Исследование временных характеристик многопоточного приложения в ОСРВ QNX во время обработки информации от периферийных устройств

Отчет по расчетно-графической работе

Ревизия: 0.1

Последняя модификация: Vlad Kovtun

Дата последней модификации: 19.12.2011 10:20:00

История изменений

28.02.2011 – Версия 0.1. Первичный документ. Влад Ковтун

Содержание

Расчетно-графическая работа. Исследование временных характеристик многопоточного приложения в ОСРВ QNX во время обработки информации от периферийных устройств

Постановка задачи

Общие положения

Целью работы является исследование времени реакции сервера на сетевые запросы клиентов, формируемые в реальном масштабе времени.

Для достижения цели необходимо решить **следующие задачи**:

- Разработать серверное приложение, которое получает запросы по протоколу TCP|UDP от некоторого количества клиентов и выполняет обработку этих запросов.
- Разработать консольное клиентское приложение, которое производит подключение к заданному серверу и через некоторые промежутки времени формирует запрос к нему по протоколу TCP|UDP.
- Исследовать время передачи и обработки пакетов серверным приложением.

Объект исследования является процесс обработки пакетов сервером, полученных от удаленных клиентов.

Задание

Задание выполняется согласно варианту 25:

- Разработка приложения осуществляется на языке C++ в среде разработки QNX Momentics v4.7 для операционной системы реального времени QNX 6.5 с ядром Neutrino 2.
- Задание на разработку приложения и проведения эксперимента:
  - в качестве протокола передачи данных используется UDP;
  - количество одновременных подключений – 4;
  - закон изменения времени между временем начал отправки следующего пакета (порции) данных от клиента к серверу: $n = \exp\ k \cdot t + b \mod maxconn$, где $maxcon = 15$-максимальный промежуток времени, мс, $k = v$, где v-номер по журналу (25); $b = 70$; величина t увеличивается на 1 каждые 45 мс; период моделирования 15 мин;
  - закон изменения количества передаваемой информации от клиента к серверу: $n = k \cdot t + b \mod maxsize$, $maxsize = 3072$-максимальный объем $k = v \cdot 64$, где v-номер по журналу (25); $b = 128$; величина t увеличивается на 1 после каждой передачи.
- После окончания проведения эксперимента следует проанализировать записи лог файла:
  - Построить график зависимости интервала времени приема и обработки пакета, от размера пакета.
  - Построить график загрузки процессора за весь период моделирования.
  - Оценить пропускную способность сервера при заданных условиях моделирования.
- Сделать выводы о возможностях оптимизации серверного приложения, для увеличения его пропускной способности.

Требования

- Архитектура приложения строится по модульному принципу.
- За основу принимается стандартная библиотека C++.
- Рекомендуется использовать защищенные ресурсы и указатели.
- Обязательным является обработка исключений.
- Во время работы приложения обязательным является отображение информации о статусе приложения, т.е. оно работает, обработано столько-то процентов текста.

- Следует предусмотреть возможность работы приложения в отладочном режиме, когда вся информация заносится в лог приложения. Допускается ведение персонального лога для каждого из процессов (потоков).
- Исходный код обязан быть комментирован. Для C# следует использовать нотацию Microsoft XML Documentation [1], для C++ следует использовать нотацию doxygen [2].

Введение

Общие положения

Управление различными процессами в АСУ технологическими процессами (АСУ ТП) необходимо осуществлять не только эффективно, но и в заданные промежутки времени. Основу таких систем управления составляют сложные вычислительные комплексы, которые функционируют под управлением операционных систем реального времени (ОС РВ). Одной из наиболее эффективных и широко используемых в промышленности ОС является OS QNX.

Одним из главных отличий ОСРВ от универсальных ОС является способность гарантировать минимальный латентный период. В ОС общего назначения внешние прерывания попадают в очередь и обрабатываются только после того, как ОС завершит свою текущую операцию и обработает все остальные прерывания, находящиеся в очереди. Латентный период в ОС РВ непостоянен по величине, и никогда не достигает своего минимально возможного значения.

Другими словами, ОС РВ умеют прерывать текущую операцию для немедленной обработки прерывания. Как следствие, ОС РВ гарантируют обработку события в течение определенного временного интервала.

Встроенные приложения

Многие приложения реального времени являются встроенными, но не все встроенные системы работают в режиме реального времени. Встроенные системы и системы реального времени часто путают друг с другом, хотя для их описания используются различные характеристики. Для приложений реального времени главное – их временные характеристики, а встроенные системы охарактеризовать не так просто.

Встроенные системы - это нечто, чьи программные компоненты не взаимодействуют с пользователем непосредственно, как, например, сотовые телефоны или индустриальные микрокомпьютеры. Нет каких-то общих определяющих признаков для встроенных систем, тем не менее, они обычно требуют меньше памяти, используют более слабые процессоры и поддерживают меньшее количество устройств ввода-вывода, например, только клавиатуру. Поскольку они уступают в универсальности ОС на настольных компьютерах, используете их только для решения каких-то конкретных задач.

«Безголовые» устройства

«Безголовые» устройства – это системы без периферии. Программы часто загружаются в эти устройства по сети. Это же сетевое соединение часто используется для конфигурирования этих устройств. Поскольку это очень ограниченные средства, то можете создавать «безголовые» устройства с использованием сети и умного программного обеспечения, работая в режиме простейшего ниспадающего меню.

Процессы в реальном времени

Множеству измерительных и управляющих приложений требуется возможность работы в режиме реального времени. Для того, чтобы понять работу этих приложений реального времени, необходимо иметь представление о базовых принципах реального времени, и о том, насколько оно важно при управлении, обработке сигналов и реагировании на события.

Управление в реальном времени

Занимаясь управлением в реальном времени, непрерывно отслеживается или моделируется состояние физической системы. Управляющая программа циклически выполняет некоторую определенную пользователем процедуру, работа которой разделяется на временные отрезки, рис. 1. Человека окружает множество

управляющих систем реального времени, таких как системы управления движением машины или термостатическая система управления климатом в доме. Есть несколько параметров, которыми можно охарактеризовать приложение реального времени:

- продолжительность управляющего цикла;
- предсказуемость;
- разброс (jitter).

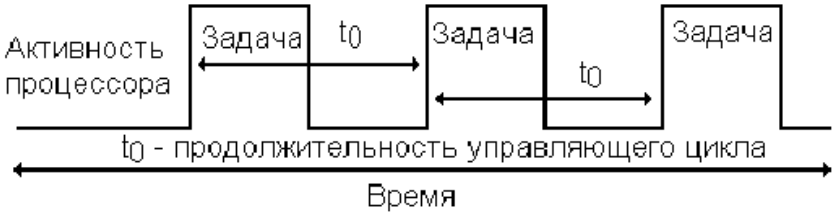

Рис. 1. Управление в реальном времени

Продолжительность управляющего цикла

Большинство АСУ РВ взаимодействуют с физической системой путем сравнения ее текущего состояния с ожидаемым состоянием и дальнейшим предсказанием поведения системы на основе результата этого сравнения. Интервал времени между двумя такими сравнениями и есть продолжительность управляющего цикла. Необходимая продолжительность этого цикла зависит от системы. Например, продолжительность цикла в системе управления температурой в духовке довольно велика. Эта система измеряет температуру, сравнивает ее с желаемой температурой и включает/выключает нагревательные горелки. В этом случае вполне достаточен цикл продолжительностью 1 сек. С другой стороны, печи, используемые в промышленности, требуют очень точно выдержанного температурного режима, например, при выращивании кристаллов. В этом случае продолжительность управляющего цикла должны быть существенно меньше, чтобы удовлетворить жестким температурным ограничениям. В общем случае управление в режиме реального времени соответствует рис. 2.

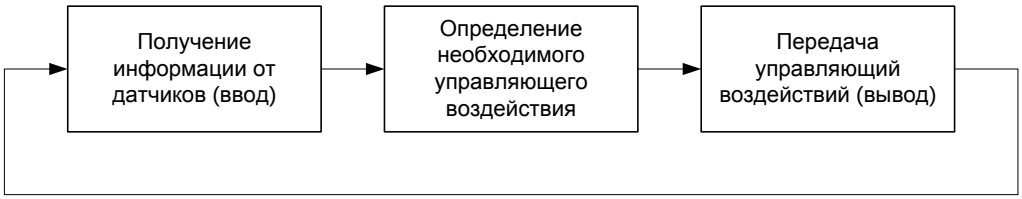

Рис. 2. Типичное управляющее приложение реального времени

Предсказуемость

Предсказуемость характеризует постоянство определенных временных интервалов между событиями. Многие управляющие алгоритмы, требуют очень предсказуемого поведения. Например, подъемник постепенно перемещается к правильному этажу вследствие предсказуемого поведения управляющего алгоритма. Если бы предсказуемости не было, подъемник все еще добирался бы до нужного этажа, но не всегда.

Разброс

Все системы реального времени в той или иной мере встречаются с проблемой разброса. Разброс – другой способ охарактеризовать непредсказуемость систем реального времени. Вы можете оценить его как разницу между отдельной временной задержкой и ее желаемой величиной (см. рис. 3).

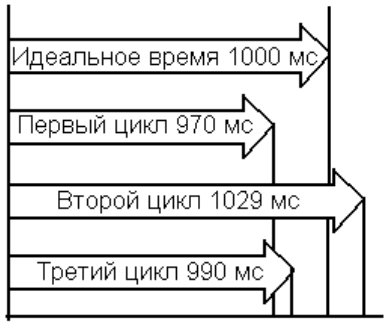

Рис. 3. Расчет разброса в допустимом интервале ±30 мс

Жесткое и мягкое реальное время

АСУ РВ могут быть по-разному распределены по шкале быстродействия. На одном краю располагаются системы жесткого реального времени, которые очень предсказуемы и никогда не теряют событий. Примером таких систем может служить измеритель мощности двигателя. Если пропускается событие, то собранные данные или моделируемое состояние дороги будут некорректны. На другом конце шкалы расположены системы мягкого реального времени, от которых не требуется высокой предсказуемости и которым позволяется пропускать события. Примером может служить система слежения за температурой, когда пропуск одного отсчета измерительной информации не влияет на общее поведение системы, поскольку температура изменяется очень медленно.

