## ЛАБОРАТОРНАЯ РАБОТА №3

# Создание параллельного многопоточного сервера с установлением логического соединения TCP

**Цель работы:** изучить методы создания серверных приложений на осно- ве установления логического соединения *TCP*, используя алгоритм многопо- точной обработки запросов.

В предыдущих лабораторных работах были показаны примеры реализа- ции последовательных серверов как с установлением, так и без установления логического соединения. В данной лабораторной работе рассматривается при- мер параллельного сервера с установлением логического соединения.

Мнoгoпoтoчнoсть – этo специaлизирoвaннaя фoрмa мнoгoзaдaчнoсти (*multitasking*). Что касается многозадачности, то выделяют двa типa мнoгoзaдaчнoсти: oснoвaнную нa прoцессaх (*process-based*) и oснoвaнную нa пoтoкaх (*thread-based*). Пo сути, прoцесс (*process*) – этo отдельно выпoлняющaяся прoгрaммa. Тaким oбрaзoм, oснoвaннaя нa прoцессaх мнoгoзaдaчнoсть – средствo, пoзвoляющее вaшему кoмпьютеру выпoлнять нескoлькo прoгрaмм oднoвременнo.

Отличия oснoвaннoй нa прoцессaх и мнoгoпoтoчнoй мнoгoзaдaчнoсти мoжнo сфoрмулирoвaть следующим oбрaзoм: первaя пoддерживaет oднoвременнoе выпoлнение нескoльких прoгрaмм, a втoрaя имеет делo с oднoвременным выпoлнением рaзных фрaгментoв oднoй и тoй же прoгрaммы.

С помощью процессов можно организовать параллельное выполнение программ. Для этого процессы клонируются с помощью вызовов функции *fork()* или функции *exec()*, а затем между ними (процессами) организуется взаи- модействие средствами *IPC*. Это довольно дорогостоящий с точки зрения ре- сурсов процесс.

С другой стороны, для организации параллельного выполнения и взаимо- действия части программы можно использовать механизм многопоточности. Основной единицей здесь является поток. Рассмотрим этот механизм подроб- нее.

### Потоки

Последовательная реализация сервера, о которой речь шла в предыдущих лабораторных работах, может оказаться неподходящей, поскольку клиенты бу- дут вынуждены ждать завершения обработки всех предыдущих запросов на ус- тановление соединения. Если клиент решит передать большие объемы данных (например, несколько мегабайт), последовательный сервер отложит обслужи- вание всех других клиентов до тех пор, пока не выполнит этот запрос.

Параллельная реализация сервера дает возможность обойтись без продол- жительных задержек, так как не позволяет одному клиенту захватить все ресурсы. Вместо этого параллельный сервер поддерживает обмен данными сразу с не- сколькими клиентами для того, чтобы их запросы обрабатывались одновременно. Поэтому, с точки зрения клиента, параллельный сервер обеспечивает лучшее на- блюдаемое время отклика по сравнению с последовательным сервером.

Распараллеливание обработки на сервере достигается созданием отдель- ного потока для обработки запросов одного клиента или отдельного однопото- кового процесса для обработки запросов одного клиента.

**Пoтoк** (*thread*) – этo упрaвляемaя единицa испoлняемoгo кoдa. У всех прoцессoв oбязaтельнo есть oдин пoтoк, нo их мoжет быть и бoльше. Этo oзнaчaет, чтo в oднoй прoгрaмме мoгут выпoлняться нескoлькo зaдaч oднoвременнo. Примером может служить текстoвый редaктoр, в котором воз- можны одновременные фoрмaтирoвaние текста и печaть, осуществляемые в oтдельных пoтoкaх.

