



ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ
КВАЛИФИКАЦИИ

Кафедра «Программное обеспечение вычислительной тех-
ники и автоматизированных систем»

Учебно-методическое пособие по дисциплине

«РАСПРЕДЕЛЕННЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ»

Автор
Кудинов Н. В.

Ростов-на-Дону, 2019



Аннотация

Учебно-методическое пособие предназначено для студентов очной формы обучения направления 09.04.04 «Программная инженерия».

Авторы

К.т.н.,
доцент каф. ПОВТиАС
Кудинов Н.В.



Оглавление

1. Лабораторная работа №1: Исследования приринговой файлообменной сети	5
1.1. Цель работы.....	5
1.2. Общие сведения.....	5
1.3. Принцип работы протокола.....	6
1.4. Алгоритм обмена данными	7
1.5. Ход выполнения работы.....	9
2. Лабораторная работа №2: Исследование трафика P2P сетей	10
2.1. Цель работы.....	10
2.2. Ход выполнения работы.....	10
3. Лабораторная работа №3: Установка и настройка системы обмена мгновенными сообщениями по протоколу XMPP	11
3.1. Цель работы.....	11
3.2. Общие сведения.....	11
Произведём конфигурирование сервера:.....	12
Добавим службу в автозапуск и запустим.....	13
3.3. Порядок выполнения работы	13
4. Лабораторная работа №4: Распределённые приложения на языке Erlang	15
4.1. Цель работы.....	15
4.2. Общие сведения.....	15
Модули	19
Компиляция	22
4.3. Задание.....	28
5. Лабораторная работа №5: Исследование P2P подсистемы .NET.....	29
5.1. Общие сведения.....	29
5.2. Обзор корпоративных технологий P2P	30
5.3. Архитектура типа “клиент-сервер”	30
5.4. Архитектура типа P2P	32
5.5. Особенности архитектуры P2P	33
5.6. Терминология P2.....	35

5.7. Решения P2P	36
5.8. Платформа Microsoft Peer-to-Peer Networking	36
Протокол PNRP	37
Служба People Near Me	42
5.9. Создание приложений P2P	43
Общая инфраструктура	43
5.10. Преобразование имён равноправных участников.....	46
5.11. Безопасный доступ кода в System.Net.PeerToPeer	49
Пример приложения P2P.....	49
5.12. Пространство имён	
System.Net.PeerToPeer.Collaboration	60
Осуществление входа и выхода	61
Обнаружение соседей.....	62
Управление контактами и определение присутствия соседей в сети.....	62
5.13. Задание.....	64
6. Лабораторная работа №6: Распределённые файловые системы	64
6.1. Цель работы.....	64
6.2. Задание.....	64

1. ЛАБОРАТОРНАЯ РАБОТА №1: ИССЛЕДОВАНИЯ ПРИРИНГОВОЙ ФАЙЛООБМЕННОЙ СЕТИ

1.1. Цель работы

Получить навыки настройки и эксплуатации централизованной приринговой сети.

1.2. Общие сведения

В файлообменных р2р сетях файлы передаются частями, каждый узел сети, участвуя в обмене, получает эти части, в то же время передаёт их другим узлам, что снижает нагрузку и зависимость от каждого клиента-источника и обеспечивает необходимую избыточность данных.

Протокол был создан Брэмом Коэном, написавшим первый torrent-клиент «BitTorrent» на языке Python 4 апреля 2001 года. Запуск первой версии состоялся 2 июля 2001 года.

БитТоррент (англ. BitTorrent) это протокол, разработанный для кооперативного обмена файлами через сеть Интернет. По сути, он является р2р-протоколом, где каждый узел сети устанавливает связь с другим напрямую для приема или передачи информации. Принципиальное отличие БитТоррента от других р2р-систем в том, что имеется центральный сервер, координирующий все существующие связи между пользователями. Благодаря этому, пропускная способность сети используется наилучшим образом. При этом, чем большее количество людей участвует в процессе конкретной раздачи, тем большую битовую скорость может получить каждый из них.

Файл-идентификатор, который содержит в себе информацию о запрошенных файлах, а именно, о размере и количестве фрагментов, и контрольной сумме скачиваемого файла(ов), о трекере на котором можно получить информацию об участниках файлообмена называется Торрентом (англ. Torrent).

Центральный сервер, который является координатором действий всех существующих связей между пользователями называется Трекером (англ. Tracker). На трекере хранятся все торрент-файлы, и он обеспечивает их распространение и связь между ними во время файлообмена. Трекер только управляет

соединениями, но не содержит самих файлов, участвующих в обмене. Он лишь предоставляет торрент-клиентам информацию об участниках файлообмена, т.е. является связующим звеном между ними.

Программа, с помощью которой возможен файлообмен по протоколу БитТоррент называют Торрент-клиентом (англ. Torrent Client).

1.3. Принцип работы протокола

Принцип работы BitTorrent: нагрузка на распространителя файла уменьшается благодаря тому, что клиенты начинают обмениваться данными сразу же, даже если файл не докачан ими до конца.

Перед началом скачивания клиент подсоединяется к трекеру по адресу, указанному в торрент-файле, сообщает ему свой адрес и хеш-сумму торрент-файла, на что в ответ клиент получает адреса других клиентов, скачивающих или раздающих этот же файл. Далее клиент периодически информирует трекер о ходе процесса и получает обновлённый список адресов. Этот процесс называется объявлением (англ. announce).

Клиенты соединяются друг с другом и обмениваются сегментами файлов без непосредственного участия трекера, который лишь хранит информацию, полученную от подключенных к обмену клиентов, список самих клиентов и другую статистическую информацию. Для эффективной работы сети BitTorrent необходимо, чтобы как можно больше клиентов были способны принимать входящие соединения. Неправильная настройка NAT или брандмауэра могут этому помешать.

При соединении клиенты сразу обмениваются информацией об имеющихся у них сегментах. Клиент, желающий скачать сегмент (личер), посылает запрос и, если второй клиент готов отдавать, получает этот сегмент. После этого клиент проверяет контрольную сумму сегмента. Если она совпала с той, что записана в торрент-файле, то сегмент считается успешно скачанным, и клиент оповещает всех присоединенных пиров о наличии у него этого сегмента. Если же контрольные суммы различаются, то сегмент начинает скачиваться заново. Некоторые клиенты банят тех пиров, которые слишком часто отдают некорректные сегменты.

Таким образом, объём служебной информации (размер торрент-файла и размер сообщений со списком сегментов)

напрямую зависит от количества, а значит, и размера сегментов. Поэтому при выборе сегмента необходимо соблюдать баланс: с одной стороны, при большом размере сегмента объём служебной информации будет меньше, но в случае ошибки проверки контрольной суммы придётся скачивать ещё раз больше информации. С другой стороны, при малом размере ошибки не так критичны, так как необходимо заново скачать меньший объём, но зато размер торрент-файла и сообщений об имеющихся сегментах становится больше.

1.4. Алгоритм обмена данными

Каждый клиент имеет возможность временно заблокировать отдачу другому клиенту (англ. choke). Это делается для более эффективного использования канала отдачи. Кроме того, при выборе — кого разблокировать, предпочтение отдаётся пирам, которые сами передали этому клиенту много сегментов. Таким образом, пиры с хорошими скоростями отдачи поощряют друг друга в соответствии с паритетным принципом.

Обмен сегментами ведётся по паритетному принципу симметрично в двух направлениях. Клиенты сообщают друг другу об имеющихся у них сегментах при подключении и затем при получении новых сегментов, и поэтому каждый клиент может хранить информацию о том, какие сегменты есть у других подключенных пиров. Порядок обмена выбирается таким образом, чтобы сначала клиенты обменивались наиболее редкими сегментами: таким образом повышается доступность файлов в раздаче. В то же время выбор сегмента среди наиболее редких случаен, и поэтому можно избежать ситуации, когда все клиенты начинают скачивать один и тот же самый редкий сегмент, что негативно бы отразилось на производительности.

Обмен данными начинается, когда обе стороны в нём заинтересованы, то есть, каждая из сторон имеет сегменты, которых нет у другой. Количество переданных сегментов подсчитывается, и если одна из сторон обнаруживает, что передаёт в среднем больше, чем принимает, она блокирует (англ. choke) на некоторое время отдачу другой стороне. Таким образом, в протокол заложена защита от личеров.

Сегменты делятся на блоки размером 16-4096 килобайт, и каждый клиент запрашивает именно эти блоки. Одновременно могут запрашиваться блоки из разных сегментов. Более того, некоторые клиенты поддерживают скачивание блоков одного

сегмента у разных пиров. В этом случае описанные выше алгоритмы и механизмы обмена применимы и к уровню блоков.

rTorrent — один из самых популярных torrent клиентов в мире Unix-подобных операционных систем (файлообменный клиент Transmission, µTorrent также популярен). Использование mmap для отображения файлов в память позволяет добиться на широкополосных каналах трехкратного преимущества перед официальным клиентом bittorrent. Еще одной характерной чертой rTorrent является наличие множества различных web-интерфейсов для удаленного управления.

Создание хэш-таблицы для поиска данных в файлообменной сети

Параметры запуска утилиты ctorrent

-t	Создать новый torrent файл. Режим создания нового фала. Это должна быть первая опция при создании хэш таблицы в torrent-файле.
-s filename	Задаёт имя файла для метаданных (обязательно), передаётся имя файла, для которого создаётся хэш-таблица
-u URL	URL трекера. Обязательный параметр, параметр часто задаётся в виде структуры "http://tracker.example.com:port/announce".
-l piece_len	размер фрагмента (default 262144). Указывает размер фрагмента для создаваемого торрент-файла. Этот параметр также определяет число фрагментов в торренте.
-p	Параметр приватности, запрещает заменять пилов в процессе загрузки. Запрещает использовать DHT, PEX, или другой не централизованный метод смены пилов.
-c comment	Описание к публикации.

Работа выполняется под группами студентов по 3 человека. Один контролирует торрент-трекер и его трафик, два других производят манипуляции с клиентским ПО.

1.5. Ход выполнения работы

1. Запустить под управлением ПО VirtualBox виртуальную машину на основе операционной системы GNU/Linux, Ubuntu для организации среды работы трекера и организации работы клиентских программ.

2. Найти проект битторрент клиента «ctorrent» и попытаться его скомпилировать в среде msys2. Если сборка неудачна, установить пакет ctorrent в Ubuntu. То же самое сделать для rtorrent и opentracker.

Получить исх. код opentracker командой: `git clone https://github.com/bazaarvoice/opentracker`

1. Провести эксперименты с созданием незначащего контента и торрент-файла его сопровождающего

```
sudo dd if=/dev/zero of=/file bs=1024 count=65536
```

```
или fsutil file createnew \file 67108864
```

```
sudo ctorrent -t /file -s /file.torrent -u  
udp://192.168.43.63/announce -l 65536
```

```
или ctorrent -t -u "http://192.168.43.161:6969/announce" -s  
pic.torrent Firewatch.jpg
```

3. Запустить раздачу «контента»

```
sudo rtorrent /file.torrent
```

или

```
ctorrent -D 0 -U 0 -e 500 pic.torrent
```

4. Запустить tcpdump и netstat -anp и определить, какие соединения пытается установить файлообменный клиент. Сделать вывод.

5. Активировать opentracker, предварительно изучив его интерфейс: опция `—help` (вероятные параметры запуска `-i 192.168.43.161 -p 6969`), исправить адрес анонсирования в torrent-файле.

6. На другом компьютере ЛВС также запустить клиентское ПО

```
sudo rtorrent /file.torrent
```

и наблюдать передачу контента по сети. Если процесс не идёт, попробовать установить `transmission-gtk`, создать в этом приложении torrent-файл и повторить попытку передачи.

7. Для проверки корректности передачи файла проверить HASH-сумму файла и размер, убедиться в наличии файла в локальной файловой системе.

Примечание: torrent-файл лучше всего передать посредством протокола ssh. Для этого в сетевую ОС нужно установить ssh-сервер командой `apt install openssh-server`. А файл

передавать при помощи утилиты тс или через nautilus (файловый проводник).

Содержание отчёта:

1. Цель работы;
2. Подробное описание хода выполнения работы, с выводами касающихся обобщений наблюдаемых фактов;
3. Выводы по работе.

2. ЛАБОРАТОРНАЯ РАБОТА №2: ИССЛЕДОВАНИЕ ТРАФИКА P2P СЕТЕЙ

2.1. Цель работы

Получить навыки моделирования и идентификации моделей процессов сетевого паритетного обмена информацией; освоить технологию регистрации характеристик сетевого трафика и технику его анализа. *Задание к лабораторной работе*

На лабораторном стенде, разработанном в ходе выполнения лабораторной работы №1 реализовать процесс регистрации статистики паритетного обмена в ходе функционирования файлообменной сети. Для анализа и регистрации параметров структуры трафика использовать opensource (GPL v.2) ПО WireShark, iptraph (ng), tcpdump. Зафиксировать таблицы параметров в энергонезависимой памяти и провести расчёт расходных характеристик трафика. Исследование нужно проводить на максимально возможном количестве реер-ов (участников файлообменной сети), на начало эксперимента анонсируемый файл должен находиться на одном раздающем узле (seed), описание контента передать по сети по протоколу ssh.

2.2. Ход выполнения работы

1. Сконфигурировать файлообменную сеть
2. Запустить параллельными процессами с файлообменными клиентами ПО сбора статистики трафика WireShark, основанного на использовании библиотеки libpcap.
3. Реализовать раздачу файла.
4. Зафиксировать таблицы параметров в файловой системе.
5. Передать файлы таблиц на выделенную машину, на

которой запустить octave и загрузить таблицы в память, используя функцию форматного чтения `sscanf`.

6. Найти суммарную интенсивность обменных информационных потоков в пиринговой сети, составив и решив линейную систему уравнений, состоящую из уравнений каждого из узлов сети.

7. Решить линейную систему уравнений в среде Octave, используя функцию `inv` или матричное деление «/». Найти процент загрузки каналов передачи информации от теоретически возможной.

