

# Системное программное обеспечение локальных компьютерных сетей

## Модель сетевого взаимодействия клиент-сервер

Автор: Денис Пынькин  
Читает: Алексей Демидчук

2014 – 2015

e-mail: [denis.pynkin@bsuir.by](mailto:denis.pynkin@bsuir.by)

**<http://goo.gl/32cTV>**

СЧАСТЬЕ ДЛЯ ВСЕХ, ДАРОМ, И ПУСТЬ НИКТО НЕ УЙДЕТ ОБИЖЕННЫЙ!

(с)Стругацкие, Пикник на обочине

Термин "клиент-сервер" означает такую архитектуру программного комплекса, в которой его функциональные части взаимодействуют по схеме "запрос-ответ".

# Клиент или сервер?

## С сетевой точки зрения

Если рассмотреть две взаимодействующие части этого комплекса, то одна из них (клиент) выполняет активную функцию, т. е. инициирует запросы, а другая (сервер) пассивно на них отвечает.

По мере развития системы роли могут меняться, например некоторый программный блок будет одновременно выполнять функции сервера по отношению к одному блоку и клиента по отношению к другому.

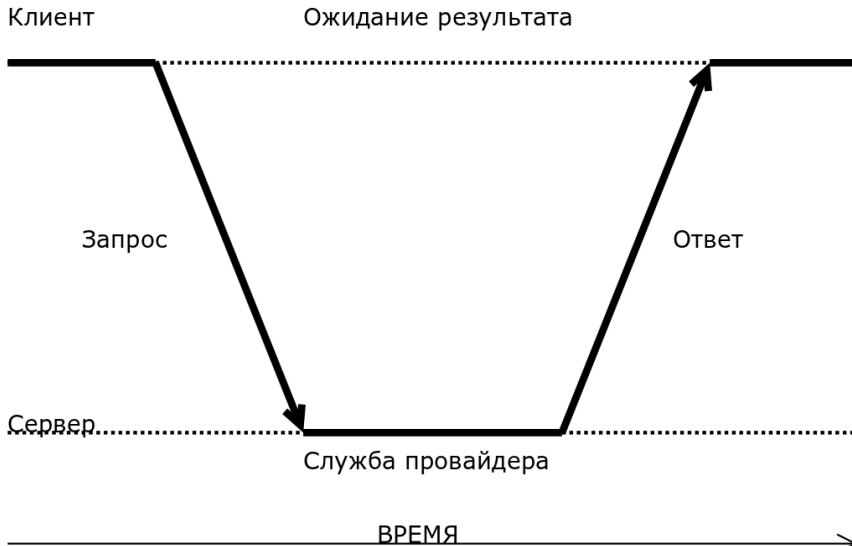
# Клиент или сервер?

## С функциональной точки зрения

Процессы, реализующие некоторую службу, например ФС или БД называются серверами.

Процессы, запрашивающие службы у серверов службы путем пересылки запроса и последующего ожидания ответа от сервера, называются клиентами.

# Сетевое взаимодействие



# Модель?

Модель клиент-сервер всегда была предметом множества дебатов и споров. Один из главных вопросов состоял в том, как точно разделить клиента и сервер. Четкого разделения здесь не может быть.

# Трехуровневая модель

- уровень пользовательского интерфейса
- уровень обработки
- уровень данных

# Уровень пользовательского интерфейса

Уровень пользовательского интерфейса обычно реализуется на клиентах. Этот уровень содержит программы, посредством которых пользователь может взаимодействовать с приложением.

Простейший вариант пользовательского интерфейса не содержит ничего, кроме символьного (или графического) дисплея.



# Уровень обработки

На этом уровне трудно выделить какие-то закономерности, обычно здесь реализуется логика работы программы.

# Уровень данных

На этом уровне находятся программы, которые поставляют данные обрабатывающим их приложениям.

В простейшем варианте уровень данных реализуется файловой системой, однако часто может использоваться и полнофункциональная база данных. В модели клиент-сервер этот уровень обычно находится на стороне сервера.

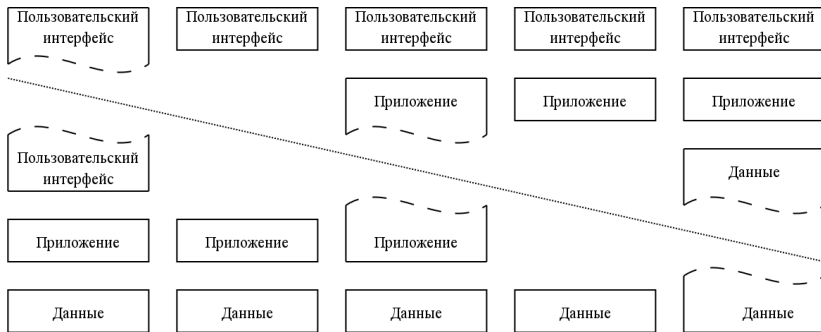
# Уровень данных

На этом уровне находятся программы, которые поставляют данные обрабатывающим их приложениям.