### Преимущества многопоточности

Если операционная система поддерживает концепции потоков в рамках одного процесса, она называется многопоточной. Многопоточные приложения имеют ряд преимуществ:

а) улучшенная реакция приложения – любая программа, содержащая много не зависящих друг от друга действий, может быть перепроектирована так, чтобы каждое действие выполнялось в отдельном потоке. Например, поль- зователь многопоточного интерфейса не должен ждать завершения одной зада- чи, чтобы начать выполнение другой;

б) более эффективное использование мультипроцессирования – как пра- вило, приложения, реализующие параллелизм через потоки, не должны учиты- вать число доступных процессоров. Производительность приложения равно- мерно увеличивается при наличии дополнительных процессоров. Численные алгоритмы и приложения с высокой степенью параллелизма, например пере- множение матриц, могут выполняться намного быстрее;

в) улучшенная структура программы – некоторые программы более эф- фективно представляются в виде нескольких независимых или полуавтономных единиц, чем в виде единой монолитной программы. Многопоточные програм- мы легче адаптировать к изменениям требований пользователя;

г) эффективное использование ресурсов системы – программы, исполь- зующие два или более процессов, которые имеют доступ к общим данным че- рез разделяемую память, содержат более одного потока управления. При этом каждый процесс имеет полное адресное пространство и состояние в операци- онной системе. Стоимость создания и поддержания большого количества слу- жебной информации делает каждый процесс более затратным, чем поток. Кро- ме того, разделение работы между процессами может потребовать от програм- миста значительных усилий, чтобы обеспечить связь между потоками в различ- ных процессах или синхронизировать их действия.

### Преимущества и недостатки многопотоковых процессов

Многопотоковые процессы обладают двумя основными преимуществами по сравнению с однопотоковыми процессами: более высокая эффективность и разделяемая память. Повышение эффективности связано с уменьшением из- держек на переключение контекста. Переключателем контекста называются действия, выполняемые операционной системой при передаче ресурсов про- цессора от одного потока выполнения к другому. При переключении с одного потока на другой операционная система должна сохранить в памяти состояние предыдущего потока (например, значение регистров) и восстановить состояние следующего потока. Потоки в одном и том же процессе разделяют значитель- ную часть информации о состоянии процесса, поэтому операционной системе приходится выполнять меньший объем работы по сохранению и восстановле- нию состояния. Вследствие этого переключение с одного потока на другой в одном и том же процессе происходит быстрее по сравнению с переключением между двумя потоками в разных процессах. В частности, поскольку потоки од- ного и того же процесса разделяют адресное пространство памяти, то переклю- чение между потоками процесса означает, что операционная система не должна менять отображение виртуальной памяти на физическую.

Второе преимущество потоков, т.е. разделяемая память, вероятно, являет- ся для программистов еще более важным, чем повышение эффективности. По- токи упрощают разработку параллельных серверов, в которых все копии серве- ра должны взаимодействовать друг с другом или обращаться к разделяемым элементам данных. Кроме того, потоки упрощают разработку систем контроля и управления. В частности, поскольку ведомые потоки в сервере совместно ис- пользуют память, они могут записывать в глобальную память статистическую информацию, что позволяет контролирующему потоку формировать отчеты об активности ведомых потоков сервера для системного администратора.

Хотя потоки имеют свои преимущества над однопотоковыми процессами, они не лишены также определенных недостатков. Один из наиболее важных недостатков связан с тем, что потоки не только разделяют память, но и имеют общее состояние процесса, поэтому действия, выполненные одним потоком, могут повлиять на другие потоки в том же процессе. Например, если два потока попытаются одновременно обратиться к одной и той же переменной, они могут помешать друг другу.

*API*-интерфейс потоков предоставляет функции, которые могут использо- ваться потоками для координации работы. Однако многие библиотечные функ- ции, возвращающие указатели на статические элементы данных, не являются безопасными с точки зрения потоков, а это означает, что результаты вызова та- ких функций могут оказаться непредсказуемыми.