8. В среде пакета Octave промоделировать взаимодействие большего количества клиентов с теми же характеристиками. Определить максимальное количество обменных узлов, обеспечивающих полную загрузку каналов в сети в данной топологией.

Содержание отчёта:

1. Цель работы;
2. Подробное описание хода выполнения работы, с выводами касающихся обобщений наблюдаемых фактов;
3. Схема сетевой топологии;
4. Таблицы интенсивностей обмена;
5. Математические модели, выкладки, результаты;
6. Выводы по работе.

3. ЛАБОРАТОРНАЯ РАБОТА №3: УСТАНОВКА И НАСТРОЙКА СИСТЕМЫ ОБМЕНА МГНОВЕННЫМИ СООБЩЕНИЯМИ ПО ПРОТОКОЛУ XMPP

3.1. Цель работы

Получить навыки конфигурирования централизованно-распределённой сети передачи сообщений

3.2. Общие сведения

XMPP (Extensible Messaging and Presence Protocol — расширяемый протокол обмена сообщениями и информацией о присутствии), ранее известный как Jabber — основанный на

XML, открытый, свободный для использования протокол для мгновенного обмена сообщениями и информацией о присутствии (см. список контактов) в режиме, близком к режиму реального времени. Изначально спроектированный легко расширяемым, протокол, помимо передачи текстовых сообщений, поддерживает передачу голоса, видео и файлов по сети.

В отличие от коммерческих систем мгновенного обмена сообщениями, таких как AIM, ICQ, WLM и Yahoo, XMPP является децентрализованной, расширяемой и открытой системой. Любой желающий может открыть свой сервер мгновенного обмена (шлюз, маршрутизатор, координатор) сообщениями, зарегистрировать на нём пользователей и взаимодействовать с другими серверами XMPP. На основе протокола XMPP уже открыто множество частных и корпоративных серверов XMPP. Среди них есть достаточно крупные проекты, такие как Facebook, Google Talk, Одноклассники.ru, QIP, LiveJournal, Juick и др.

Ejabberd - это высокопроизводительный, широко распространенный и надежный jabber-сервер.

Сеть Jabber, в сравнении с другими системами общения, имеет такие преимущества, как широкая функциональность (поддержка чатов, транспортов в другие сети), децентрализованность и независимость, безопасность (поддержка SSL-шифрования заложена в протокол), отсутствие проблем с кодировками, быстрый вход в сеть. Отсутствуют ограничения на объем сообщений, спам, присутствует возможность находиться в режиме online одновременно с нескольких мест.

Для получения функциональности XMPP-сервера необходимо в системе GNU/Linux использующей debian совместимый менеджер пакетов установить интерпретатор языка erlang , нереляционную СУБД и собственно сам xmpp (jabber) сервер, выполнив команду:

```
apt-get install erlang-asn1 erlang-base erlang-ssl ejabberd
```

Произведём конфигурирование сервера:

```
cp /usr/local/etc/ejabberd/ejabberd.cfg.example
```

```
/usr/local/etc/ejabberd/ejabberd.cfg
```

```
cp /usr/local/etc/ejabberd/ejabberd.defaults.example
```

```
/usr/local/etc/ejabberd/ejabberd.defaults
```

Переходим управление текстовому редактору, и проведём с его использованием изменения конфигурации:

```
vi /usr/local/etc/ejabberd/ejabberd.cfg
```

```
{acl, admin, {user, "vasya"}}}.
```

Добавим сюда описание пользователя, который будет иметь права (выполнять функцию) администратора, и пользоваться этими правами как через jabber, так и через веб-интерфейс

Исправим localhost названием настраиваемого сервера

```
% Host name:
```

```
{hosts, ["jabberd.hostname.ru"]}
```

Добавим службу в автозапуск и запустим

```
jabberd # vi /etc/rc.conf
```

```
ejabberd_enable="YES"
```

```
# /usr/local/etc/rc.d/ejabberd start
```

В случае проблем с конфигурацией инициализировать хранилище cookie командой:

```
rm /usr/local/lib/erlang/lib/ejabberd-*/.erlang.cookie
```

Провести конфигурирование DNS -сервера, добавляя записи о доступных xmpp-серверах в виде записей типа A и PTR в конфигурационный файл named.conf

Работу целесообразно проводить в подгруппах из 4-х человек, при реализации обмена две контролируют трафик сервера ejabberd на двух разных узлах, а двое создают диалоговую активность посредством клиентского ПО.

3.3. Порядок выполнения работы

1. Установить и запустить xmpp-сервер (выполняется каждым студентом индивидуально, поверх операционной системы, работающей под управлением эмулятора виртуальных машин VirtualBox). Пример команд установки:

```
apt-install ejabberd
```

2. Установить приложения для обмена мгновенными сообщениями, пример команд установки:

```
Sudo add-apt-repository -y ppa:pigdin-developers/ppa
```

```
sudo apt-get update
```

```
sudo apt-get install pidgin-dev
```

Прим.: для подгруппы нужно два таких сервера на разных узлах локальной сети.

Для использования ПО вне виртуальных машин можно использовать мультипротокольный мессенджер kopete из проекта kdewin (<https://download.kde.org/stable/kdewin/installer/>).

3. Кооперативно всей группой настроить один DNS-сервер,

обеспечивающий локальную сеть информацией о работающих хтпр-серверах, для доступа по имени владельца. Это действие можно заменить редактированием файлов `/etc/hosts` сетевых операционных систем. Данные во всех файлах компьютеров сети должны быть согласованы.

4. Внести изменения в конфигурационный файл сервера (`/etc/ejabberd/ejabberd.yml`)

hosts:

- "localhost"
- "bat.local" //имя первого сервера из `/etc/hosts`
- "norm.local" // имя второго сервера из `/etc/hosts`

5. Зарегистрировать пользователей на серверах
`ejabberd register user bat.local 1234`

6. Запустить процессы серверов командой: `ejabberdctl -live`

7. Для обмена сообщениями запустить любой доступный клиент сети хтпр (jabber), например `pidgin`, `kopete`, `qip`, `miranda`, протестировать возможность обмена, настроив учётную запись (ссылку на сервер) пользователя клиента.

8. Используя утилиту `tcpdump` пронаблюдать за последовательностью прохождения tcp пакетов. Попытаться сформулировать обобщенный алгоритм обмена информацией в процессе передачи сообщения.

9. Оценить сохранность сообщений при недоступности пользователей и мобильность пользователей при перемещении их между узлами.

Содержание отчёта:

1. Цель работы;
2. Подробное описание хода выполнения работы, с выводами касающихся обобщений наблюдаемых фактов;
3. Листинги протокола `tcpdump`
4. Выводы по работе.

4. ЛАБОРАТОРНАЯ РАБОТА №4: РАСПРЕДЕЛЁННЫЕ ПРИЛОЖЕНИЯ НА ЯЗЫКЕ ERLANG

4.1. Цель работы

Получить представление о тенденциях использования функциональных языков программирования для организации распределенных вычислений. Получить представление о минимальных средствах поддерживающих автоматическое распределение нагрузки и/или ресурсов в вычислительных и телекоммуникационных сетях.

4.2. Общие сведения

Новые высокоуровневые языки программирования, позволяют создавать сложные программные системы за более короткое время, но для каждого типа задач есть свои наиболее эффективные инструменты. Для быстрой разработки систем распределенных вычислений можно рассчитывать использовать корпоративные решения, основанные на парадигмах безопасности, авторизации, инкасулирования кода, но зачастую такие системы содержат избыточный функционал. Для понимания необходимого состава функций, поддерживающих распределенные вычисления, предлагается создать приложение на языке Erlang.

Родина языка — лаборатория **Ericsson Computer Science Laboratory (CSLab)**. По большому счету, все началось с потребности немного усовершенствовать язык Prolog, добавив в него поддержку параллелизма. В 1990 году выделился уникальный синтаксис. Тогда же была разработана и виртуальная машина Erlang. А в 1998-м, через восемь лет после публикации, язык Erlang с набором библиотек был опубликован под открытой лицензией.

Erlang относится к функциональным языкам программирования, основанный на использовании λ -выражений. Язык похож на Prolog. В противоположность объектно-ориентированному подходу, предлагающему решать любые задачи, производя декомпозицию предметной области на объекты (объектно-ориентированное моделирование) erlang предлагает моделировать предметную область в терминах процессов. программа, написанная на этом языке, представляет собой

совокупность процессов, взаимодействующих через обмен сообщениями. В Erlang ситуация исключений, связанная с нарушением доступа к памяти, невозможна в принципе, потому что в Erlang, используется обмен сообщениями между процессами. Процессы не изменяют своего состояния, перезаписывая участки общей памяти, поэтому они хорошо распараллеливаются, а решаемая задача успешно масштабируется. Erlang является кросс-платформенным языком, программа транслируется в байт-код, исполняемый виртуальной машиной (Erlang Emulator).

Чтобы передать информацию от одного процесса к другому, используются сообщения. У каждый процесс снабжён своим почтовым ящиком, из которого он извлекает поступившие сообщения. Особенности Erlang в том, что процесс может не читать все сообщения из своего почтового ящика, возможна организация фильтрации, т.е. избирательная выборка сообщений. Ещё одна особенность программы на Erlang заключается в том, что процесс, выполнивший недопустимую операцию, принудительно уничтожается, при этом расходы на взаимодействие со снимаемым процессом минимальны. Процессы в Erlang изолированы друг от друга, значит, при распараллеливании задачи можно обойтись без мьютексов, семафоров, взаимных блокировок. Работающий экземпляр виртуальной машины называется узлом (Node). Каждый узел обладает информацией о существовании других узлов на этой же машине и может с ними взаимодействовать. Также узел может взаимодействовать и с узлами, существующими на других машинах, достаточно задать в программе адрес удаленной системы.

В современных дистрибутивах GNU/Linux интерпретатор erlang устанавливается из репозитариев, выполнением команды:

```
apt-get install erlang-base
```

Для разработки программ на Erlang можно использовать Emacs, NetBeans, Eclipse (с плагином **ErliBird**).

Для работы с Erlang с Emacs понадобится пакет erlang-mode. Настройка производится следующим образом:

```
// Указываем путь к erlang-mode и загружаем
erlang-start
(add-to-list 'load-path ".....")
(require 'erlang-start)
```

```
// типы файлов, для которых будет активирована мод
(add-to-list 'auto-mode-alist '(("\\.erl"?$"
```



```
. erlang-mode))  
(add-to-list 'auto-mode-alist '(("\\.hrl?")  
. erlang-mode))  
// пути к erlang окружению  
(setq erlang-root-dir "/opt/local/lib/erlang")  
(add-to-list 'exec-path "/opt/local/lib/erlang/bin")  
(setq erlang-man-root-dir "/opt/local/lib/  
erlang/man")
```

Выполнение простых операций:

```
1> "hello, world!"  
"hello, world!"  
2> 1 + 2.  
3  
3> (2 * 3) + 4.  
10
```

Одно из требований языка заключается в том, что идентификаторы переменных должны быть набраны в верхнем регистре:

```
1> "hello, world!"  
  
"hello, world!"  
2> 1 + 2.  
3  
3> (2 * 3) + 4.  
10
```

Переменные в Erlang являются переменными одноразового присваивания. Если переменной еще не присваивалось никакого значения, она считается открытой. Закрытая переменная — та, которой уже однажды было присвоено значение и, следовательно, поменять его на другое уже нельзя. Знак равенства представляет собой всего лишь оператор приведения — сначала в переменной неопределенное значение, а потом — это конкретное число. При таком подходе не нужен механизм блокировок при многопоточном программировании.

Другой элемент языка Erlang - атом. Атом — это именованная константа. В отличие от переменной, название атома может быть написано строчными буквами. Следующий важный элемент языка — кортежи (Tuples). Кортёж — это набор значений ограниченной длины. Кортёж ограничивается фигурными скобками, а элементы кортежа отделяются друг от друга запятыми. Кортёжи могут быть вложенными друг в друга. Вот примеры кортежей:

```
1> {ok, 9}.
```

```
{ok,9}
2> {true, {127, 0, 0, 1}}.
{true,{127,0,0,1}}
3> {box, {width, 10}, {height, 35}}.
```

Кортеж не ограниченный по длине, является списком. Списки в отличие от кортежей заключаются в квадратные скобки. Еще список можно разделить на две части — заголовок (head) и хвост (tail). Одно от другого отделяется с помощью вертикальной черты.

```
1> [one, two, three].
[one,two,three]
2> [1, 2|[3, 4, 5]].
[1,2,3,4,5]
3> [{key1, value1}, {key2, value2}].
[{key1,value1},{key2,value2}]
```

Для завершения работы интерпретатора нужно набрать команду `halt()` с точкой в конце строки. Для ввода и интерпретации полноценных листингов можно использовать любой текстовый редактор, не сохраняющий элементы форматирования. Файл должен иметь расширение `(.erl)`. Логическая структура Erlang-программы представляет собой набор модулей. Модули оформляются в виде отдельных файлов. В начале файла объявляется начало модуля и его название. Для этого используется конструкция

```
-module(название).
```

Название модуля должно совпадать с именем файла. Если потребуется обратиться к модулю из другого файла, достаточно указать его имя и – через двоеточие — имя функции, которую нужно вызвать. В круглых скобках после имени функции, как обычно, передаем необходимые аргументы:

```
mymod:test(8).
```

Если внутри модуля требуется объявить функцию, доступную извне, используется конструкция:

```
-export([имя_функции/количество_аргументов]).
```

После того, как файл с модулем создан, его можно скомпилировать в байт-код простой командой

```
с(имя_модуля).
```

Чтобы запустить на выполнение Erlang-программу в обход диалогового режима (то есть, как обычное приложение), нужно всего лишь подsunуть интерпретатору пару ключей: `-noshell`

(подавляет интерактивный режим) и `-run`, после которого нужно последовательно указать имя модуля, имя функции, и опционально — параметры, передаваемые функции при запуске.

