Содержание:

ВВЕДЕНИЕ

Темпы современного развития информационных технологий поистине поражают воображения. Всё больший спектр задач подвластны искусственному интеллекту, и всё большие объемы работы выполняют компьютеры ежедневно. Грани, которые представители этой области вычерчивали, как недостижимые постепенно стираются, и мы уже сейчас можем видеть поколение программистов, которые в работе иногда даже не задумываются о том объёме вычислительной мощности, которые потребляют их программы. Темпы развития в этой области настолько стремительны, что поспеть за ними в экономическом аспекте становится всё сложнее, особенно это актуально для людей, которым не нужно использовать всё многообразие возможностей, предлагаемых на рынке услуг, а необходимо выполнять лишь определённый набор команд. В следствии этих фактов и возникли такие технологии как «облачные вычисления», многие веб-сервисы (как например почтовые сервисы), data mining, глобальные сети и прочие. В корне понимания озвученных технологий и многих других, как только проникающие в нашу жизнь, так и совсем привычных, лежит понимание клиент-серверной модели, поскольку именно эта модель лежит в основе всего вышеуказанного. Актуальность данного вопроса иллюстрируется большим количеством инвестиций в проекты, основой которой является модель клиент-серверных приложений. Исходя из этого может быть сформулирована гипотеза о том, что для понимания основ работы большого количества современных технологий необходимо изначально изучить модель клиент-сервера в базовом виде. Чтобы подтвердить или опровергнуть данную гипотезу необходимо решить следующие задачи:

- Изучить основные понятия связанные с клиент-серверной моделью
- Изучить протоколы взаимодействия между клиентской и серверной частью
- Изучить основные технологии, использующие клиент-серверную модель
- Составить критерии выбора стека технологий для составления клиент-серверной модели под конкретные задачи
- Разработать клиент-серверное приложение и на его (их) основе изучить прикладные приёмы создания клиент серверной модели

Методами исследования в данном случае будут выступать различные учебные пособия, описывающие клиент-серверную архитектуру, а также прикладные решения, используемые в современных технологиях.

Объектом исследования выступают технологии, которые используют или могут использовать клиент-серверную модель

Предметом исследования в данном случае является сама технология клиент-сервера и технологии, непосредственно связанные с ней.

ГЛАВА 1. ОСНОВЫ КЛИЕНТ-СЕРВЕРНОЙ МОДЕЛИ

1.1 ИСТОРИЯ РАЗВИТИЯ КЛИЕНТ-СЕРВЕРНОЙ МОДЕЛИ

Всё более обширной становится использования информационных систем, при всё более усложняющихся технологиях. Можно привести примеры систем, которые разрослись и усложнились настолько, что приобрели настолько глобальных характер, что от их работы зависит деятельность огромного количества людей. В виду своего глобального характера, что проявляется в том числе в необходимости обеспечить доступ к географически отдалённых друг от друга объектов, а также в силу разного рода других причин, такие системы имеют очень сложную архитектуру, которая предполагает функционирование в виде выполняющихся на отдельных узлах наборов компонентов. [1] Стоит учитывать, что со временем растёт как количество таких систем, так и требования, которые к ним предъявляют. Сложность создания таких систем высока, а стек технологий, задействованный в разработке отличается от того, что используется в «монолитных» системах. Тем не менее, ошибочно полагать, что такие распределенные системы являются изобретением последних лет. [3] Уже три десятилетия назад активно разрабатывались архитектурные модели «хост-компьютер +терминалы», реализованных на основе мейнфреймов (в качестве примера можно взять IBM-360/370 и компьютеры серии ЕС ЭВМ), или на основе мини-ЭВМ (таких как например PDP-11, или его отечественный аналога cm-4). [2] Отличительной особенностью этих систем являлась полная зависимость терминалов, которые представляли собой рабочие станции, от вычислений и команд, поступавших от хост-компьютера. Такой подход имел свои преимущества,

по сравнению с другими реализациями управления, существоваших в то время. Так, множеству людей начало открываться большое разнообразие ресурсов хост-компьютера для использования одновременно, а также и довольно дорогие для тех времен периферийные устройства (принтеры, графопостроители, устройства ввода с магнитных лент и гибких дисков, дисковые накопители).Задействованное программное обеспечение в таком случае имело дело только с "локальными" ресурсами - с локальной файловой системой, локальной оперативной памятью и т.д. [4]

Начавшийся бурный рост индустрии персональных компьютеров поначалу мало что изменил в идеологии построения программных систем -по-прежнему в большинстве своем программы имели дело с локальными ресурсами. Правда, часть этих ресурсов была уже "псевдо локальной",например, файлы на сетевом диске. [5] Однако по-прежнему файл обрабатывался непосредственно самим узлом, при этом файл сначала передавался по сети (уже на этом этапе развития возникли сложности -проблемы блокировки ресурсов и предупреждения тупиков, проблемы поддержки логической целостности для вносимых изменений и т.д.). В какой-то момент стало очевидно, что традиционные подходы не работают.

При увеличении объема перерабатываемых данных, а также по мере возрастания их стоимости стало очевидно, что доверять их обработку клиентским машинам нельзя. Любая ошибка на них (а чем больше клиентов, тем больше вероятность ошибки) приводит либо к потере данных, либо к их блокировкам в процессе работы, а, стало быть, к снижению общей производительности системы. [6] Следующим ключевым шагом стало повсеместное распространение идеологии клиент-серверной обработки. Это были "двухролевые" системы: клиент нес ответственность за отображение пользовательского интерфейса и выполнение кода приложения, а роль сервера обычно поручалась СУБД. В применении к примеру с файлом переход к клиент-серверной архитектуре может быть проиллюстрирован следующим образом: вместо того, чтобы читать файл целиком и обрабатывать его, машина-клиент передает машине-серверу запрос, в котором указывает, каким образом файл должен быть обработан. Сервер запрос клиента обрабатывает и возвращает ему результат.

Повсеместный переход на технологию "клиент-сервер" помог решить много старых проблем, но при этом создал много новых. Одной из основных трудностей было и остается определение границы между функционалом клиента и сервера. Часто решение о переносе части задач на сервер пагубно сказывается на общей производительности системы, и наоборот, перенос части нагрузки на клиента

может привести к потере централизации. [7] По мере роста популярности систем "клиент-сервер" набирала силу и технология объектно-ориентированного программирования, которая предлагала перейти к системной архитектуре с тремя слоями: слой представления отводится пользовательскому интерфейсу, слой предметной области предназначен для описания основных функций приложения, необходимых для достижения поставленной перед ним цели, а третий слой представляет источник данных.

С появлением Web всем внезапно захотелось иметь системы "клиент-сервер", где в роли клиента выступал бы Web-браузер. Появившиеся инструментальные средства конструирования Web-страниц были в меньшей степени связаны с SQL и потому более подходили для реализации третьего уровня. [8]

В современном мире считается, что быстрый темп развития иноваций, тесно переплетающихся с клиент-серверным строением, до сих пор не стоит на месте – подавляющее количество баз данных, действующих в современности, создано благодаря данным разработкам. Не менее важными выступают разделы, которые граничат, с продвижением данной разработки - так называемые трехслойные и многослойные, а также децентрализованные приложения. Данные о последних годах развития компьютерного продукта повествует нам о том, что строение баз данных необходимо подбирать при помощи потребностей производства, но не собственных желаний работников. [9]

Не секрет, что правильная и четкая организация информационных бизнес-решений является слагающим фактором успеха любой компании. Особенно важным этот фактор является для предприятий среднего и малого бизнеса, которым необходима система, которая способна предоставить весь объем бизнес-логики для решения задач компании. [1] В то же время, такие системы для компаний со средним и малым масштабом сетей часто попадают подкритерий —цена - качество‖, то есть должны обладать максимальной производительностью и надежностью при доступной цене. Первоначально системы такого уровня базировались на классической двухуровневой клиент-серверной архитектуре (Two-tier architecture). [1]

1.2 КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА

1.2.1 ОПРЕДЕЛЕНИЯ

Модель построения информационной системы это совокупность положений, определяющих прототип, устройство, предполагаемое назначение и объединение составляющих информационной системы.

Клиент-сервер есть вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг (сервисов), называемыми серверами, и заказчиками услуг, называемыми клиентами. [1]

Сервер — это программа, представляющая какие-то услуги другим программам и обслуживающая запросы клиентов на получение ресурсово пределенного вида.

Клиент — это программа, использующая услугу, представляемую программой сервера.

Часто люди клиентом или сервером просто называют компьютер, на котором работает какая-либо из этих программ. В сущности, клиент и сервер — это роли, исполняемые программами. Клиенты и сервер физически могут находиться на одном компьютере. Одна и та же программа может быть и клиентом, и сервером одновременно. [2]

1.2.2 РАЗВИТИЕ ИДЕЙ КЛИЕНТ-СЕРВЕРНОЙ КОНЦЕПЦИИ

Одними из главных недочётов современных компьютеров выступают их низкая вычислительная мощность, безопасность и необходимость

покупки вспомогающего компьютерного оборудования, способствующего ликвидации обособленности различных ПК друг против друга.

Зачастую множеству людей, активно использующим ЭВМ требуются как достойная вычислительная мощность так и подходящие возможности личных ПК. [2] Из-за этого в сферах, где для работы требуются сложные расчёты и применяются мощные изолированные мейнфреймы или обособленные кластеры с терминалами, работникам часто бывает необходимо использовать другие ЭВМ для операций, обслуживающих командный ввод в окно терминала. В связи с этим такие люди вынуждены знать минимум 2 разные ОС (внушительные ЭВМ, в классическом

варианте, обладают такими операционными системами, как ОС MVS, VMS, VM, UNIX, а ПК в свою очередь - MS DOS/MS Windows, OS/2 или Mac), а также искать пути преодоления проблем с двусторонней передачей сведений. По данным анкетирования, проводившегося среди работников около 300 крупнейших фирм США, пользователей ЭВМ оказалось, что 81% представителей нуждаются в доступности информации с двух и более компьютеров. [2] Для решения данной проблемы ПК стали группировать в ЛВС и обеспечивать их уникальными ОС (NetWare фирмы Novell) для одновременного получения данных, расположенных в некоторых разных узлах сети. Данное изобретение является файл-сервером, но, к сожалению, они обладают некоторыми несовершенствами. Например, эта разработка не может в полном объёме гарантировать анонимность подключения и безопасность файлов. В сеть данные поступают в полном объёме, невзирая на то, что людям, использующим эти файлы необходима лишь их часть и эта проблема снижает скорость передачи данных и ощутимо загружает виртуальное пространство. [3] Неуязвимость структуры файл-серверов довольно низкая, например неполадки одного из серверов во время регистрации данных может обеспечить утрату или изменение нужного файла. Для обеспечения непротиворечивости данных приходится блокировать файлы, что также приводит к замедлению работы. Естественным желанием

Задачей высокопрофессиональных кадров, связанных с информационными технологиями, в этом контексте было объединение возможностей персональных компьютеров с тем функционалом, который может предоставить мощный центральный компьютер. Для достижения этой цели на начальном этапе было использование персональных компьютеров в роле представляющих гибкий интерфейс терминалов. [3] При использовании этого метода в ПК, который использует особый протокол связи с центральным компьютером, может быть запущено немодальный виртуальный процесс работы терминала. Это позволяет получить рабочую архитектуру, способную содержать в себе все преимущества мощного центрального компьютера на персональном компьютере, который может эксплуатироваться как обычно. Это позволяет не использовать в рабочем пространстве две независимые архитектуры (что может привести к путанице), однако все недостатки мощных центральных компьютеров всё ещё остаются. [3]

Помимо этого, хотя персональные компьютеры, имеющие дисплеи с картой VGA, позволяют работать с графикой, однако использовать их в качестве графического терминала большой центральной машины неудобно. Задача выполняется в центральном компьютере и по проводам передаются графические образы экрана.