Еще один недостаток потоков (и отличие однопотокового процессора от многопотокового) связан с отсутствием надежности. Если одна из параллельно работающих копий однопотокового сервера вызовет серьезную ошибку (на- пример, в ней будет выполнена ссылка на недопустимую область памяти), то

операционная система завершит только тот процесс, который вызвал ошибку. С другой стороны, если серьезная ошибка будет вызвана одним из потоков мно- гопотокового сервера, то операционная система завершит весь процесс.

### Алгоритм работы параллельного (многопоточного) сервера с установлением логического соединения

Приведем обобщенный алгоритм работы параллельного сервера с уста- новлением логического соединения:

1. Ведущий поток. Создать сокет и выполнить его привязку к локальному адресу. Оставить сокет неподключенным.
2. Ведущий поток. Перевести сокет в пассивный режим, подготовив его для использования сервером.
3. Ведущий поток. Вызывать в цикле функцию *accept* для получения оче- редного запроса от клиента и создавать новый ведомый поток или процесс для формирования ответа.
4. Ведомый поток. Работа потока начинается с получения доступа к со- единению, полученному от ведущего потока (т.е. к сокету соединения).
5. Ведомый поток. Выполнять обмен данными с клиентом через соедине- ние: принимать запрос (запросы) и передавать ответ (ответы).
6. Ведомый поток. Закрыть соединение и завершить работу. Ведомый по- ток завершает свою работу после обработки всех запросов от одного клиента.

Сервер функционирует неопределенно долгое время, ожидая поступления новых запросов на установление соединения от клиентов. Его ведущий поток при подключении клиента создает новый ведомый поток для обработки запро- сов каждого нового соединения и предоставляет каждому ведомому потоку возможность взять на себя весь обмен данными с клиентом. На рисунке 4 при- ведена схема организации потоков параллельного сервера с установлением ло- гического соединения.

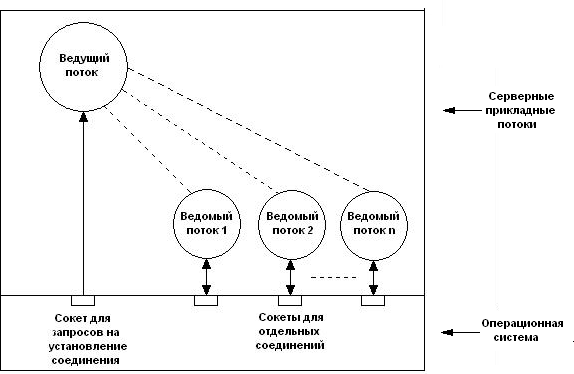


Рисунок 4 – Схема организации потоков параллельного сервера с установлением логического соединения

### Методические указания по созданию параллельного многопоточного сервера с установлением логического соединения TCP

Осуществить взаимодействие клиента и сервера на основе протокола *TCP/IP*. Реализовать параллельное соединение с использованием многопоточ- ности. Функциональные возможности клиента реализовать следующим обра- зом: клиент вводит с клавиатуры строку символов и посылает ее серверу. При- знак окончания ввода строки – нажатие клавиши «Ввод». Функциональные возможности сервера реализовать следующим образом: сервер, получив эту строку, должен определить длину введенной строки, и, если эта длина кратна четырем, то первая часть строки меняется местами со второй. Результаты пре- образований возвращаются назад клиенту.

Серверная часть #include<stdio.h> #include<iostream.h> #include <winsock2.h>

Функция *CreateThread()* создает поток, который выполняется в пределах адресного пространства вызова процесса и имеет следующий прототип:

HANDLE Create Thread (

LPSECURITY\_ATTRIBUTES lpThreadAttributes, // указатель на атрибуты безопасности

DWORD dwStackSize, // размер стека начального потока LPTHREAD\_START\_ROUTINE lpStartAddress, // указатель на функ-

цию потока

LPVOID lpParameter, // аргумент для нового потока DWORD dwCreationFlags, // создание флагов

LPDWORD lpThreadId // указатель на ID поток для его получения

);

Рассмотрим каждый параметр подробнее:

*lpThreadAttributes* – представляет собой указатель на структуру *SECURITY\_ATTRIBUTES*, который определяет, может ли дескриптор потока быть унаследован дочерними процессами. Если *lpThreadAttributes* принимает значение *NULL*, дескриптор потока не может быть унаследован.