Обработка сигналов в режиме реального времени

Обработка сигналов в режиме реального времени имеет много общего с управлением. Она требует предсказуемых временных интервалов между повторяющимися событиями. Но вместо вычисления отклика при этом выполняется обработка сигналов, полученных в результате сбора данных. Примером может служить оценка частотных характеристик сигнала. Приложение циклически собирает отсчеты сигнала в предсказуемые моменты времени и вычисляет спектр мощности. Такое приложение должно не только анализировать сигнал блоками, но и делать это в режиме точка-за-точкой.

Процедуры поточечного анализа

Во многих приложениях процедуры поточечного анализа требуют повышенной производительности. Вместо того, чтобы анализировать блоки данных, эти процедуры накапливают данные в памяти и вычисляют новые значения на основе этих данных и нового значения (имеется в виду, что процедуры сохраняют не собираемые данные, а только предыдущий результат, и вычисляют новый результат на основе предыдущего результата и вновь полученного отсчета). В процессе того, как эти процедуры анализа получают отдельные отсчеты, они генерируют и точечные оценки, и массивы значений. Например, поточечный цифровой фильтр получает отдельный отсчет и выводит отфильтрованное значение. Функция спектра мощности могла бы получать очередной отсчет и генерировать массив частотного профиля. Жесткое реальное время в этой ситуации необходимо, поскольку при потере отсчета или задержке отсчета во времени нарушится целостность накопленных данных.

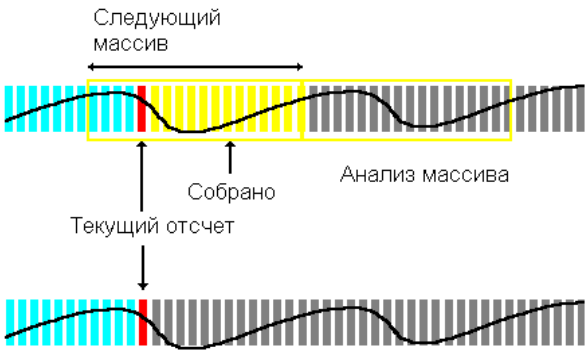

Рис. 4. Поточечный анализ

Собственно производительность

Процедуры поточечного анализа позволяют уменьшить временную задержку между моментом сбора данных и моментом генерации результирующего массива. Например, простой цифровой фильтр собирает 1000 отсчетов в секунду, фильтрует данные и генерирует массив отфильтрованных значений. Такой массив генерируется в течение всей секунды, когда был принят отсчет, что вызывает большую временную задержку или сдвиг по фазе. При поточечной фильтрации массив выходных значений формируется после сбора каждого отсчета, что приводит к уменьшению временной задержки до микросекунд.

Улучшенное принятие решений

Можно достичь лучших результатов при принятии решений, если использовать в своем приложении процедуры поточечного анализа. Производя фильтрацию по приходу каждого нового отсчета, увеличивается разрешающую способность вашего процесса принятия решений. Если хотите отключить систему, если низкочастотная величина превышает некоторый уровень, можете обратиться к отфильтрованному сигналу и сравнить его значение с установкой. При поблочном анализе решение может занять всю секунду, в течение которой сигнал регистрируется и фильтруется. При поточечном анализе можете анализировать значения по мере их сбора и сравнивать их с установкой каждую миллисекунду.

Наивысшая производительность

Процедуры поточечного анализа часто более эффективны, чем их поблочные аналоги. Вместо того, чтобы пересчитывать выход по всему набору данных, поточечный анализ вычисляет текущее выходное значение на основе нескольких предыдущих значений, текущего значения и предыдущего результата.

Реагирование на события в реальном времени

Реагируя в реальном времени на события, можно откликаться на отдельное событие в течение отведенного временного промежутка. Система реального времени гарантирует минимальное время реакции на событие. События могут приходить периодически или случайно. Пример приложения, реагирующего на внешние события в реальном времени - фабричная система мониторинга безопасности. Если технологический процесс вступает в опасное состояние, система реального времени обязана гарантированно быстро отреагировать на это. Реагирование может заключаться в отключении критических компонентов.

Латентный период

Латентный период (latency) используется для описания времени реакции на событие. Эта характеристика подобна характеристике предсказуемости в управляющих системах. При реагировании на события в режиме реального времени гарантировать минимально возможный латентный период. Латентный период на промышленном производстве может рассматриваться как максимальное время, требуемое на отключение критических компонентов при возникновении «опасного» события.

Сети в системах реального времени

Построение АСУ РВ требует организации множества распределенных вычислительных систем в единую сеть. Отметим, что особой задачей построения АСУ РВ, является именно организация сетевого взаимодействия в реальном времени.

В системах, построенных на основе ОС РВ OS QNX получили распространение несколько типов сетей:

- Ethernet.
- QNET.

О сетях Ethernet сказано достаточно много, отметим лишь, что для целей АСУ РВ они не совсем подходят. В связи с этим и были разработаны сети QNET. Остановимся более детально на сетях QNET.

Ведение

В простейшем случае локальная сеть обеспечивает разделяемый доступ к файлам и периферийным устройствам для нескольких соединенных между собой компьютеров. OS QNX идет гораздо дальше этого простейшего представления и объединяет всю сеть в единый, однородный набор ресурсов.

Любой процесс на любом компьютере в составе сети может непосредственно использовать любой ресурс на любом другом компьютере. С точки зрения приложений не существует никакой разницы между местным или удаленным ресурсом, и использование удаленных ресурсов не требует каких-либо специальных средств. Более того, чтобы определить, находится ли такой ресурс как файл или устройство на локальном компьютере или на другом узле сети, в программу не потребуется включать специальный дополнительный код!

Пользователи могут иметь доступ к файлам по всей сети, использовать любое периферийное устройство, запускать программы на любом компьютере сети (при условии, что они имеют надлежащие полномочия). Связь между процессами осуществляется единообразно, независимо от их местоположения в сети. В основе такой прозрачной поддержки сети в QNX лежит всеобъемлющая концепция Inter Process Communication (IPC) — взаимодействие между отдельными процессами на основе передачи сообщений.

Модель единого компьютера

QNX изначально проектировалась как сетевая ОС. В некоторых отношениях сеть QNX напоминает скорее большую ЭВМ, нежели набор мини-компьютеров. Однако, в отличие от большой ЭВМ, «единый компьютер» QNX обеспечивает гораздо более быструю реакцию системы, т.к. соответствующий объем вычислительных ресурсов может быть выделен на каждом узле в соответствии с потребностями каждого пользователя.

В условиях управления производством используются программируемые контроллеры и другие устройства ввода/вывода, работающие в режиме реального времени, которые могут потребовать больше ресурсов, чем другие менее ответственные приложения, такие как, например, текстовый редактор. Сеть QNX достаточно «отзывчива», чтобы поддерживать оба этих типа приложений одновременно — QNX позволяет сфокусировать вычислительную мощность системы на производственном оборудовании там, где это необходимо, не жертвуя в то же время интерфейсом пользователя.

Гибкая поддержка сети

Сеть QNX может быть построена с использованием различного оборудования и стандартных промышленных протоколов. В силу своей полной прозрачности для прикладных программ и пользователей, новые сетевые архитектуры могут быть внедрены и исключены динамически, в любое время, не нарушая при этом целостности операционной системы и не прерывая ее работы.

Такая степень прозрачности является еще одним примером больших возможностей архитектуры QNX, основанной на передаче сообщений (IPC). Во многих ОС такие важные функции как поддержка сети и IPC выполнены в виде надстроек над ОС, а не интегрированы непосредственно в ее сердцевину. Результатом такого подхода является неуклюжий и неэффективный интерфейс с «двойным стандартом», когда связь между процессами — это одно дело, в то время как проникновение в скрытый интерфейс таинственного монолитного ядра — совершенно другое дело!

QNX, напротив, исходит из того, что эффективная связь является ключом к эффективной работе. Передача сообщений является, таким образом, краеугольным камнем архитектуры QNX, увеличивает эффективность всех без исключения транзакций между процессами в системе, независимо от того, идет ли речь о передаче данных по внутренней шине персонального компьютера или по коаксиальному кабелю на расстояние нескольких миль.

Структуры сети QNX.

Согласно принципам архитектуры QNX, сервисы, отвечающие за организацию сетевой поддержки в QNX выполняются вне ядра, как отдельные процессы.

Преимущества такой архитектуры:

- сетевые драйверы могут быть динамически подключены в состав системы в любой момент времени, легко заменить тот или иной драйвер на другой, или изменить параметры сети, не приостанавливая систему;
- различные типы сетевых протоколов, такие как QNET и TCP/IP могут сосуществовать в системе одновременно;

В QNX Neutrino сетевая подсистема делится на три основные составляющие:

- менеджер сети (io-net), исполняемый модуль.
- интерфейс сетевого протокола (npm-qnet.so, npm-tcpip.so и т.д.) — разделяемые библиотеки.
- интерфейс драйвера сетевого оборудования (devn-ne2000.so) — разделяемые библиотеки.

Примерно так выглядит схема взаимодействия приложения с аппаратным обеспечением. Компонент io-net может одновременно поддерживать несколько сетевых протоколов и несколько драйверов.

Менеджер сети io-net

Менеджер сети io-net выполняет ключевую функцию «моста» между интерфейсами сетевых протоколов и интерфейсами сетевых драйверов. Наибольшей гибкости удается достичь за счет независимости менеджера сети от используемых типов протоколов или аппаратной реализации сети. io-net подгружает протоколы и драйверы как динамические библиотеки, которые в свою очередь осуществляют взаимодействие с программой клиента через соответствующий API протокола и с железом через соответствующий API драйвера.

Интерфейс протокола передачи данных

Интерфейс протокола отвечает за реализацию того или иного протокола передачи данных, например таких как QNET, TCP/IP и др. Каждый интерфейс протокола поставляется в виде разделяемой библиотеки, например npm-qnet.so, протокол QNET, он же Native Neurino Networking. Одновременно в системе могут быть запущены один и более протоколов.