Эти образы довольно велики и скорость смены изображения на экране может быть очень низкой. Следующим шагом в решении описанной выше проблемы явилось использование архитектуры клиент-сервер. В такой архитектуре все компьютеры сети разделены на 2 группы: клиенты и серверы. Компьютер-сервер - это мощный компьютер с большой оперативной памятью и большим количеством дискового пространства. [5] На нем хранится база данных и выполняется сложная обработка, требующая больших вычислительных ресурсов. На компьютерах-клиентах выполняются первичная обработка данных при вводе, форматирование данных, а также окончательная (финишная) обработка данных, извлеченных с сервера. В качестве компьютеров-клиентов обычно используются персональные компьютеры типа IBM PC или Macintosh.

Преимущества архитектуры клиент-сервер очевидны. Каждый тип компьютера используется по своему назначению, а следовательно, обеспечивается более полное использование возможностей компьютеров.[5] На компьютерах-клиентах работают знакомые пользователям PC пакеты, позволяющие предоставлять результаты работы всей системы в удобном для анализа и принятия решений виде. На этих компьютерах легко можно реализовать дружественный пользовательский интерфейс приложения, использующий графику, цвет, звук, работу с окнами и мышью и т.д. Компьютер-клиент позволяет быстро выполнять ввод и первичный контроль данных. [5] Для финишной обработки данных могут использоваться те редакторы или пакеты электронных таблиц, которые пользователь считает наиболее удобными. В качестве компьютеров-клиентов могут одновременно использоваться компьютеры разных типов с различными операционными системами.

Архитектура клиент-сервер позволяет реализовать распределенную обработку, поскольку часть работы (интерфейс с пользователем, финишная обработка) выполняется на компьютере-клиенте, а часть - на компьютере-сервере. Это позволяет снизить загрузку сервера и оптимизировать его работу, а также увеличить число клиентов, одновременно работающих с сервером. Наиболее часто архитектура клиент-сервер применяется для приложений, созданных с использованием систем управления базами данных (СУБД). [7]

Дальнейшим развитием архитектуры клиент-сервер явилось использование в сети не одного, а нескольких серверов баз данных. Это позволило перейти от работы с локальной БД к работе с распределенной БД. Причем работа с распределенной базе данных (БД) "прозрачна" для пользователя, т.е. он работает с ней так же, как с локальной БД, не задумываясь о том, на каком сервере лежат его данные. [7]

Пользователь использует один сервер, а если он не найдёт необходимую информацию на своих страницах, то в этом случае он запрашивает данные с других серверов. Многохостовая модель связи в наши дни является очень перспективной т. к. дает возможность поменять главную производительную ЭВМ на некоторое количество менее мощных и, следовательно, более дешевых, и еще больше распараллелить обработку данных. Кроме того, такая архитектура увеличивает безопасность модели в целом, ведь если выйдет из строя один хост, то нагрузка может быть распределена между другими, и приложения смогут продолжить работу. [7] В случае, если промежуток местной или глобальной сети вышел из строя, устройство будет стараться отыскать другое направление к конечным данным (по остальным путям ЛВС). С другой стороны, местные сервера сохраняют информацию, которая задействуется подавляющее большинство времени в глобальном модуле - это дает возможность уменьшить раздачу информации по сети от сервера к серверу.

1.3 МОДЕЛИ КЛИЕНТ-СЕРВЕР

На сегодняшний момент есть, как минимум, три строения клиент-сервер:

1. Модель доступа к удаленным данным (RDA-модель);
2. Модель сервера базы данных (DBS-модель);
3. Модель сервера приложений (AS-модель).

Строение 1 и 2 классифицируются как двухзвенные и не имеют возможности применения как основное строение распределенной системы, а строение 3 — трехзвенное. [5] Также как остальное многозвенное строение, оно выгодно, т. к. в данной модели части для непосредственного контакта с потребителем абсолютно не зависят от составляющей разбора информации. [8] В общем, трехзвенной ее справедливо называть потому, что в ней определенно присутствуют следующие компоненты:

- Составляющая диалога с потребителем;
- ПО среднего уровня (middleware);
- Составляющая поиска и обработки информации.

Middleware — основная составляющая трехзвенного дистрибутивного устройства. Данная утилита обладает возможностью контроля переводов и связи, перемещения обработки данных, контроля информации, а также другими

возможностями. Есть одно очень важное отличие возможностей "сервер запросов — клиент запросов" и трехзвенных технологий : в начальном примере клиент посылает запрос на конкретную информацию, т. к. осведомлен о строении информационного хранилища (это действие именуется "поставкой данных" клиенту). Клиент делает SQL-запрос системе управления базами данных и после она их отправляет, в результате происходит непременное соединение алгоритмов, в качестве исполнения которой все системы управления базами данных пользуются скрытым SQL-каналом. [8]

Данный канал создается несколькими процессами: SQL/Net на компьютере-клиенте и SQL/Net на компьютере-сервере и производится с помощью утилиты connect по желанию пользователя. Канал является закрытым т. к. нет возможности, к примеру, создать утилиту, которая станет кодировать SQL-запросы с помощью определенного кода или другим образом будет вмешиваться в процесс передачи данных между клиентским и серверным приложением. [5]

В случае трехзвенной схемы клиент явно запрашивает один из сервисов (предоставляемых прикладным компонентом), например, передавая ему некоторое сообщение, и получает ответ также в виде сообщения. Клиент направляет запрос во внешнюю среду, ничего не зная о месте расположения сервиса. Имеет место так называемая "поставка функций" клиенту. [7]

Для клиента сама база данных видна исключительно посредством набора сервисов. Более того, он вообще ничего не знает о ее существовании, т. к. все операции над базой данных выполняются внутри сервисов. Таким образом, речь идет о двух принципиально разных подходах к построениюинформационных систем клиент-сервер. Двухзвенная архитектура на сегодняшний день может считаться достаточно устаревшей и, в связи с развитием распределенных информационных систем, постепенно отходит на второй план. И если для быстрого создания несложных приложений с небольшим числом пользователей этот метод подходит как нельзя лучше, то при построении корпоративных распределенных информационных систем он абсолютно непригоден в силу вышеперечисленных причин.

1.4 ТОЛСТЫЙ И ТОНКИЙ КЛИЕНТЫ

Как значится в словаре, тонкий клиент - это клиентское устройство (или программа), передающее большую часть исполняемых им функций серверу.

Толстый клиент определить намного проще - это все клиенты, не являющиеся тонкими. [7]

Тонкий клиент (thin client) — терминал сети без жестких дисков, вычислительная мощность которого и объем памяти определяются задачами пользователя. Все программы и приложения, хранящиеся на сервере, становятся доступными для пользователя при включении его устройства и выполнении процедуры регистрации на сервере. Тонким клиентом называют также ПК (в том числе и мобильный) с минимизированной мощностью процессора, оперативной и внешней памятью, позволяющий пользователю осуществлять ввод и отображение данных за счет выполнения вычислений и сохранения данных на более мощном ПК или сервере, с которыми он может осуществлять связь при помощи каналов средней пропускной способности. [7]

К тонкому клиенту могут подключаться внешние устройства ввода/вывода данных (сканеры, мониторы, принтеры и проекторы).Клиент называется тонким, если он не содержит вовсе или содержит лишь малую часть бизнес-логики, т. е. представляет собой исключительно презентационный слой. К толстым относятся клиенты со значительной долей бизнес-логики. [7]

Лучший пример тонкого клиента — Web-браузер, настолько универсальный, что способен подключаться к абсолютно разным прикладным программам, о которых "не знает" ничего, и, тем не менее обеспечивать приемлемый интерфейс пользователя. Вся концепция сетевого компьютера строится на идее создания дешевого небольшого устройства, на котором будет работать Web-браузер:

- Централизация
- Администрирования настольных устройств за счет централизованного оперирования приложениями и их модификациями, выполняемыми на сервере. Они становятся доступными для всех пользователей сразу, не требуется контакт с отдельными пользователями.
- Упрощение технологии
- Обслуживания рабочих мест, применяя соответствующие сервисные средства, администратор системы может одновременно обслуживать множество устройств.
- Возможность контроля за действиями пользователя, благодаря отсутствию накопителей на рабочем месте, пользователь не может привносить в конфигурацию программного обеспечения что-то свое, устанавливая собственные программы.

- Мобильность пользователей. Пользователь, не привязанный к конкретному рабочему месту, может произвольно перемещаться в пределах локальной сети, применять устройства дистанционного доступа.
- Повышение производительности труда операторов. Сведение всех сервисных операций на сервер заметно повышает производительность труда операторов.
- Снижение стоимости эксплуатации оборудования, при не слишком большом различии в стоимости оборудования тонкий клиент заметно дешевле в эксплуатации.

Преимущества и недостатки данного подхода можно представить в виде таблицы.

Преимущества	Недостатки
Ещё меньшая мощность компьютеров-клиентов	Усложняется разработка программного обеспечения
Ещё дешевле при большом количестве клиентов	Усложняется администрирование программного обеспечения
Ещё меньшая нагрузка на сеть	Более дорогой сервер
Ещё реже требуется обновление ПО на клиентах	При повреждении сервера останавливается работа всех клиентов

Таблица 1. Преимущества и недостатки тонких клиентов

Технология «тонкий клиент-сервер» базируется на трех основных составляющих:

- Стопроцентное выполнение прикладных задач на терминальном сервере,
- Многопользовательская операционная система,
- Технология распределенного отображения пользовательского интерфейса приложений.