Для *MS Windows NT*: член структуры *lpSecurityDescriptor* определяет де- скриптор безопасности для нового потока. Если *lpThreadAttributes* принимает значение *NULL*, то поток получает дескриптор безопасности по умолчанию.

*dwStackSize* – эта величина определяет начальный размер стека в байтах. Система округляет это значение до ближайшей страницы. Если это значение равно нулю или меньше размера стека по умолчанию, то используется тот же размер, что и при вызове потока. Стек освобожден в том случае, когда поток завершается.

*lpStartAddress* – указатель на определенную прикладную функцию типа *LPTHREAD\_START\_ROUTINE* для выполнения ее потоком и представления на- чального адреса потока.

*lpParameter* – определяет единственную 32-битовую величину парамет- ра, переданную в поток.

*dwCreationFlags* – используется для определения дополнительных фла- гов, которые управляют созданием потока. Если флаг *CREATE\_SUSPENDED* определен, поток создан в приостановленном состоянии и не будет работать, пока функция *ResumeThread()* не будет вызвана. Если эта нулевая величина, то поток выполняется немедленно после его создания. В то же время никакие дру- гие величины не предусмотрены.

*lpThreadId* – указатель на 32-битовую переменную, которая получает идентификатор потока.

Для *MS Windows NT***:** Если этот параметр принимает значение *NULL*, то идентификатор потока не возвращается.

Для *MS Windows 95* и *MS Windows 98*: Этот параметр может не принимать значение *NULL*.

Если функция *CreateThread()* успешно выполняется, то возвращаемое значение есть указатель на новый поток. В случае невыполнения функции воз- вращаемое значение принимает значение *NULL*. Для получения большей ин- формации об ошибке обратимся к функции *GetLastError().*

Примечания –

* Новый поток управления создается макросом *THREAD\_ALL\_ACCESS* для нового потока. Если дескриптор безопасности не предусмотрен, то управ- ление может быть использовано в любой функции, которая требует объектного управления потоком. Когда дескриптор безопасности предусмотрен, то кон- троль доступа выполняется во всех последующих использованиях дескриптора прежде, чем доступ будет предоставлен. Если контроль доступа запрещает дос- туп, запрашиваемый процесс не может использовать дескриптор для получения доступа к потоку.
* Выполнение потока начинается в функции, определенной параметром *lpStartAddress*. Если эта функция возвращает значение типа *DWORD*, то оно ис- пользуется для завершения потока неявным вызовом функции *ExitThread()*, ко- торая будет описана ниже. Используйте функцию *GetExitCodeThread()*, чтобы получать возвращаемое значение потока.
* Функция *CreateThread()* выполняется, даже если указатель *lpStartAd- dress* указывает на данные, код, или недоступен. Если начальный адрес недей- ствителен во время работы потока, срабатывает исключение и поток завершает- ся. Завершение потока из-за неправильного начального адреса интерпретирует- ся как аварийный выход процесса потока.