Например, следующая команда запускает менеджер сети с двумя протоколами:

io-net -d ne2000 -p ttcpip -p qnet

В комплект поставки QNX 6 входит четыре основных интерфейса сетевых протоколов:

- QNET – «родной» протокол ОС QNX,
- TTcpIp – «tiny» (крошечный) tcp/ip стек для встраиваемых приложений (только базовые функции tcp/ip),
- TcpIp - полная реализация TCP/IP стека от BSD 4.4 (BSD sockets API),
- Pppmgr - Point-to-Point протокол.

Интерфейс сетевого драйвера

Через сетевой драйвер осуществляется взаимодействие менеджера сети с конкретным типом сетевого адаптера. Каждый драйвер поставляется в виде разделяемой библиотеки и подключается при старте менеджера сети. Например, команда:

io-net -d ne2000

запускает менеджер сети, который в свою очередь подгружает интерфейс драйвера сетевой платы NE2000, размещенный в виде динамической библиотеки devn-ne2000.so. Между драйвером и менеджером сети устанавливается двустороннее взаимодействие. Драйвер может обрабатывать вызовы от менеджера сети при приеме пакетов и, наоборот, менеджер сети, в свою очередь, обращается к драйверу при попытке осуществить передачу пакетов.

Компания QSSL (разработчик ОС QNX) относительно недавно выпустила в свет бета-версии комплектов разработчиков драйверов для своей ОС, в т.ч. сетевых. Комплект содержит примеры исходных текстов реально существующих сетевых драйверов и руководство программиста в формате PDF.

Протокол QNET

Native Neutrino Networking — это, прежде всего, высокоскоростной сетевой протокол. Уникальность QNET заключается в том, что он превращает объединенные в сеть компьютеры в один виртуальный суперкомпьютер (кластер), создавая единый однородный набор ресурсов, доступ к которым возможен из любого места сети. Следует заметить, что QNET предусматривает возможность наличия нескольких параллельных физических сетей. Для этого следует поместить в каждый компьютер несколько сетевых плат соединенных отдельными кабелями.

- FLEET расшифровывается как:
- Fault-tolerant — отказоустойчивость;
- Load-balancing on the fly — регулировка нагрузки на лету;
- Efficient performance — эффективное действие;
- Extensible architecture — расширяемая архитектура;
- Transparent distributed processing — прозрачная распределенная обработка.

Отказоустойчивость

При выходе из строя сети, например в результате неисправности сетевой платы или обрыва кабеля, FLEET автоматически перенаправляет данные по другой сети. Это происходит на лету, незаметно для прикладных программ.

Регулировка нагрузки на лету

Пропускная способность сети обычно ограничена скоростью работы сетевого оборудования. FLEET дает возможность одновременно передавать данные по нескольким сетям, позволяя увеличить пропускную способность сети в несколько раз.

Эффективное действие

Сетевые драйверы FLEET позволяют буквально «выжать» все возможное из сетевого оборудования. Например, при передаче больших блоков данных между двумя процессами по сети Ethernet достигает высокой пропускной способностью, см. таблица 1.

Таблица 1. Пропускная способность по сети Ethernet через драйверы FLEET

Сеть	Пропускная способность
10 Mbit Ethernet	1.1 Mbytes/sec
100 Mbit Ethernet	7.5 Mbytes/sec

Расширяемая архитектура

Сетевые драйверы и менеджер сети не входят в ядро ОС и являются независимыми процессами, которые могут быть запущены или остановлены в любой момент. Компьютеры могут динамически подключаться к сети либо отсоединяться от нее не требуя реконфигурации системы. Благодаря автоматическому сетевому «мосту», можно объединять различные физические сети в одну логическую локальную сеть.

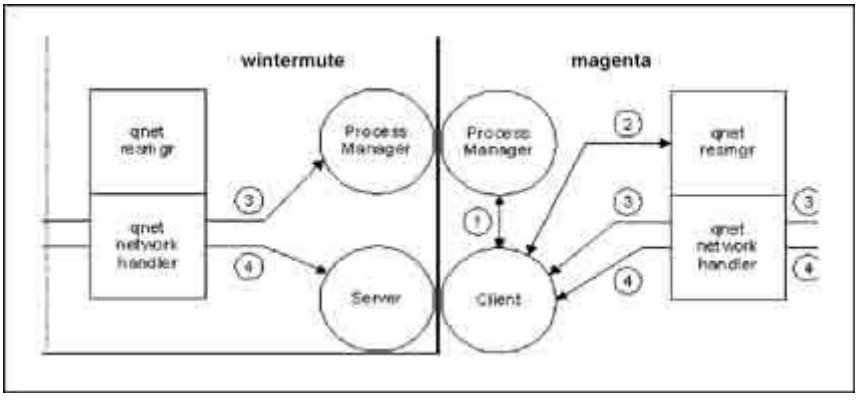

Рис. 5. Пример объединения нескольких разнородных сетей в единую логическую сеть

Распределенная обработка данных

Поддержка сети QNET глубоко интегрирована в примитивы передачи сообщений и управления процессами, и, как следствие, связь между процессами на локальном узле и по сети — это одно и то же. Такая прозрачная поддержка связи между процессами по сети превращает сеть в единый виртуальный компьютер. Результат? Приложения без внесения каких-либо изменений в код, могут взаимодействовать по сети.

Соглашения об именах узлов

Чтобы понять, как работает QNET, рассмотрим пример взаимодействия "клиент-сервер". В случае единственного узла (процессы клиента и сервера работают в рамках одного компьютера), клиент просто создает связь с сервером, используя функцию ConnectAttach(). Затем посылается сообщение, например, через MsgSend().

Теперь рассмотрим случай простой сети с двумя машинами — одна содержит процесс-клиент, другая — процесс-сервер. Соглашение о связи клиент-сервер идентично соглашению в случае единственного узла. Клиент создает связь с сервером и посылает серверу сообщение. Единственное различие в случае сети — то, что клиент определяет другой дескриптор узла для запроса функции ConnectAttach ().

Как клиент узнает, какой именно дескриптор узла использовать для связи с сервером?

Клиент получает адрес сервера, опираясь на сетевое имя запрашиваемого удаленного ресурса. В случае с одной, локальной машиной, результатом могут быть дескриптор узла, идентификатор процесса и идентификатор канала. В случае с сетью — результаты аналогичны — разница только в значении дескриптора узла. Если значение дескриптора узла 0, то это один и тот же узел, если число, отличное от нуля, то это удаленный узел.

И в том и в другом случаях, при соединении клиента с сервером, клиент получает идентификатор из ConnectAttach(). Этот идентификатор используется в дальнейшем при отправке/приеме сетевых сообщений.

Давайте рассмотрим, что произойдет в системе после выполнения функции open ().

Предположим, что клиент на одном узле (magenta) желает использовать последовательный порт (/dev/ser1) на другом узле (wintermute). Клиент выполняет open(), указывая путь /net/wintermute/ dev/ser1.

Рассмотрим, что при этом произойдет:

1. Передача сообщения от клиента к местному менеджеру процессов. Осуществляется запрос, с кем войти в контакт, чтобы выполнить распознавание сетевого пути /net/wintermute/ dev/ser1. Т.к. менеджер сети работает через пространство имен "/net", менеджер процессов вернет сообщение о перенаправлении, указывающее, что клиент должен соединиться с локальным менеджером сети для получения дополнительной информации.

2. Клиент передает сообщение локальному менеджеру сети, заново запрашивая информацию о том, с кем он должен соединиться, в соответствии с именем сетевого пути. Локальный менеджер сети отвечает другим сообщением перенаправления, возвращая дескриптор узла, идентификаторы процесса и канала от менеджера процессов на узле wintermute.

3. Клиент создает связь с менеджером процессов на узле wintermute, еще раз запрашивая с кем нужно войти в контакт, согласно сетевому пути. Менеджер процессов на узле wintermute возвращает переадресацию, на сей раз с описателем узла, идентификаторами канала и процесса драйвера последовательного порта на его собственном узле.

4. Клиент создает связь с драйвером последовательного порта на узле wintermute, и, наконец, получает идентификатор, который можно использовать впоследствие для посылки/приема сообщений.

Теперь все сообщения клиент-сервер являются прямыми и проходят аналогично локальному случаю.

Терминология

Имя узла - Последовательность, идентифицирующая узел. Имя узла не может содержать "слешей" или "точек". В примере выше, использовали wintermute в качестве одного из имен узлов.

Доменное имя узла - Последовательность, присоединяемая npm-qnet к имени узла. Вместе имя узла и доменное имя должны образовывать единую последовательность. Доменное имя является уникальным для всех узлов, которые общаются друг с другом.

Полное сетевое имя узла (FQNN) - Полная последовательность, объединяющая вместе имя узла и доменное имя. Например, имя узла — wintermute, и доменное имя — qnx.com.ru, то полное сетевое имя будет: wintermute. qnx.com.ru.

Сетевой каталог - Каталог, имя которого определяется через npm-qnet. Каждый сетевой каталог (их может быть на одном узле больше одного) имеет предопределенное имя. По умолчанию — /net.

Распознавание имен - Процесс, с помощью которого npm-qnet конвертирует полные сетевые имена в список адресов, для того, чтобы транспортный уровень протокола знал маршрут.

Преобразование имен - Участок кода, осуществляющий преобразование полного сетевого имени в список конечных адресов. Каждый сетевой каталог имеет список преобразователей имен, которые в свою очередь используются для преобразования полных сетевых имен. По умолчанию — ndp.

Политика обслуживания (QoS)

Способ взаимодействия между двумя узлами. По умолчанию QoS — loadbalance.

Варианты преобразования имен:

Менеджер сети включает следующие типы преобразования:

- NDP — Node Discovery Protocol — посылает широковещательный запрос в сеть.
- DNS — к имени узла добавляется точка, результат передается функции TCP/IP gethostbyname ().
- File — ищет полное сетевое имя в статическом файле.

Качество обслуживания (QoS)

Качество обслуживания — это одна из проблем, возникающей в сетях повышенной надежности, применяемых в ОС РВ. Если осуществляется пересылка сигнальных данных через сеть, в системе управления производственным процессом, то естественно, хотелось бы гарантировать степень допуска возможных ошибок и т.д.