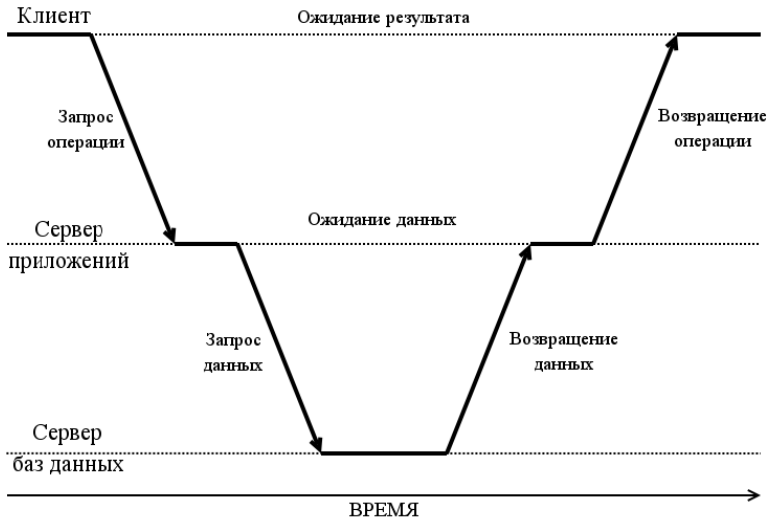
В простейшем варианте уровень данных реализуется файловой системой, однако часто может использоваться и полнофункциональная база данных. В модели клиент-сервер этот уровень обычно находится на стороне сервера.

Специфическим требованием этого уровня является требование сохранности – это означает, что когда приложение не работает, данные должны сохраняться в определенном месте в расчете на дальнейшее использование.

# Физически двухзвенная архитектура



# Физически трехзвенная архитектура



## Вертикальное распределение

Во множестве приложений обработка данных организована как многозвенная архитектура приложений клиент-сервер.

Особенностью такого типа является то, что он о достигается размещением логически разных компонентов на различных физических машинах.

## Вертикальное распределение

Во множестве приложений обработка данных организована как многозвенная архитектура приложений клиент-сервер.

Особенностью такого типа является то, что он о достигается размещением логически разных компонентов на различных физических машинах.

## Горизонтальное распределение

При таком типе распределения клиент или сервер может содержать физически разделенные части логически однородного модуля, причем работа с каждой из частей может протекать независимо. Это делается для выравнивания нагрузки.

## Вертикальное распределение

Во множестве приложений обработка данных организована как многозвенная архитектура приложений клиент-сервер.

Особенностью такого типа является то, что он о достигается размещением логически разных компонентов на различных физических машинах.

## Горизонтальное распределение

При таком типе распределения клиент или сервер может содержать физически разделенные части логически однородного модуля, причем работа с каждой из частей может протекать независимо. Это делается для выравнивания нагрузки.

## peer-to-peer

Для некоторых несложных приложений выделенного сервера может не быть вообще. Такую организацию обычно называют одноранговым распределением.



# Модель взаимодействия типа клиент-сервер и проектирование ПО

С точки зрения прикладного программиста, протоколы TCP/IP, как и большинство других протоколов компьютерной связи, просто предоставляет основные механизмы передачи данных. В частности протоколы TCP/IP, позволяют устанавливать соединение между 2-мя прикладными программами и передавать данные в прямом и обратном направлении. Поэтому *принято считать*, что протоколы TCP/IP обеспечивают одноранговую или прямую связь.

# Основная модель сетевого взаимодействия

Необходимость применения принципа организации взаимодействия типа клиент-сервер связана с решением проблемы согласования условий соединения.

Модель взаимодействия типа клиент-сервер предусматривает согласование условий соединения наиболее простым способом:

она требует, чтобы для взаимодействия любой пары приложений, один из участников приступал к работе заранее и ждал (возможно в течение неопределенного времени) до тех пор, пока к нему не обратится второй участник соединения.

# Архитектура клиента

Приложения действующие в качестве клиентов концептуально проще приложений выполняющих функции серверов.

- основной части клиентского программного обеспечения не нужно взаимодействовать с несколькими серверами

Приложения действующие в качестве клиентов концептуально проще приложений выполняющих функции серверов.

