Оқушылардың ойлау қабілетін жетілдіруде геометрияның берер үлесі мол. Оқушыларды дұрыс ойлауға, ойдан ой туындатуға, ой тізбегін құруға және соның нәтижесінде нақты да дұрыс тұжырымға келуге, қорытынды жасауға үйретеді.

Мектеп геометриясын оқытудың басты мақсаттарының бірі – оның теориялық негіздерін білу және оларды практикада қолдану дағдыларын меңгеру. Сонымен қатар логикалық ойлауын, дәлелдеу қабілетін, талқылауларды себептеу, ойды дәл және анық тұжырымдай білу мәселелері де маңызды міндеттер болып табылады. Геометрияның мектептік курсын оқытуда оқушылардың кеңістік түсінігі мен елестете білуін дамыту, қоршаған ортаны геометриялық тұрғыдан «көре білу» және т.б. мәселелері шешіледі.

Қолданылған әдебиеттер:

1. Гусев А.В. Преподавание геометрии в 6-8 классах - «просвещение» 1979
2. Геометрия: Учеб.для 7 кл.сред.шк./ред Колмогоров А.Н., 1977.-158с.с.
3. Кутузов Б.В. Геометрия: Пособия для учительских и пед.инст-тов/Кутузов Б.В., 1950.-283с.с.
4. Колмогоров А.Н. Геометрия:Учеб пособие для 6-8кл.сред.шк./ Колмогоров А.Н., Семенович А.Ф., Черкасов Р.С., 1982.-383с.
5. ШыныбековӘ.Н., Геометрия: жалпы білім беретін мектептің 7-сыныбына арналған оқулық. 4-басылымы. – Алматы: Атамұра, 2012 – 96 бет.

Власенко В., Карагандинский государственный университет имени академика Е.А. Букетова, факультет математики и информационных технологий, студентка гр. Инф-408
*(Научный руководитель – старший преподаватель **Допира Р.И.**)*

КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ НА C#

В настоящее время компьютерные сети используются повсеместно, позволяя пользователям объединять вычислительные и дорогостоящие ресурсы для совместного решения различных задач. Например, очень часто используются сетевые принтеры.

Работа в рамках компьютерной сети потребовала создания ряда протоколов для организации взаимодействия между компьютерами в рамках сети. И одним из наиболее популярных в настоящее время стеков (наборов) таких протоколов является стек протоколов TCP/IP.

Для построения программ, работающих с сетью и предоставляющих конечным пользователям какие-либо сервисы, используется архитектура «клиент-сервер». Эта популярная парадигма основана на понятиях «сервера» (пассивный компонент, обслуживающий запросы от клиентов) и «клиента» (активный компонент, обращающийся с каким-либо запросом к серверу). Средства .NET и библиотеки языка C# позволяют организовать разработку клиент-серверных приложений на базе сетей TCP/IP. В частности, для реализации сокетов могут быть использованы классы в пространстве имен System.Net.Sockets [1].

Протоколы транспортного уровня, следующие в иерархии за IP, используются для передачи данных между прикладными процессами, реализующимися в сетевых узлах. Пакет данных, поступивший от одного компьютера другому через Интернет, должен быть передан процессу-обработчику, и именно по конкретному назначению. Транспортный уровень принимает на себя ответственность за это. На этом уровне два основных протокола – TCP и UDP.

TCP (Transmission Control Protocol) – транспортный протокол передачи данных в сетях TCP/IP, предварительно устанавливающий соединение с сетью. Обмен данными, ориентированный на соединения, может использовать надежную связь, для обеспечения которой протокол уровня 4 посылает подтверждения о получении данных и запрашивает повторную передачу, если данные не получены или искажены. Протокол TCP использует именно такую надежную связь. TCP используется в таких прикладных протоколах, как HTTP, FTP, SMTP и Telnet.

Протокол TCP требует, чтобы перед отправкой сообщения было открыто соединение. Серверное приложение должно выполнить так называемое пассивное открытие (passive open), чтобы создать соединение с известным номером порта, и, вместо того чтобы отправлять вызов в сеть, сервер переходит в ожидание поступления входящих запросов. Клиентское приложение должно выполнить активное открытие (active open), отправив серверному приложению синхронизирующий порядковый номер (SYN), идентифицирующий соединение. Клиентское приложение может использовать

динамический номер порта в качестве локального порта. Сервер должен отправить клиенту подтверждение (ACK) вместе с порядковым номером (SYN) сервера. В свою очередь клиент отвечает ACK, и соединение устанавливается. После этого может начаться процесс отправки и получения сообщений. При получении сообщения в ответ всегда отправляется сообщение ACK. Если до получения ACK отправителем истекает тайм-аут, сообщение помещается в очередь на повторную передачу.