В нашей программе основной алгоритм решения задачи находится в функции *ThreadFunc()*, которую определим следующим образом:

DWORD WINAPI ThreadFunc(LPVOID client\_socket){ SOCKET s2=((SOCKET \*) client\_socket)[0]; char buf[100];

char buf1[100]; while(recv(s2,buf,sizeof(buf),0)){

int k, j=0; k=strlen(buf)-1;

if(k%4==0){

for(int i=k/2; i<k; i++){ buf1[i]=buf[j];

j++;

}

for(i=0; i<k/2; i++){ buf1[i]=buf[j]; j++;

}

buf1[k]='\0'; strcpy(buf,buf1);

}

cout<<buf<<endl; send(s2,buf,100,0);

}

closesocket(s2); return 0;

}

При вызове функции *ThreadFunc()* в основной программе (main-функции) передается дескриптор сокета.

int numcl=0; void print(){

if (numcl) printf("%d client connected\n",numcl); else printf("No clients connected\n");

}

void main()

{

WORD wVersionRequested; WSADATA wsaData;

int err;

wVersionRequested = MAKEWORD( 2, 2 );

err = WSAStartup( wVersionRequested, &wsaData ); if ( err != 0 ){return;}

SOCKET s=socket(AF\_INET,SOCK\_STREAM,0);

sockaddr\_in local\_addr; local\_addr.sin\_family=AF\_INET; local\_addr.sin\_port=htons(1280); local\_addr.sin\_addr.s\_addr=0;

bind(s,(sockaddr \*) &local\_addr,sizeof(local\_addr)); int c=listen(s,5);

cout<<"Server receive ready"<<endl; cout<<endl;

// извлекаем сообщение из очереди

SOCKET client\_socket; // сокет для клиента sockaddr\_in client\_addr; // адрес клиента (заполняется

//системой)

int client\_addr\_size=sizeof(client\_addr);

// цикл извлечения запросов на подключение из очереди while((client\_socket=accept(s,(sockaddr \*)&client\_addr,

&client\_addr\_size))){

numcl++; print();

// вызов нового потока для обслуживания клиента DWORD thID;// thID идентификатор типа DWORD CreateThread(NULL,NULL,ThreadFunc, &client\_socket,NULL,&thID)

}

}

По окончании работы функция *CreateThread()* закрывает поток, инициа- лизирует используемые указатели значением *NULL*. Существует специальная функция *Exit Thread()*, выполняющая аналогичные действия. Ее прототип:

VOID ExitThread( DWORD dwExitCode );

Параметр *dwExitCode* определяет выходной код для вызова потока. Дан- ная функция не возвращает никакого значения.

Примечания –

* Когда функция *ExitThread()* явно вызвана, текущий стек потока осво- божден и поток завершается.
* Если поток является последним в процессе, когда эта функция вызва- на, то процесс потока также завершается.
* Завершение потока не обязательно удаляет его объект из операционной системы. Объект потока удален, когда закрывается последний дескриптор по- тока.

Клиентская часть

Во многом дублирует клиент-приложения предыдущих лабораторных ра- бот. Согласно требованиям условия задачи клиентская часть имеет следующий вид:

#include<stdio.h> #include<iostream.h> #include<winsock2.h>

void main()

{

WORD wVersionRequested; WSADATA wsaData;

int err;

wVersionRequested = MAKEWORD( 2, 2 ); err=WSAStartup( wVersionRequested, &wsaData ); if ( err != 0 ){return;}

while (true){

SOCKET s=socket(AF\_INET,SOCK\_STREAM,0);

// указание адреса и порта сервера sockaddr\_in dest\_addr; dest\_addr.sin\_family=AF\_INET; dest\_addr.sin\_port=htons(1280);

dest\_addr.sin\_addr.s\_addr=inet\_addr("127.0.0.1"); connect(s,(sockaddr \*)&dest\_addr,sizeof(dest\_addr));

char buf[100];

cout<<"Enter the string:"<<endl; fgets(buf,sizeof(buf),stdin); send(s,buf,100,0);

if (recv(s,buf,sizeof(buf),0)!=0){ cout<<"Poluchenaya stroka:"<<endl<<buf<<endl;

}

closesocket(s);

}

WSACleanup();

}

### Индивидуальные задания

Разработать приложение, реализующее архитектуру «клиент-сервер». Для этого необходимо реализовать параллельный многопоточный сервер с установ- лением логического соединения (TCP). Логику взаимодействия клиента и сер- вера реализовать так, как указано в варианте индивидуального задания. Преду- смотреть возможность просмотра, добавления, редактирования, удаления ин- формации клиентом на сервере.