Сеть Neutrino предполагает следующие варианты QoS:

- **Балансировка нагрузки (loadbalance).** На уровне протокола npm-qnet принимаются решения, какие соединения использовать для передачи пакетов, в зависимости от текущей нагрузки и скорости передачи данных, определяемой io-net. Осуществляется попытка доставить пакеты, накопившиеся в очереди, как можно быстрее к удаленному узлу, используя все доступные и возможные физические соединения. Если возникает обрыв связи, периодически посылаются служебные пакеты с целью обнаружить восстановление соединения. Если соединение восстанавливается, оно включается в работу. Используется по умолчанию.

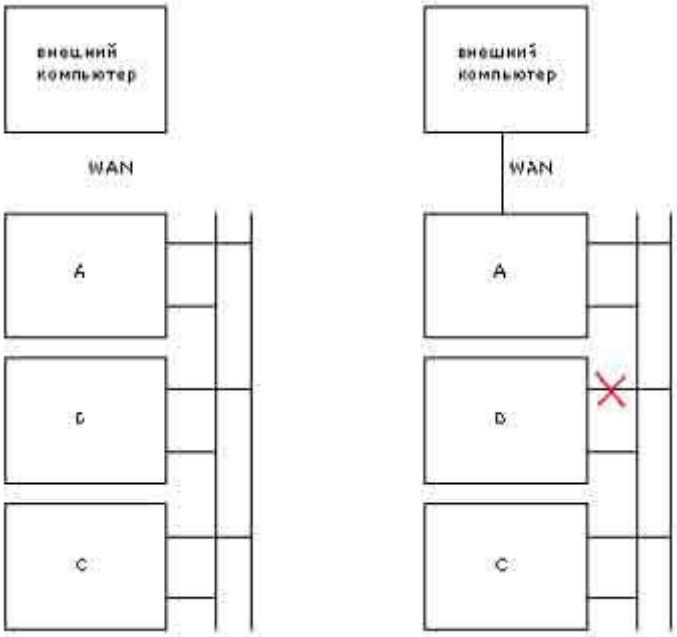

Рис. 6. Пример сети предприятия, с удаленным компьютером

- **Последовательный (sequential).** Используется первое доступное соединение, до возникновения ошибки, тогда используется следующее доступное соединение, и так далее. Приоритеты соединений определяются пользователем. Как и с loadbalance QoS, вышедшие из строя соединения периодически проверяются на восстановление и, если соединение восстановлено, оно помещается в пул доступных соединений. Если используется соединение с меньшим приоритетом, и восстанавливается соединение с большим, то предпочтение отдается использованию соединения с большим приоритетом.
- **Привилегированный (preferred)**. Аналогичен последовательному, с некоторыми отличиями. Когда последнее из списка доступных соединений, выходит из строя, включается политика loadbalance.
- **Избыточный (redundant).** Пакеты пересылаются по всем возможным соединениям одновременно. Если хотя бы одно соединение выходит из строя, то передача возобновляется только при восстановлении.

Политику качества обслуживания можно определять как часть сетевого пути. Например /net/wintermute~redundant/dev/ser1. Политика должна всегда начинаться с символа «тильды». Для одной и той же сети с разными политиками качества обслуживания можно использовать символьные ссылки в каталоге «/net».

Давайте рассмотрим на примере, как вышеописанные схемы работают на практике. Пример реальной сети на предприятии, состоящей из трех компьютеров A, B, C и удаленной ЭВМ, доступ к которой осуществляется через глобальную сеть (как пример — Интернет).

Предположим, что все доступные соединения используются в режиме балансирования нагрузки. A, B, и C при взаимодействии между собой используют пропускную способность обеих сетей, чем обеспечивается в данном случае наибольшая производительность, рис. 6(а). Теперь представим, что один из соединительных кабелей, идущих к B, отсоединен, рис. 6(б).

Теперь, A или C, при взаимодействии с B, обнаруживают, что они ничего не получают по одному из интерфейсов. Будет предпринята попытка протестировать интерфейс на некоторой очень низкой скорости передачи данных. До тех пор, пока кабель не будет вновь подключен к B, взаимодействие будет осуществляться с примерно ? пропускной способности по сравнению с изначальной. Время обнаружения выхода из строя соединения зависит от величины таймаута носителя и частоты транзакций (опрос не будет осуществляться, если нет невыполненных транзакций). Другой случай — интерфейс часто отказывает (например автокар переехал 10base5, но разъемы остались в сетевых платах). В этом случае, если потери скорости в интерфейсе превысят некоторое допустимое значение в процентах (например 5%), протокол транспортного уровня снизит скорость передачи данных, чтобы предпринять тестирование интерфейса.

Предположим, что С взаимодействует с внешней машиной. Для связи могут использоваться оба соединения с узлом А, который выступает в этом случае в качестве шлюза. Теперь рассмотрим этот пример с использованием избыточной политики. При применении избыточной политики в локальной сети ни один из узлов не сможет установить связь с внешней машиной если хотя бы одно соединение в локальной сети будет повреждено. Поэтому нужно быть весьма осторожным при применении в данном случае политики избыточности, гораздо выгоднее использовать другие методы, возможно loadbalance или последовательный.

Экономический эффект от применения QNET, по сравнению с традиционными сетевыми технологиями, выражается в сокращении затрат на программные и аппаратные средства при проектировании конкретных узлов сети и является прямым следствием свойств прозрачной сетевой поддержки.

За счет сетевой прозрачности мы можем отказаться от использования графического интерфейса и прикладного ПО, отвечающего за вывод информации на рабочем месте оператора — можно подгружать эти компоненты с ЭВМ, выполняющей функции подсистемы сбора информации через QNET. В результате, на рабочем месте оператора понадобятся только драйвера графики и ввода. Аналогично и с остальными компонентами системы. В результате мы сможем сократить и затраты на аппаратную часть всего комплекса.

Протокол TCP/IP

С ростом Internet, все более широкораспространенным становится протокол, основанный на — IP (Internet, протокол). Даже если Вы не подключаетесь к Internet, все равно, протокол IP и базирующиеся на нем инструментальные средства, общеприменимы и в результате IP-протокол становится де факто для многих частных сетей.

IP используется от простых задач (удаленный вход в систему) до более сложных (например поставка биржевых цен в реальном масштабе времени). И, сейчас, достаточно сложно найти ОС, в которой не было бы средств поддержки IP-протокола.

Для удовлетворения большинства пользовательских требований, QSSL разработала «облегченный» стек Neutrino TCP/IP (npm-ttcpip), с целью уменьшить затраты ресурсов, по сравнению с использованием обычного BSD API.

Neutrino TCP/IP также соответствует концепции модульности, принятой в QNX. Например, клиент NFS реализован в виде отдельного модуля и т.п. Именно принцип модульности в Neutrino TCP/IP позволил разработчикам эффективно и быстро проектировать и создавать встраиваемые системы, ориентированные на использование IP-протокола.

Структура

Интерфейс протокола npm-ttcpip разработан в виде многопоточного ресурс-менеджера. За счет использования мультипоточности стало возможным одновременное обслуживание большого количества клиентов.

В Neutrino TCP/IP используется BSD-совместимый socket-API. Это обеспечивает совместимость как с другими Unix-системами, так и с Windows. С практической точки зрения это позволяет безболезненный перенос сетевых приложений для TCP/IP из этих ОС в ОС Neutrino.

Функции разрешения имен

Функции базы данных имен были изменены, в сторону лучшего соответствия потребностям встраиваемых систем.

/etc/resolv.conf

Информация, обычно содержащаяся в/etc/resolv.conf может быть помещена в переменную среды RESCONF. Это позволяет использовать сервер имен без /etc/resolv.conf. Это также касается вызовов gethostbyname () и других функций преобразования имен.

/etc/protocols

В функции Getprotobyname () и getprotobynumber () были внесены изменения для обеспечения поддержки нескольких протоколов, включая IP, ICNP, UDP, и TCP. Для большинства случаев это означает, что можно обойтись без файла /etc/protocols.

/etc/services

Функция Getservbyname () была подвержена изменениям с целью поддержки нескольких сервисов, включая ftp, telnet, smtp, domain, nntp, netbios-ns, netbios-ssn, sunrpc, и nfsd. Для большинства случаев это означает, что можно обойтись без файла /etc/services.

Способность к взаимодействию

Код npm-ttcpip реализует функциональные возможности, предложенные в RFC 1122, ARP, IP, ICMP, UDP, и поддерживает протоколы TCP. npm-ttcpip также обеспечивает поддержку DHCP.

Мы не будем здесь рассматривать подробно состав пакета TCP/IP, об этом достаточно много и так написано. Остановимся лишь на особенностях реализации, исключительных для QNX Neutrino.

Встраиваемый сервер slinger

Отдельно имеет смысл рассмотреть входящий в состав пакета TCP/IP компактный веб-сервер slinger — собственная разработка QSSL, предназначенный для применения во встраиваемых приложениях. Он поддерживает возможности SSI и CGI, что позволяет, например, при применении в контроллерах, выдавать по запросу определенную информацию в виде динамических HTML-документов на рабочее место оператора. Это позволяет значительно сократить расходы на разработку клиентского программного обеспечения на рабочем месте оператора, осуществляющего, например, мониторинг параметров технологического процесса.

Организация рабочего места оператора сводится к установке компьютера с ОС, поддерживающей TCP/IP и содержащей в себе веб-браузер.

Полнофункциональный TCP/IP Stack

В состав ОС также входит реализация протокола TCP/IP, полностью соответствующая BSD 4.4 sockets. Это полнофункциональный TCP/IP протокол npm-tcpip. Включает в себя модуль NFS сервера, поддержку multicasting, virtual packet interface и т.д. Идеален для использования при решении таких задач, как, например, организация полнофункционального интернет-узла.

Здесь следует заметить, что приложения, удовлетворяющие API npm-ttcpip, сохранят свою работоспособность и под управлением полнофункционального стека npm-tcpip без какой либо модификации или рекомпиляции.

Фильтры (nfm-xxx)

IpFilter

В ОС Neutrino уже реализованы программные средства, расширяющие возможности TCP/IP до профессиональных. К таким средствам можно отнести модуль IPFilter. Программа распространяется с исходным кодом. Позволяет реализовать такие функции, как собственно фильтрация IP-пакетов, Firewall (что немаловажно на стыке глобальной сети и внутренней сети предприятия), NAT и комбинации вышеперечисленных возможностей.

