УДК 004.738

ИНТЕРНЕТ-СЛУЖБА ДЛЯ ПОДДЕРЖКИ РАСПРЕДЕЛЕННЫХ ИНФОРМАЦИОННО-УПРАВЛЯЮЩИХ СИСТЕМ

Р. Э. Асратян

Институт проблем управления им. В. А. Трапезникова, г. Москва

Описаны принципы организации протокола RFPP (протокола обработки удаленных пакетов файлов), специально предназначенного для поддержки взаимодействия компонентов распределенных управляющих систем. Основной акцент сделан на обеспечении разнообразных форм взаимодействия (вызов удаленных обработчиков, связь с постоянно активными агентами через очереди обслуживания, межсерверный обмен данными) и обеспечение устойчивости к сетевым сбоям. Главным объектом обработки служит "удаленный пакет наборов данных", обладающий способностью мигрировать с сервера на сервер и аккумулировать результаты работы программных компонентов.

ВВЕДЕНИЕ

Развитие Интернета дало новый импульс разработкам распределенных приложений и систем управления. Системы с "большой географией", включающие в себя сотни взаимодействующих компонентов, удаленных друг от друга на сотни и тысячи километров, уже сегодня являются реальностью [1, 2]. Тем не менее, выбор сетевых протоколов для построения таких систем и сегодня представляет собой проблему для разработчиков и администраторов. В большинстве случаев разработчики вынуждены пользоваться широкоизвестными Интернет-службами и протоколами, изначально спроектированными для решения совсем других задач: поддержки массовых коммуникаций (SMTP/POP3), электронной прессы (HTTP) и др. [2, 3].

Возникающие при этом трудности связаны, прежде всего, с тем, что семантика этих Интернет-служб не вполне адекватна потребностям распределенных информационно-управляющих систем, а акцент в их реализации сделан скорее на достижение максимальной производительности, чем на надежности и устойчивости к сетевым сбоям (что особенно четко проявляется в условиях работы с некачественными каналами связи). Наверное, многим знакома ситуация, когда вследствие разрыва сетевого соединения результаты длительной работы, например, CGI-приложения, оказываются потерянными. Этот недостаток особенно характерен для Интернет-служб, которые обычно относят к классу "online", т. е. служб, реализующих все фазы взаимодействия (передача параметров, обработка, получение результатов и т. п.) в течение одного сеанса связи. Риск потери данных объясняется тем, что эти службы, как правило, жестко ассоциируют "контекст" сеанса связи с TCP-соединением. Поэтому разрушение соединения означает потерю "контекста" и, как следствие, результатов обработки.

Понятие "контекста" сеанса связи является основополагающим для описываемого ниже протокола RFPP (Remote File Packets Protocol — протокол обработки удаленных пакетов файлов). Главная особенность протокола заключается в том, что этот контекст ассоциируется не с сетевым соединением, а с новым, явно определяемым понятием — удаленным пакетом наборов данных (далее будем называть его просто "пакетом"). Клиент RFPP сам открывает пакет на RFPP-сервере при необходимости удаленного взаимодействия и закрывает его, когда он больше не нужен. В промежутке между этими событиями он может наполнить пакет данными, вызвать те или иные удаленные обработчики данных, получить результаты их работы, передать пакет другому клиенту и т. п. Причем все это может быть сделано или за одно TCP-соединение, или за несколько. Сохраняя идентификатор пакета, клиент всегда может восстановить контекст сеанса после случайного (или намеренного) разрыва соединения и продолжить работу.

Главная цель разработки RFPP заключалась в решении следующих задач.

- Создание простого, интуитивно понятного средства сетевой поддержки распределенных приложений и информационно-управляющих систем.
- Обеспечение защиты от потери данных в условиях некачественных каналов связи.
- Преодоление “водораздела” между так называемыми “online” и “offline”-взаимодействиями. Фактически, применительно к RFPP термины “online” и “offline” скорее характеризуют стиль “поведения” клиента, а не являются имманентными свойствами протокола.

1. КРАТКИЙ ПРИМЕР

Как и всякий протокол, основанный на TCP/IP [2, 3], RFPP поддержан клиентским и серверным программным обеспечением. Сервер RFPP представляет собой постоянно активную программу, обслуживающую запросы на обработку от клиентов. Клиентское программное обеспечение представляет собой библиотеку функций, реализующих прикладной программный интерфейс (API) к RFPP. Этот интерфейс является “лицом” RFPP с точки зрения пользователя. Чтобы сразу же дать представление о нем, рассмотрим следующий “скелет” кода в нотации языка C++.

```
Rfppclnt Rfppc;
char PackId[9];
Rfppc.Config(“192.168.1.77”,8126,”alibaba”,”sezam”);
Rfppc.Connect();
Rfppc.Open(PackId);
Rfpp.SendFile(PackId,”c:\\out”,”file1”);
Rfpp.SendFile(PackId,”c:\\out”,”file2”);
Rfpp.Process(PackId,”zip”,””);
while(Rfpp.GetFile(PackId, “c:\\in”, “*.*”)==0);
Rfppc.Close(PackId);
Rfppc.Disconnect();
```

Первые две строки определяют две переменные: строку символов PackId и объект класса Rfppclnt. Этот класс содержит в себе все методы, необходимые клиенту для работы с RFPP. Третья строка настраивает объект на работу с определенным RFPP-сервером. В качестве параметров объекту передаются IP-адрес (или имя) сервера, номер порта, имя и пароль пользователя (RFPP-сервер обслуживает только зарегистрированных пользователей).