Пользователи имеют возможность одновременно заходить в систему и выполнять приложения на сервере в разных, защищенных друг от друга сессиях сервера.[5] В системе с использованием тонкого клиента по сети или коммутируемой

телефонной линии на сервер передаются сигналы, отражающие нажатие на ту или иную клавишу либо то или иное движение мыши. А сервер производит соответствующие действия и формирует изменения экрана пользователя и передаѐт эти изменения тонкому клиенту. [5]

Тонкий клиент принимает от сервера видоизменённое отображение экрана и проецирует его на устройство (в утилитах нашего времени посылается как видоизмененный экран целиком, так и только доля проекции, в соответствии с необходимостью, с помощью которой ПО тонкого клиента проецирует новое изображение). В качестве клиента могут быть различные персональные компьютеры, однако, т. к. он поддерживает очень малое количество возможностей для редактирования информации, а как тонкий клиент также есть возможность использовать небольшие терминалы, располагающие маленькой вычислительной мощностью, не имеющие устройств с подвижными составляющими (жесткие диски, вентиляторы), располагающие, обычно, компонентами, имеющие существенные ограничения по объемам памяти (ОЗУ). [3]

В процессе обработки информации в терминальной модели все обрабатывающие программы, а также обрабатываемы данные, и параметры настроек располагаются непосредственно на терминальном сервере. Такой подход располагает большим количеством преимуществ в контексте базовой развёртки рабочей области (это проявляется, в частности, в отсутствии нужды в установке программного обеспечения на всех терминалах), более удобного проведения резервного копирования данных (надо копировать только содержимое сервера), восстановления сессий после сбоев (все пользовательские сессии автоматически сохраняются на сервере). [3]

Еще одно преимущество технологии тонких клиентов состоит в том, что она ориентирована на сотрудников, пользующихся дистанционным доступом. Если какое-то время сотрудникам компании нужно работать вне офиса (скажем, в командировке, в другом офисе или дома), эта проблема легко и безболезненно решается с помощью систем на базе тонких клиентов. [3]

Технология тонких клиентов обеспечивает высокую производительность даже на низко производительных рабочих местах, за счѐтиспользования вычислительных ресурсов терминального сервера. При необходимости повысить вычислительную мощность всей системы достигается заменой всего лишь одного устройства – терминального сервера, все рабочие места автоматически переходят на более высокий уровень производительности без необходимости замены каких-либо

устройств.

Толстый или Rich-клиент - это приложение, обеспечивающее (в противовес тонкому клиенту) расширенную функциональность независимо от центрального сервера. [2] Часто сервер в этом случае является лишь хранилищем данных, а вся работа по обработке и представлению этих данных переносится на машину клиента.
Достоинства:

- Толстый клиент обладает широким функционалом в отличие от тонкого.
- Режим многопользовательской работы.
- Предоставляет возможность работы даже при обрывах связи с сервером.
- Имеет возможность подключения к банкам без использования сети Интернет.
- высокое быстродействие.

Недостатки:

- Большой размер дистрибутива.
- Многое в работе клиента зависит от того, для какой платформы он разрабатывался.
- При работе с ним возникают проблемы с удаленным доступом к данным.
- Довольно сложный процесс установки и настройки.
- Сложность обновления и связанная с ней неактуальность данных.

Большинство современных средств быстрой разработки приложений(RAD), которые работают с различными базами данных, реализует стратегию: "толстый" клиент обеспечивает интерфейс с сервером базы данных через встроенный SQL. Такой вариант реализации системы с "толстым" клиентом, кроме перечисленных выше недостатков, обычно обеспечивает недопустимо низкий уровень безопасности. [1] Например, в банковских системах приходится всем операционистам давать права на запись в основную таблицу учетной системы. Кроме того, данную систему почти невозможно перевести на Web-технологию, так как для доступа к серверу базы данных используется специализированное клиентское ПО. [1]

Итак, рассмотренные выше модели имеют следующие недостатки.

"Толстый" клиент:

- сложность администрирования;
- усложняется обновление ПО, поскольку его замену нужно производить одновременно по всей системе;

- усложняется распределение полномочий, так как разграничение доступа происходит не по действиям, а по таблицам;
- перегружается сеть вследствие передачи по ней необработанных данных;
- слабая защита данных, поскольку сложно правильно распределить полномочия.

2. "Толстый" сервер:

- Усложняется реализация, так как языки типа PL/SQL не приспособлены для разработки подобного ПО, и нет хороших средств отладки;
- Производительность программ, написанных на языках типа PL/SQL, значительно ниже, чем созданных на других языках, что имеет важное значение для сложных систем;
- Программы, написанные на СУБД-языках, обычно работают недостаточно надежно; ошибка в них может привести к выходу из строя всего сервера баз данных;
- Получившиеся таким образом программы полностью непереносимы на другие системы и платформы.

Для решения перечисленных проблем используются многоуровневые (три и более уровней) архитектуры клиент-сервер.

Рассмотрим следующие компоненты:

1. Презентационная логика (Presentation Layer - PL);
2. Бизнес-логика (Business Layer - BL);
3. Логика доступа к ресурсам (Access Layer - AL).

Таким образом, можно придти к нескольким моделям клиент-серверного

взаимодействия:

"Толстый" клиент. Наиболее часто встречающийся вариант реализации архитектуры клиент-сервер в уже внедренных и активно используемых системах. [1] Такая модель подразумевает объединение в клиентском приложении как PL, так и BL. Серверная часть, при описанном подходе, представляет собой сервер баз данных, реализующий AL. К описанной модели часто применяют аббревиатуру RDA - Remote Data Access. [2]

"Тонкий" клиент. Модель, начинающая активно использоваться в корпоративной среде в связи с распространением Internet-технологий и, в первую очередь, Web-

браузеров. В этом случае клиентское приложение обеспечивает реализацию PL, а сервер объединяет BL и AL.

Сервер бизнес-логики. Модель с физически выделенным в отдельное приложение блоком BL. [3]

Хотя, рассматриваемые в этой части варианты разделения функциональности между клиентом и сервером являются "классическими", далее будет использоваться не только устоявшаяся традиционная, но и более новая терминология, возникшая вследствие распространения в корпоративных средах Internet/intranet-технологий и стандартов. [5]

Хотя в качестве серверной части, в общем случае, выступает менеджер многопользовательского доступа к информационным ресурсам, в этой статье будет сохраняться ориентация на серверы баз данных, как оконченое серверное звено.

Модели, основанные на Internet-технологиях и применяемые для построения внутрикорпоративных систем получили название intranet. Хотя intranet-системами сегодня называют все, что, так или иначе использует стек протоколов TCP/IP, с ними скорее следует связать использование Web-браузеров в качестве клиентских приложений. При этом важно отметить тот факт, что браузер не обязательно является HTML-"окном", но, в не меньшей степени, представляет собой универсальную среду загрузки объектных приложений/компонент -Java или ActiveX. [7]

Описанные три модели организации клиент-серверных систем в определенной степени являются ориентирами в задании жесткости связей между различными функциональными компонентами, чем строго описываемыми программами в реальных проектах. Жесткость связей в схеме взаимодействия компонент системы часто определяется отсутствием (или наличием) транспортного или сетевого уровня (Transport Layer - TL),обеспечивающего обмен информацией между различными компонентами.

Посмотрим на то, что же происходит в реальной жизни. С точки зрения применения описанных моделей, при проектировании прикладных систем разработчик часто сталкивается с правилом 20/80. Суть этого правила заключается в том, что 80% пользователей обращаются к 20%функциональности, заложенной в систему, но оставшиеся 20% задействуют основную бизнес-логику - 80%. [8] В первую группу пользователей попадают операторы информационных систем (ввод и редактирование информации), а также рядовые сотрудники и менеджеры,

обращающиеся к поисковым и справочным механизмам (поиск и чтение данных). Во вторую группу пользователей попадают эксперты, аналитики и менеджеры управляющего звена, которым требуются как специфические возможности отбора информации, так и развитые средства ее анализа и представления. С точки зрения реализации моделей необходимо обеспечить прозрачность взаимодействия между различными компонентами системы, а, следовательно, обратиться к существующим стандартам такого взаимодействия. [9]

Любая прикладная система, вне зависимости от выбранной модели взаимодействия, требует такой инструментарий, который смог бы существенно ускорить сам процесс создания системы и, одновременно с этим, обеспечить прозрачность и наращиваемость кода. На фоне разработки и внедрения систем корпоративного масштаба явно присутствует тенденция использования объектно-ориентированных компонентных средств разработки. Соответственно, полноценное применение объектов в распределенной клиент-серверной среде требует и распределенного объектно-ориентированного взаимодействия, то есть возможности обращения к удаленным объектам. Таким образом, мы приходим к анализу существующих распределенных объектных моделей. [9]

На настоящий момент наибольшей проработанностью отличаются COM/DCOM/ ActiveX и CORBA/DCE/Java. Если в первом случае требуемые механизмы поддержки модели являются неотъемлемой частью операционной платформы Win32 (Windows 95/NT/CE),то во втором случае предусмотрена действительная кросс-платформенность (например, везде, где есть виртуальная машина Java). Если попытаться объективно оценить (хотя любая такая попытка во многом субъективна) перспективы применения этих моделей, то для этого необходимо понять требования к операционным платформам, выдвигаемые различными функциональными компонентами системы. [8] При построении реальных систем корпоративного масштаба уже мало обходиться их разделением на три базовых фрагмента PL, BL, AL. Так как бизнес-логика является блоком, наиболее емким и специфичным для каждого проекта, именно ее приходится разделять на более мелкие составляющие. Такими составляющими могут быть, например, функциональные компоненты обработки транзакций(Transaction Process Monitoring), обеспечения безопасности (Security) при наличии разграничения прав доступа и выходе в Internet (Fire-wall), публикование информации в Internet (Web-access), подготовки отчетов(Reporting), отбора и анализа данных в процессе принятия решений (DecisionSupport), асинхронного уведомления о событиях (Event Alerts),тиражирования данных (Replication), почтового обмена (Mailing) и др. [7]

Вследствие наличия такого огромного количества функций, закладываемых в блоки поддержки бизнес-логики, появляется понятие сервера приложений(Application Server - AS). Причем, сервер приложений не, просто является некоим единым универсальным средним BL-звеном между клиентской и серверной, частью системы, но AS существует во множественном варианте, как частично изолированные приложения, выполняющие специальные функции, обладающие открытыми интерфейсами управления и поддерживающие стандарты объектного взаимодействия.[5]

Проникновение информационных технологий в сферу бизнеса в качестве неотъемлемого условия успешного управления приводит к тому, что системы корпоративных масштабов требуют сочетания различных клиент-серверных моделей в зависимости от задач, решаемых на различных конкретных направлениях деятельности предприятия. Вспомнив, снова, о правиле 20/80 можно придти к выводу, что наиболее оптимальным выбором, с точки зрения управляемости и надежности системы, является сочетание различных моделей взаимодействия клиентской и серверной части. [3] По сути, мы приходим даже не к трехуровневой, а многоуровневой (N-tier) модели, объединяющей различных по "толщине" клиентов, серверы баз данных и множество специализированных серверов приложений, взаимодействующих на базе открытых объектных стандартов. [2]