Berkley Packet filter

Существует в стадии внутренней бета-версии QSSL. Применяется в библиотеке libcap, использующейся например для реализации сниффера пакетов tcpdump. Пользователь, используя фильтры собственной разработки, может значительно расширить существующие возможности TCP/IP в ОС QNX Neutrino.

Программное обеспечение

Если говорить о программном обеспечении для TCP/IP в QNX Neutrino, то его уже достаточно много. Это как разработки «с нуля» так и софт, перенесенный из других

*nix-совместимых ОС. Вкратце перечислим наиболее важные из них. Это веб-серверы Apache 1.3.x, Boa, популярный FTP-сервер ProFTPD, прокси-сервер squid, текстовые и графические браузеры, ICQ-, IRC- и FTP-клиенты. Большое количество программного обеспечения, распространяемого под лицензией GNU, также перенесено в среду QNX Neutrino.

Из сказанного следует, что сети типа QNET, являются хорошей альтернативой сетям типа Ethernet в распределенных системах для управления процессами в реальном времени.

Построение распределенных приложений

Ранее упоминалось, что наибольшую популярность получили Ethernet-сети с использованием стека протоколов TCP/IP. При построение таких приложений принято использовать многозвенную архитектуру, однако в рамках данной работы будем рассматривать ее упрощенную версию – клиент-серверную.

Так, согласно задания, необходимо разработать клиент-серверное приложение, в которое входит:

- Сервер, поддерживает до 4-х клиентских подключений.
- Клиент, может функционировать параллельно с несколькими такими же клиентами в одной сети и подключаться к заданному клиенту.

Алгоритм

Введение

Исходя из постановки задачи, необходимо описать алгоритм работы сразу 2-х приложений:

- Сервера.
- Клиента.

Диаграмма последовательности взаимодействия клиента и сервера приведена на рис. 1. На примере клиента 1 детализирован весь жизненный цикл потока 1 по его обслуживанию.

Коротко остановимся на протоколе взаимодействия. После запуска сервера, он переходит в режим ожидания подключений клиентов. После подключения клиента, сервер создает новый поток, на который ложится задача информационного обмена с клиентом. После завершения процедуры информационного взаимодействия (или возникновения критической ошибки) поток прекращает свою жизнедеятельность.

Отметим, что в основном потоке сервера установлен барьер, который ожидает завершения деятельности всех потоков. Сервер завершает свою работу лишь после завершения деятельности всех потоков.

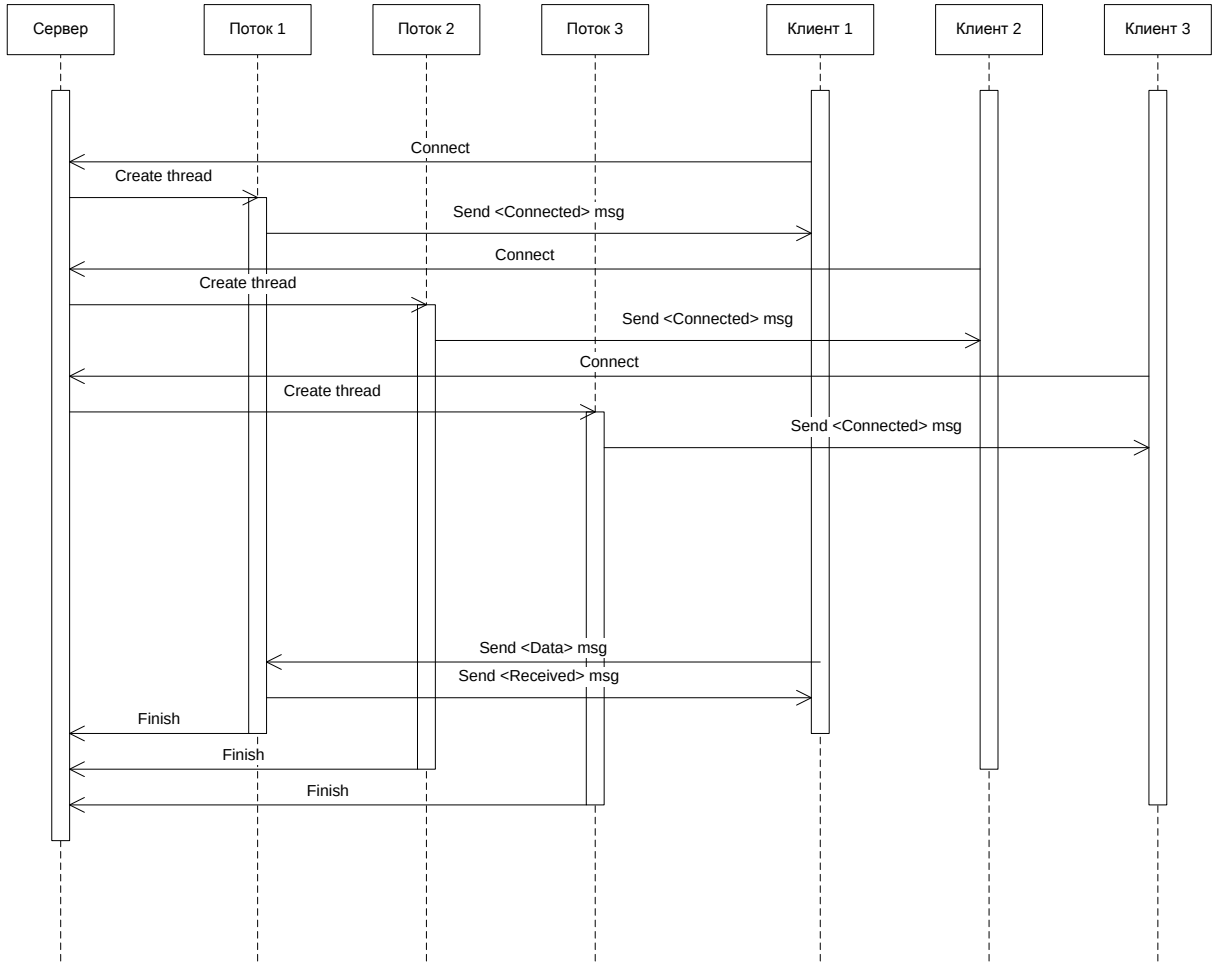

Рис. 7. Диаграмма последовательностей взаимодействия многопоточного сервера и 3-х клиентов

Сервер

На рис. 2 приведена блок-схема алгоритма основного потока выполнения на сервере. После запуска процесса сервера, производится чтения файла конфигурации:

- К какому IP адресу производится привязка.
- К какому порту производится привязка.
- Максимальное количество клиентов, которых необходимо обслужить.
- Имя файла лога, куда заносится информация о полученных пакетах

После прочтения файла конфигурации, производится проверка корректности параметров конфигурации. В случае их некорректности, производится завершение процесса сервера.

Далее создается серверный сокет, к которому и предполагается подключение удаленных клиентов. Вновь созданный серверный сокет конфигурируется, после чего производится связывание локального адреса и серверного сокета. Серверный сокет начинается «слушаться» на возможные клиентские подключения, в цикле, до тех пор пока не будут подключены все клиенты, и не будет выставлен соответствующий статус.

В случае подключения клиента, запускается новый поток для взаимодействия клиента с сервером (получения данных от клиента).

Отметим, что лишь после завершения всех потоков взаимодействия с клиентами, возможно завершение работы сервером. Для этого используется примитив синхронизации - барьер.

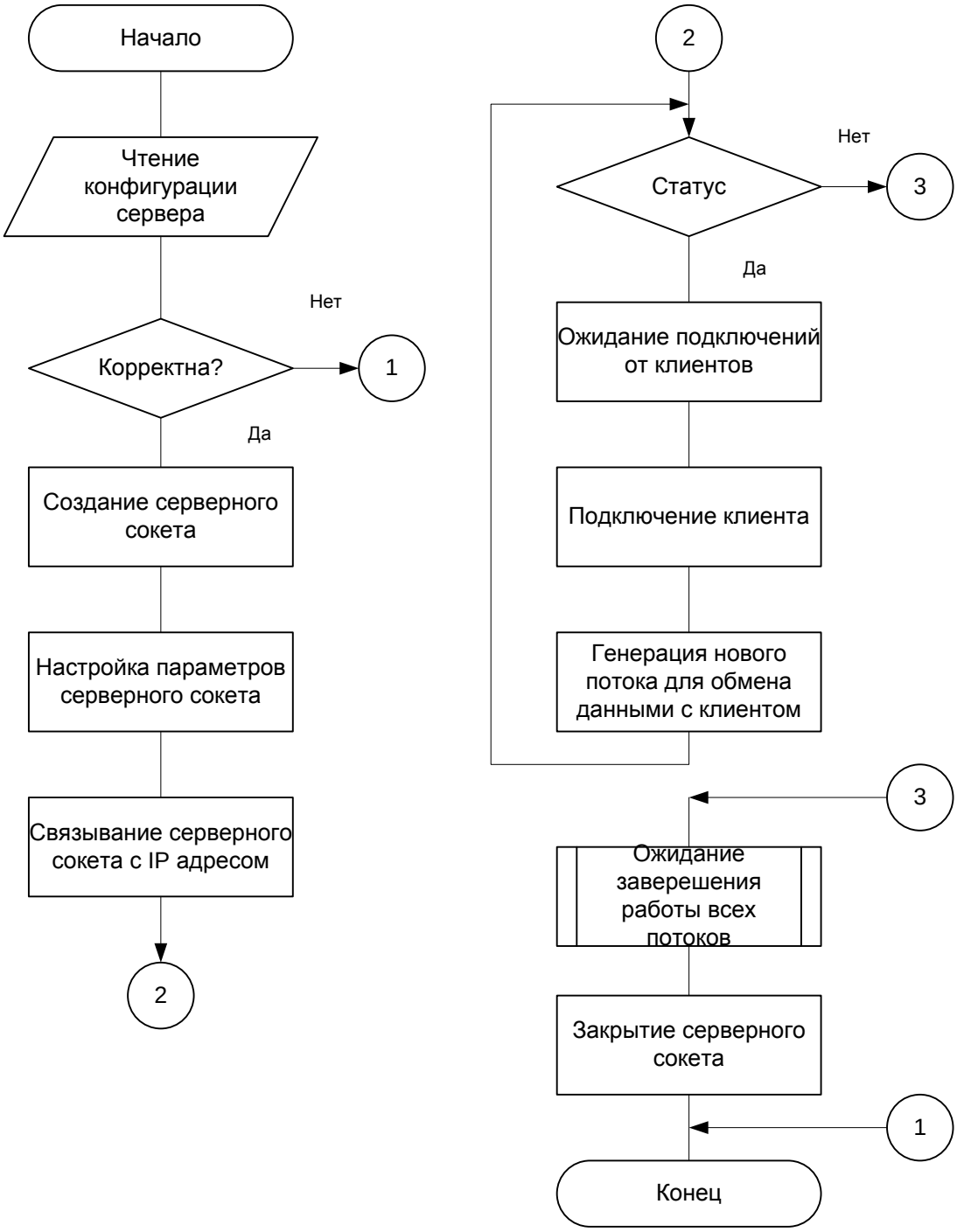

Рис. 8. Блок-схема основного потока сервера

Перейдем к описанию алгоритма работы потока обслуживания клиентских подключений, на рис. 3 приведена блок-схема его алгоритма.