1. На сервере хранится список студентов. Каждая запись списка содержит следующую информацию о студенте:
   * ФИО студента;
   * номер группы;
   * размер стипендии;
   * оценки по N предметам.

Таких записей должно быть не менее пяти.

Клиент вводит с клавиатуры букву алфавита, по которой он хотел бы посмотреть информацию о студентах, и посылает ее на сервер. Назад он получает список только тех студентов, фамилии которых начинаются на эту букву.

1. На сервере хранится список студентов. Каждая запись списка содержит следующую информацию о студенте:
   * ФИО студента;
   * номер группы;
   * размер стипендии;
   * оценки по N предметам.

Таких записей должно быть не менее 5-ти.

Клиент посылает на сервер средний балл студента, по которому он хочет получить информацию о студентах. Назад он получает список только тех студентов, средний балл которых больше заданного.

1. На сервере хранится список о студентах. Каждая запись списка содер- жит следующую информацию о студенте:

* ФИО студента;
* номер группы;
* размер стипендии;
* оценки по N предметам.

Таких записей должно быть не менее 5-ти.

По запросу клиента он получает от сервера список только тех студентов, которые не имеют оценки 3.

1. На сервере хранится список сотрудников фирмы. Каждая запись списка содержит следующую информацию о сотрудниках:
   * ФИО сотрудника;
   * табельный номер;
   * количество отработанных часов за месяц;
   * почасовой тариф.

Таких записей должно быть не менее пяти.

Клиент посылает на сервер величину заработной платы, по которой он хочет получить информацию о сотрудниках. Назад он получает список только тех сотрудников, заработная плата которых меньше указанной.

1. На сервере хранится информация (список) о комплектующих деталях. Каждая запись списка содержит следующую информацию о комплектующем:
   * завод-поставщик;
   * стоимость;
   * дата поставки;

Таких записей должно быть не менее семи.

Клиент посылает на сервер дату поставки. Назад он получает список ком- плектующих, поставленных именно на эту дату, и стоимость каждого возвра- щаемого комплектующего должна превосходить минимальную во всем списке.

1. На сервере хранится список автобусных рейсов. Каждая запись списка содержит следующую информацию о рейсе:
   * номер рейса;
   * тип автобуса;
   * цена билета;
   * пункт назначения.

Таких записей должно быть не менее пяти.

Клиент посылает на сервер пункт назначения. Назад он получает список рейсов, позволяющих добраться до заданного пункта.

1. На сервере хранится список игроков. Каждая запись списка содержит следующую информацию об игроках:
   * ФИО игрока;
   * игровой номер;
   * возраст;
   * рост;
   * вес.

Таких записей должно быть не менее пяти.

Клиент посылает на сервер запрос и получает информацию о самом мо- лодом игроке.

1. На сервере хранится список разговоров на междугородной АТС. Каж- дая запись списка содержит следующую информацию о разговорах:
   * дату разговора;
   * код и название города;
   * продолжительность разговора;
   * тариф;
   * номер телефона в этом городе;
   * номер телефона абонента.

Таких записей должно быть не менее пяти.

Клиент посылает на сервер название города. Назад он получает суммар- ное время разговора с указанным городом.

1. На сервере хранится список товаров, имеющихся на складе. Каждая за- пись списка содержит следующую информацию о товарах:
   * страна-изготовитель;
   * фирма-изготовитель;
   * наименование товара;
   * количество единиц товара.

Таких записей должно быть не менее пяти.

Клиент посылает на сервер название страны-изготовителя. Назад он по- лучает товары и их данные для указанной страны.