Существенным облегчением в реализации многоуровневых гетерогенных систем является активная работа ряда производителей программного обеспечения, направленная на создание переходного ПО. В отличие от продуктов middleware, обеспечивающих верхний транспортный уровень (универсальные интерфейсы доступа к данным ODBC, JDBC, BDE; MessageOrientedMiddleware - MOM; ObjectRequest Broker - ORB;), переходное ПО отвечает за трансляцию вызовов в рамках одного стандарта обмена в вызовы другого - мосты ODBC/JDBC и BDE/ODBC, COM/CORBA, Java/ActiveX и т.п.[1]

В общем случае, многоуровневая модель клиент-серверной системы может быть представлена, например, следующим образом:

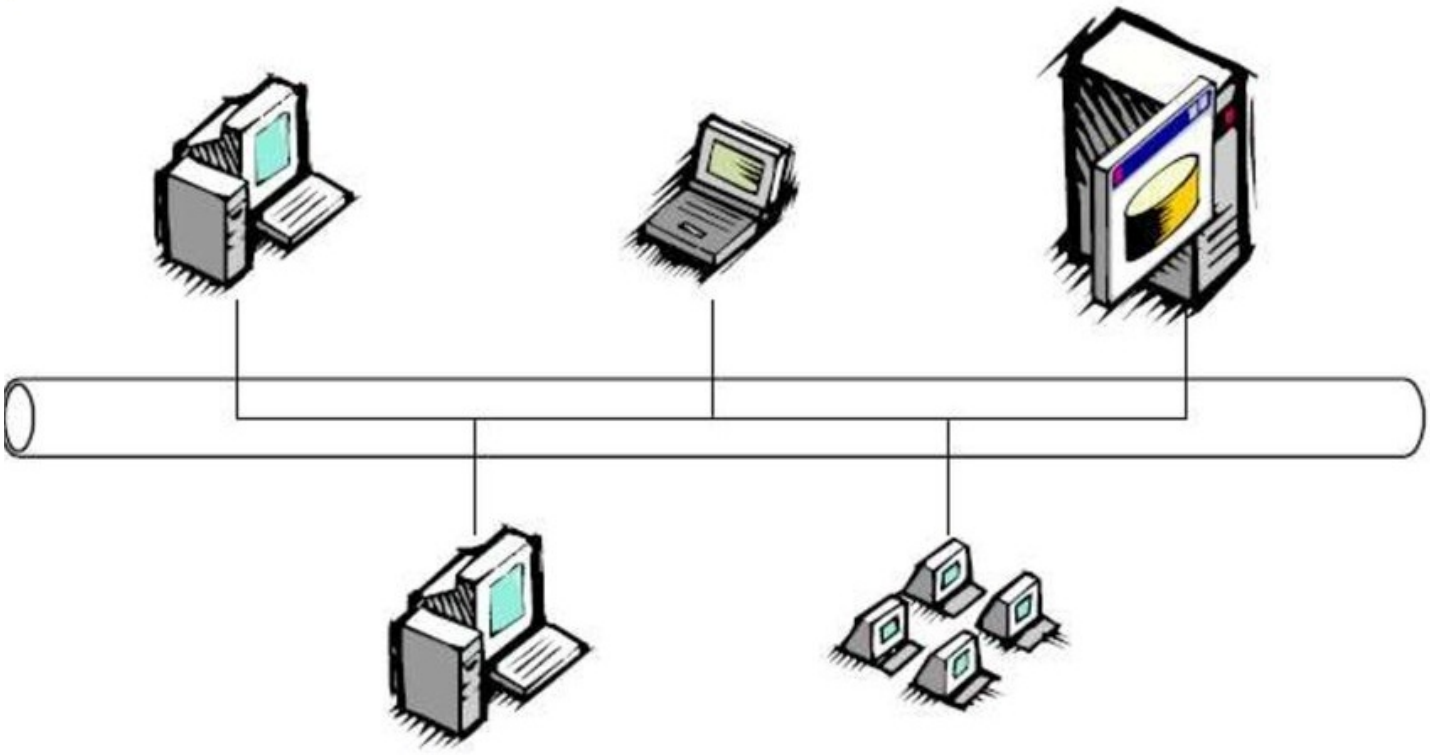

Рисунок 6. Многоуровневая клиент-серверная модель

Причем, различные стандарты взаимодействия могут применяться в различных связках узлов системы, а мосты встраиваться в любой узел или выделяться в своеобразные серверы приложений, с физическим выделением в узлах сети. [1] Двигаясь между клиентами слева-направо на диаграмме, мы можем наблюдать переход между различными моделями распределенных вычислений - через intranet к Internet.

1.5 КЛИЕНТ-СЕРВЕРНЫЕ ВЫЧИСЛЕНИЯ

В 1970-х и 1980-х годов была эпоха централизованных вычислений на мэйнфреймах IBM занимающих более 70% в компьютерном бизнесе мира. Бизнес транзакции, деятельности и базы данных, запросы и техническое обслуживание - все исполнялось на мэйнфреймах IBM. [1] Этап перехода к клиент-серверным вычислениям представляет совершенно новую концепцию и технологию реорганизации всего делового мира. Вычислительные парадигмы 1990-х годов называли «волной будущего». [1] Машина-клиент обычно управляет интерфейсами процессов, таких как графический интерфейс (графический пользовательский

интерфейс), отправкой запросов на сервер программ, проверкой данных, введенных пользователем, а также управляет местными ресурсами, а пользователь взаимодействует с такими, как монитор, клавиатура, рабочие станции, процессора и других периферийных устройств. С другой стороны, сервер выполняет запрос клиента, с помощью службы. После того как сервер получает запросы от клиентов, он выполняет поиск базы данных, обновления и управляет целостностью данных и отправляет ответы на запросы клиентов. [2]

Цель клиент-серверных вычислений - позволить каждой сетевой рабочей станции (клиента) и принимающей (сервера) быть доступными, по мере необходимости приложения, а так же обеспечивать доступ к существующему программному обеспечение и аппаратным компонентам от различных поставщиков для совместной работы. Когда эти два условия соединены, становятся очевидными преимущества клиент-серверной архитектуры, такие как: экономия средств, повышение производительности, гибкости и использования ресурсов.[2] Клиент-серверные вычисления состоят из трех компонентов: клиентского процесса, запрашивающего обслуживание и серверного процесса предоставления запрашиваемых услуг, с Middleware между ними для их взаимодействия.

Машина клиент обычно управляет пользовательским интерфейсом частей приложения, проверкой данных, введенных пользователем, отправкой запросов на сервер программы. Кроме того, клиентский процесс также управляет местными ресурсами, что позволяет пользователю взаимодействовать с монитором, клавиатурой, рабочими станциями, процессорами и другими периферийными устройствами. [2]

Машина Сервер выполняет служебные запросы клиента. После того как сервер получает запросы от клиентов, он производит поиск базы данных, обновления, управляет целостностью данных и отправляет ответы на запросы клиентов. Серверных процесс может работать на другой машине в сети; тогда сервер используется как файловая система услуг и приложений сервисов. Или в некоторых случаях, другой рабочий стол машины обеспечивает применение услуг. Сервер выступает в качестве программного обеспечения двигателя, который управляет общим ресурсам, таким как базы данных, принтеры, линии связи, или процессоров высокой мощности. [3]

Основная цель серверного процесса - выполнение фоновых задач, которые являются общими для приложений. Простейшая форма серверов - это дисковый сервер и файл-сервер. Если клиент передает запросы на файл или группы файлов

по сети на файловый сервер, эта форма обслуживания данных требует большой пропускной способности и может замедлить сеть с большим количеством пользователей. Более продвинутые формы серверов - это серверы баз данных, сервер транзакций и серверов приложений. [3]

Middleware позволяет приложениям прозрачно контактировать с другими программами или процессами независимо от местоположения. Ключевым элементом Middleware является NOS (NetworkOperatingSystem),которая предоставляет такие услуги, как маршрутизация, распределение, обмен сообщениями и управления сервисной сети. NOS полагается на коммуникацию протоколов предоставления конкретных услуг. Прежде чем пользователь может получить доступ к услугам сети, клиент-серверный протокол требует установку физического соединения и выбор транспортных протоколов. Клиент-серверный протокол диктует, каким образом клиенты запрашивают информацию и услуги от сервера, а также как сервер отвечает на эту просьбу. [3]

1.6 ПИРАМИДА МОДЕЛИ «КЛИЕНТ-СЕРВЕР»

Мартин Батлер, председатель Butler Group предложил новые рамки для реализации клиент-серверной стратегии. Это пятислойная модель под названием VAL (ValueAddedLayers) Модель. Основная структура напоминает по форме пирамиду, со слоями Инфраструктура и Middleware в нижней части пирамиды, а приложения, хранилища и бизнес модели на вершине. [5]

Характеристики каждого слоя:

Уровень 1 - Инфраструктурный слой. Слой инфраструктуры состоит из всех тех компонентов, которые являются пассивными и не выполняет бизнес-функции. Примеры относятся к этой категории компьютерных операционных систем, сетей, пользовательских интерфейсов и системы управления базой данных.

Уровень 2 – Middleware. Middleware позволяет приложениям прозрачно коммуникатировать с другими программами или процессами независимо от их местоположения. Это средство отображения приложений используемых ими ресурсов. Middleware является ключом к интеграции гетерогенных аппаратных и программных сред, обеспечивая тот уровень интеграции, который необходим многим организациям. Типичные Middleware проявляются после сетевых соединений, соединений с базой данных и осуществляют взаимодействие между

базой данных и приложениями. [5]

Уровень 3 – Программный. Приложения являются активными компонентами, которые выполняют работы по организации, и именно сюда, многие компании инвестируют большое количество усилий, времени и денег. Приложения, которые не имеют ключевого значения. Они все чаще приобретаются как готовый пакет. Приложения же, которые являются жизненно важными для увеличения конкурентоспособности компании в отрасли, разрабатываются на месте. [5]

Уровень 4 – Хранилище. Роль хранилища - изоляция бизнес-модели / спецификации от технологических инструментов, которые используются для ее осуществления.

Уровень 5 - Бизнес-модели. Бизнес-модель должна быть независимой, чтобы все технологии, которые используются для ее осуществления были применимы к аппаратной и программной среде в зависимости от того, что является наиболее подходящим. Это будет все больше и больше опираться на объектно-ориентированные методы, и уже существует поколение инструментов, которые поддерживают объект моделирования. [7]

ГЛАВА 2. РАЗВИТИЕ КОНЦЕПЦИЙ КЛИЕНТ-СЕРВЕРНОЙ МОДЕЛИ

2.1. КЛИЕНТ-СЕРВЕРНЫЕ ВЫЧИСЛЕНИЯ

Технология перехода к клиент-серверным вычислениям проявляется, главным образом, за счет более сложной ситуации в бизнесе в последние годы, такие как глобальный маркетинг, дистанционные онлайн продажи распределения, децентрализованной корпоративной стратегии и т.д. Все это требует быстрое реагирование, легкий доступ к информации данных, и более эффективной координации между людьми на всех уровнях как внутри, так и вне организации. [2] Клиент-серверные вычисления позволяют решать все эти проблемы и, следовательно, являются одним из приоритетных вопросов в умах управления ИТ. Ясно, что клиент-серверная технология приносит много преимуществ для бизнеса, а также расширение возможностей для расширения и конкурирования. К сожалению, все это может быть достигнуто только за счет более высокой стоимости сооружений и более сложной совместимости систем.