- основной части клиентского программного обеспечения не нужно взаимодействовать с несколькими серверами
- основная часть клиентского программного обеспечения применяется в виде обычных прикладных программ – не требуются привилегии

Приложения действующие в качестве клиентов концептуально проще приложений выполняющих функции серверов.

- основной части клиентского программного обеспечения не нужно взаимодействовать с несколькими серверами
- основная часть клиентского программного обеспечения применяется в виде обычных прикладных программ – не требуются привилегии
- большая часть клиентского обеспечения не заботится о правилах защиты, поскольку в этом вопросе оно полагается на ОС

# Определение местонахождения сервера

*Определение местонахождения сервера – первейшая и одна из самых выжных задач клиентского программного обеспечения!!!*

# Определение местонахождения сервера

*Определение местонахождения сервера – первейшая и одна из самых выжных задач клиентского программного обеспечения!!!*

- доменное имя или IP-адрес сервера могут быть заданы в виде константы во время трансляции клиентской программы



# Определение местонахождения сервера

*Определение местонахождения сервера – первейшая и одна из самых важных задач клиентского программного обеспечения!!!*

- доменное имя или IP-адрес сервера могут быть заданы в виде константы во время трансляции клиентской программы
- клиентская программа может требовать у пользователя указывать имя сервера при ее вызове

# Определение местонахождения сервера

*Определение местонахождения сервера – первейшая и одна из самых выжных задач клиентского программного обеспечения!!!*

- доменное имя или IP-адрес сервера могут быть заданы в виде константы во время трансляции клиентской программы
- клиентская программа может требовать у пользователя указывать имя сервера при ее вызове
- информация о местонахождении сервера предоставляется из постоянного хранилища данных

# Определение местонахождения сервера

*Определение местонахождения сервера – первейшая и одна из самых выжных задач клиентского программного обеспечения!!!*

- доменное имя или IP-адрес сервера могут быть заданы в виде константы во время трансляции клиентской программы
- клиентская программа может требовать у пользователя указывать имя сервера при ее вызове
- информация о местонахождении сервера предоставляется из постоянного хранилища данных
- для поиска сервера используется отдельный протокол

# Определение местонахождения сервера

Еще один аспект поиска сервера следует из того, какой тип сервиса предоставляется сервером.

От этого зависит нужно ли использовать какой-либо определенный сервер или же можно использовать первый ответивший.

В качестве альтернативы в ответе сервера может также содержаться список других серверов.

# Алгоритм клиентов с установлением логического соединения

Задача построения клиента с использованием протокола ТСР является самой простой из всех задач сетевого программирования.

Алгоритм клиента ТСР:

- 1 Найти IP-адрес и номер порта протокола сервера, с которым необходимо установить связь.
- 2 Распределить сокет
- 3 Указать, что для соединения нужен произвольный, неиспользуемый порт протокола на локальном компьютере и позволить ПО ТСР выбрать такой порт
- 4 Подключить сокет к серверу
- 5 Выполнить обмен данными с сервером по протоколу прикладного уровня
- 6 Закрывать соединение

# Анализ параметра адреса

Хотя каждая клиентская программа может указывать адрес и порт по-своему, существует общепринятый синтаксис этих параметров:

- сначала имя, а вторым параметром порт  
`telnet server port`
- имя и порт рассматриваются как один параметр, разделенный двоеточием  
`lftp -e ls ftp.mgts.by:21`

# Пример: функция для создания сокета

```
1 int mksock( char *host, char * service, char * proto, struct sockaddr_in *sin)
2 {
3     struct hostent *hptr;
4     struct servent *sptr;
5     struct protoent *pptr;
6     int sd=0, type;
7
8     memset( sin, 0, sizeof( *sin));
9     sin->sin_family = AF_INET;
10
11     if( hptr = gethostbyname( host))
12         memcpy( & sin->sin_addr, hptr->h_addr, hptr->h_length);
13     else return -1;
14
15     if ( ! ( pptr = getprotobyname( proto)) ) return -1;
16
17     if( sptr = getservbyname( service, proto))
18         sin->sin_port = sptr->s_port;
19     else
20         if( (sin->sin_port = htons(( unsigned short) atoi (service))) == 0) return -1;
21
22     if ( strcmp( proto, "udp") == 0)
23         type = SOCK_DGRAM;
24     else
25         type = SOCK_STREAM;
26
27     if ( (sd = socket( PF_INET, type, pptr->p_proto)) < 0){
28         perror( "Ошибка при распределении сокета");
29         return -1;
30     }
31     return sd;
32 }
```