Модули

Модуль — это группа логически связанных функций, объединенных под одним именем. Грубо говоря, модули в Erlang — это аналог пространств имен из императивных языков. Они используются для того, что бы объединить функции имеющие сходное назначение. Например, функции для работы со списками находятся в модуле `lists`, а функции ввода-вывода в модуле `io`. Для того, что бы вызвать функцию, необходимо воспользоваться следующей конструкцией: `ModuleName:FunctionName(Arg1, Arg2, ..., ArgN)`. Для примера вызовем функцию, которая возвращает элемент переданного кортежа с указанным номером. Эта функция называется `element` и находится в модуле `erlang`.

```
1> erlang:element(3, {23,54,34,95}).
```

```
34
```

Так же есть возможность вызывать функции без явного указания модуля. Об этом написано немного дальше. Модули содержат в себе функции и атрибуты.

Атрибуты модуля

Атрибуты — это не переменные, как в императивных языках. В Erlang атрибуты модуля — это его метаданные, такие как название, версия, автор, список импортированных функций и т.д. Атрибуты используются компилятором. Так же из них человек может получить полезную для себя информацию о модуле без необходимости разбираться в исходном коде (например версию и автора).

Атрибуты указываются в самом начале файла с модулем и имеют следующий вид: `-Name(Arg)..` Название модуля должно быть атомом. Каждый атрибут указывается на отдельной строке. Вы можете присвоить модулю любые атрибуты, которые захотите, например описать ваше настроение, которое было у вас во время его создания. Так же есть ряд предопределенных атрибутов. И сейчас мы сможем рассмотреть самые часто используемые. Для наглядности мы создадим модуль, который будет содержать функции, выполняющие самые элементарные математические операции: сложение, вычитание, умножение и деление.

-module(Name).

Название модуля — это единственный обязательный атрибут и он обязательно должен быть указан первым. Без него ваш модуль

просто-напросто не скомпилируется. В качестве аргумента принимает атом — название модуля. Назовем наш модуль mySuperModule.

```
-module(mySuperModule).
```

Теперь мы имеем вполне работоспособный модуль. Мы объявили единственный обязательный атрибут и теперь наш модуль может быть скомпилирован. Правда он абсолютно бесполезен, ведь в нем нет ни одной функции. Но фактически — это готовый модуль.

```
-export([Funct1/Arity, Funct2/Arity, ..., FunctN/Arity])
```

Список экспортируемых функций — список функций модуля, которые будут доступны извне. Принимаемый атрибут — список функций. Здесь Funct — название функции, а Arity — количество аргументов, принимаемых ей(арность). Наш модуль будет экспортировать четыре функции: add, subtr, mult, divis (сложение, вычитание, умножение, деление). Каждая функция будет принимать по два аргумента.

```
-export([add/2, subtr/2, mult/2, divis/2]).
```

Помните, что функции, которые вы не укажете в списке экспорта будет невозможно вызвать извне модуля. Работать с ними можно будет только внутри модуля. Экспорт является средством достижения инкапсуляции в модуле. Как вы могли догадаться, экспортированные функции — это аналог открытых методов класса из императивных языков, а остальные — аналог закрытых.

```
-import(ModuleName, [Funct1/Arity, Funct2/Arity, ..., FunctN/Arity]).
```

Этот атрибут указывает, что мы хотим импортировать из модуля ModuleName функции указанные в списке, который передается вторым аргументом. Каждый импортируемый модуль указывается в *отдельном* атрибуте.

Зачем импортировать функции? Как упоминалось выше, для обращения к функции из другого модуля необходимо указать ее полное имя вида ModuleName:FunctionName(). Если вы не хотите каждый раз указывать имя модуля, его нужно импортировать. Этот атрибут — аналог директивы #using из языка C++. Но не стоит злоупотреблять импортированием. Полное имя функции гораздо нагляднее. Увидев его можно сразу сказать к какому модулю принадлежит вызываемая функция. В случае короткого имени, вам придется запоминать из какого модуля была

импортирована эта функция.

Мы будем использовать полные имена функций, но если бы мы хотели использовать короткие имена, мы могли бы написать что то вроде следующего:

```
-import(io, [format/2]).
```

Ну и для примера давайте укажем какой-нибудь произвольный атрибут. Пусть это будет имя автора.

```
-author("Haru Atari").
```

Полный список предопределенных атрибутов можно изучить в официальной документации.

Если сейчас вы попытаете скомпилировать наш модуль, то получите ошибку:

```
1> c(mySuperModule).  
./mySuperModule.erl:2: function add/2 undefined  
./mySuperModule.erl:2: function divis/2 undefined  
./mySuperModule.erl:2: function mult/2 undefined  
./mySuperModule.erl:2: function subtr/2 undefined
```

Как понятно из текста ошибки, компилятор не может найти в нашем файле функции, которые мы указали в списке импорта. И это логично, ведь мы еще их не добавили. Давайте исправим эту ошибку и создадим наши функции.

Функции

В базовом случае, функции в Erlang имеют следующий вид: `FunctName(Arg1, Arg2, ..., ArgN) -> FunctionBody`. Имя функции — атом, а ее тело — это одно или несколько выражений, разделенных *запятыми*. В конце тела функции ставится точка. Если функция содержит всего одно выражение, нагляднее будет записать ее в одну строку.

```
add(X, Y) -> X + Y.
```

Наша функция принимает два аргумента и возвращает их сумму. Обратите внимание на отсутствие слова `return`. Дело в том, что в Erlang функция всегда возвращает результат последнего выражения. В нашем случае — это результат сложения. Поэтому слово `return` просто-напросто не нужно.

Но далеко не всегда функция состоит из одного выражения. В таком случае, тело функции выделяется отступом слева от остального кода. В таком случае наша функция будет выглядеть так:

```
add(X, Y) ->
```

```
doSomething(),  
X + Y.
```

Теперь самостоятельно добавьте оставшиеся три функции. Давайте скомпилируем наш модуль, что бы испытать то, что мы написали.

Компиляция

Программы, написанные на Erlang, компилируются в промежуточный байт код, который потом выполняется в виртуальной машине. Благодаря этому приложения написанные на Erlang кроссплатформенны. Существует несколько виртуальных машин для Erlang. Но самая распространенная — это BEAM (Bogdan/Björn's Erlang Abstract Machine). Существует еще ряд виртуальных машин (JAM и WAM), но они почти не используются и рассматривать их мы не будем. Есть два способа компиляции: из терминала или командной строки Erlang. Давайте рассмотрим оба варианта.

Для компиляции из терминала необходимо перейти в директорию с файлом и вызвать команду `erlc FileName.erl`. Для нашего модуля это будет выглядеть так (путь у вас будет свой).

```
cd ~/Erlang-for-the-little-ones/02/sources  
erlc mySuperModule.erl
```

Для того, что бы сделать это из командной строки Erlang необходимо так же перейти в необходимую директорию командой `cd("DirName")`, а затем вызвать команду `c(ModuleName)`. Обратите внимание, мы передаем название модуля, а не файла. Расширение указывать не надо.

```
1> cd("~/Erlang-for-the-little-ones/02/sources").  
/home/haru/Erlang-for-the-little-ones/02/sources  
ok  
2> c(mySuperModule).  
{ok,mySuperModule}
```

В результате компиляции рядом с файлом `mySuperModule.erl` появится файл `mySuperModule.beam`. Это и есть скомпилированный модуль. Теперь его можно использовать. Давайте попробуем:

```
1> mySuperModule:add(2, 4).  
6  
2> mySuperModule:divis(6,4).  
1.5
```

Стоит упомянуть о том, что есть возможность передавать компилятору «флаги компиляции». Для этого в функцию `c()`

необходимо передать второй аргумент — список флагов. Для примера давайте скомпилируем наш модуль в дебаг режиме:
`c(mySuperModule, [debug_info]).`

Конкурентный бинарный поиск в древовидной структуре данных

Для организации поиска в основной памяти особое значение имеют упорядоченные двоичные (бинарные) деревья (как, например, на рисунке 1). В каждом таком дереве естественно определяются левое и правое поддеревья. Двоичное дерево называется идеально сбалансированным, если число вершин в его левом и правом поддеревьях отличается не более, чем на 1 (легко видеть, что при соблюдении этого условия длины пути до любой листовой вершины дерева отличаются не больше, чем на 1). Примеры идеально сбалансированных деревьев показаны на рисунке 2

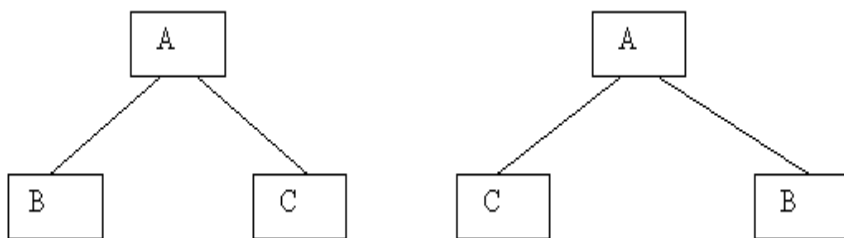


Рис.1

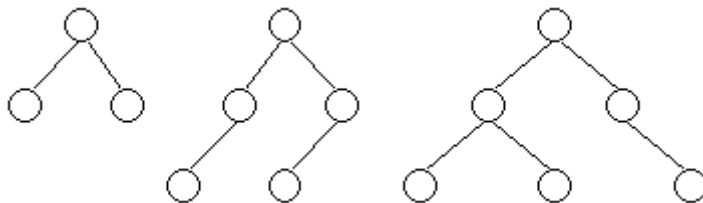


Рис 2.

Программа на языке Erlang реализующая рекурсивный бинарный поиск

```
% Fuad Tabba
% cs.auckland.ac.nz AT fuad
-module(pbst).
-export([rpcContains/2, rpcInsert/2, rpcDelete/2, rpcDump/1]).
-compile(export_all). % for debugging
% To get started:-
% createTree() creates a sample tree
```

```

% randCreate(KeyRange) creates a random tree within a certain
% range
% testTree(KeyRange, Operations) creates and performs a bunch of
% operations
% on the tree.
% Insert, Delete and Contains should always be correct. Dump is
% used for
% debugging and can show an inconsistent state if changes are
% pending. Dump
% must always be correct for a stable tree.
% Node Structure: {Key, PidLeft, PidRight}
% true if the tree contains the Key, false otherwise
rpcContains(_, nil) ->
false;
rpcContains(Key, Pid) ->
Pid ! {contains, Key, self()},
receive
Result -> Result
end.
% Inserting into an empty (nil) node - spawn a new process for the
% new node
rpcInsert(Key, nil) ->
spawn(fun() -> nodeState({Key, nil, nil}) end);
% Inserting into an existing node - ask its process to deal with it
rpcInsert(Key, Pid) ->
Pid ! {insert, Key},
Pid.
% Deleting an empty (nil) node - just results in an empty leaf
% (not found)
rpcDelete(_, nil) ->
nil;
% Deleting from an existing node send a request to it
rpcDelete(Key, Pid) ->
Pid ! {delete, Key},
Pid.
% For debugging: returns the whole tree
rpcDump(nil) ->
nil;
rpcDump(Pid) ->
Pid ! {dump, self()},
receive
Tree -> Tree
end.

```



```

% Maintains the state of the node
nodeState(Node) ->
receive
{contains, Key, Requester} ->
contains(Key, Node, Requester),
nodeState(Node);
{insert, Key} ->
nodeState(insert(Key, Node));
{delete, Key} ->
nodeState(delete(Key, Node));
{node, Requester} ->
Requester ! Node,
nodeState(Node);
{dump, Requester} ->
dump(Node, Requester),
nodeState(Node)
end.
% Get the dump of the subtree
dump(Node, Requester) ->
if
nil == Node ->
Requester ! Node;
true ->
{Key, PidLeft, PidRight} = Node,
Requester ! {Key, rpcDump(PidLeft), rpcDump(PidRight)}
end.
% Check whether the subtree contains the key.
contains(Kfind, {Key, PidLeft, _}, Requester) when Kfind < Key ->
relayContain(Kfind, PidLeft, Requester);
contains(Kfind, {Key, _ PidRight}, Requester) when Kfind > Key ->
relayContain(Kfind, PidRight, Requester);
contains(Key, {Key, _ _}, Requester) ->
Requester ! true;
contains(_, nil, Requester) ->
Requester ! false.
% Relays the contain request to the next node,
% Or informs the requestor that it's not there if it's a nil node
relayContain(_, nil, Requester) ->
Requester ! false;
relayContain(Key, NodePid, Requester) ->
NodePid ! {contains, Key, Requester}.
insert(Knew, {Key, PidLeft, PidRight}) when Knew < Key ->
{Key, rpcInsert(Knew, PidLeft), PidRight};