TCP – это сложный, требующий больших затрат времени протокол, что объясняется его механизмом установления соединения, но он берет на себя заботу о гарантированной доставке пакетов, избавляя нас от необходимости включать эту функциональную возможность в прикладной протокол. Протокол TCP имеет встроенную возможность надежной доставки. Если сообщение не отправлено корректно, мы получим сообщение об ошибке. Протокол TCP определен в RFC 793.

В отличие от TCP UDP – очень быстрый протокол, поскольку в нем определен самый минимальный механизм, необходимый для передачи данных. Конечно, он имеет некоторые недостатки. Сообщения поступают в любом порядке, и то, которое отправлено первым, может быть получено последнимени Доставка сообщений UDP вовсе не гарантируется, сообщение может потеряться, и могут быть получены две копии одного и того же сообщения. Последний случай возникает, если для отправки сообщений в один адрес использовать два разных маршрута.

UDP (User Datagram Protocol)– транспортный протокол, передающий сообщения-датаграммы без необходимости установки соединения в IP-сети. UDP не требует открывать соединение, и данные могут быть отправлены сразу же, как только они подготовлены. UDP не отправляет подтверждающие сообщения, поэтому данные могут быть получены или потеряны. Если при использовании UDP требуется надежная передача данных, ее следует реализовать в протоколе более высокого уровня.

Так в чем же преимущества UDP, зачем может понадобиться такой ненадежный протокол? Чтобы понять причину использования UDP, нужно различать однонаправленную передачу, широковещательную передачу и групповую рассылку.

Однонаправленное (unicast) сообщение отправляется из одного узла только в один другой узел. Это также называется связью "точка-точка". Протокол TCP поддерживает лишь однонаправленную связь. Если серверу нужно с помощью TCP взаимодействовать с несколькими клиентами, каждый клиент должен установить соединение, поскольку сообщения могут отправляться только одиночным узлам.

Широковещательная передача (broadcast) означает, что сообщение отправляется всем узлам сети. Групповая рассылка (multicast) - это промежуточный механизм: сообщения отправляются выбранным группам узлов.

UDP может использоваться для однонаправленной связи, если требуется быстрая передача, например для доставки мультимедийных данных, но главные преимущества UDP касаются широковещательной передачи и групповой рассылки.

Обычно, когда мы отправляем широковещательные или групповые сообщения, не нужно получать подтверждения из каждого узла, поскольку тогда сервер будет наводнен подтверждениями, а загрузка сети возрастет слишком сильно. Примером широковещательной передачи является служба времени. Сервер времени отправляет широковещательное сообщение, содержащее текущее время, и любой хост, если пожелает, может синхронизировать свое время с временем из широковещательного сообщения.

UDP – это быстрый протокол, не гарантирующий доставки. Если требуется поддержание порядка сообщений и надежная доставка, нужно использовать TCP. UDP главным образом предназначен для широковещательной и групповой передачи. Протокол UDP определен в RFC 786.

Сокет – это один конец двустороннего канала связи между двумя программами, работающими в сети. Соединяя вместе два сокета, можно передавать данные между разными процессами (локальными или удаленными). Реализация сокетов обеспечивает инкапсуляцию протоколов сетевого и транспортного уровней.

Первоначально сокеты были разработаны для UNIX в Калифорнийском университете в Беркли. В UNIX обеспечивающий связь метод ввода-вывода следует алгоритму open/read/write/close. Прежде чем использовать ресурс, его нужно открыть, задав соответствующие разрешения и другие параметры. Как только ресурс открыт, из него можно считывать или в него записывать данные. После использования ресурса пользователь должен вызывать метод Close(), чтобы подать сигнал операционной системе о завершении его работы с этим ресурсом.

Когда в операционную систему UNIX были добавлены средства межпроцессного взаимодействия (Inter-Process Communication, IPC) и сетевого обмена, был заимствован привычный

шаблон ввода-вывода. Все ресурсы, открытые для связи, в UNIX и Windows идентифицируются дескрипторами. Эти дескрипторы, или описатели (handles), могут указывать на файл, память или какой-либо другой канал связи, а фактически указывают на внутреннюю структуру данных, используемую операционной системой. Сокет, будучи таким же ресурсом, тоже представляется дескриптором. Следовательно, для сокетов жизнь дескриптора можно разделить на три фазы: открыть (создать) сокет, получить из сокета или отправить сокету и в конце концов закрыть сокет [2].

