локальной ОС. Такое приложение совмещает компонент представления данных (графический пользовательский интерфейс ОС) и прикладной компонент (вычислительные мощности клиентского компьютера).

В последнее время все чаще используется еще один термин: «rich»-client. «Rich«-клиент своего рода компромисс между «толстым» и «тонким» клиентом. Как и «тонкий» клиент, «rich»-клиент также представляет графический интерфейс, описываемый уже средствами XML и включающий некоторую функциональность толстых клиентов (например интерфейс drag-and-drop, вкладки, множественные окна, выпадающие меню и т.п.)

Прикладная логика «rich»-клиента также реализована на сервере. Данные отправляются в стандартном формате обмена, на основе того же XML (протоколы SOAP, XML-RPC) и интерпретируются клиентом.

Некоторые основные протоколы «rich»-клиентов на базе XML приведены ниже:

* XAML (eXtensible Application Markup Language) — разработан Microsoft, используется в приложениях на платформе .NET;
* XUL (XML User Interface Language) — стандарт, разработанный в рамках проекта Mozilla, используется, например, в почтовом клиенте Mozilla Thunderbird или браузере Mozilla Firefox;
* Flex — мультимедийная технология на основе XML, разработанная Macromedia/Adobe.

## **Заключение**

Итак, *основная идея архитектуры «клиент-сервер» состоит в разделении сетевого приложения на несколько компонентов*, каждый из которых реализует специфический набор сервисов. Компоненты такого приложения могут выполняться на разных компьютерах, выполняя серверные и/или клиентские функции. Это позволяет повысить надежность, безопасность и производительность сетевых приложений и сети в целом.