```

```

insert(Knew, {Key, PidLeft, PidRight}) when Knew > Key ->
{Key, PidLeft, rpcInsert(Knew, PidRight)};
insert(Key, {Key, PidLeft, PidRight}) ->
{Key, PidLeft, PidRight};
% Inserting into a node whose state is nil, it takes on the key
insert(Key, nil) ->
{Key, nil, nil}.
delete(Kdel, {Key, PidLeft, PidRight}) when Kdel < Key ->
{Key, rpcDelete(Kdel, PidLeft), PidRight};
delete(Kdel, {Key, PidLeft, PidRight}) when Kdel > Key ->
{Key, PidLeft, rpcDelete(Kdel, PidRight)};
% There's the possibility of having a node who's state is "nil"
% I've decided to take the lazy approach and leave such leaves,
% only to prune
% them when necessary
%
% Three possibilities; left is a process with a nil state, right
% is a process
% with a nil state, or neither, which means I need to find the
% maximum node
% in the left
delete(Key, {Key, PidLeft, PidRight}) ->
NodeLeft = getNode(PidLeft),
NodeRight = getNode(PidRight),
if
NodeLeft == nil ->
NodeRight;
NodeRight == nil ->
NodeLeft;
true ->
% Find the biggest node in the left subtree assume its
% identity
% then delete it. This step is tricky; if not done
% right it could
% either deadlock or have the tree in an inconsistent
% state.
Kmax = max(NodeLeft),
rpcDelete(Kmax, PidLeft),
{Kmax, PidLeft, PidRight}
end;
% Not found
delete(_, nil) ->
nil.
    
```

```

% Gets the state of the node from its associated process
getNode(nil) ->
nil;
% Ask the process for its state then wait for it to respond.
getNode(Pid) ->
Pid ! {node, self()},
receive
Node -> Node
end.
% returns the biggest key in this subtree
% can't really be parallelized
max({Key, _, PidRight}) ->
NodeRight = getNode(PidRight),
if
NodeRight ::= nil ->
Key;
true ->
max(NodeRight)
end.
% Create a pre-determined test tree
createTree() ->
Root = rpcInsert(260, nil),
rpcInsert(240, Root),
rpcInsert(140, Root),
rpcInsert(320, Root),
rpcInsert(250, Root),
rpcInsert(170, Root),
rpcInsert(60, Root),
rpcInsert(100, Root),
rpcInsert(20, Root),
rpcInsert(40, Root),
rpcInsert(290, Root),
rpcInsert(280, Root),
rpcInsert(270, Root),
rpcInsert(30, Root),
rpcInsert(265, Root),
rpcInsert(275, Root),
rpcInsert(277, Root),
rpcInsert(278, Root).
% Creates a random tree within the specified range. KeyRange/2 is
the root of
% the tree
randCreate(KeyRange) ->
    
```

```

Kroot = trunc(KeyRange/2),
Operations = KeyRange,
Root = rpcInsert(Kroot, nil),
lists:foreach(fun(_) -> rpcInsert(rand:uniform(KeyRange),
Root) end, lists:seq(1, Operations)),
Root.
% First creates a random tree, then performs randomly selected
% operations on it
testTree(KeyRange, Operations) ->
Root = randCreate(KeyRange),
lists:foreach(fun(_) -> spawn (fun() -> randOperation(Root,
KeyRange) end) end, lists:seq(1,Operations)),
Root.
% Perform a randomly select
randOperation(RootPid, KeyRange) ->
Op = rand:uniform(3),
Key = rand:uniform(KeyRange),
case Op of
1 -> rpcInsert(Key, RootPid);
2 -> rpcDelete(Key, RootPid);
3 -> rpcContains(Key, RootPid)
end.
    
```

4.3. Задание

Ввести и оттранслировать программу в байт-код, выполнить под управлением виртуальной машины, разобрать алгоритм, выявить сложные участки кода, разобрать отдельно.

Ход выполнения работы:

1. создать файл с текстом программы и сохранить его в файловой системе с именем pbst.erl
2. Оттранслировать программу в байт-код, для этого в консоли ввести команду:


```
erlc pbst.erl
```
3. Вызвать консольный интерпретатор командой:


```
erl
```
4. Дать интерпретатору следующие команды:
 - а) Перейдем в папку с домашним каталогом.


```
Cd ("~/").
```
 - б) вызвать выполнение подпрограмм, обращаясь к функциям:
 - 1) создать тестовое дерево и вернуть строку вида <A.V.C>, где А – номер узла(0 означает локальный узел - программа запу-

цена на локальной машине), В - первые 15 бит номера процесса, С – 16-18 биты номера процесса:

`pbst:createTree()`.

2) создает дерево и выполняет над ним заданную операцию. Первый параметр – диапазон ключей. Второй параметр – код операции. Коду 1 соответствует операция вставки, коду 2 соответствует операция удаления, коду 3 соответствует операция проверки на содержание переданного узла:

`pbst:testTree(260, 2)`.

3) создать случайное дерево в указанном диапазоне ключей:
`pbst:randCreate(512)`.

5. Исследовать программу реализующую алгоритм конкурентного древовидного бинарного поиска.

6. Сопоставить синтаксические элементы программы с описанием языка Erlang.

7. Сделать вывод о соответствии языка Erlang задачам разработки распределённых информационных систем. Определить, какие особенности исполнительской среды определяют эти предпосылки.

Содержание отчёта: цель работы, фиксация успешности каждого пункта порядка выполнения работы, заключение о более рациональной последовательности действий, выводы.

5. ЛАБОРАТОРНАЯ РАБОТА №5: ИССЛЕДОВАНИЕ P2P ПОДСИСТЕМЫ .NET

5.1. Общие сведения

Технология организации **одноранговых сетей (peer-to-peer networking)**, часто называемая технологией P2P, является одной из самых полезных и при этом часто неправильно понимаемых среди средств, появившихся в последние несколько лет. Когда люди думают о P2P, им на ум, как правило, приходит лишь одна вещь: возможность обмена музыкальными или видео файлами, зачастую незаконным образом. Это связано с тем, что приложения для обмена файлами наподобие BitTorrent стали очень популярными, а в них для работы используется именно технология P2P.

Однако, хотя технология P2P применяется в приложениях для обмена файлами, это вовсе не означает, что она не может

использоваться в других приложениях. На самом деле эта технология может применяться в целом ряде других приложений, и она становится все более и более важной в современном мире повсеместных коммуникаций.

В Microsoft тоже не обошли стороной появление технологии P2P и стали разрабатывать собственные инструменты и средства для ее применения. Так появилась платформа **Microsoft Windows Peer-to-Peer Networking**, исполняющая роль своего рода каркаса для коммуникаций в приложениях P2P. В состав этой платформы входят такие важные компоненты, как *PNRP (Peer Name Resolution Protocol — протокол преобразования имен членов)* и *PNM (People Near Me — соседние пользователи)*.

Кроме того, в версию .NET Framework 3.5 было включено новое пространство имен System.Net.PeerToPeer и несколько новых типов и средств, позволяющих создавать приложения P2P с минимальными усилиями.

5.2. Обзор корпоративных технологий P2P

Технология P2P представляет собой альтернативный подход к организации сетевых коммуникаций. Для того чтобы понять, чем P2P отличается от "стандартного" подхода к обеспечению коммуникаций, не помешает сделать шаг назад и вспомнить, что собой представляет связь типа "клиент-сервер". Коммуникации такого типа очень часто применяются в современных сетевых приложениях.

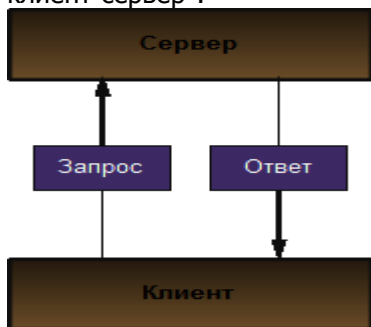
5.3. Архитектура типа "клиент-сервер"

Традиционно взаимодействие с приложениями по сети (в том числе Интернет) организуется с использованием архитектуры типа "клиент-сервер". Прекрасным примером могут служить веб-сайты. При просмотре веб-сайта происходит отправка по Интернет соответствующего запроса веб-серверу, который затем возвращает требуемую информацию. Если необходимо загрузить какой-то файл, это делается напрямую с веб-сервера.

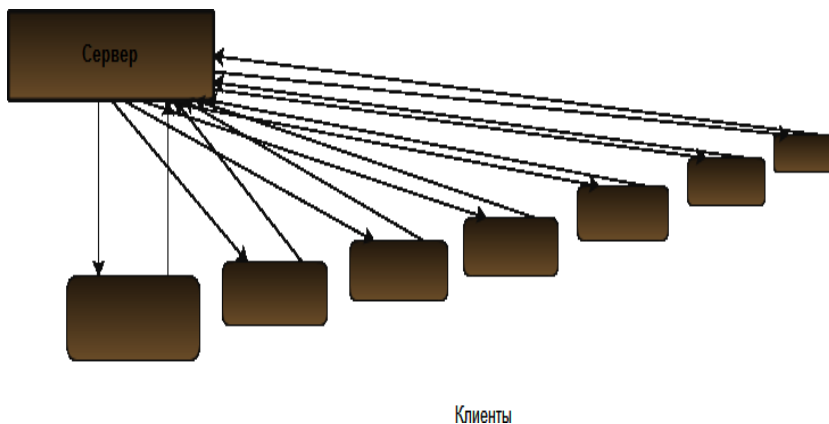
Аналогично, настольные приложения, имеющие возможность подключения к локальной или глобальной сети, обычно устанавливают соединение с каким-то одним сервером, например, сервером баз данных или сервером, предоставляющим набор служб.

На рисунке ниже показан простой вариант архитектуры

типа "клиент-сервер":



Ничего по сути неправильного в такой архитектуре нет, и на самом деле во многих случаях она будет оказываться именно тем, что нужно. Однако ей присуща проблема с масштабируемостью. На следующем рисунке показано, как она будет масштабироваться при добавлении дополнительных клиентов:



С добавлением каждого клиента нагрузка на сервер, который должен взаимодействовать с каждым клиентом, будет увеличиваться. Если снова взять пример с веб-сайтом, то такое увеличение нагрузки может стать причиной выхода веб-сайта из строя. При слишком большом трафике сервер просто перестанет реагировать на запросы.

Конечно, существуют варианты масштабирования, с помощью которых можно смягчить подобную ситуацию. Один из них предусматривает масштабирование "вверх" за счет увеличения мощности и ресурсов сервера, а другой — масштабирование "вширь" путем добавления дополнительных

серверов. Первый способ, естественно, ограничивается доступными технологиями и стоимостью более мощного оборудования. Второй способ потенциально более гибкий, но требует добавления дополнительного уровня в инфраструктуру для обеспечения клиентов возможностью либо взаимодействовать с отдельными серверами, либо поддерживать состояние сеанса независимо от сервера, с которым осуществляется взаимодействие. Для этого доступна масса решений, таких как продукты, позволяющие создавать веб-фермы или фермы серверов.

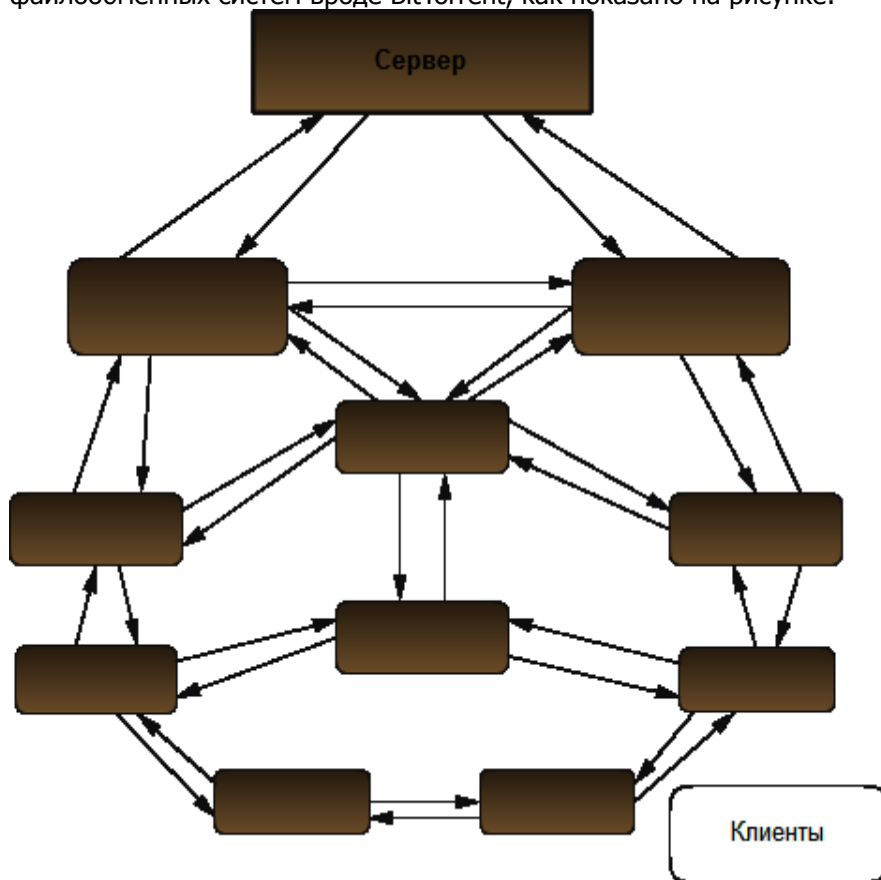
5.4. Архитектура типа P2P

Одноранговый (peer-to-peer) подход полностью отличается от подхода с масштабированием "вверх" или "вширь". В случае применения P2P вместо того, чтобы сосредоточить усилия на попытках улучшить коммуникации между сервером и его клиентами, все внимание уделяется поиску способов, которыми клиенты могут взаимодействовать между собой.

Давайте для примера представим, что веб-сайтом, с которым взаимодействуют клиенты, является www.williamspublishing.com, а издательство Williams объявило о выходе новой книги на этом сайте и предоставлении его для бесплатной загрузки всем желающим, но лишь на протяжении одного дня. Не трудно догадаться, что при таком положении дел накануне появления книги веб-сайт начнет просматривать масса людей, которые будут постоянно обновлять его содержимое в своих браузерах и ожидать появления файла. Как только файл станет доступным, все они одновременно начнут пытаться загрузить его и, скорее всего, веб-сервер, который обслуживает веб-сайт, не выдержит такого натиска и выйдет из строя.

Чтобы предотвратить выход веб-сервера из строя, можно воспользоваться технологией P2P. Вместо отправки файла прямо с сервера сразу всем клиентам он может быть отправлен только определенному числу клиентов. Несколько остальных клиентов могут далее загрузить его у тех клиентов, у которых он уже есть. После этого еще несколько клиентов могут загрузить его у клиентов, получивших его вторыми, и т.д. По сути, этот процесс может происходить даже быстрее благодаря разбиению файла на куски и распределению этих кусков среди клиентов, одни из которых будут загружать их прямо с сервера, а другие — из

других клиентов. Именно так и работают технологии файлообменных систем вроде BitTorrent, как показано на рисунке:



5.5. Особенности архитектуры P2P

Тем не менее, в описанной здесь архитектуре обмена файлами все равно остались кое-какие проблемы, которые должны быть решены. Для начала, каким образом клиенты узнают о том, что существуют другие клиенты, и как они будут обнаруживать фрагменты файла, которые, возможно, имеются у других клиентов? Кроме того, каким образом гарантировать оптимальное взаимодействие между клиентами, если их могут отделять друг от друга континенты?

Каждый клиент, участвующий в работе сетевого приложения P2P, для преодоления этих проблем должен быть способен выполнять следующие операции:

1. обнаруживать других клиентов;
2. подключаться к другим клиентам;
3. взаимодействовать с другими клиентами.