В качестве параметра, потоку передается вновь созданный сокет клиентского подключения. Первым, что необходимо сделать потоку – отправить сообщение клиенту о том, что было успешно создано подключение. После чего, клиент сможет передать большой пакет данных. Перед началом приема пакета данных от клиента, сервер фиксирует время начала приема данных и по окончании его приема, также фиксирует время. Разница между конечным и начальным моментом времени является время передачи-приема пакета данных.

После окончания приема пакета данных, серверу необходимо уведомить клиента об успешном приеме пакета данных, после чего поток закрывает клиентский сокет и прекращает свое существование.

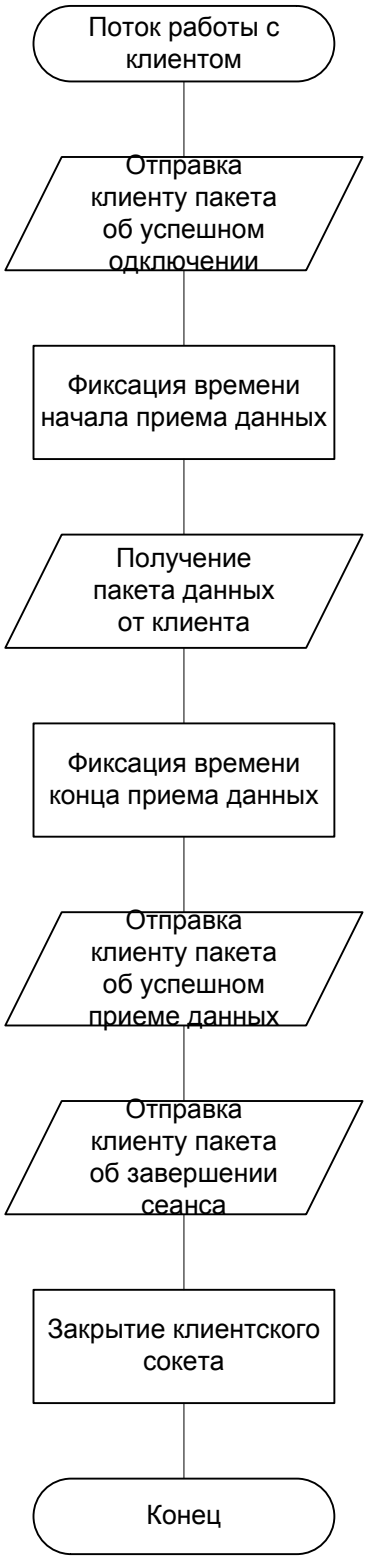

Рис. 9. Блок-схема потока по обслуживанию клиентских подключений

Далее опишем алгоритм функционирования клиентского приложения.

Клиент

Блок схема клиентского приложения представлена на рис. 4. Коротко опишем алгоритм работы приложения. После запуска, производится чтение из файла конфигурации:

- К какому IP адресу необходимо привязать клиентское приложение.
- Параметры закона формирования интервалов времени, через которые происходит генерация пакетов.
- Параметры закона формирования размеров пакетов.
- К какому IP адресу сервера необходимо подключаться.
- К какому порту сервера необходимо подключаться.

После чтения конфигурации, производится проверка ее корректности. В случае некорректности конфигурации происходит завершение приложения.

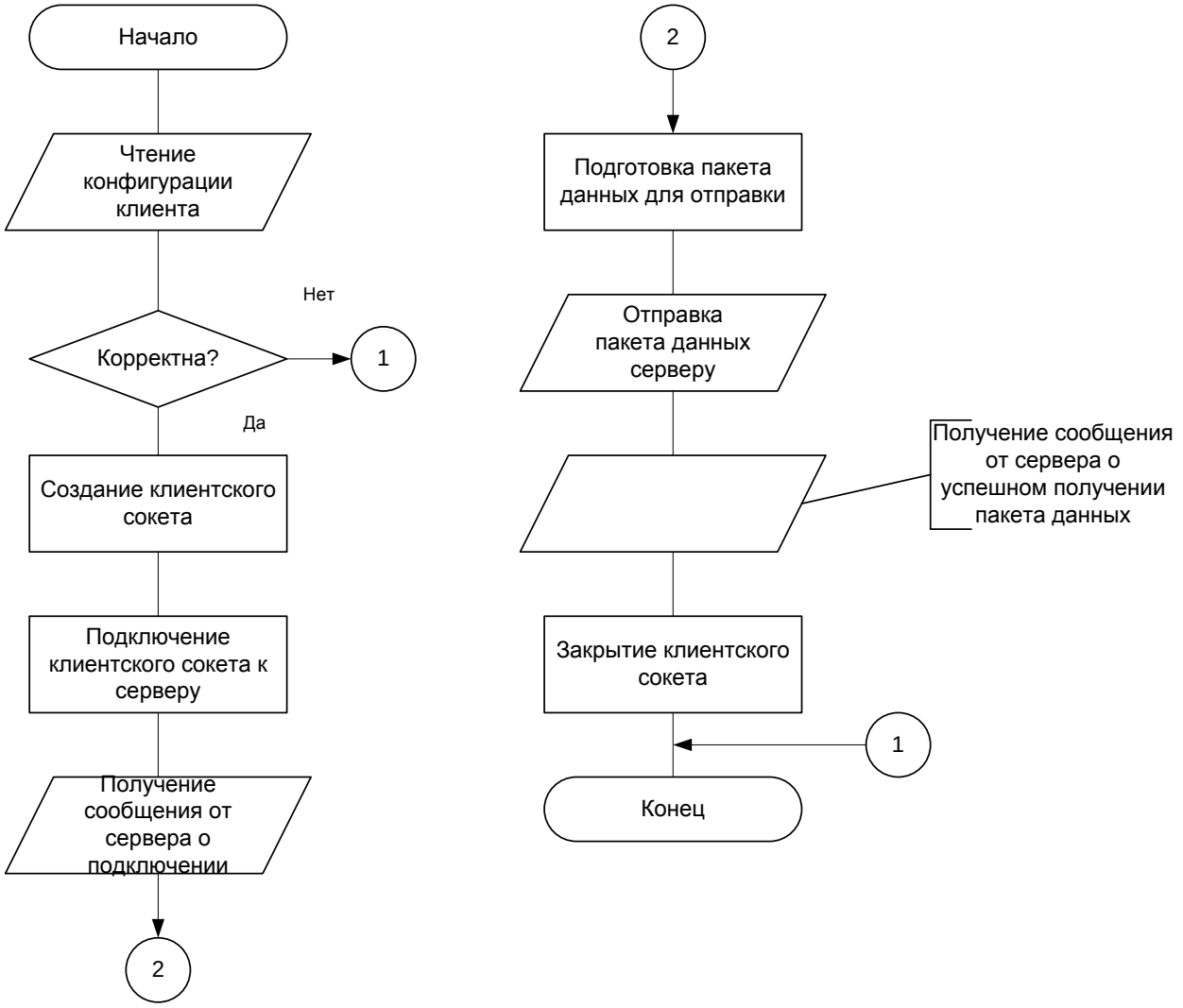

Рис. 10. Блок-схема алгоритма основного потока клиента

Далее происходит создание клиентского сокета и привязка его к IP адресу. В случае успешного создания клиентского сокета производится подключение к серверу. После успешного подключения к серверу, ожидается получение пакета от сервера с информацией об успешности подключения. Далее, клиент формирует пакеты заданного размера, согласно закона формирования, а также производит отправку сформированных пакетов, через заданные интервалы времени, согласно закона формирования.

После успешной отправки, от сервера получается пакет с информацией об успешном приеме пакетов данных. Перед завершением, клиентское приложение закрывает клиентский сокет.

Далее рассмотрим особенности программной реализации алгоритмов клиента и сервера для ОС РВ.

Программная реализация

Обзор

При разработке приложения, предполагается использовать среду разработки QNX Momentics v4.7, данная среда разработки получена от компании разработчика по академической лицензии для кафедры безопасности информационных технологий Национального авиационного университета. В качестве языка программирования будет использоваться язык высокого уровня C++.

Тестирование приложения и эксперименты будет производиться на реальной ОС РВ QNX 6.5 Neutrino v2, развернутой на виртуальной машине VMWare. Данная виртуальная

машина, также получена по академической лицензии от компании разработчика по академической лицензии для кафедры безопасности информационных технологий Национального авиационного университета.

Сервер

Предполагается передача конфигурации сервера через командную строку.

server -addr:<ipaddress> –port:<port> –maxconn:<maxconn> –log:<logfilename>

где

- -addr:<ipaddress> - указывает IP адрес, на котором сервер «слушает» клиентские подключения.
- -port:<port> - указывает номер порта, на котором сервер «слушает» клиентские подключения.
- –maxconn:<maxconn> - указывает максимальное количество одновременных подключений клиентов к серверу.
- –log:<logfilename> - указывает имя файла лога, куда пишется информация о пакетах, полученных от клиентов.