# Пример: создание сокета

```
1
2 #include <stdio.h>
3 #include <errno.h>
4 #include <string.h>
5 #include <netdb.h>
6 #include <sys/types.h>
7 #include <sys/socket.h>
8 #include <arpa/inet.h>
9
10 int mksock( char *host, char * service, char * proto, struct sockaddr_in *sin);
11
12 main( void)
13 {
14
15     char * host = "pop.gmail.com";
16     char * service = "imap";
17     char * proto = "tcp";
18     struct sockaddr_in sin;
19     int sd;
20
21     if ( sd = mksock( host, service, proto, &sin) == -1) {
22         printf( "Ошибка при создании сокета\n");
23         return 1;
24     }
25
26     printf( "Адрес сервера %s = %s\n", host, inet_ntoa( sin.sin_addr));
27     printf( "Адрес порта %s = %X\n", service, sin.sin_port);
28
29     return 0;
30 }
```



# Пример: создание сокета результат работы программы

Адрес сервера pop.gmail.com = 74.125.39.108

Адрес порта imap = 143

# Подключение сокета к TCP серверу

Производится с помощью вызова `connect`. Этот вызов выполняет 4 задачи:

- 1 проверяет, является ли сокет действительным и не был ли он подключен
- 2 заполняет поле адреса конечной точки в дескрипторе сокета (из 2-го параметра)
- 3 выбирает локальный адрес в дескрипторе сокета, если он еще не задан
- 4 иницирует соединение TCP и возвращает результат в вызывающую программу

# Взаимодействие с сервером при использовании ТСР

Если соединение установлено, то обычно прикладной протокол определяет взаимодействие по принципу запрос-ответ. Для этого используются вызовы `send` и `recv` (`write` и `read`).

Поскольку ПО ТСР не учитывает границ между записями, то в любой программе, принимающей данные из соединения ТСР, необходимо предусмотреть возможность получения данных в виде фрагментов, составляющих лишь несколько байтов.

```
1 char *req, *buf;
2
3 send( sd, req, strlen(req), 0)
4
5 while( n = recv( sd, buf, buflen, 0) > 0 ){
6     buf += n;
7     buflen -= n;
8 }
```

# Заккрытие соединения TCP

Для корректного завершения соединения и освобождения сокета вызывается функция `close`. Однако часто возникает ситуация, когда сервер не знает будут ли еще приходить запросы от клиента и наоборот, когда клиент не знает все ли данные выдал сервер.

Механизм частичного закрытия `shutdown` позволяет устранить неопределенность в работе прикладных протоколов, которые передают произвольный объем информации в ответ на запрос. В таких случаях клиент выполняет операцию частичного закрытия после передачи последнего запроса; затем сервер закрывает соединение после передачи последнего ответа.

# Алгоритм клиентов без установления логического соединения

В этом алгоритме проблема надежности игнорируется.

- 1 Найти IP-адрес и номер порта протокола сервера, с которым необходимо установить связь.
- 2 Распределить сокет
- 3 Указать, что для соединения нужен произвольный, неиспользуемый порт протокола на локальном компьютере и позволить ПО UDP выбрать такой порт
- 4 Указать сервер, на который должны передаваться сообщения.
- 5 Выполнить обмен данными с сервером по протоколу прикладного уровня
- 6 Закрывать сокет

# Подключенный и неподключенный режимы

В клиентском приложении сокет UDP может использоваться в одном из 2-х основных режимов: подключенном и неподключенном.

Подключенные сокеты удобно применять для взаимодействия с конкретным сервером, а неподключенные – если адресат может меняться, в этом случае нужно для каждого сообщения указывать адрес сервера.

Если используется подключение, то вызов функции `connect` только записывает информацию об удаленной точке в дескриптор сокета. Даже если вызов успешен, это еще не означает, что адрес удаленной точки действителен или что сервер является достижимым.

# Заккрытие соединения UDP

Применение функции `close` приводит к тому, что ПО UDP будет отбрасывать все дальнейшие пакеты. При этом удаленная сторона не информируется о закрытии соединения, поэтому приложения нужно проектировать таким образом, чтобы удаленный участник знал, как долго сокет должен оставаться доступным до его закрытия.

Вызов функции `shutdown` не отправляет каких-либо сообщений удаленной стороне и сокет просто помечается, как неприменимый для передачи данных в указанном направлении.

# Ненадежность UDP

Клиентское ПО, в котором используется протокол UDP, должно обеспечивать надежность с помощью различных методов, таких как контроль за последовательностью поступления пакетов, подтверждения, тайм-ауты и повторная передача. Проектирование правильных, надежных и эффективных протоколов для больших сетей требуют большого опыта.



Спасибо за внимание!  
Вопросы?