В том, что касается способности обнаруживать других клиентов, возможны два очевидных решения: поддержка списка клиентов на сервере, чтобы клиенты могли получать его и связываться с другими клиентами (называемыми peers—равноправными участниками), либо использование инфраструктуры (например, PNRP), которая позволяет клиентам обнаруживать друг друга напрямую. В большинстве файлообменных систем применяется решение с поддержкой списка на сервере и используются серверы, называемые *"трекерами"* (*trackers*).

В файлообменных системах в роли сервера может также выступать и любой клиент, как показано на рисунке выше, объявляя, что у него имеется доступный файл, и регистрируя его на сервере-трекере. На самом деле в чистой сети P2P вообще не нужны никакие серверы, а лишь равноправные участники.

Проблема подключения к другим клиентам является более тонкой и распространяется на всю структуру используемой приложением P2P сети. При наличии одной группы клиентов, в которой все должны иметь возможность взаимодействовать друг с другом, топология соединений между этими клиентами может приобретать чрезвычайно сложный вид. Зачастую производительность удается улучшать за счет создания нескольких групп клиентов с возможностью установки подключения между клиентами в каждой из них, но не с клиентами в других группах.

В случае создания этих групп по принципу локальности можно добиться дополнительного повышения производительности, поскольку в таком случае клиенты получают возможность взаимодействовать друг с другом по более коротким (с меньшим числом прыжков) сетевым путям между машинами.

Способность взаимодействовать с другими клиентами, пожалуй, не так важна, поскольку существуют хорошо зарекомендовавшие себя протоколы вроде TCP/IP, которые вполне могут применяться и здесь. Конечно, допускается привносить свои улучшения, как в высокоуровневые технологии (например, использовать службы WCF, получая в распоряжение все предлагаемые ими функциональные возможности), так и в низкоуровневые протоколы (например, применять протоколы многоадресной рассылки и тем самым обеспечивать отправку

данных во множество конечных точек одновременно).

Обеспечение клиентов возможностью обнаруживать, подключаться и взаимодействовать друг с другом играет центральную роль в любой реализации P2P.

5.6. Терминология P2

В предыдущих разделах уже было представлено понятие **равноправного участника (peer)** — именно так называют клиентов в сети P2P. Слово "клиент" в сети P2P не имеет никакого смысла, потому что здесь нет обязательного сервера, клиентом которого нужно быть.

Группы равноправных участников, которые соединяются друг с другом, называются *ячейками (meshes)*, *облаками (clouds)* или *графами (graphs)*. Каждая отдельная группа считается хорошо соединенной, если соблюдено хотя бы какое-то одно из следующих условий:

- Между каждой парой равноправных участников существует путь соединения, позволяющий каждому участнику подключаться к другому равноправному участнику требуемым образом.
- Между каждой парой равноправных участников существует относительно небольшое количество соединений, по которым они могут связываться.
- Удаление одного равноправного участника из группы не лишает остальных равноправных участников возможности взаимодействия друг с другом.

Обратите внимание, что это вовсе не означает, что каждый равноправный участник должен обязательно иметь возможность подключаться к каждому другому равноправному участнику напрямую. На самом деле, если проанализировать сеть с математической точки зрения, то можно обнаружить, что для соблюдения упомянутых выше условий равноправным участникам необходимо иметь возможность подключаться к относительно небольшому количеству других равноправных участников.

Еще одним понятием в технологии P2P, о котором следует знать, является **волновое распространение (flooding)**. Под волновым распространением подразумевается способ, которым один фрагмент данных может передаваться по сети всем равноправным участникам и которым может производиться опрос других узлов в сети для обнаружения конкретного фрагмента данных. В неструктурированных сетях P2P этот процесс протекает

довольно произвольно; при этом сначала устанавливается связь с ближайшими соседними равноправными участниками, которые затем, в свою очередь, связываются со своими ближайшими соседями, и т.д. до тех пор, пока не будет охвачен каждый равноправный участник в сети.

Также допускается создавать и структурированные сети P2P с четко определенными путями, по которым должно происходить распространение запросов и данных среди равноправных участников.

5.7. Решения P2P

При наличии подходящей инфраструктуры для P2P можно начинать разрабатывать не просто улучшенные версии клиент-серверных приложений, но и совершенно новые приложения. Технология P2P особенно подходит для приложений следующих классов:

- приложения, предназначенные для распространения содержимого, в том числе упоминавшиеся ранее приложения обмена файлами;
- приложения, предназначенные для совместной работы, такие как приложения, позволяющие открывать общий доступ к рабочему столу и "белой доске" (whiteboard);
- приложения, предназначенные для обеспечения многопользовательской связи и позволяющие пользователям общаться и обмениваться данными напрямую, а не через сервер;
- приложения, предназначенные для распределения обработки, как альтернатива приложениям для суперкомпьютеров, которые обрабатывают огромные объемы данных;
- приложения Web 2.0, объединяющие в себе некоторые или все перечисленные выше приложения и превращающие их в динамические веб-приложения следующего поколения.

5.8. Платформа Microsoft Peer-to-Peer Networking

Платформа Microsoft Windows Peer-to-Peer Networking (Платформа для создания одноранговых сетей Windows производства Microsoft) представляет собой предлагаемую Microsoft реализацию технологии P2P. Она поставляется в составе операционных систем Windows XP SP2, Windows Vista, Windows 7 и Windows 8, а также доступна в виде дополнения для Windows XP

SP1. Она включает в себя две технологии, которые можно использовать для создания приложений P2P в .NET:

Протокол Peer Name Resolution Protocol (PNRP)

Его можно применять для публикации и преобразования адресов равноправных участников сети.

Сервер People Near Me

Его можно применять для обнаружения локальных равноправных участников (и который в настоящее время доступен только в Windows Vista и Windows 7).

Протокол PNRP

Естественно, для реализации приложения P2P можно использовать любой имеющийся в распоряжении протокол, но при работе в среде Microsoft Windows имеет смысл хотя бы рассмотреть вариант применения протокола PNRP. К настоящему времени успело выйти две версии этого протокола. Первая версия предлагалась в составе Windows XP SP2, Windows XP Professional x64 Edition и Windows XP SP1 с пакетом Advanced Networking Pack for Windows XP. Вторая версия была выпущена вместе Windows Vista и сделана доступной для пользователей Windows XP SP2 в виде отдельного загружаемого компонента

(<http://support.microsoft.com/kb/920342>).

Первая и вторая версии PNRP не совместимы между собой, и потому здесь рассматривается только вторая версия.

Сам по себе протокол PNRP не предоставляет всего, что необходимо для создания приложения P2P. Скорее, он является лишь одной из основополагающих технологий, которые применяются для преобразования адресов равноправных участников сети P2P.

Протокол PNRP позволяет клиенту регистрировать конечную точку (называемую именем равноправного участника(peer name)), которая автоматически распространяется между остальными равноправными участниками в группе (облаке). Это имя инкапсулируется в идентификатор PNRP (PNRP ID). Любой равноправный участник, который обнаруживает идентификатор PNRP, может использовать PNRP для его преобразования в имя самого равноправного участника и затем начинать взаимодействовать с соответствующим клиентом напрямую.

Например, можно определить имя равноправного участника, представляющее конечную точку службы WCF, и воспользоваться PNRP для его регистрации в группе (облаке) в виде идентификатора PNRP ID. Равноправный участник,

выполняющий подходящее клиентское приложение, которое применяет механизм обнаружения, способный распознавать имена равноправных участников для предоставляемой службы, тогда сможет обнаружить этот идентификатор PNRP ID. После обнаружения участник с помощью протокола PNRP установит местонахождение конечной точки службы WCF и начнет пользоваться этой службой.

Важным моментом является то, что PNRP не делает никаких предположений относительно того, что на самом деле скрывается за именем равноправного участника. Право решать, как использовать это имя после обнаружения оставляется за самими равноправными участниками.

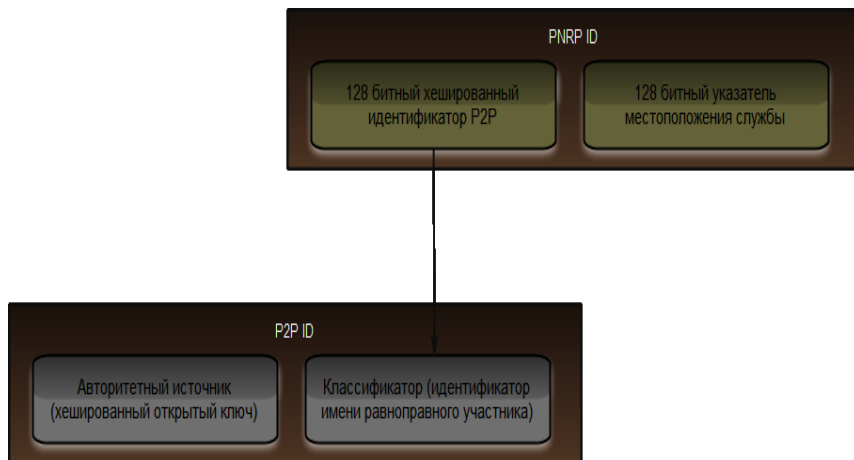
Информация, которую равноправный участник получает от PNRP при преобразовании идентификатора PNRP, включает в себя адрес IPv6, а также обычно и адрес IPv4 участника, который опубликовал этот идентификатор, вместе с номером порта и, необязательно, небольшим количеством дополнительных данных. Если равноправный участник не знает, что означает имя другого равноправного участника, он вряд ли сможет сделать с этой информацией что-нибудь полезное.

Идентификаторы PNRP

Идентификаторы PNRP ID являются 256-битными значениями. Младшие 128 бит используются для обозначения уникальным образом индивидуального равноправного участника сети, а старшие 128 бит — для представления его имени. Старшие 128 бит представляют собой хеш-комбинацию, состоящую из хешированного значения открытого ключа, которое получается от осуществляющего публикацию равноправного участника, и строки длиной до 149 символов, которая представляет имя этого участника.

Хешированный открытый ключ (называемый *авторитетным источником (authority)*) в сочетании с этой строкой (называемой *классификатором (classifier)*) называются идентификатором P2P. Вместо хешированного открытого ключа также может использоваться значение 0, в случае чего имя равноправного участника считается незащищенным (если применяется открытый ключ, то имя считается защищенным).

На рисунке ниже схематично показана структура идентификатора PNRP:



Служба PNRP на стороне равноправного участника отвечает за поддержание списка идентификаторов PNRP, как тех, которые публикует сама, так и тех, которые она получает в виде кэшированного списка от экземпляров служб PNRP, находящихся в других местах облака. Когда равноправный участник пытается преобразовать идентификатор PNRP, служба PNRP либо использует кэшированную копию конечной точки для выяснения адреса того равноправного узла, который опубликовал данный идентификатор PNRP, либо спрашивает его соседей, не могут ли они сделать это.

В конечном итоге с узлом, опубликовавшим идентификатор PNRP, устанавливается соединение, и служба PNRP получает возможность преобразовать его.

Обратите внимание, что все это не требует участия администратора. Все, что понадобится сделать — позаботиться о том, чтобы равноправные участники знали, что следует делать с именами после их преобразования с помощью своей локальной службы PNRP.

Равноправные участники сети могут применять PNRP для нахождения идентификаторов PNRP, совпадающих с определенным идентификатором P2P. Эту возможность можно использовать для реализации простейшего механизма, позволяющего обнаруживать незащищенные имена равноправных участников. Дело в том, что в случае отображения несколькими равноправными участниками незащищенного имени с одинаковым классификатором, их идентификатор P2P ID будет совпадать.

Конечно, из-за того, что любой равноправный участник

может использовать незащищенное имя, нет никакой гарантии, что конечная точка, с которой будет устанавливаться соединение, будет именно той, которая ожидается, поэтому такое решение подходит лишь для реализации механизма обнаружения по локальной сети.

Облака PNRP

Выше было рассказано, как PNRP осуществляет регистрацию и преобразование имен равноправных участников в облаке (группе). Облако (или группа) поддерживается seed-сервером, которым может быть любой сервер с запущенной службой PNRP, которая поддерживает запись хотя бы об одном равноправном участнике. Для службы PNRP доступны облака двух типов:

Облака локальных соединений (Link local)

В такие облака входят компьютеры с подключением к локальной сети. Каждый ПК может подключаться к более чем одному облаку такого типа при условии наличия у него нескольких сетевых адаптеров.

Глобальные облака (Global)

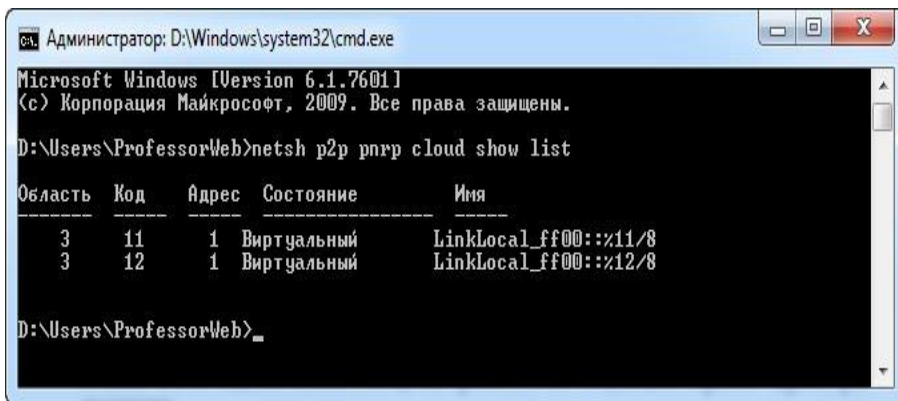
В такие облака по умолчанию входят компьютеры с подключением к Интернету, хотя также можно определять приватное глобальное облако. Разница состоит в том, что для глобального облака с подключением к Интернету Microsoft поддерживает seed-сервер, а для определяемого самостоятельно приватного глобального облака должен использоваться собственный seed-сервер. В последнем случае необходимо позаботиться о том, чтобы все равноправные участники могли подключаться к нему, настроив соответствующим образом параметры политики.