Интерфейс IPC для взаимодействия между разными процессами построен поверх методов ввода-вывода. Они облегчают для сокетов отправку и получение данных. Каждый целевой объект задается адресом сокета, следовательно, этот адрес можно указать в клиенте, чтобы установить соединение с целью.

Обычно приложение клиент-сервер, использующее сокеты, состоит из двух разных приложений - клиента, инициирующего соединение с целью (сервером), и сервера, ожидающего соединения от клиента [3].

Например, на стороне клиента, приложение должно знать адрес цели и номер порта. Отправляя запрос на соединение, клиент пытается установить соединение с сервером. Если события развиваются удачно, при условии что сервер запущен прежде, чем клиент попытался с ним соединиться, сервер соглашается на соединение. Дав согласие, серверное приложение создает новый сокет для взаимодействия именно с установившим соединение клиентом (рисунок 1).

Рисунок 1. Взаимодействие сервера и клиента

При разработке клиент-серверного приложения на языке C# были решены следующие задачи пользовательского уровня:

1. Осуществить взаимодействие клиента и сервера на основе протокола TCP/IP. Реализовать параллельное соединение с использованием многопоточности.
2. Функциональные возможности клиента реализовать следующим образом: клиент вводит с клавиатуры строку символов и посылает ее серверу.
3. Функциональные возможности сервера реализовать следующим образом: сервер получив эту строку, выясняет, сколько символов входит в эту строку.
4. Количество вхождений символов в строку передать назад клиенту.

Для успешного запуска клиента сначала необходимо запустить сервер. Запуск сервера возможен как из командной строки, так и непосредственно при запуске файла клиент.exe. Запуск клиента производится из командной строки или с помощью файла сервер.exe.

Для отправки строки текста серверу для проверки необходимо в окне консоли напечатать эту строку и нажать клавишу «Enter». Сервер вернет клиенту результат проверки строки (будет выведен в окно консоли клиента). Ограничений по отправке символов серверу от клиента обнаружено не было. Завершение сеанса между клиентом и сервером осуществляется при помощи команды <TheEnd>.

В ходе выполнения работы было разработано клиент-серверное приложение с многопоточным сервером, позволяющее серверу проверять отправляемые клиентом строки на предмет количества символов в строке (рисунок 2). В ходе тестирования было установлено, что разработанное ПО работает корректно. Алгоритм обработки данных сервером успешно обрабатывает отправляемые клиентом данные.

Рисунок 2. Демонстрация работы приложения

Литература:

1. Электронный справочник MSDN: http://www.msdn.microsoft.com
2. Дуглас Камер, Дэвид Л. Стивенс Сети TCP/IP, том 3. Разработка приложений типа клиент/сервер для Linux/POSIX – Изд.:Вильямс, 2002.
3. Исаченко О.В. Программное обеспечение компьютерных сетей: Учебное пособие. – Изд.:М.: ИНФРА-М, 2012.

Вознюк С.С., Карагандинский государственный университет имени академика Е.А. Букетова, химический факультет, студент гр. Хе-42;
Ашеева А.А., Смайлова Г.Б., Карагандинский государственный университет имени академика Е.А. Букетова, химический факультет, магистранты гр. М(Хе)-22;
Рахимжанова Г.А., Карагандинский государственный университет имени академика Е.А. Букетова, химический факультет, магистрант гр. М(Хо)-22
*(Научный руководитель – к.х.н., доцент **Пустолайкина И.А.**)*

ИССЛЕДОВАНИЕ ПРОТОЛИТИЧЕСКИХ СВОЙСТВ НЕКОТОРЫХ СЕМИХИНОННЫХ РАДИКАЛОВ МЕТОДАМИ КВАНТОВОЙ ХИМИИ

Стабильные радикалы нашли в настоящее время широкое применение в различных областях химии, физики и биологии в качестве специфических спиновых зондов, меток, ловушек и т.д. Благодаря спектроскопии ЭПР высокого разрешения долгоживущие органические парамагнетики могут быть использованы при изучении кинетики быстропротекающих по времени химических процессов в растворах [1]. Стабильные радикалы дают ценную информацию о механизме реакций, протекающих между компонентами системы, которые либо не реагируют, либо слабо реагируют с