Грани, которые представители этой области вычерчивали как недостижимые постепенно стираются, и мы уже сейчас можем видеть поколение программистов, которые в работе иногда даже не задумываются о том объёме вычислительной мощности, которые потребляют их программы. Темпы развития в этой области настолько стремительны, что поспеть за ними в экономическом аспекте становится всё сложнее, особенно это актуально для людей, которым не нужно использовать всё многообразие возможностей, предлагаемых на рынке услуг, а необходимо выполнять лишь определённый набор команд. В следствии этих фактов и возникли такие технологии как «облачные вычисления», многие веб-сервисы (как например почтовые сервисы), data mining, глобальные сети и прочие. В корне понимания озвученных технологий и многих других, как только проникающие в нашу жизнь, так и совсем привычных, лежит понимание клиент-серверной модели, поскольку именно эта модель лежит в основе всего вышеуказанного.

Клиент-сервер не единственный способ решения бизнес-проблем, он также получил свои ограничения, один из которых, дороговизна. Но пока клиент-сервер осуществляется мудро, он может принести конкурентные преимущества. [1]

2.2 ТРЕХ-УРОВНЕВАЯ АРХИТЕКТУРА

В настоящее время клиент-серверная архитектура имеет гибкую модульную архитектуру. Она может быть изменена или дополнена.Различные подходы могут быть скомбинированы в различных комбинаторных последовательностях, удовлетворяющих практически любые вычислительным потребностям. Поскольку Интернет становится важным фактором в вычислительных средах клиент-серверных приложений, работающих через Интернет станет важным нового типа распределенных вычислений. [1]

Интернет расширит охват и мощь клиент-серверной архитектуры. С помощью общепринятых стандартов, это позволит облегчить и расширить клиент-серверную архитектуру как внутри, так и между компаниями. Так же из-за интернета будут происходить изменения в языках программирования к технологии распределенных объектов. [2]

Клиент-серверная архитектура по-прежнему остается единственной и лучшей архитектурой с точки зрения использования Интернета и других новых технологий. Но, независимо от развитий других архитектурных подходов, клиент-серверная архитектура, вероятно, останется основой для большинства вычислительных

событий в течение следующего десятилетия. [2]

Тем не менее, ошибочно полагать, что такие распределенные системы являются изобретением последних лет. [3] Уже три десятилетия назад активно разрабатывались архитектурные модели «хост-компьютер +терминалы», реализованных на основе мейнфреймов (в качестве примера можно взять IBM-360/370 и компьютеры серии ЕС ЭВМ), или на основе мини-ЭВМ (таких как например PDP-11, или его отечественный аналог cm-4). [2] Отличительной особенностью этих систем являлась полная зависимость терминалов, которые представляли собой рабочие станции, от вычислений и команд, поступавших от хост-компьютера. Такой подход имел свои преимущества, по сравнению с другими реализациями управления, существоваших в то время. Так, множеству людей начало открываться большое разнообразие ресурсов хост-компьютера для использования одновременно, а также и довольно дорогие для тех времен периферийные устройства (принтеры, графопостроители, устройства ввода с магнитных лент и гибких дисков, дисковые накопители). Задействованное программное обеспечение в таком случае имело дело только с "локальными" ресурсами - с локальной файловой системой, локальной оперативной памятью и т.д.

2.3 ПРОШЛОЕ И БУДУЩЕЕ КЛИЕНТОВ

Конечно же, с появлением большого количества людей, соединенных посредством компьютерной сети, сразу пошло бурное развитие клиент-серверных приложений. Но сервер в этом случае используется крайне экономно – это средство для хранения данных и выполнения несложных операций. Основную вычислительную роль берет на себя именно клиент, присоединенных к этой сети. Довольно удачная архитектура дает возможность использовать максимально ресурсы компьютера, богатый пользовательский интерфейс, важные данные пользователя хранятся на компьютере дома, а не сервере неизвестно какой страны.[2] Проблем несколько – заставить человека установить какое-либо приложение можно лишь действительно предоставив ему его как средство решения проблемы. А как же быть тому количеству предпринимателей, которые любыми способами хотят привлечь человека своим выгодным товаром? Пока что клиент-серверная архитектура не сильно им в этом помогает. Да и приложения пишутся для определенной платформы и версии операционной системы, что напрочь лишает взаимодействия пользователей разных операционных систем.

Но прогресс не стоит на месте, и в 1989 году была предложена концепция всемирной паутины. А уже через четыре года появляется первый браузер Mosaic. Сразу же в мире приложений начинает вырисоваться два класса приложений – тонкие клиенты, примером которого может служить тот же браузер Mosaic и толстые клиенты, которые по прежнему берут на себя значительную часть обработки информации и используют сервер для хранения и взаимодействия. [3]

Клиентские приложения не собирается сдавать позиций, потому что все, что может сейчас браузер - это отображать информацию. Динамика отсутствует. Одно лишь значительное преимущество – сайты становятся визитной картой, имеющие свой оригинальный дизайн, расположение меню, цветовую гамму. [3]

Для клиентских приложений это не было распространено –все создавалось на основе стандартных вариантов, стандартное месторасположения меню, стандартные цвета и шрифты. С одной стороны простота и функциональность, с другой стороны – красота и изящество. Конечно же, красота и изящество не могла не найти своих сторонников. Также пользователи разных операционных систем могли спокойно просматривать сайты. Возможны некоторые несоответствия в отображении информации, но все же лучше чем ничего. С появлением JavaScript в 1996 году html-страницы получают небывалую динамику и немного начинают походить на обычные клиентские приложения. [3] Конечно же, до полноценных клиентских приложений им, скорее всего, не дойти никогда, но страницы оживают. Живые страницы делают переворот в интернете, и идет бурное совершенствование скриптовой технологии, что приводит к рождению в 2005 году AJAX, а именно идеи асинхронного обращения к серверу для получения лишь необходимой части страницы, а не всей страницы. Инновационные решения, основанные на AJAX, типа карт Windows Live Local, приблизили веб-приложения к уровню удобства обычных клиентских программ. Вот тут веб-страницу можно было уже спутать с обычным клиентским приложением. Но выглядеть как в клиентском приложении, и работать как клиентское приложение – это разные вещи. Все же доступ к ресурсам ограничен, соединение между клиентом и сервером одноразовое – уже просмотренные страницы при следующем обращении необходимо загружать опять. Но это проблемы, которые могут быть решены с появлением нового http-протокола.[5]

В свое время, толстые клиенты, работающие на компьютере пользователя, при грандиозном развитии веб-технологий становятся в прямом смысле толстыми – они сложны для клиента при обновлении версии, они прихотливы к семействам операционных систем, плохо взаимодействуют с веб-серверами, потому что

зачастую построены с использованием клиент-серверной структуры. Здесь надо было срочно искать способы исправить ситуацию. Корпорация Microsoft не задержалась с предложением и выпустила на рынок программную технологию Microsoft .NET Framework, призванную объединить множество различных служб, написанных на разных языках, для общей совместимости. [5] Эта виртуальная машина может быть установлена на разных семействах Windows, а также на других операционных системах, что позволяет использовать любой из языков NET-семейства для написания работоспособных приложений для всех операционных систем, на которых установлен framework. Одна из проблем, которая так долго преследовала клиентские приложения, частично была решена.

Итак, гонка клиентских и веб-приложений находится на той стадии, когда веб просто физически не может проникнуть глубже в ресурсы пользователя, чтобы увеличить быстродействие. [5] Возможности в реализации отдельных техник все же еще не доступны по сравнению с клиентскими приложениями и есть проблема постоянного обращения к серверу, потому что все еще мы работаем с обычным html-кодом. Клиентские же приложение со своим богатством и простотой реализации сложнейших техник веба слишком неохотно потребляются пользователями, привыкшими к этому времени с помощью браузера решать самые сложные свои задачи. [7] К тому же, клиентские приложения по прежнему привязаны к определенным протоколам передачи данных для обмена информацией. А это влечет за собой дублирование определенных сервисов.

Так мы плавно перешли от истории к дням сегодняшним и видим, что бесспорного лидера нет, и каждая из технологий имеет свои достоинства инедостатки. После долгих блужданий возле клиентского компьютера интернет-гиганты все же готовы смирится с тем, что для увеличения быстродействия отдельных сервисов, для дальнейшего усложнения систем придется рассчитывать на мощности своих серверов, а не пытаться максимально глубоко влезть в ресурсы пользователей. [7] В связи с этим последнее время очень интенсивно пошло развитие сервисов, построенных для использования облачных вычислений (cloud computing) – технология обработки данных, в которой программное обеспечение предоставляется пользователю как Интернет-сервис. При этом пользователь не заботится об архитектуре облака, а лишь получает необходимые ему мощности от целых кластеров. Такие сервисы на данный момент уже предоставляют Microsoft, Amazon (Elastic Compute Cloud).Тем самым вся необходимая пользователю функциональность перемещается на сервера тех фирм, которые ее предоставляют. [7] Доступ осуществляет через браузер, а, значит, отсутствует привязанность к

разным семействам операционных систем.

Ярким примером может служить: Gmail - почтовый клиент Google, предоставляющий богатый инструментарий для работы с почтой прямо из браузера.

Тем же временем продолжается совершенствование способов приблизить веб в клиентским приложениям. В 2006 году корпорация Microsoft выпустила плагин к IE – Silverlight, который позволяет запускать приложения, содержащие анимацию, векторную графику и аудио-видеоролики, что характерно для RIA (Rich Internet application). [8]

Софтверные же компании имеют другую позицию – а именно видят будущее в smart client-ах – локальных приложениях, которые всецело риентированы на потребление всевозможных сервисов из вне.

Smart Client — это легко устанавливаемое и управляемое клиентское приложение, предоставляющее пользователю адаптивный, отзывчивый и богатый пользовательский интерфейс, полностью использующее возможности локальных ресурсов компьютера и интеллектуально управляющее взаимодействием с распределенными источниками данных. [8]

Ключевыми особенностями, отличающими Smart Client, являются: Богатый пользовательский интерфейс. Чтобы называться «умным»,клиентское приложение должно иметь удобный пользовательский интерфейс, подстраиваясь под нужды пользователя, допуская персонализацию и предоставляя все современные способы управления(drag’n’drop, контекстные меню, дочерние окна, нотификации и т. д.). Простая установка, не требующая участия пользователя. Приложение должно предлагать пользователю автоматическую установку, не требующую перезагрузки, долгого ожидания или большого объема закачиваемых файлов. Автоматическая установка обновлений. [8] Появление новых версий приложения должно автоматически проверяться, их установка так же должна происходить в автоматическом режиме. Возможность работы при отсутствии соединения с сервером. Если приложение в своей работе взаимодействует с удаленными источниками данных, оно также должно работать и предоставлять максимум возможной функциональности и при «отсоединенной» (оффлайн) работе.