В прошлых выпусках PNRP существовали облака еще и третьего типа, которые назывались облаками локальных сайтов (Site local). Они больше уже не применяются и потому не рассматриваются.

Для выяснения, в какие облака входит данный компьютер, служит следующая команда:

```
netsh p2p pnrp cloud show list
```

Ниже показано, как обычно выглядит результат запуска этой команды:



```

Администратор: D:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
(с) Корпорация Майкрософт, 2009. Все права защищены.

D:\Users\ProfessorWeb>netsh p2p pnrp cloud show list

Область  Код  Адрес  Состояние  Имя
-----  ---  -----  -
3        11   1       Виртуальный  LinkLocal_ff00::%11/8
3        12   1       Виртуальный  LinkLocal_ff00::%12/8

D:\Users\ProfessorWeb>_
    
```

В данном случае результат указывает, что доступно два облака локальных соединений (Link local). Понять это можно как по значению в столбце Name(Имя), так и по значению в столбце Score(Область), в котором для облаков локальных соединений всегда отображается значение 3, а для глобальных облаков — значение 1. Для подключения к глобальному облаку необходимо иметь глобальный адрес IPv6. Компьютер, на котором запускалась команда netshc p2p, таковым не обладал, потому для него оказалось доступным только локальное облако.

При наличии проблем с сетевой связью следует проверить, не препятствует ли брандмауэр передаче локального сетевого трафика через порты UDP 3540 или 1900. UDP-порт 3540 используется протоколом PNRP, а UDP-порт 1900 — протоколом SSDP (Simple Service Discovery Protocol — простой протокол обнаружения служб), который, в свою очередь, используется службой PNRP (а также устройствами UPnP).

Особенности PNRP в Windows 7

В Windows 7 протокол PNRP предусматривает использование нового компонента под названием **DRT (Distributed Routing Table — таблица распределенной маршрутизации)**.

Этот компонент отвечает за определение структуры используемых PNRP ключей, роль которой в реализации по умолчанию исполняет описанный выше идентификатор PNRP. С помощью API-интерфейса DRT можно определить альтернативную схему ключей при условии, что они представляют собой 256-битные целочисленные значения (подобно идентификаторам PNRP ID).

Это означает возможность использования любой схемы, но при этом, разумеется, нужно самостоятельно заботиться о генерации и защите ключей. За счет применения этого компонента можно создавать совершенно новые топологии облаков, выходящие за рамки действия PNRP.

В Windows 7 также появилась новая опция для подключения к другим пользователям в приложении Remote Assistance, которая называется **Easy Connect**. Эта опция использует PNRP для обнаружения пользователей, к которым необходимо подключиться. После создания сеанса с помощью Easy Connect или других средств (например, отправки приглашения по электронной почте) пользователи могут открыть общий доступ к своим рабочим столам и оказывать помощь друг другу через интерфейс Remote Assistance.

Служба People Near Me

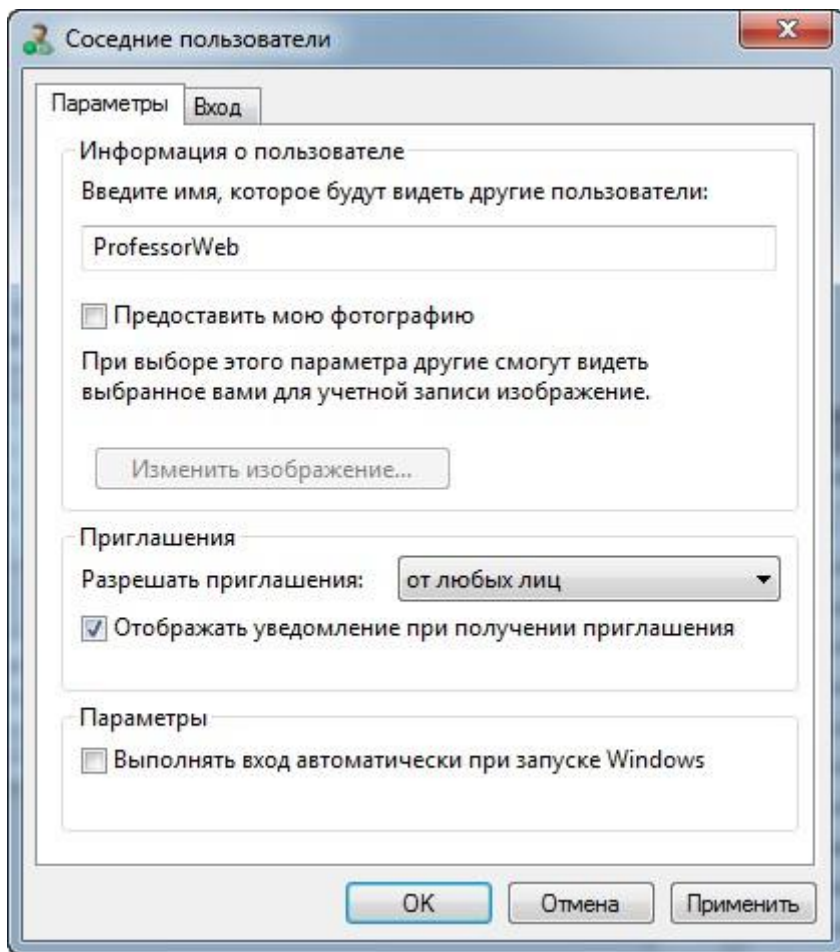
Служба PNRP, как было показано в предыдущем разделе, применяется для определения местонахождения равноправных участников сети P2P. Очевидно, что она является важной действующей технологией при продумывании процесса обнаружения, подключения и взаимодействия в приложении P2P, но сама по себе она ни одного из этих этапов полностью не реализует.

В реализации этапа обнаружения ей помогает служба People Near Me (Соседние пользователи), которая позволяет находить равноправных участников, которые зарегистрировались в локальной сети (т.е. подключены к тому же локальному облаку, что и данный компьютер).

Эта служба наверняка уже попадалась на глаза, поскольку она является встроенным компонентом Windows Vista и Windows 7, а также используется в приложении Windows Meeting Space, которое позволяет открывать равноправным участникам общий доступ к различным приложениям. Для конфигурирования этой службы применяется элемент Change People Near Me (Изменить соседних пользователей) панели управления, быстро перейти к которому можно путем ввода следующей команды:

```
collab.cpl
```

Этот элемент позволяет получить доступ к диалоговому окну, показанному на рисунке ниже. После регистрации эта служба становится доступной в любом приложении, которое построено с учетом возможности ее использования:



На момент написания этих строк служба PNM было доступна только в семействе операционных систем Windows Vista и Windows 7. Возможно, в будущем появятся пакеты обновлений или дополнительные загружаемые модули, которые позволят сделать ее доступной в Windows XP.

5.9. Создание приложений P2P

Общая инфраструктура

Теперь, когда известно, что собой представляет технология P2P, и какие технологии доступны разработчикам приложений

.NET для реализации приложений P2P, пришла пора посмотреть, как создавать такие приложения. Из приведенного ранее материала понятно, что в любом приложении P2P должен применяться протокол PNRP для публикации, распространения и преобразования имен равноправных участников. Поэтому в первую очередь здесь будет показано, как достичь этого в .NET, а затем — каким образом использовать службу PNM в качестве каркаса для приложения P2P. Последнее может быть выгодно, поскольку в случае применения PNM реализовать собственные механизмы обнаружения не понадобится.

Перед изучением этих тем сначала необходимо ознакомиться с классами, которые предлагаются в следующих пространствах имен: System.Net.PeerToPeer и System.Net.PeerToPeer.Collaboration. Для работы с этими классами нужно обязательно ссылаться на сборку System.Net.dll.

Классы в пространстве имен System.Net.PeerToPeer инкапсулируют API-интерфейс для PNRP и позволяют взаимодействовать со службой PNRP. Их можно применять для решения двух основных задач: регистрация имен равноправных участников и преобразование имен равноправных участников.

Регистрация имен равноправных участников

Для регистрации имени равноправного участника потребуется выполнить перечисленные ниже шаги:

1. Создайте защищенное или незащищенное имя равноправного участника с определенным классификатором.
2. Настройте процесс регистрации этого имени, предоставив следующие сведения:

- Номер порта TCP.
- Облако или облака, в которых должно быть зарегистрировано имя участника (если не указано, PNRP будет регистрировать имя равноправного участника во всех доступных облаках).

- Комментарий длиной до 39 символов.
- Дополнительные данные объемом до 4096 байт.
- Информация о том, должны ли для имени равноправного участника автоматически генерироваться конечные точки (поведение по умолчанию, при котором конечные точки автоматически генерируются на основе IP-адреса или адресов равноправного члена и номера порта, если таковой указан).

- Коллекция конечных точек.
3. Зарегистрируйте имя равноправного участника в локальной службе PNRP.

После выполнения третьего шага имя равноправного члена становится доступным для всех соседей в выбранном облаке (или облаках). Регистрация равноправного члена продолжается то тех пор, пока не будет прекращена явным образом, или до тех пор, пока не будет завершен процесс, зарегистрировавший имя равноправного участника.

Для создания имени равноправного участника применяется класс **PeerName**. Экземпляр этого класса создается из строкового представления идентификатора P2P ID в форме *авторитетный_источник.классификатор*, либо из строки классификатора и типа **PeerNameType**. Можно использовать как тип *PeerNameType.Secured*, так и тип *PeerNameType.Unsecured*. Например:

```
PeerName pn = new PeerName("Peer classifier",  
PeerNameType.Secured);
```

Поскольку в незащищенном имени равноправного участника значением авторитетного источника является 0, следующие строки кода эквивалентны:

```
PeerName pn = new PeerName("Peer classifier",  
PeerNameType.Unsecured);  
PeerName pn = new PeerName("0.Peer classifier");
```

После создания экземпляра **PeerName**, можно использовать вместе с номером порта для инициализации объекта **PeerNameRegistration**:

```
PeerNameRegistration pnr = new PeerNameRegistration(pn, 8080);
```

В качестве альтернативного варианта, можно установить для объекта **PeerNameRegistration**, создаваемого с использованием его параметра по умолчанию, свойство **PeerName** и (необязательно) свойство **Port**. Кроме того, желаемый экземпляр **Cloud** можно передать либо в третьем параметре конструктору **PeerNameRegistration**, либо указать с помощью свойства **Cloud**. Экземпляр **Cloud** можно получить либо из имени облака, либо с применением одного из следующих статических членов класса **Cloud**:

Cloud.Global

Это статическое свойство позволяет получить ссылку на глобальное облако, которое в зависимости от конфигурации политики равноправных участников может быть и приватным глобальным облаком.

Cloud.AllLinkLocal

Это статическое поле позволяет получить облако, содержащее все облака локальных соединений, доступные для данного

равноправного участника.

Cloud.Available

Это статическое поле позволяет получить облако, содержащее все облака, доступные для данного равноправного участника, т.е. локальные и глобальные (если таковые имеются).

После создания экземпляра `PeerNameRegistration` можно при желании установить его свойства `Comment` и `Data`. При этом следует помнить об ограничениях этих свойств. При попытке установить для свойства `Comment` строку длиной более 39 символов `Unicode`, будет сгенерировано исключение **PeerToPeerException**, а при попытке установить для свойства `Data` байтовый массив размером более 4096 байт — исключение `ArgumentOutOfRangeException`.

Можно также с помощью свойства `EndPointCollection` добавить конечные точки. Это свойство представляет собой коллекцию типа `System.Net.IPEndPointCollection` объектами `System.Net.IPEndPoint`. В случае применения свойства `EndPointCollection` может понадобиться установить свойство `UseAutoEndPointSelection` в `false`, чтобы предотвратить автоматическую генерацию конечных точек.

После того как все готово к регистрации имени равноправного участника, можно вызвать метод `PeerNameRegistration.Start()`. Для удаления записи о регистрации имени равноправного члена из службы PNRP служит метод `PeerNameRegistration.Stop()`.

Ниже приведен пример регистрации защищенного имени равноправного участника вместе с комментарием:

```
PeerName pn = new PeerName("Peer classifier",
PeerNameType.Unsecured);
PeerNameRegistration pnr = new PeerNameRegistration(pn,
8080);
pnr.Comment = "Комментарий";
pnr.Start();
```

5.10. Преобразование имён равноправных участников

Для преобразования имени равноправного участника должны быть выполнены следующие шаги:

1. Сгенерируйте имя равноправного члена на основе известного идентификатора P2P, либо идентификатора P2P ID,

полученного посредством операции обнаружения.

2. Воспользуйтесь распознавателем (resolver) для преобразования имени равноправного участника и получения коллекции записей с именами равноправных членов. Распознаватель можно ограничить отдельным облаком и/или максимальным количеством возвращаемых результатов.

3. Из любой полученной записи с именем равноправного участника извлеките само имя, а также необходимую информацию о конечной точке, комментарии и дополнительные данные, если таковые присутствуют.

Подобно процессу регистрации имени равноправного участника, этот процесс начинается с создания объекта PeerName. Отличие состоит в том, что здесь используется имя равноправного участника, которое уже было зарегистрировано одним или более удаленными участниками. Простейшим способом для получения списка активных членов в локальном облаке является регистрация незащищенного имени для каждого равноправного участника с одним и тем же классификатором и применение этого имени на стадии преобразования.

Однако для глобальных облаков пользоваться такой стратегией не рекомендуется, поскольку незащищенные имена могут быть легко подделаны. Для преобразования имен равноправных участников используется класс **PeerNameResolver**. Получив в распоряжение экземпляр этого класса, можно выбрать, как должно выполняться преобразование — синхронно с применением метода *Resolve()* или же асинхронно с помощью метода *ResolveAsync()*.

Метод *Resolve()* можно вызывать с указанием как лишь одного параметра PeerName, так и дополнительно экземпляра Cloud, в котором должно производиться преобразование, максимального количества возвращаемых записей с именами равноправных участников в виде целого числа, или того и другого вместе. Этот метод возвращает экземпляр *PeerNameRecordCollection*, представляющий собой коллекцию объектов **PeerNameRecord**.