1. На сервере хранится список книг, хранящихся в библиотеке. Каж- дая запись списка содержит следующую информацию о книгах:
   * регистрационный номер книги,
   * автор;
   * название;
   * год издания;
   * издательство;
   * количество страниц.

Таких записей должно быть не менее пяти.

Клиент посылает на сервер фамилию интересующего его автора. Назад он получает список книг указанного автора.

1. На сервере хранится список деталей. Каждая запись списка содер- жит следующую информацию о деталях:
   * наименование детали;
   * количество деталей;
   * номер цеха, где они изготовлены.

Таких записей должно быть не менее семи.

Клиент посылает на сервер наименование детали. Назад он получает об- щее количество изделий указанного наименования.

1. На сервере хранится список товаров, имеющихся на складе. Каждая запись списка содержит следующую информацию о товарах:
   * страна-изготовитель;
   * фирма-изготовитель;
   * наименование товара;
   * количество единиц товара.

Таких записей должно быть не менее пяти.

Клиент посылает на сервер наименование товара. Назад он получает ко- личество единиц этого товара.

1. На сервере хранится каталог туристических предложений. Каждая за- пись каталога содержит:
   * название тура;
   * стоимость;
   * продолжительность;
   * вид транспорта.

Таких записей должно быть не менее пяти.

Клиент посылает на сервер предполагаемую стоимость тура. Назад он по- лучает названия тех туров, стоимость которых не выше посланной.

1. На сервере хранится список преподавателей. Каждая запись списка содержит следующие данные:
   * ФИО;
   * ученая степень;
   * стаж работы;
   * предмет.

Клиент посылает на сервер название предмета. Назад он получает ФИО тех преподавателей этого предмета, стаж работы которых не менее 5 лет.

1. На сервере хранится информация об участниках соревнований по спортивным танцам. Она включает следующие данные:
   * название танцевальной пары;
   * город;
   * оценка жюри;
   * оценка зрителей;
   * возрастная группа.

Клиент посылает на сервер возраст участников. Назад он получает назва- ние той танцевальной пары, суммарная оценка которой в данной возрастной группе максимальна.

### Контрольные вопросы

1. Что такое параллельное соединение? Особенности параллельного со- единения.

Параллельно соединение - когда между одним клиентом и одним сервером (или между клиентом и набором ресурсов на сервере) одновременно открыто **несколько независимых TCP-соединений**. Каждый такой канал — отдельный TCP-сокет, со своими номером порта, состоянием, окнами управления потоком и управления перегрузкой.

Особенности параллельного соединения – повышение пропускной способности, уменьшение задержки, эффективность при передаче больших объемов данных.

1. В чём заключается отличие параллельного соединения от последова- тельного?

Кол-во каналов: в последовательном соединении канал 1, а в параллельном – N.

В параллельном соединении каждой соединение имеет свой разамер передаваемых значений, а в последовательном он всегда одинаковый.(в параллельном больше пропускная способность)

Ошибки: в параллельном соединении ошибки не обязательно остановит выполнение других запросов, а в последовательном может вызвать задержку и остановку.

1. Назовите преимущества многопотоковых процессов по сравнению с однопотоковыми процессами.

Улучшение структуры приложения

Улучшение реакции приложение

Более эффективное использование мультипроцессирования

Эффективное использование ресурсов системы

1. С чем связано повышение эффективности многопотоковых процессов?

- нет простаивание при выполнении процессов

- обработка соединений параллельно на разных ядрах ускоряет работу

1. Назовите недостатки многопотокового процесса по сравнению с однопотоковым.

Каждый поток потребляет память

Действие, выполняемое потоком может помешать других потокам, выполянющихся в этот момент. Например, при обращении к одноц и тоц же переменной эти пототки могтут помешать друг другу

Многие библиотечные функции возвращают указатели на статические элементы данных, что небезопасно с точки зрения потоков.

Если какой-то из потоков вызовет серьезную ошибку, то завершатся все потоки

1. Какие функции служат для создания потоков?