Примерами существующих смарт-клиентов могут быть:

IssueVision - help desk management application

TaskVision – клиентское приложение, которое позволяет подключенным пользователям создавать задачи, проекты и распределять их между другими пользователями. [9]

Взаимодействие между пользователями построено с использованием веб-сервисов. Поскольку обмен структурированными данными между клиентом ссервисом производится с помощью стандартного языка XML – то приложение может взаимодействовать с большинством существующих сервисов, не зависимо от языка реализации. Однако даже с этими решениями у «smart» клиентов в случае прерывания связи с Internet только один выбор— отключаться, поэтому для устранения этого неудобства в Microsoft предложили технологию Live Mesh, позволяющую локально запускать Web-приложения. [9] Звучит немного парадоксально – имеется в виду, чтоприложение может работать с данными и при следующем подключении ужесинхронизировать их с сервером.Такая возможность (работать оффлайн) также будет включена впоследний Silverlight, что позволит даже с веб-страницами работать воффлайн режиме.

В ближайшее время, как и последние много лет, основной средой обмена информацией останется интернет. Судя по тенденциям, клиентские и веб приложения будут развиваться параллельно, только немного другим путем –теперь это будут не монолитные порталы, написанные одной командой и использующие ресурсы одной эко-системы. Это будут наряжено - один костяк и множество подключенных сервисов, возможно даже разработанные разными фирмами. [9] Это приведет к тому, что основное внимание и львиная доля времени будет расходоваться на разработку сложных сервисов, но потом они легко будут подключаться к всевозможным порталам, приложениях и другим сервисам. Тем самым в скором будущем мы будем находиться не только во всемирной паутине, но еще и каждая ниточка этой паутины будет состоять из такого же сложного смешения различных сервисов, потребляемых различными устройствами. [8] Но это огромное разнообразие сервисов будет полезно после окончательного внедрения нового протокола IPV6, что позволит подключать к интернету даже микроволновые печи, холодильники и т.д. Именно управление таким огромным количеством устройств в сети приведет к созданию множеств асервисов и порталов, которые в онлайн режиме помогут управлять вашими электроприборами.[9]

ГЛАВА 3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ КЛИЕНТ-СЕРВЕРНЫХ ПРИЛОЖЕНИЙ

3.1 ВЫБОР ПЛАТФОРМЫ ДЛЯ СЕРВЕРНОЙ ЧАСТИ МОДЕЛИ

Теперь, когда у нас есть практически вся общая информация про модель клиент-сервер, мы можем приступить к рассмотрению конкретных реализациях. [7] Итак, первый вопрос, который задаёт себе практически любой человек, который приступает к созданию клиент-сервера, это вопрос «Linux или Windows»? Особенностям, которые отличают эти две операционные системы, а также программы, которые написаны под определенную определённую ОС может быть посвящена не одна книга, однако, если посмотреть на клиент-серверные приложения со стороны непосредственного пользователя, то разница на самом деле будет невелика, и оба варианта подходят для большинства клиентов. [5] Linux и Windows на сегодняшний день являются самыми популярными системами для создания хост-сервера, и, хотя нельзя не отметить, что Linux является бесспорным лидером этой гонки, но если рассматривать предлагаемый этими системами функционал, то различия будут незначительны (в большинстве случаев), так что выбор не такой и простой. Под Linux обычно понимают общее название для всех архитектурных решений, разработанных на одном программном ядре. [3] Пакетные реализаций существует огромное количество, их отличия, однако, не будут в дальнейшем рассматриваться в ходе этой работы. Рассмотрим отличие этих операционных систем по некоторым критериям. Опустим критерии оценки удобства пользования графическим интерфейсом, поскольку это является скорее вопросом удобства использования. Одним из самых важных вопросов при выборе сервера является его безопасность, поскольку обычно такие модели содержат данные, чья конфиденциальность должна быть обеспечена. Обычно безопасность обеспечивается не только самой реализацией клиент-серверных приложений, но и хостом и операционной системой. Анти-лидером по количеству программных ошибок в операционной системе можно назвать Windows, различные ошибки в этой операционной системе возникают практически ежедневно и связаны с функциональными возможностями, которые не могут быть заменены в рамках этой операционной системы. [2] Поэтому, можно сказать, что если выбор операционной системы для сервера обусловлен только выбором безопасности, то операционная система Linux лидирует в этом показателе, хотя безусловно нельзя полагать, что Windows server является совершенно уязвимой ОС – использование современных методов защиты и технологий компьютерной безопасности делают её достаточной. Рассмотрим стабильность работы операционных систем. Если провести достаточно

подробную аналитику, то можно бесспорно заявить, что операционная система Windows является более уязвимой к ошибкам, которые могут привести к аварийной остановке работы сервера. [1] Чаще всего такие ошибки связаны с определённой несовместимостью программных компонентов, но диагностика и нормализация этой проблемы может отнять много времени. Средняя частота перезагрузок на серверах на базе Windows составляет 2.3 перезагрузки в день, некоторые же сервера на базе Linux фактически работают без единой перезагрузки на протяжении нескольких лет, что вовсе не означает, что на этой операционной системе не случаются ошибки, однако, как правило, они более редкие и реже приводят к фатальным изменениям. Возможности обеих операционных систем ограничены лишь набором технологий, которые можно в них использовать. [1] Практически все существующие технологии имеют какую-то систему интеграции с обеими операционными системами, однако, всё же интеграция технологий обычно не поспевает за созданием этих технологий, да и их использование лучше оптимизировано под «родную» операционную систему. [1] Примерный список технологий, который лучше использовать на какой-то конкретной операционной системе приведён в виде таблицы (технологии не указанные в таблице работают практически одинаково на обеих операционных системах). [2]

Технологии	Linux	Windows
Web-приложений	+	-
WYSIWYG	+	-
PHP, MySQL,CGI,Perl,Python	+	-
ASP,ASP.NET,MS SQL	-	+
ASP,ASP.NET,PHP, MySQL	-	+

Таблица 2. Сравнение используемых технологий

Как видно из приведённой таблицы технологии, непосредственно разрабатываемые Microsoft обычно работают лучше на операционных системах

Windows, все прочие либо одинаково работают на обеих платформах (к примеру, весь стек-технологий Java) или же имеют некоторое преимущество в ОС Linux (Что, как в случае с Python, может служить основополагающим моментом не только в рамках первоначального выбора, но и может стать серьёзным аргументом по переносу серверной части. [2] Поскольку тот Python имеет на сегодняшний день самое большое количество эффективных библиотек для работы в сфере data science и машинного обучения, что бессомнения может потребоваться на каком-то этапе развития клиент-серверной модели), однако, и этот критерий не может быть заключительным, поскольку каждая конкретная задача должна решаться индивидуально и гибкость разрабатываемой модели является основополагающим моментом при разработке архитектуры. [2] Разумеется в этой таблице приведены лишь небольшое количество элементов, которые могут участвовать в анализе – традиционное лидерство операционной системы Linux можно отметить и в FTP-серверах, Web-серверах под управлением Apache, DNS-серверы, файловые серверы, серверы электронной почты, однако в указанном списке не всё так очевидно, поскольку в операционной системе Windows также может успешно функционировать все указанные выше технологии (WEB-серверы с ограничениями), но и у Windows есть ряд ещё неозвученных возможностей: широкое применение нашли внедряемые локальные сети под управлением службы каталогов Active Directory (которая активно развивается в каждом новом выпуске Windows server). [3] Последний сравниваемый критерий будет заключаться в цене. Большинство дистрибутивов Linux распространяются согласно лицензии GPL, а значит совершенно бесплатны, исключением являются некоторые специализированные программные дистрибутивы. Программный продукт компании Microsoft является полностью коммерческим продуктом, поэтому стоимость устанавливается самой компанией и, к сожалению, эта стоимость является довольно высокой, иногда настолько, что цена одного только этого программного обеспечения может превышать стоимость всего оборудования. [3] Для более подробного изучения модельной структуры клиент-сервера в дальнейшем в этой работе будут рассмотрены особенности реализации клиент-серверных приложений на базе каждой из операционных систем.

3.2 РЕАЛИЗАЦИЯ КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ В LINUX

Для реализации клиент-серверного приложения в Linux будем использовать сильные стороны этой операционной системы. Напишем TCP/IP-сервер на C, который обменивался бы с клиентом информацией о времени суток через заданный порт на указанном удаленном узле. Пользователь должен быть способен указать адрес удаленного хоста в виде его имени или точечного IP-адреса через терминал. [3] Клиент должен выводить имя/адрес локального хоста. Клиент также должен выводить номер порта локального сокета после соединения с сервером. Клиент также должен вывести имя/адрес удаленного хоста и данные о времени, возвращаемые сервером. [5] Сервер после отправки информации должен очистить неотправленные байты, а в последствии закрыть соединения с клиентом. Поскольку переброс информации о времени –достаточно простой протокол передачи информации, то можно рассмотреть лишь однопоточный сервер (более подробно об способах реализации многопоточного сервера в Linux на fork'ах и процессах можно прочитать выше в работе). [5] Итак, поскольку мы решили использовать C для реализации нашей задачи, то неплохо было бы ознакомиться с теми возможностями, которыми располагает этот язык. Для начала, базовая структура, которая используется при работе с интернет адресами: struct sockaddr_in {short sin_family;unsigned short sin_port; struct in_addr sin_addr;char sin_zero[8];};struct in_addr {unsigned long s_addr;};

Поле sin-family определяет используемый формат адреса (набор протоколов), для протокола TCP/IP должно иметь значение AF_INET.Поле sin_addr содержит адрес узла сети.Поле sin_port содержит номер порта на узле сети.Поле sin_zero служебное

Далее необходимо разобрать структуру изменения действия процесса. Эта структура используется для установления сигналов между процессами.

structsigaction {void (*sa_handler)(int);void (*sa_sigaction)(int, siginfo_t *, void *);sigset_t sa_mask;int sa_flags;void (*sa_restorer)(void);}

В некоторых архитектурных решениях используют объединение элементов, но не sa_handler и sa_sigaction одновременно. [7] Соответсвенно, эти два параметра используются для задачи типа действия процесса. Sa_mask –задаёт маску сигналов для блокировки, sa_flags проставляет набор флагов для корректировки поведения процесса.