Например, ниже показан пример преобразования незащищенного имени равноправного участника во всех локальных облаках с возвратом максимум 5 результатов:

```
PeerName pn = new PeerName("0.Peer classifier");
PeerNameResolver pnres = new PeerNameResolver();
PeerNameRecordCollection pnrc = pnres.Resolve(pn,
Cloud.AllLinkLocal, 5);
```

Метод `ResolveAsync()` предусматривает использование стандартной схемы вызова асинхронного метода, а именно — передачу уникального объекта `userState`, после чего ожидание поступления событий `ResolveProgressChanged`, свидетельствующих об обнаружении равноправных участников и выполнении преобразования, и события `ResolveCompleted`, свидетельствующего об окончании процесса обнаружения участников и преобразования.

Метод `ResolveAsyncCancel()` позволяет отменить любой незавершенный асинхронный запрос. В обработчиках события `ResolveProgressChanged` используется параметр аргументов события `ResolveProgressChangedEventArgs`, унаследованный от стандартного класса `System.ComponentModel.ProgressChangedEventArgs`.

Свойство `PeerNameRecord` объекта аргумента события, получаемого в обработчике событий, можно применять для получения ссылки на запись с именем равноправного участника, которая была обнаружена.

Аналогичным образом для события `ResolveCompleted` требуется обработчик, принимающий параметр типа `ResolveCompletedEventArgs`, который наследуется от `AsyncCompletedEventArgs`. Этот тип имеет параметр `PeerNameRecordCollection`, который можно использовать для получения полного списка записей с именами равноправных членов, которые были обнаружены.

В следующем коде показан пример реализации обработчиков для этих событий:

```
private pnrres_ResolveProgressChanged(object sender,
    ResolveProgressChangedEventArgs e)
{
    // Использование e.ProgressPercentage (унаследованного от
    базовых аргументов события)
    // Обработка PeerNameRecord из e.PeerNameRecord
}

private pnrres_ResolveCompleted(object sender,
    ResolveCompletedEventArgs e)
{
    // Проверка наличия e.IsCancelled и e.Error
    (унаследованных
    // от базовых аргументов события)
```



```
// Обработка PeerNameRecordCollection из  
e.PeerNameRecordCollection  
}
```

После получения одного или более объектов `PeerNameRecord` можно переходить к их обработке. Этот класс `PeerNameRecord` предоставляет в распоряжение свойства `Comment` и `Data` для изучения комментариев и дополнительных данных, которые были добавлены при регистрации имени равноправного участника (если были), свойство `PeerName` для извлечения объекта `PeerName` и записи с именем участника и, что наиболее важно, свойство `EndPointCollection`.

Как и в случае с `PeerNameRegistration`, здесь это свойство тоже представляет собой коллекцию типа `System.Net.IPEndPointCollection`, содержащую объекты `System.Net.IPEndPoint`. Эти объекты можно использовать для подключения к предоставляемым равноправным участникам конечным точкам любым желаемым образом.

5.11. Безопасный доступ кода в `System.Net.PeerToPeer`

Пространство имен `System.Net.PeerToPeer` также включает два следующих класса, которые можно использовать вместе с моделью **CAS (Code Access Security — безопасный доступ кода)**:

`PnrpPermission`, унаследованный от класса `CodeAccessPermission` и **`PnrpPermissionAttribute`**, унаследованный от класса `CodeAccessSecurityAttribute`.

Эти классы удобно применять для обеспечения возможности настройки доступа PNRP с помощью полномочий обычным для CAS образом.

Пример приложения P2P

Давайте рассмотрим пример приложения P2P, использующего графический интерфейс WPF, в котором для конечной точки равноправного участника используется служба WCF.

Конфигурационные настройки этого приложения содержатся в конфигурационном файле `App.config` и указывают, как должно выглядеть имя равноправного участника и какой порт должен прослушиваться:

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>
```

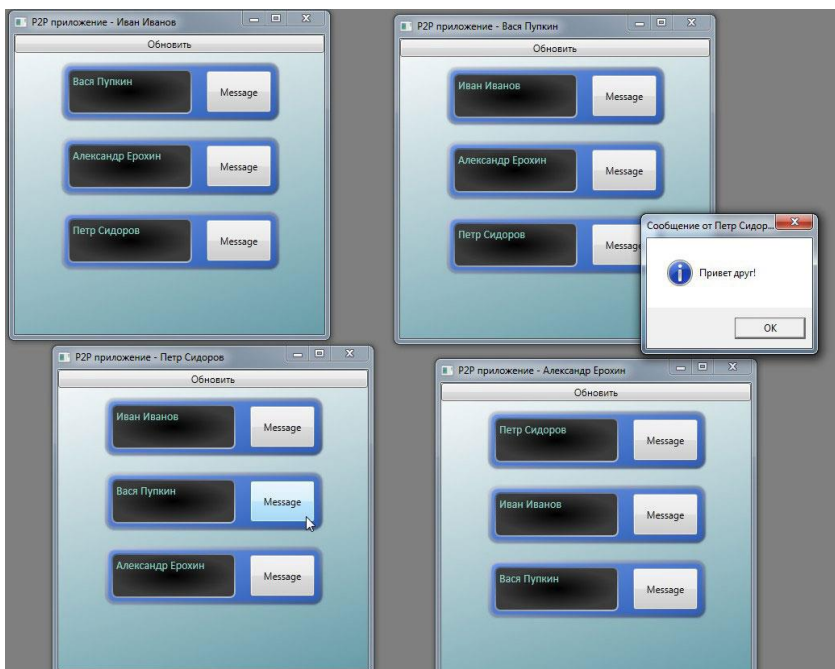
```
<appSettings>  
  <add key="username" value="Вася Пупкин" />  
  <add key="port" value="4590" />  
</appSettings>  
</configuration>
```

После сборки этого приложения можно приступить к его тестированию, либо скопировав его на другие компьютеры в локальной сети и запустив все его экземпляры, либо запустив множество его экземпляров на одном компьютере. В последнем варианте нужно будет поменять используемый для каждого экземпляра порт, внося соответствующие изменения в отдельные конфигурационные файлы (понадобится скопировать содержимое каталога Debug на свой локальный компьютер и по очереди отредактировать каждый конфигурационный файл).

В обоих вариантах результаты будут выглядеть яснее, если изменить также имя пользователя в каждом экземпляре.

После запуска всех экземпляров приложения можно щелкнуть на кнопке Refresh (Обновить), чтобы получить список доступных равноправных участников асинхронным образом. Обнаружив в этом списке нужного участника, можно с помощью щелчка на кнопке Message отправить ему стандартное сообщение.

На рисунке ниже показан пример того, как приложение выглядит в действии при запуске четырех его экземпляров на одной машине. На рисунке видно, что один равноправный участник только что отправил сообщение другому равноправному участнику, и оно отобразилось в диалоговом окне:



Итак, создайте новый проект приложения WPF по имени P2P. Изначально нужно будет добавить в проект ссылки на сборки System.Configuration.dll, System.Net.dll, System.Runtime.Serialization.dll, System.ServiceModel.dll, а также указать пространства имен, которые мы будем использовать:

```
using System.Net;
using System.Net.PeerToPeer;
using System.ServiceModel;
using System.Configuration;
```

Добавьте в проект XML-файл App.config, который описан выше, а также следующие файлы классов (предоставляют информацию о пире, создают WCF-контракт):

```
// Файл PeerEntry.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net.PeerToPeer;
```

```
namespace P2P
{
    class PeerEntry
    {
```

```

    public PeerName PeerName { get; set; }
    public IP2PService ServiceProxy { get; set; }
    public string DisplayString { get; set; }
    public bool ButtonsEnabled { get; set; }
}
}
// Файл P2PService.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.Serialization;
using System.ServiceModel;

namespace P2P
{
    [ServiceBehavior(InstanceContextMode =
InstanceContextMode.Single)]
    public class P2PService : IP2PService
    {
        private MainWindow hostReference;
        private string username;

        public P2PService(MainWindow hostReference, string
username)
        {
            this.hostReference = hostReference;
            this.username = username;
        }

        public string GetName()
        {
            return username;
        }

        public void SendMessage(string message, string from)
        {
            hostReference.DisplayMessage(message, from);
        }
    }
}
// Файл IP2PService.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.Serialization;
using System.ServiceModel;

```

```
namespace P2P
{
    [ServiceContract]
    public interface IP2PService
    {
        [OperationContract]
        string GetName();

        [OperationContract(IsOneWay = true)]
        void SendMessage(string message, string from);
    }
}
```

Большая часть работы в этом приложении происходит в обработчике события Window_Loaded() окна MainWindow. Ниже показана XAML-разметка окна и исходный код приложения:

```
<Window x:Class="P2P.MainWindow"
```

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
    xmlns:P2P="clr-namespace:P2P"
    Title="MainWindow" Height="400" Width="400"
    Loaded="Window_Loaded" Closing="Window_Closing">
    <Window.Resources>
        <LinearGradientBrush x:Key="bevelBrush"
        EndPoint="0.369,-1.362" StartPoint="0.631,2.362">
            <GradientStop Color="#FF001E56" Offset="0"/>
            <GradientStop Color="#FFFFFFFF" Offset="1"/>
        </LinearGradientBrush>
        <DataTemplate x:Key="PeerEntryDataTemplate">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="100" />
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition />
                </Grid.RowDefinitions>
                <Rectangle RadiusX="10" RadiusY="10"
                Grid.ColumnSpan="2" Stroke="{DynamicResource
                bevelBrush}" StrokeThickness="4" >
                    <Rectangle.Fill>
                        <LinearGradientBrush EndPoint="0.369,-1.362"
                StartPoint="0.631,2.362">
                            <GradientStop Color="#FF1346A6"
```

```

Offset="0"/>
    <GradientStop Color="#FF85ADF6"
Offset="1"/>
    </LinearGradientBrush>
    </Rectangle.Fill>
    <Rectangle.BitmapEffect>
    <BlurBitmapEffect Radius="3"/>
    </Rectangle.BitmapEffect>
</Rectangle>
    <TextBlock Margin="10" Text="{Binding
Path=DisplayString}" Padding="4" TextWrapping="Wrap"
Width="150" Opacity="0.995" FontFamily="Calibri"
FontSize="14" Foreground="#FF8ED1C3" >
    <TextBlock.Background>
    <RadialGradientBrush>
    <GradientStop Color="#FF000000" Offset="0"/>
    <GradientStop Color="#FF3C3C3C" Offset="1"/>
    </RadialGradientBrush>
    </TextBlock.Background>
</TextBlock>
    <Rectangle RadiusX="6" RadiusY="6" Margin="8"
Fill="{x:Null}" StrokeThickness="2" >
    <Rectangle.Stroke>
    <LinearGradientBrush EndPoint="0.631,2.362"
StartPoint="0.369,-1.362">
    <GradientStop Color="#FF001E56" Offset="0"/>
    <GradientStop Color="#FFFFFFF" Offset="1"/>
    </LinearGradientBrush>
    </Rectangle.Stroke>
</Rectangle>
    <StackPanel Grid.Column="1">
    <Button Name="MessageButton"
Margin="10,10,10,10" Height="50" IsEnabled="{Binding
Path=ButtonsEnabled}" Content="Message"
BorderBrush="{DynamicResource bevelBrush}"/>
    </StackPanel>
</Grid>
</DataTemplate>
</Window.Resources>

    <Window.Background>
    <LinearGradientBrush EndPoint="0.444,-0.183"
StartPoint="0.778,1.12">
    <GradientStop Color="#FF6199A5" Offset="0"/>
    <GradientStop Color="#FFFFFFF" Offset="1"/>
    </LinearGradientBrush>
</Window.Background>

```

```

<StackPanel>
  <Button Name="RefreshButton"
Click="RefreshButton_Click">Обновить</Button>
  <ListBox Name="PeerList"
ItemTemplate="{DynamicResource PeerEntryDataTemplate}"
ButtonBase.Click="PeerList_Click" Background="{x:Null}"
BorderBrush="{x:Null}">
    <ListBox.ItemContainerStyle>
      <Style TargetType="ListBoxItem">
        <Setter Property="Margin" Value="10" />
        <Setter Property="HorizontalAlignment"
Value="Center" />
      </Style>
    </ListBox.ItemContainerStyle>
    <P2P:PeerEntry DisplayString="Обновите, чтобы
увидеть пиров." ButtonsEnabled="False" />
  </ListBox>
</StackPanel>
</Window>
public partial class MainWindow : Window
{
    private P2PService localService;
    private string serviceUrl;
    private ServiceHost host;
    private PeerName peerName;
    private PeerNameRegistration peerNameRegistration;

    public MainWindow()
    {
        InitializeComponent();
    }

    private void Window_Loaded(object sender,
RoutedEventArgs e)
    {
        // Получение конфигурационной информации из
app.config
        string port = ConfigurationManager.AppSettings["port"];
        string username =
ConfigurationManager.AppSettings["username"];
        string machineName = Environment.MachineName;
        string serviceUrl = null;

        // Установка заголовка окна
        this.Title = string.Format("P2P приложение - {0}",
username);

        // Получение URL-адреса службы с использованием
    
```

```

адресаIPv4
    // и порта из конфигурационного файла
    foreach (IPAddress address in
        Dns.GetHostAddresses(Dns.GetHostName()))
    {
        if (address.AddressFamily ==
            System.Net.Sockets.AddressFamily.InterNetwork)
        {
            serviceUrl =
                string.Format("net.tcp://{0}:{1}/P2PService", address, port);
            break;
        }
    }

    // Выполнение проверки, не является ли адрес null
    if (serviceUrl == null)
    {
        // Отображение ошибки и завершение работы
        приложения
        MessageBox.Show(this, "Не удастся определить адрес
        конечной точки WCF", "Networking Error",
            MessageBoxButton.OK, MessageBoxImage.Stop);
        Application.Current.Shutdown();
    }

    // Регистрация и запуск службы WCF
    localService = new P2PService(this, username);
    host = new ServiceHost(localService, new
        Uri(serviceUrl));
    NetTcpBinding binding = new NetTcpBinding();
    binding.Security.Mode = SecurityMode.None;
    host.AddServiceEndpoint(typeof(IP2PService), binding,
        serviceUrl);
    try
    {
        host.Open();
    }
    catch (AddressAlreadyInUseException)
    {
        // Отображение ошибки и завершение работы
        приложения
        MessageBox.Show(this, "Не удастся начать
        прослушивание, порт занят.", "WCF Error",
            MessageBoxButton.OK, MessageBoxImage.Stop);
        Application.Current.Shutdown();
    }

    // Создание имени равноправного участника (пира)
    
```



```

        peerName = new PeerName("P2P Sample",
PeerNameType.Unsecured);

        // Подготовка процесса регистрации имени
равноправного участника в локальном облаке
        peerNameRegistration = new
PeerNameRegistration(peerName, int.Parse(port));
        peerNameRegistration.Cloud = Cloud.AllLinkLocal;

        // Запуск процесса регистрации
        peerNameRegistration.Start();
    }

    private void Window_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
    {
        // Остановка регистрации
        peerNameRegistration.Stop();

        // Остановка WCF-сервиса
        host.Close();
    }

    private void RefreshButton_Click(object sender,
RoutedEventArgs e)
    {
        // Создание распознавателя и добавление
обработчиков событий
        PeerNameResolver resolver = new PeerNameResolver();
        resolver.ResolveProgressChanged +=
            new
EventHandler<ResolveProgressChangedEventArgs>(resolver_Re
solveProgressChanged);
        resolver.ResolveCompleted +=
            new
EventHandler<ResolveCompletedEventArgs>(resolver_ResolveC
ompleted);

        // Подготовка к добавлению новых пиров
        PeerList.Items.Clear();
        RefreshButton.IsEnabled = false;

        // Преобразование незащищенных имен пиров
асинхронным образом
        resolver.ResolveAsync(new PeerName("0.P2P Sample"),
1);
    }

