## Лабораторная работа №9 распределенный UDP сервер/ UDP клиент

Цель работы: написать приложения клиенты и серверы, имитирующие работу grid системы, обеспечить надежность доставки данных или пакетов, автоматическое обнаружение расчетчиков;

User Datagram Protocol

Транспортный протокол для передачи данных в сетях IP без установления соединения. Он является одним из самых простых протоколов транспортного уровня модели OSI.

В отличие от TCP, UDP не подтверждает доставку данных, не заботится о корректном порядке доставки и не делает повторов. Поэтому аббревиатуру UDP иногда расшифровывают как Unreliable Datagram Protocol (протокол ненадёжных датаграмм). Зато отсутствие соединения, дополнительного трафика и возможность широковещательных рассылок делают его удобным для применений, где малы потери, в массовых рассылках локальной подсети, в медиапротоколах и т.п.

Распределенный UDP сервер/ UDP клиент

Задача написать приложение раздающее задание, которое посылало бы данные на несколько приложений расчетчиков, затем все расчетчики производили бы какие-либо преобразования над данными и отсылали их обратно. В случае, если приложение раздающее задание не получило обработанные данные обратно, оно должно отправить начальные данные другому расчетчику.

Обеспечение надежности доставки

Для обеспечения надежности доставки можно использовать следующие методы: хэш-функции, кода Хэмминга, контрольные суммы. На примере хэш-функции:

Вычисляем хэш-функцию от данных, отправляем ее вместе с данными, получатель вычисляет свою хэш-функцию от полученных данных, сравнивает ее с полученной хэш-функцией, если все передалось успешно - они совпадут.

Автоматическое обнаружение расчетчиков

Например:

Создаем сервер с заранее известным портом. И несколько клиентов, один из которых будет раздавать задания, а остальные производить расчеты. Все клиенты регистрируются на сервере, как «расчетчики» и один, как «раздающий задания». Затем сервер передает данные о расчетчиках раздающему задания, тот в свою очередь посылает данные для расчета напрямую расчетчикам.

Данный вариант не является эталонным. Принимаются любые другие, лишь бы выполняли поставленную задачу.

Рекомендуемая литература:

http://helper10.narod.ru/i27.htm

http://tools.ietf.org/html/rfc768

http://docs.oracle.com/javase/1.4.2/docs/api/java/net/DatagramSocket.html

http://docs.oracle.com/javase/1.4.2/docs/api/java/net/DatagramPacket.html

Пример Java:

Для работы с датаграммными сокетами приложение должно создать сокет на базе класса DatagramSocket, а также подготовить объект класса DatagramPacket, в который будет записан принятый от партнера по сети блок данных. Канал, а также входные и выходные потоки создавать не нужно. Данные передаются и принимаются методами send и receive, определенными в классе DatagramSocket.

В классе DatagramSocket определены два конструктора, прототипы которых представлены ниже:

public DatagramSocket(int port);

public DatagramSocket();

Первый из этих конструкторов позволяет определить порт для сокета, второй предполагает использование любого свободного порта.

Обычно серверные приложения работают с использованием какого-то заранее определенного порта, номер которого известен клиентским приложениям. Поэтому для серверных приложений больше подходит первый из приведенных выше конструкторов.

Клиентские приложения, напротив, часто применяют любые свободные на локальном узле порты, поэтому для них годится конструктор без параметров.

Прием и передача данных на датаграммном сокете выполняется с помощью методов receive и send, соответственно:

public void receive(DatagramPacket p);

public void send(DatagramPacket p);

В качестве параметра этим методам передается ссылка на пакет данных (соответственно, принимаемый и передаваемый), определенный как объект класса DatagramPacket. Этот класс будет рассмотрен позже.

Еще один метод в классе DatagramSocket, которым вы будете пользоваться, это метод close, предназначенный для закрытия сокета:

public void close();

Перед тем как принимать или передавать данные с использованием методов receive и send вы должны подготовить объекты класса DatagramPacket. Метод receive запишет в такой объект принятые данные, а метод send - перешлет данные из объекта класса DatagramPacket узлу, адрес которого указан в пакете.

Подготовка объекта класса DatagramPacket для приема пакетов выполняется с помощью следующего конструктора:

public DatagramPacket(byte ibuf[],

int ilength);

Этому конструктору передается ссылка на массив ibuf, в который нужно будет записать данные, и размер этого массива ilength.

Если вам нужно подготовить пакет для передачи, воспользуйтесь конструктором, который дополнительно позволяет задать адрес IP iaddr и номер порта iport узла назначения:

public DatagramPacket(byte ibuf[],

int ilength,

InetAddress iaddr, int iport);

Таким образом, информация о том, в какой узел и на какой порт необходимо доставить пакет данных, хранится не в сокете, а в пакете, то есть в объекте класса DatagramPacket.

Помимо только что описанных конструкторов, в классе DatagramPacket определены четыре метода, позволяющие получить данные и информацию об адресе узла, из которого пришел пакет, или для которого предназначен пакет.

Метод getData возвращает ссылку на массив данных пакета:

public byte[] getData();

Размер пакета, данные из которого хранятся в этом массиве, легко определить с помощью метода getLength:

public int getLength();

Методы getAddress и getPort позволяют определить адрес и номер порта узла, откуда пришел пакет, или узла, для которого предназначен пакет:

public InetAddress getAddress();

public int getPort();

Если вы создаете клиент-серверную систему, в которой сервер имеет заранее известный адрес и номер порта, а клиенты - произвольные адреса и различные номера портов, то после получения пакета от клиента сервер может определить с помощью методов getAddress и getPort адрес клиента для установления с ним связи.

Варианты заданий.

1. Расчет определенного интеграла методом левых прямоугольников.
2. Расчет определенного интеграла методом правых прямоугольников.
3. Расчет определенного интеграла методом трапеций.
4. Сглаживание с помощью метода скользящего среднего (сумма соседних точек)
5. Медианное сглаживание (упорядочивание в окне и получение значения центрального элемента)
6. Сортировка методом Хоара (быстрая сортировка). Дерево расчетчиков.
7. Генетический алгоритм. Поиск минимума квадратичной функции. Если сложно, то вместо этого умножение матриц (распараллелить).
8. Клональный алгоритм. Поиск минимума квадратичной функции. Если сложно, то распараллелить сложение матриц.
9. Поиск значения в упорядоченном массиве. Дерево расчетчиков.
10. Поиск значения хэш-суммы (от случайной строки и данных) соответствующей заданному условию сравнения.