Невозможно представить работу процесса без функции socket, которая используется для создания сокетов. Её прототив выглядит следующим образом:

intsocket (intdomain, inttype, intprotocol); где domain –параметр накладывающий ограничения на формат используемых адресов, type – определяет тип канала для связи с сокетом, protocol позволяет выбрать протокол для канала связи. При равенстве нулю ОС выбирает такой протокол автоматически. [7]

Для перевода данных из узлового порядка в сетевой и обратно используют функцию htons или её аналоги. [7] Прототип этой функции: uint16_t htons(uint16_t hostshort);

Для присвоения сокету какого-то имени в зависимости от адресного домена используется функция bind, имеющая следующий прототип: int bind(int s, char * name, int namelen); В качестве параметра s указывают дескриптор сокета, которому дают имя. Name –имя сокета, namelen –длина имени. [8]

Чтобы информировать клиента об ожидании запросов связи используют функцию listen, которая имеет следующий прототип: int listen(int s, int backlog); В качестве первого параметра указывают сокет от которого ожидают передачу, а в качестве второго – количество одновременных запросов. Для проведения операций над наборами сигналов используется функции sigsetops, которые в зависимости от вызова могут инициализировать набор сигналов, добавить сигнал или удалить сигнал и т.д. [8]

Для принятия сервером связи на сокет используется функция accept. Вызов этой функции является блокирующим. Прототип функции: int accept(int s, char * name, int* anamelen); аргумент s –дескриптор сокета для принятия связей от клиента, 2 аргумент –имя клиента, 3 аргумент –длина структуры адреса.

Для работы с адресами используют функции inet_..., которые выполняют различные функции, как например: преобразование IP-адреса в двоичный код (в заданном порядке), извлечь сетевой номер, преобразование в строковый вид, создать сетевой адрес и т.д. [8] Для закрытия установленных соединений используют функцию close.

Тогда, серверная часть может быть написана в таком виде: #include <stdio.h>#include <stdlib.h>#include <unistd.h>#include <errno.h>#include <string.h>#include <sys/types.h>#include <sys/socket.h>#include <netinet/in.h>#include <arpa/inet.h> #include <time.h> #include <sys/wait.h>#include <signal.h>#define MYPORT 1025#define BACKLOG 10

void sigchld_handler(int s){while(wait(NULL) > 0);}int main(void){time_t timer;int sockfd, new_fd;struct sockaddr_in my_addr;struct sockaddr_in their_addr;sin_size;struct sigaction sa;int yes=1;if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1){perror("socket");exit(1);}if(setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int)) == -1){perror("setsockopt");exit(1);}my_addr.sin_family = AF_INET;my_addr.sin_port = htons(MYPORT);my_addr.sin_addr.s_addr = INADDR_ANY;memset(&(my_addr.sin_zero), 0, 8);if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1){perror("bind");exit(1);}if (listen(sockfd, BACKLOG) == -1){perror("listen");exit(1);}sa.sa_handler = sigchld_handler;sigemptyset(&sa.sa_mask);sa.sa_flags = SA_RESTART;if (sigaction(SIGCHLD, &sa, NULL) == -1) {perror("sigaction");exit(1);}while(1){sin_size = sizeof(struct sockaddr_in);if ((new_fd=accept(sockfd, (struct sockaddr *)&their_addr,&sin_size)) == -1){continue;}printf("Received request from Client: %s:%d\n",inet_ntoa(their_addr.sin_addr),MYPORT);if (!fork()){close(sockfd);timer = time(NULL);if (send(new_fd, ctime(&timer), 30, 0) == -1)perror("send");close(new_fd);exit(0);}close(new_fd); }return 0;}

Мы получили серверную часть нашей модели –теперь необходимо поговорить о клиентской части.[9] Клиент должен получить информацию о машине в сети. Для этих целей служит структура hostent. struct hostent {char *h_name;char **h_aliases;int h_addrtype;int h_length; char **h_addr_list; }#define h_addr h_addr_list[0]

Полями структуры hostent являются:h_name-(официальное имя машины); h_aliases-(оканчивающийся нулем массив альтернативных имен машины); h_addrtype-(тип адреса; в настоящее время всегда AF_INET); h_length-(длина адреса в байтах); h_addr_list-(оканчивающийся нулем массив сетевых адресов машины в сетевом порядке байтов); h_addr-(первый адрес в h_addr_list определен для совместимости с более ранними версиями).

Для доступа к машине и для изменения имени машины используются функции gethostname и sethostname со следующими прототипами: int gethostname(char *name, size_t len);int sethostname(const char *name, size_t len);

Для установки соединения с сервером используют функцию connect со следующим прототипом: int connect(int s, char * name, int namelen); где аргумент s- это дескриптор клиента, name –имя сервера и namelen –длина name. Тогда клиентскую часть можно записать следующим образом: #include <stdio.h>#include <stdlib.h>#include <unistd.h>#include <errno.h>#include <string.h>#include <netdb.h>#include <sys/types.h>#include <netinet/in.h>#include <sys/socket.h>

#define PORT 1025 #define MAXDATASIZE 500

int main(int argc, char *argv[]){int sockfd, numbytes;char buf[MAXDATASIZE];struct hostent *he;struct sockaddr_in their_addr;char hostn[400];char ipadd[400];struct hostent *hostIP;if ((gethostname(hostn, sizeof(hostn))) == 0){hostIP = gethostbyname(hostn);}else{printf("ERROR:FC4539 - IP Address not found.");}if (argc != 2){fprintf(stderr,"usage: client hostname\n");exit(1);}if ((he=gethostbyname(argv[1])) == NULL){perror("gethostbyname");exit(1);}if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1){perror("socket");exit(1);}their_addr.sin_family = AF_INET;their_addr.sin_port = htons(PORT);their_addr.sin_addr = *((struct in_addr *)he->h_addr);memset(&(their_addr.sin_zero), 0, 8);(connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1){perror("connect");exit(1);}if ((numbytes=recv(sockfd, buf, MAXDATASIZE-1, 0)) ==-1){perror("recv");exit(1);}buf[numbytes] = 0;printf("\n\nLocalhost: %s\n", inet_ntoa(*(struct in_addr *)hostIP->h_addr));printf("Local Port: %d\n", PORT);printf("Remote Host: %s\n",inet_ntoa(their_addr.sin_addr));printf("Received daytime data: %s\n",buf);close(sockfd);return 0;}

Мы получили базисный каркас клиент-серверного приложения, который абсолютно справляется со своей задачей, но что самое главное – рабочую модель, которая может быть разобрана с проектировочной стороны вопроса. [1] Данная модель может быть усложнена, но базовые принципы сохраняются. Программирование клиент-серверных приложений на C под Linux отличается своей простотой и элегантностью, все технологии или дополнительные задачи, которые нужно будет добавить могут быть добавлены уже в готовую модель без нужды переписывания кода заново. [2]

3.3 ОСОБЕННОСТИ РЕАЛИЗАЦИИ КЛИЕНТ-СЕРВЕРА ПОД WINDOWS

В 3.1 была довольно разгромная оценка выбора Windows в качестве операционной системы для серверной части (важно отметить, что серверная и клиентская части могут быть реализованы в разных операционных системах и даже используя разный стек технологий. Оценки даются с позиции содержания толстой серверной части «в отрыве» от клиентской), из-за чего могло сложиться неправильное представление о том, что Windows безнадёжна и ничего путного в плане клиент-серверной архитектуры на ней нельзя добиться. [3] Чтобы это опровергнуть

(напоминаю, что все оценки в главе 3.1 давались лишь в сравнении, а однозначный вывод так и не был сформулирован) попробуем реализовать WEB-сокетный клиент-сервер, который был указан в главе 3.1 как тот, что более приспособлен для Linux (так это и есть, но это не мешает ему прекрасно работать под Windows). Итак, перед началом непосредственного разбора стоит упомянуть, что в отличие от более элегантного кода Linux, решительно невозможно добиться той же лаконичности в данном случае, несмотря даже на гораздо более высокий уровень абстракции (разумеется будет использоваться стек-технологий Microsoft, в частности .NET и C#) – поэтому, чтобы не растягивать работу искусственно будут приведены лишь выдержки из этой программы (выделение этого кода в приложение также считаю излишним, поскольку большая часть кода будут занимать служебные функции, которые не имеют непосредственного отношения к разбираемой теме). [5] Итак, основная работа по соединению происходит инкапсулировано в реализациях класса WebSocket или классов наследуемых от него. Работа с сокетами происходит в классе System.NET.Sockets.Socket. И вообще, лаконичность и возможность сильного контроля С, сменилось на большую семантическую связь и большой вариативности (что обеспечивается иногда даже низкоуровневой реализацией отдельных классов, что безусловно было сделано для более тесной интеграции с Winapi). [7] В целом, сами по себе эти классы не предоставляют какой-то уникальный набор инструментов по управлению соответствующими структурами и, по сути, внутри имеют очень много общего с уже рассмотренными структурами и функциями в СИ. Приведём пример реализации соединения: using System;using System.Collections.Generic;using System.Linq;using System.IO;using System.Threading.Tasks;namespace Fleck{public class WebSocketConnection : IWebSocketConnection{public WebSocketConnection(ISocket socket, Action<IWebSocketConnection> initialize, Func<byte[], WebSocketHttpRequest> parseRequest, Func<WebSocketHttpRequest, IHandler> handlerFactory, Func<IEnumerable<string>, string> negotiateSubProtocol){Socket = socket;OnOpen = () => { };OnClose = () => { };OnMessage = x => { };OnBinary = x => { };OnPing = x => SendPong(x);OnPong = x => { };OnError = x => { };_initialize = initialize;_handlerFactory = handlerFactory;_parseRequest = parseRequest;_negotiateSubProtocol = negotiateSubProtocol;}public ISocket Socket { get; set; }private readonly Action<IWebSocketConnection> _initialize;private readonly Func<WebSocketHttpRequest, IHandler> _handlerFactory;private readonly Func<IEnumerable<string>, string> _negotiateSubProtocol;readonly Func<byte[], WebSocketHttpRequest> _parseRequest;public IHandler Handler { get; set; }private bool

```
_closing;private bool _closed;private const int ReadSize = 1024 * 4;public Action OnOpen { get; set; }public Action OnClose { get; set; }public Action<string> OnMessage { get; set; }public Action<byte[]> OnBinary { get; set; }public Action<byte[]> OnPing { get; set; }public Action<byte[]> OnPong { get; set; }public Action<Exception> OnError { get; set; }public IWebSocketConnectionInfo ConnectionInfo { get; private set; }public bool IsAvailable {get { return !_closing && !_closed && Socket.Connected; }}public Task Send(string message){return Send(message, Handler.FrameText);}public Task Send(byte[] message){return Send(message, Handler.FrameBinary);}public Task SendPing(byte[] message){return Send(message, Handler.FramePing);}public Task SendPong(byte[] message){return Send(message, Handler.FramePong);}private Task Send<T>(T message, Func<T, byte[]> createFrame){if (Handler == null)throw new InvalidOperationException("Cannot send before handshake");if (!IsAvailable){const string errorMessage = "Data sent while closing or after close. Ignoring.";FleckLog.Warn(errorMessage);var taskForException = new TaskCompletionSource<object>(); taskForException.SetException(new ConnectionNotAvailableException(errorMessage));return taskForException.Task;}var bytes = createFrame(message);return SendBytes(bytes);}public void StartReceiving(){var data = new List<byte>(ReadSize);var buffer = new byte[ReadSize]; Read(data,buffer);}public void Close(){Close(WebSocketStatusCodes.NormalClosure);}public void Close(int code){if (!IsAvailable)return;_closing = true;if (Handler == null) {CloseSocket();return;}var bytes = Handler.FrameClose(code);if (bytes.Length == 0)CloseSocket();else SendBytes(bytes, CloseSocket);}public void CreateHandler(IEnumerable<byte> data){var request = _parseRequest(data.ToArray()); if (request == null)return; Handler = _handlerFactory(request); if (Handler == null) return; var subProtocol = _negotiateSubProtocol(request.SubProtocols); ConnectionInfo = WebSocketConnectionInfo.Create(request, Socket.RemoteIpAddress, Socket.RemotePort, subProtocol); _initialize(this);var handshake Handler.CreateHandshake(subProtocol);SendBytes(handshake, OnOpen);}private void Read(List<byte> data, byte[] buffer) {if (!IsAvailable)return;Socket.Receive(buffer, r =>{if (r <= 0) {FleckLog.Debug("0 bytes read. Closing."); CloseSocket();return;}FleckLog.Debug(r + " bytes read");var readBytes = buffer.Take(r); if (Handler != null) {Handler.Receive(readBytes);} else {data.AddRange(readBytes);CreateHandler(data);}Read(data, buffer);},HandleReadError);}private void HandleReadError(Exception e){if (e is AggregateException) {var agg = e as AggregateException; HandleReadError(agg.InnerException);return;}if (e is
```