```

```

void resolver_ResolveCompleted(object sender,
ResolveCompletedEventArgs e)
{
    // Сообщение об ошибке, если в облаке не найдены
    пиры
    if (PeerList.Items.Count == 0)
    {
        PeerList.Items.Add(
            new PeerEntry
            {
                DisplayString = "Пиры не найдены.",
                ButtonsEnabled = false
            });
    }
    // Повторно включаем кнопку "обновить"
    RefreshButton.IsEnabled = true;
}
    
```

```

void resolver_ResolveProgressChanged(object sender,
ResolveProgressChangedEventArgs e)
{
    PeerNameRecord peer = e.PeerNameRecord;

    foreach (IPEndPoint ep in peer.EndPointCollection)
    {
        if (ep.Address.AddressFamily ==
            System.Net.Sockets.AddressFamily.InterNetwork)
        {
            try
            {
                string endpointUrl =
                string.Format("net.tcp://{0}:{1}/P2PService", ep.Address,
                ep.Port);
                NetTcpBinding binding = new NetTcpBinding();
                binding.Security.Mode = SecurityMode.None;
                IP2PService serviceProxy =
                ChannelFactory<IP2PService>.CreateChannel(
                    binding, new EndpointAddress(endpointUrl));
                PeerList.Items.Add(
                    new PeerEntry
                    {
                        PeerName = peer.PeerName,
                        ServiceProxy = serviceProxy,
                        DisplayString = serviceProxy.GetName(),
                        ButtonsEnabled = true
                    });
            }
            catch (EndpointNotFoundException)
                
```

```

        {
            PeerList.Items.Add(
                new PeerEntry
                {
                    PeerName = peer.PeerName,
                    DisplayString = "Неизвестный пир",
                    ButtonsEnabled = false
                });
        }
    }
}

e) private void PeerList_Click(object sender, RoutedEventArgs
    {
        // Убедимся, что пользователь щелкнул по кнопке с
        именем MessageBoxButton
        if (((Button)e.OriginalSource).Name == "MessageButton")
        {
            // Получение пира и прокси, для отправки
            сообщения
            PeerEntry peerEntry =
            ((Button)e.OriginalSource).DataContext as PeerEntry;
            if (peerEntry != null && peerEntry.ServiceProxy != null)
            {
                try
                {
                    peerEntry.ServiceProxy.SendMessage("Привет
                    друг!", ConfigurationManager.AppSettings["username"]);
                }
                catch (CommunicationException)
                {
                }
            }
        }
    }

    internal void DisplayMessage(string message, string from)
    {
        // Показать полученное сообщение (вызывается из
        службы WCF)
        MessageBox.Show(this, message,
            string.Format("Сообщение от {0}", from),
            MessageBoxButton.OK, MessageBoxImage.Information);
    }
}

```

В обработчике события `Window_Loaded()` сначала производится загрузка конфигурационной информации и установка в заголовке окна имени пользователя.

Далее с использованием адреса хоста равноправного участника и сконфигурированного порта определяется конечная точка, в которой должна предоставляться служба WCF. Привязкой этой службы будет `NetTcpBinding`, поэтому в URL-адресе конечной точки применяется протокол `net.tcp`.

Затем производится проверка полученного URL-адреса конечной точки, после чего выполняется регистрация и запуск службы WCF. Обратите внимание на использование одиночного (singleton) экземпляра класса службы для налаживания простых коммуникаций между приложением-хостом и службой (для отправки и получения сообщений). Кроме того, для простоты режим безопасности в конфигурации привязки отключен.

После щелчка на кнопке Обновить в обработчике события `RefreshButton_Click()` вызывается метод `PeerNameResolver.ResolveAsync()` для обнаружения соседних равноправных участников асинхронным образом.

Остальная часть кода отвечает за отображение и взаимодействие с соседними равноправными участниками; ее можно изучить самостоятельно.

Предоставление конечных точек WCF через облака P2P является замечательным способом для обеспечения возможности установки местонахождения служб внутри предприятия, а также налаживания связи между равноправными участниками сети подобно тому, как было сделано в рассмотренном примере.

5.12. Пространство имён `System.Net.PeerToPeer.Collaboration`

Классы в пространстве имен `System.Net.PeerToPeer.Collaboration` предоставляют каркас, который можно использовать для создания приложений, предусматривающих работу со службой `People Neat Me` и API-интерфейсом совместной работы посредством P2P (P2P Collaboration API). Как упоминалось ранее, это было возможно только в ОС Windows Vista или Windows 7.

Классы, определенные в этом пространстве имен, можно использовать для взаимодействия с соседними равноправными участниками и приложениями, включая выполнение следующих функций:

- осуществление входа (sign in) и выхода (sign out);
- обнаружение других равноправных участников;
- управление контактами и выявление присутствия других равноправных участников в сети.

Классы из этого пространства имен можно также применять для приглашения других пользователей присоединиться к работе в каком-то приложении или для обеспечения возможности обмена данными между пользователями и приложениями. Однако для того чтобы делать это, необходимо создавать собственные приложения, поддерживающие PNM.

Осуществление входа и выхода

Одним из наиболее важных классов в пространстве имен System.Net.PeerToPeer.Collaboration является **PeerCollaboration**. Это статический класс с множеством статических методов, которые можно применять для самых различных целей, как будет показано в этом и следующем разделах. В частности, для осуществления входа и выхода из службы People Near Me предназначены его методы *SignIn()* и *SignOut()*. Оба метода принимают единственный параметр типа PeerScope, который может иметь одно из перечисленных ниже значений:

PeerScope.None

В случае использования этого значения методы *SignIn()* и *SignOut()* не будут иметь никакого эффекта.

PeerScope.NearMe

Указание этого значения позволяет осуществлять вход и выход из локальных облаков.

PeerScope.Internet

Применение этого значения позволяет осуществлять вход и выход из глобального облака (это может понадобиться для установки связи с контактным лицом, которого в текущий момент нет в локальной подсети).

PeerScope.All

Указание этого значения позволяет осуществлять вход и выход из всех доступных облаков.

При необходимости вызов *SignIn()* будет приводить к отображению диалогового окна конфигурирования People Near Me. После входа в службу можно для свойства *PeerCollaboration.LocalPresenceInfo* установить значение типа *PeerPresenceInfo*. Это сделает доступной стандартную функциональность мгновенного обмена сообщениями (IM), например, возможность установки состояния в "отошел".

Распределенные информационные системы

Для свойства `PeerPresenceInfo.DescriptiveText` можно установить строку Unicode длиной до 255 символов, а для свойства `PeerPresenceInfo.PresenceStatus` — значение перечисления `PeerPresenceStatus`. Значения этого перечисления описаны ниже:

 Перечисление `PeerPresenceStatus`

Значение	Описание
<code>PeerPresenceStatus.Away</code>	Равноправного участника нет на месте.
<code>PeerPresenceStatus.BeRightBack</code>	Равноправного участника нет на месте, но он скоро вернется.
<code>PeerPresenceStatus.Busy</code>	Равноправный участник занят.
<code>PeerPresenceStatus.Idle</code>	Равноправный участник неактивен.
<code>PeerPresenceStatus.Offline</code>	Равноправного участника нет в сети.
<code>PeerPresenceStatus.Online</code>	Равноправный участник находится в сети и доступен для связи.
<code>PeerPresenceStatus.OnThePhone</code>	Равноправный участник разговаривает по телефону.
<code>PeerPresenceStatus.OutToLunch</code>	Равноправного участника нет на месте, но он вернется после обеда.

Обнаружение соседей

После подключения к локальному облаку можно получить список всех равноправных участников сети, которые находятся по соседству. Это делается с помощью метода `PeerCollaboration.GetPeersNearMe()`, который возвращает объект `PeerNearMeCollection`, содержащий в себе объекты **PeerNearMe**.

Свойство `Nickname` объекта `PeerName` позволяет получить имя обнаруженного соседа, свойство `IsOnline` — определить, находится ли он в текущий момент в сети, а (в случае низкоуровневых операций) свойство `PeerEndpoints` — определить имеющиеся к нему отношение конечные точки. С помощью свойства `PeerEndpoints` также можно выяснить статус объекта `PeerNearMe` в сети.

Для получения объекта **PeerPresenceInfo** необходимо передать методу `GetPresenceInfo()` информацию о конечной точке, как было описано в предыдущем разделе.

Управление контактами и определение присутствия соседей в сети

Контакты представляют собой способ, которым можно запоминать других равноправных участников сети. Обнаруженных посредством службы `People Near Me` пользователей сети можно добавить в свои контакты и после этого связываться с ними в любой момент, когда они будут появляться в сети. Связываться с контактами можно как через локальные, так и через

глобальные облака (при условии наличия IPv6-подключения к Интернету).

Чтобы добавить обнаруженного пользователя в контакты, необходимо вызвать метод *PeerNearMe.AddToContactManager()*. При вызове этого метода можно закрепить за контактом отображаемое имя, псевдоним (*nickname*) и адрес электронной почты. Однако обычно управление контактами осуществляется с помощью класса *ContactManager*.

В любом случае при манипулировании контактами приходится иметь дело с объектами **PeerContact**. Класс *PeerContact*, как и *PeerNearMe*, унаследован от абстрактного базового класса *Peer*, но имеет больше свойств и методов, чем *PeerNearMe*.

Так, например, он включает в себя свойства *DisplayName* и *EmailAddress*, которые позволяют более подробно описывать обнаруженного PNM пользователя. Другое отличие между этими двумя классами связано с тем, что *PeerContact* имеет более явные отношения с классом *System.Net.PeerToPeer.PeerName*. С помощью свойства *PeerContact.PeerName* можно извлечь объект *PeerName*, после чего приступить к взаимодействию с любыми конечными точками, которые этот объект *PeerName* предоставляет, используя описанные ранее приемы.

Информация о локальном соседе также доступна в статическом свойстве *ContactManager.LocalContact*. Это позволит далее иметь дело со свойством *PeerContact* и деталями о локальном соседе.

Для добавления объектов *PeerNearMe* в локальный список контактов применяется либо метод *ContactManager.CreateContact()*, либо метод *CreateContactAsync()*, а объектов *PeerName* — метод *GetContact()*. Удаление контактов, представляемых объектами *PeerNearMe* или *PeerName*, производится с помощью метода *DeleteContact()*.

И, наконец, существуют события, которые можно обрабатывать в ответ на изменения в контактах. Например, событие *PresenceChanged* можно обрабатывать в ответ на изменения в информации о присутствии любого из известных *ContactManager* контактов.

5.13. Задание

Исследовать описанные возможности, предоставляемые операционной системой Windows 7 для организации p2p сетей. Создать проект системы обмена сообщениями, внести в него копии приведенных листингов программ, отладить, изучить алгоритм обмена сообщениями и синхронизации процессов с использованием отладчика IDE, описать общие черты алгоритма, сделать выводы.

6. ЛАБОРАТОРНАЯ РАБОТА №6: РАСПРЕДЕЛЁННЫЕ ФАЙЛОВЫЕ СИСТЕМЫ

6.1. Цель работы

Получить представление о распределённых файловых системах, текущем состоянии разработки, сложности настройки и эксплуатации СХД.

6.2. Задание

Ознакомиться с обзорами сетевых файловых систем. Постараться провести аналитический обзор систем организации сетевого и распределённого хранения. Установить распределённую файловую систему и привести нагрузочное тестирование. Сравнить с SMB/NFS.

Лабораторная работа выполняется в подгруппах.

Ход выполнения работы:

1. В зависимости от варианта получить исходные коды, скопировать и установить одну из распределённых файловых систем на группу компьютеров (по кол-ву участников эксперимента). Возможные варианты: lustre, HDFS (Hadoop file system), GFS2, CEPH.

Инструкции (методика установки):

а) HDFS <http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/>

б) Lustre <http://samag.ru/archive/article/1865>

2. Определить ip-адреса всех узлов, на которых установлен

серверами

3. Выбрать имена и внести адресную информацию в /etc/hosts

4. Произвести настройку серверов Ф.С. на узлах сети.

5. Провести нагрузочное тестирование, посредством копирования /dev/null на перемонтированную кластерную файловую систему.

Содержание отчёта: цель работы, фиксация успешности каждого пункта порядка выполнения работы, заключение о более рациональной последовательности действий, выводы.