При программной реализации серверного приложения необходимо:

- Организовать сетевое взаимодействие. За основу следует взять сокеты BSD формата.
- Заложить возможность одновременной обработки нескольких пакетов от множества клиентов. Это подразумевает многопоточную обработку пакетов, другими словами, для обслуживания каждого клиента предполагается создать собственный поток. За основу следует взять Posix реализацию потоков.

Сетевое взаимодействие

Сетевое взаимодействие реализуется посредством системных вызовов (необходимо коротко описать каждую функцию):

- socket().
- setsockopt().
- connect().
- bind().
- listen().
- accept().
- send().
- recv(). И варианты этой команды: recvfrom().
- read().
- write().

Многопоточная обработка

Многопоточную обработку реализуется посредством системных вызовов Posix нотации, которые представлены в библиотеке pthread (необходимо коротко описать каждую функцию):

- Функции синхронизации:
  - pthread_mutex_init().
  - pthread_mutex_lock().
  - pthread_mutex_unlock().
  - pthread_mutex_destroy().
  - pthread_mutex_trylock().
  - pthread_join().
- Функции управления потоками:
  - pthread_create().
  - pthread_exit().

Фиксация времени обработки пакета предполагается посредством системного вызова clock().

Каждый клиент имеет таймер работающий в отдельном потоке определенное время (согласно варианту – 1200 секунд). Работа таймера базируется на функции clock(void) (заголовочный файл time.h), которая возвращает время, измеряемое процессором в

тактах от начала выполнения программы, или −1, если оно не известно. Пересчет этого времени в секунды выполняется по формуле clock () / CLOCKS_PER_SEC.

Часть материала, который может подойти для описания необходимых вызовов.

Для создания сокета в операционной системе служит системный вызов socket(). Для транспортных протоколов семейства TCP/IP существует два вида сокетов: UDP-сокет – сокет для работы с датаграммами, и TCP-сокет – потоковый сокет. Однако понятие сокета не ограничивается рамками только этого семейства протоколов.

При создании сокета необходимо точно специфицировать его тип. Эта спецификация производится с помощью трех параметров вызова socket(). Первый параметр указывает, к какому семейству протоколов относится создаваемый сокет, а второй и третий параметры определяют конкретный протокол внутри данного семейства.

Второй параметр служит для задания вида интерфейса работы с сокетом – будет это потоковый сокет, сокет для работы с датаграммами или какой-либо иной. Третий параметр указывает протокол для заданного типа интерфейса. В стеке протоколов TCP/IP существует только один протокол для потоковых сокетов – TCP и только один протокол для датаграммных сокетов – UDP, поэтому для транспортных протоколов TCP/IP третий параметр игнорируется.

Для транспортных протоколов TCP/IP в качестве первого параметра указывается предопределенная константа AF_INET (Address family – Internet) или ее синоним PF_INET (Protocol family – Internet) в некоторых вариациях (для IPv4 и IPv6).

Второй параметр принимает значения SOCK_STREAM для потоковых сокетов и SOCK_DGRAM – для датаграммных.

Для привязки сервера к IP-адресу и номеру порта используется системный вызов bind(). Для процесса клиента эта привязка объединена с процессом установления соединения с сервером в новом системном вызове connect() и скрыта от глаз пользователя. Внутри этого вызова операционная система осуществляет настройку сокета на выбранный ею порт и на адрес любого сетевого интерфейса. Для перевода сокета на сервере в пассивное состояние и для создания очереди соединений служит системный вызов listen(). Сервер ожидает соединения и получает информацию об адресе соединившегося с ним клиента с помощью системного вызова accept().Поскольку установленное логическое соединение выглядит со стороны процессов как канал связи, позволяющий обмениваться данными с помощью потоковой модели, для передачи и чтения информации оба системных вызова используют системные вызовы send() и recv(), а для завершения соединения – системный вызов close().

Среди системных вызовов со стороны клиента появляется только один новый – connect(). Системный вызов connect() при работе с TCP-сокетами служит для установления логического соединения со стороны клиента. Вызов connect() скрывает внутри себя настройку сокета на выбранный системой порт и произвольный сетевой интерфейс. Вызов блокируется до тех пор, пока не будет установлено логическое соединение, или пока не пройдет определенный промежуток времени.

Для установления соединения необходимо задать три параметра: дескриптор активного сокета, через который будет устанавливаться соединение, полный адрес сокета сервера и его длину.

Системный вызов connect служит для организации связи клиента с сервером. Чаще всего он используется для установления логического соединения, хотя может быть применен и при связи с помощью датаграмм (connectionless).

Параметр sockd является дескриптором созданного ранее коммуникационного узла, т. е. значением, которое вернул системный вызов socket().

Параметр servaddr представляет собой адрес структуры, содержащей информацию о полном адресе сокета сервера. Он имеет тип указателя на структуру-шаблон struct sockaddr, которая должна быть конкретизирована в зависимости от используемого семейства протоколов и заполнена перед вызовом.

Параметр addrlen должен содержать фактическую длину структуры, адрес которой передается в качестве второго параметра.

Системный вызов listen используется сервером, ориентированным на установление связи путем виртуального соединения, для перевода сокета в пассивный режим и установления глубины очереди для соединений.

Параметр sockd является дескриптором созданного ранее сокета, который должен быть переведен в пассивный режим, т. е. значением, которое вернул системный вызов socket(). Системный вызов listen требует предварительной настройки адреса сокета с помощью системного вызова bind().

Параметр backlog определяет максимальный размер очередей для сокетов, находящихся в состояниях полностью и не полностью установленных соединений.

Системный вызов accept используется сервером, ориентированным на установление связи путем виртуального соединения, для приема полностью установленного соединения.

Параметр sockd является дескриптором созданного и настроенного сокета, предварительного переведенного в пассивный (слушающий) режим с помощью системного вызова listen().

Системный вызов accept требует предварительной настройки адреса сокета с помощью системного вызова bind().

Параметр cliaddr служит для получения адреса клиента, установившего логическое соединение, и должен содержать указатель на структуру, в которую будет занесен этот адрес.

Параметр clilen содержит указатель на целую переменную, которая после возвращения из вызова будет содержать фактическую длину адреса клиента.

Порядок осуществления создания серверного сокета и ожидания подключения клиентов:

- Создание сокета, посредством системного вызова socket. Здесь указывается тип подключения: TCP|UDP.
- Связывание сокета с заданным сетевым интерфейсом, через вызов bind. Здесь указывает IP адрес и порт.
- Начало прослушивания сокета для возможного подключения клиентов, через вызов listen.
- В случае успешного подключение клиента, производится создание нового потока через вызов pthread_create, в котором производится присоединение клиентского сокета к серверному, через вызов accept (возвращает сокет текущего подключения клиента).
- Асинхронная передача информации между клиентом и сервером осуществляется в рамках вновь созданного потока, через системные вызовы read/write, также можно использовать recv*/send*.

Логирование

Согласно задания, серверу необходимо заносит информацию о полученных и обработанных пакетах от всех клиентов в некий журнал. В данном случае, в качестве журнала используется обычный текстовый файл, в который построчно пишется информации необходимая информация:

- Номер полученного пакета от клиента (указывается IP адрес клиента).
- Идентификатор клиента.
- Время начала приема пакета.
- Время окончания приема пакета.
- Период времени, за который обрабатывался (принимался) пакет.
- Размер пакета.

Отметим, что информация в журнал заносится уже после полной обработки (приема) пакета.

Клиент

Предполагается передача конфигурации клиента через командную строку.

client -saddr:<serveripaddress> –sport:<serverport> –caddr:<clientipaddress> -tp:<timeperiod> -tk:<ktimeparam> -tb:<btimeparam> -maxsize:<maxpacketsize> -pk:<kpacketparam> -pb:<bpacketparam>

где

- -saddr:<serveripaddress> - указывает IP адрес, на котором сервер «слушает» клиентские подключения.
- -sport:<serverport> - указывает номер порта, на котором сервер «слушает» клиентские подключения.
- -caddr:<clientipaddress> - указывает IP адрес, с которого клиент подключается к серверу.
- –tp:<timeperiod> - указывает период моделирования в мс, т.е. промежуток времени, в время которого клиент формирует запросы серверу.
- –tk:<ktimeparam> - указывает параметр k для закона изменения времени между пакетами, отправляемыми к серверу.
- -tb:<btimeparam> - указывает параметр b для закона изменения времени между пакетами, отправляемыми к серверу.
- -maxsize:<maxpacketsize> - максимальный размер пакета в килобайтах.
- -pk:<kpacketparam> - указывает параметр k для закона изменения размера пакетов, отправляемым к серверу.
- -pb:<bpacketparam> - указывает параметр b для закона изменения размера пакетов, отправляемым к серверу.

Сетевое взаимодействие

Более детально об использованных библиотеках и системных вызовах, можно прочесть в соответствующем разделе сервера.

Порядок выполнения подключения к серверу:

- Создание сокета, посредством системного вызова socket. Здесь указывается тип подключения: TCP|UDP.
- Подключение к серверному сокету, посредством вызова connect.
- Далее производится асинхронная передача информации между клиентом и сервером посредством вызовов read/write.
- После чего закрывается соединение со стороны клиента и сервера, вызовом close.

Обработка ошибок

Обработка ошибок производится посредством обработки исключительных ситуаций, а также проверки состояния после выполнения соответствующих системных вызовов.

Эксперимент

Описание экспериментальной установки

По причине отсутствия необходимого количества доступных компьютеров для установки OS QNX, а также отсутствия необходимых лицензий для установки OS QNX, предполагается использование виртуальных машин VMWare, запущенных на компьютерах под управлением ОС Windows XP SP3.

Далее остановимся на конфигурации компьютеров и программном обеспечении задействованных в эксперименте.

Аппаратное обеспечение

В случае наличия очень мощного многопроцессорного и многоядерного компьютера, допускается моделирование распределенной вычислительной системы с помощью набора виртуальных машин функционирующих в рамках монитора виртуальных машин VMWare посредством виртуальной сети VMWare Ethernet. Для всех клиентов и сервера использовался один и тот же образ виртуальной машины. В настройках монитора виртуальных машин VMWare Workstation для всех сетевых адаптеров было установлено Network Adapter было выполнено соединение Briged.