ObjectDisposedException){FleckLog.Debug("Swallowing ObjectDisposedException", e); return;}OnError(e);if (e is WebSocketException) {FleckLog.Debug("Error while reading", e);Close(((WebSocketException)e).StatusCode);} else if (e is SubProtocolNegotiationFailureException) {FleckLog.Debug(e.Message); Close(WebSocketStatusCodes.ProtocolError);} else if (e is IOException) {FleckLog.Debug("Error while reading", e); Close(WebSocketStatusCodes.AbnormalClosure);}else{FleckLog.Error("Application Error", e);Close(WebSocketStatusCodes.InternalServerError);}}private Task SendBytes(byte[] bytes, Action callback = null) {return Socket.Send(bytes, () =>{FleckLog.Debug("Sent " + bytes.Length + " bytes"); if (callback != null) callback();},e =>{if (e is IOException) FleckLog.Debug("Failed to send. Disconnecting.", e); else FleckLog.Info("Failed to send. Disconnecting.", e); CloseSocket();});}private void CloseSocket() {_closing = true; OnClose();_closed = true;Socket.Close();Socket.Dispose();_closing = false;}}}

Для человека владеющим языком C# этот код не принесёт большого количества новой информации, но тем не менее, это является полностью рабочей моделью WEB соединения. [8] В ней происходит уже описанная выше логика работы, а сам код можно читать как семантически-понятный набор инструкций, что хоть и не делает сложность разработки ниже, но сокращает количество текста, необходимого для объяснений. [9] Для понимания контекста подключения приведём также код самого сервера на сокетах: public class WebSocketServer : IWebSocketServer{private readonly string _scheme;private readonly IPAddress _locationIP; private Action<IWebSocketConnection> _config;public WebSocketServer(string location){var uri = new Uri(location); Port = uri.Port; Location = location; _locationIP = ParseIPAddress(uri); _scheme = uri.Scheme; var socket = new Socket(_locationIP.AddressFamily, SocketType.Stream, ProtocolType.IP); if(!MonoHelper.IsRunningOnMono()){ #if __MonoCS__ #else #if !NET45 if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows)) #endif {socket.SetSocketOption(SocketOptionLevel.IPv6, SocketOptionName.IPv6Only, false);} #if !NET45 if (!RuntimeInformation.IsOSPlatform(OSPlatform.Windows)) {socket.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, 1); }#endif #endif}ListenerSocket = new SocketWrapper(socket); SupportedSubProtocols = new string[0];} public ISocket ListenerSocket { get; set; } public string Location { get; private set; } public int Port { get; private set; } public X509Certificate2 Certificate { get; set; } public SslProtocols EnabledSslProtocols { get; set; } public IEnumerable<string> SupportedSubProtocols { get; set; } public bool RestartAfterListenError {get; set; }public bool IsSecure;{get { return _scheme == "wss" && Certificate != null; }}public void Dispose();{ListenerSocket.Dispose();}private

IPAddress ParseIPAddress(Uri uri){string ipStr = uri.Host; if (ipStr == "0.0.0.0"){return IPAddress.Any;}else if(ipStr == "[0000:0000:0000:0000:0000:0000:0000:0000]") {return IPAddress.IPv6Any;} else {try {return IPAddress.Parse(ipStr);} catch (Exception ex){throw new FormatException("Failed to parse the IP address part of the location. Please make sure you specify a valid IP address. Use 0.0.0.0 or [::] to listen on all interfaces.", ex); }}}public void Start(Action<IWebSocketConnection> config){var ipLocal=new IPEndPoint(_locationIP, Port); ListenerSocket.Bind(ipLocal); ListenerSocket.Listen(100); Port=((IPEndPoint)ListenerSocket.LocalEndPoint).Port; FleckLog.Info(string.Format("Server started at {0} (actual port {1})", Location, Port)); if(_scheme=="wss"){if(Certificate==null){FleckLog.Error("Scheme cannot be 'wss' without a Certificate"); return; }if(EnabledSslProtocols==SslProtocols.None)EnabledSslProtocols=SslProtocols.Tls; FleckLog.Debug("Using default TLS 1.0 security protocol."); }}ListenForClients(); _config=config; }private void ListenForClients(){ListenerSocket.Accept(OnClientConnect, e =>{FleckLog.Error("Listener socket is closed", e); if(RestartAfterListenError){FleckLog.Info("Listener socket restarting"); try{ListenerSocket.Dispose(); var socket=new Socket(_locationIP.AddressFamily, SocketType.Stream, ProtocolType.IP); ListenerSocket=new SocketWrapper(socket); Start(_config); FleckLog.Info("Listener socket restarted"); }catch (Exception ex) {FleckLog.Error("Listener could not be restarted", ex); }}}); }private void OnClientConnect(ISocket clientSocket) {if (clientSocket==null) return; // socket closed FleckLog.Debug(String.Format("Client connected from {0}:{1}", clientSocket.RemoteIpAddress, clientSocket.RemotePort.ToString())); ListenForClients(); WebSocketConnection connection=null; connection=new WebSocketConnection(clientSocket, _config, bytes=>RequestParser.Parse(bytes, _scheme), r => HandlerFactory.BuildHandler(r, s=>connection.OnMessage(s), connection.Close, b=>connection.OnBinary(b), b=>connection.OnPing(b), b=>connection.OnPong(b)), s=>SubProtocolNegotiator.Negotiate(SupportedSubProtocols, s)); if(IsSecure) {FleckLog.Debug("Authenticating Secure Connection"); clientSocket, .Authenticate(Certificate, EnabledSslProtocols, connection.StartReceiving, e=>FleckLog.Warn("Failed to Authenticate", e));} else {connection.StartReceiving();}}}}

Как может быть видно из этих фрагментов кода – разработка сервера на Windows обладает рядом преимуществ, а именно: сочетание достоинств низкоуровнего подхода в сочетании с высоким уровнем абстракции, понятность кода, простого распределения нагрузки на процессы, удобную модульную структуру и в целом возможность воспроизведения всего стека современных технологий в заданных

условиях. [9] Тем не менее, нельзя однозначно назвать готовое решение для какой-то задачи, любые предпочтения без производительных оценок, полученных в конкретном контексте обречены сводится к выбору предпочтений. Однако, благодаря разбору указанных моделей можно говорить об отличии одного подхода от другого, что несомненно будет играть при объективной оценке для реализации конкретной задачи. [8]

ЗАКЛЮЧЕНИЕ

В ходе данной работы была рассмотрена модель клиент-сервера, а также технологии связанные с ней. В процессе изучения основных понятий и технологий, связанных с моделью клиент-сервера, мы пришли к выводу, что в течение развития информационных технологий и приёмов программирования модель клиент-сервера обретает всё большую популярность и имеет всё более повсеместный характер, что может являться следствием того, что клиент-серверная модель является одним из самых эффективных приёмов уменьшения требований вычислительных мощностей (данный факт также был иллюстрирован в ходе данной работы). В ходе работы постоянно проводились параллели между клиент-серверной моделью и существующими технологиями, к которым уже привыкли обыватели, что даёт представление о высокой востребованости этой технологии. В рамках данной работы были сформулированы и решены следующие задачи:

- Были Изучены основные понятия связанные с клиент-серверной моделью
- Были Изучены протоколы взаимодействия между клиентской и серверной частью
- Были Изучены основные технологии, использующие клиент-серверную модель
- Составлены критерии выбора стека технологий для составления клиент-серверной модели под конкретные задачи
- Были Разработаны клиент-серверные приложения и на их основе изучены прикладные приёмы создания клиент серверной модели

Исходя из этого, можно сказать, что наша гипотеза о том, что для понимания основ работы большого количества современных технологий необходимо изначально изучить модель клиент-сервера в базовом виде полностью подтвердилась. в следствии чего, можно говорить о базисной концепции изучаемой области, что, в частности, делает написанную курсовую работу –методическим пособием для начинающих изучать информационные технологии в целом, или же тех, кто хочет

разобраться в какой-то конкретной области современных технологических решений.

БИБЛИОГРАФИЯ

1. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений. –М.: Вильямс, 2008. -452с.
2. Снейдер И. Эффективное программирование TCP/IP. – М.: Питер, 2002. -314с.
3. Кровчик Э,Кумар В, Лагари Н, Мунгале А, Нагел К, Паркер Т, Шивакумар Ш. .NET. Сетевое программирование для профессионалов. – М.: Лори, 2005. -400с.
4. Кустов В. Руководство администратора баз данных Informix. –М.: O’Really, 1998 -45с.
5. Томас Коннолли, Каролин Бегг, Базы данных. Проектирование, реализация и сопровождение. Теория и практика. –М.: Вильямс,2017 -1440с.
6. Фаулер М. Новые методологии программирования. –М.:ThoughtWorks, 2001 -44с.
7. Барфилд Э., Уолтерс Б. Программирование клиент-сервер в локальных вычислительных сетях. –М.: Информационно-издательский дом "Филинъ", 1997. - 424 с.
8. ДустдарС, ШрейнерВ. *International Journal of Web and Grid Services (IJWGS)*. –онлайнпубликация,2005. -78с.
9. Хьюз К., Хьюз Т. Параллельное и распределенное программирование на С++. –М.: Вильямс, 2004. -672 с.

ПРИЛОЖЕНИЕ А

Список сокращений.

СУБД – Система управления базами данных.

ПО – Программное обеспечение.

БД – База данных.

ЭВМ – Электронно-вычислительная машина.

ПК – Персональный компьютер.

ИС – Информационная система.