Компьютер	Описание

Сервер	CPU: Intel(R) Core(TM)2 CPU T7200 @ 2.00GHz RAM: DDR2 2Gb HDD: Seagate 500 Gb SATA II ST9500420AS LAN: Gigabit Ethernet Marvell Yukon 88E8055
	Виртуальная машина: CPU: 2 cores RAM: 512 Мб HDD: 400 Мб LAN: VMWare Fast Ethernet
Рабочая станция 1 (Виртуальная машина)	CPU: 2 cores RAM: 512 Мб HDD: 400 Мб LAN: VMWare Fast Ethernet
Рабочая станция 2 (Виртуальная машина)	CPU: 2 cores RAM: 512 Мб HDD: 400 Мб LAN: VMWare Fast Ethernet
Рабочая станция 3 (Виртуальная машина)	CPU: 2 cores RAM: 512 Мб HDD: 400 Мб LAN: VMWare Fast Ethernet
Рабочая станция 4 (Виртуальная машина)	CPU: 2 cores RAM: 512 Мб HDD: 400 Мб LAN: VMWare Fast Ethernet
Коммутатор (имитируется монитором виртуальных машин)	Gigabit Ethernet Switch

Программное обеспечение

Компьютер	Описание
Сервер	OS: Microsoft Windows XP Professional Server Service Pack 3 Монитор: VMWare Workstation 7.0 Виртуальная ОС: QNX 6.5 Neutrino v2
Клиент 1	OS: Microsoft Windows 2003 Server Service Pack 3 Монитор: VMWare Workstation 7.0 Виртуальная ОС: QNX 6.5 Neutrino v2
Клиент 2	OS: Microsoft Windows 2003 Server Service Pack 3 Монитор: VMWare Workstation 7.0 Виртуальная ОС: QNX 6.5 Neutrino v2
Клиент 3	OS: Microsoft Windows 2003 Server Service Pack 3

	Монитор: VMWare Workstation 7.0 Виртуальная ОС: QNX 6.5 Neutrino v2
Клиент 4	OS: Microsoft Windows 2003 Server Service Pack 3 Монитор: VMWare Workstation 7.0 Виртуальная ОС: QNX 6.5 Neutrino v2

Результаты эксперимента

Для проведения эксперимента была развернута виртуальная Gigabit Ethernet сеть, которая формируется монитором виртуальных машин VMWare, с описанной конфигурацией и программным обеспечением.

Порядок выполнения эксперимента:

1. Запуск монитора виртуальных машин VMWare Workstation 7.1.

2. Запуск виртуальной машины сервера и запуск серверного приложения.

```
server –addr:192.168.1.67 –port:8099 –maxconn:4 –log:log.svr
```

где

- команда -addr:192.168.1.67 указывает адрес, на котором сервер «слушает» клиентские подключения.
- команда -port:8099 указывает номер порта, на котором сервер «слушает» клиентские подключения.
- команда –maxconn:4 указывает максимальное количество одновременных подключений клиентов к серверу.
- команда –log:log.svr указывает имя файла лога, куда пишется информация о пакетах, полученных от клиентов.

3. Запуск виртуальной машины клиента 1.

4. Запуск виртуальной машины клиента 2.

5. Запуск виртуальной машины клиента 3.

6. Запуск виртуальной машины клиента 4.

7. Последовательный запуск клиентов 1-4 для подключения к серверу.

```
client –saddr:192.168.1.67 –sport:8099 –caddr:192.168.1.69 -tp:15 –tk:25 -
tb:70 –maxsize:3072 -pk:1600 –pb:128
```

где

- -saddr:192.168.1.67 - указывает IP адрес, на котором сервер «слушает» клиентские подключения.
- -sport:8099 - указывает номер порта, на котором сервер «слушает» клиентские подключения.
- -caddr:192.168.1.69 - указывает IP адрес, с которого клиент подключается к серверу.
- –tp:15 - указывает период моделирования в мс, т.е. промежуток времени, в время которого клиент формирует запросы серверу.
- –tk:25 - указывает параметр k для закона изменения времени между пакетами, отправляемыми к серверу.
- -tb:70 - указывает параметр b для закона изменения времени между пакетами, отправляемыми к серверу.
- -maxsize:3072 - максимальный размер пакета в килобайтах.
- -pk:1600 - указывает параметр k для закона изменения размера пакетов, отправляемым к серверу.
- -pb:128 - указывает параметр b для закона изменения размера пакетов, отправляемым к серверу.

```
client –saddr:192.168.1.67 –sport:8099 –caddr:192.168.1.70 -tp:15 –tk:25 -
tb:70 –maxsize:3072 -pk:1600 –pb:128
```

```
client –saddr:192.168.1.67 –sport:8099 –caddr:192.168.1.71 -tp:15 –tk:25 -
tb:70 –maxsize:3072 -pk:1600 –pb:128
```

```
client –saddr:192.168.1.67 –sport:8099 –caddr:192.168.1.72 -tp:15 –tk:25 -
tb:70 –maxsize:3072 -pk:1600 –pb:128
```

Управление клиентами, проводилось через удаленное соединение, посредством IDE QNX Momentics через вкладку Target Navigator. Отметим, что для этого на каждой удаленной машине, необходимо запустить `qconn`. Узнать IP адрес каждого клиента можно посредством выполнения команды `ifconfig`.

На рис. 11 приведена диаграмма загрузки процессора сервера во время моделирования.

Диаграмма

Рис. 11. Диаграмма загрузки процессора сервера

На рис. 12 приведена диаграмма количества принимаемых пакетов в единицу времени, во время моделирования.

Диаграмма

Рис. 12. Диаграмма загрузки сетевого соединения сервера

В таблице 2 представлены варианты одновременного приема нескольких пакетов разной длины и время необходимо для их обработки процессором (вычислительной системой в целом, если она многопроцессорная).

Таблица 2. Соотношение времени обработки (приема) нескольких пакетов и загрузки процессора

№	Число пакетов	Загрузка процессора, мс
1		
2		
…		

На рис. 13 приведена зависимость времени обработки сервером пакетов разной длины.

Диаграмма

Рис. 13. График зависимости времени обработки пакетов различной длины

В таблице 3 представлены все размеры пакетов и усредненное время обработки пакетов данной длины процессором (вычислительной системой в целом, если она многопроцессорная).

Таблица 3. Соотношение времени обработки (приема) пакета и загрузки процессора

№	Размер пакета	Загрузка процессора, мс
1		
2		
…		

Из данных, приведенные на рис. 13 и таблицы 2, можем заметить, что время, затраченное на обработку, процессором прямо пропорционально размеру пакета.

На рис. 14 приведена зависимость времени одновременной обработки сервером пакетов разной длины, что позволяет судить об общей производительности сервера в Мб/с.

Диаграмма

Рис. 14. График зависимости времени одновременной обработки пакетов сервером

В таблице 3 представлен суммарный размер одновременно принимаемых пакетов и время обработки заданного объема информации процессором (вычислительной системой в целом, если она многопроцессорная).

Таблица 4. Соотношение времени обработки (приема) объема информации и загрузки процессора

№	Суммарный размер пакетов, Мб	Загрузка процессора, мс
1		
2		
…		

Из данных, приведенные на рис. 14 и таблицы 3, можем заметить, что время, затраченное на обработку, процессором прямо пропорционально обрабатываемой (принимаемой) объему информации.

Коротко остановимся на результатах эксперимента:

- Время моделирования: 000 с.
- Каждым клиентом было сформировано и отправлено пакетов, соответственно: 000 шт.
- Общее количество отправленных клиентами: 000 шт.
- Пропускная способность сервера: 000 Мб/с.

Что бы сделать выводы о возможной пропускной способности (количество пакетов в единицу времени) сервера, необходимо сопоставить графики загрузки процессора и количества пакетов, принимаемых в конкретный момент времени во время моделирования. Из графиков на рис. 11-12, видно, что производительности процессора сервера достаточно, для обработки количества пакетов, передаваемых во время моделирования и его загрузка не превышает 10%, исходя из этого, можно предположить, что данная конфигурация способна обработать 000 пакетов в единицу времени, что соответствует 000 одновременно функционирующим клиентам.

Выводы

В результате выполнения лабораторной работы можно сделать следующие выводы:

1. Во время выполнения расчетно-графической работы, были использованы наработки лабораторной работы № 14 «Разработка серверного и консольного клиентского приложения для обмена сообщения с ним для QNX в среде QNX Mnemonics IDE».

2. Разработано клиент-серверное приложение на языке C++ для управления в реальном времени сложным технологическим процессом.

3. Проведен эксперимент, в котором использовались технологии виртуализации VMWare, для моделирования распределенной системой управления сложным технологическим процессом в реальном времени.

4. Данная вычислительная система, позволяет принимать одновременно до 000 (необходимо посчитать исходя из текущей картины) пакетов от 000 (необходимо посчитать исходя из текущей картины) клиентов, без изменений конфигурации оборудования.

5. Пропускная способность сервера во время эксперимента составляет: 000 Мб/с, отметим, что данный сервер теоретически способен дать пропускную способность 000 Мб/с.

6. Другие выводы, которые можно сделать на основе экспериментальной информации …

Литература

1. Microsoft Developer Network. URL: http://www.msdn.com

2. Формирование документации к исходному коду с помощью средства doxygen. URL: www.nrjetix.com/r-and-d/lectures

3. Общие сведения о работе с сетями Ethernet. URL: http://docstore.mik.ua/manuals/ru/linux_parallel/node77.html

4. Beej's Guide to Network Programming. Using Internet Sockets. URL: http://www.beej.us/guide/bgnet/

5. Стивенс Р., Раго С. Unix. Професиональное программирование, 2-е издание. -СПб.: Символ-Плюс, 2007. -1040 с.

6. Стивенс У.Р., Феннер Б., Рудофф Э.М. Unix: разработка сетевых приложений. 3-е издание. - СПб.: Питер, 2007. -1039 с.

7. Алексеев Д., Ведревич Е., Волков А., и др. Практика работы с QNX. -М.: Издательский Дом «КомБук», 2004. -432с.

8. Кёртен Р. Введение в QNX Neutrino 2. Руководство по программированию приложений реального времени в QNX Realtime Platform. -Издательство «Петрополис», 2001.

9. Другие источники литературы …