Четвертая строка устанавливает TCP-соединение с сервером, а пятая — открывает на сервере новый пакет: в результате выполнения метода Open строка PackId окажется заполненной уникальным (для конкретного сервера) восьмисимвольным идентификатором открытого пакета. Все последующие методы будут использовать этот идентификатор.

Шестая и седьмая строки обеспечивают передачу на сервер двух файлов из каталога “c:\out”. Можно представлять себе, что файлы “наполняют” открытый пакет. Восьмая строка обеспечит вызов обработчика, зарегистрированного на сервере под именем “zip” (с передачей ему пустой строки параметров). Например, архиватора, который создаст архив всех переданных файлов. В результате обработки в пакете окажется единственный файл — файл архива.

Девятая строка содержит цикл выборки всех файлов из пакета в локальный каталог клиента. Десятая и одиннадцатая обеспечивают закрытие пакета и прекращение соединения с сервером, соответственно.

Разумеется, из данного фрагмента кода намеренно удалены операторы проверки успеха выполнения методов и обработки исключений.

Рассмотрим теперь альтернативный подход к построению кода. Он заключается в замене строки с вызовом метода Process следующими строками:

```
Rfpp.Start(PackId,”zip”,””);
. . .
Rfppc.Connect();
Rfpp.Wait(PackId,600);
```

Метод Start запускает обработку, разрывает соединение с сервером и немедленно возвращает управление. Ожидание окончания обработки реализуется с помощью метода Wait, который имеет два параметра: идентификатор пакета и предельное время ожидания в секундах: тайм-аут (разумеется, по возвращаемому значению пользователь может определить, чем закончилось ожидание). Существенно, что эти методы могли бы быть выполнены со значительным сдвигом во времени, и даже двумя разными процессами (если обеспечена передача идентификатора пакета от одного процесса другому).

2. ПАКЕТЫ И НАБОРЫ ДАННЫХ

Для RFPP-сервера пакет является основной единицей обработки. Он представляет собой поименованное хранилище поименованных наборов данных, загруженных клиентами или же полученных в результате обработки. Кроме того, с пакетом связана управляющая информация, позволяющая “отслеживать” его текущее состояние и историю его обработки. Мы не будем делать никаких предположений относительно способов физической реализации пакетов (в файловой системе, в оперативной памяти или в СУБД). Существенно то, что каждый пакет может логически находиться либо в “рабочей области”, либо в одной из очередей. Пакеты в рабочей области не образуют никакой структуры — доступ к ним возможен только по идентификатору. Пакеты, находящиеся в очереди, перенумерованы. Доступ к ним возможен по порядковому номеру пакета в очереди.

Загрузка данных в пакет по сети выполняется с помощью методов SendFile и SendMem класса Rfppclnt. В первом случае данные выбираются из файла, а во втором — из области памяти. Загруженные данные образуют отдельный набор данных в пакете с именем, совпадающем с именем исходного файла или же заданным отдельным аргументом метода.

Прием данных по сети из пакета выполняется с помощью методов GetFile и GetMem класса Rfppclnt. В первом случае данные загружаются в файл, а во втором — в область памяти. Успешно принятый набор данных удаляется из пакета. В аргументах методов можно указывать или конкретные или “групповые имена” наборов данных. По коду завершения метода пользователь может распознать ситуацию отсутствия наборов данных в пакете.

В случае разрыва соединения в процессе передачи данных оно автоматически восстанавливается, а процесс передачи продолжается с “точки разрыва” (как в FTP), а не с начала (как в SMTP). Отметим, что наличие стабильного контекста создает все необходимые условия для восстановления передачи.

Так как открытие и закрытие пакетов происходит по инициативе клиента, сервер периодически запускает фоновый процесс "чистки мусора". "Мусором" считаются пакеты, которые не обрабатывались в течение достаточно долгого интервала времени. Длительность интервала задается администратором.

С точки зрения управления пакетами RFPP-сервер имеет дело с объектами трех типов:

- клиенты — источники запросов на обслуживание (в том числе — на открытие и закрытие пакетов);
- обработчики — программы, запускаемые сервером для обработки наборов данных в пакетах;
- агенты — постоянно активные (или периодически запускаемые) программы, объединяющие в себе свойства клиентов и обработчиков. В отличие от обработчиков, агенты могут выполняться на отдельных машинах (т. е. не на тех, на которых работает сервер и (или) клиенты). Пакеты передаются на обработку агентам через связанные с агентами очереди пакетов;
- серверы — другие RFPP-серверы, доступные для взаимодействия.

Объекты всех четырех типов должны быть предварительно зарегистрированы на сервере RFPP. Другими словами, сервер отвергает запросы, исходящие от незарегистрированных клиентов (агентов), или же запросы на запуск незарегистрированных обработчиков. При регистрации объекту присваивается уникальное регистрационное имя; кроме того, с ним связывается определенная управляющая информация (пароль, права доступа, спецификация исполняемого модуля обработчика и т. п.).

3. ОБРАБОТЧИКИ

Чтобы программу можно было использовать в качестве RFPP-обработчика, ее поведение должно подчиняться определенным правилам:

- RFPP-сервер передает программе-обработчику три параметра командной строки: строку параметров, переданных клиентом (в методе Process или Start), спецификацию рабочего каталога и спецификацию каталога для временных файлов. Программа имеет право прочесть информацию из файлов, находящихся в рабочем каталоге, обработать ее и поместить туда же результаты своей работы в виде одного или нескольких файлов. Разумеется, содержание обработки может быть любым (архивация, преобразование форматов, запрос к СУБД и т. п.);
- программа может поместить одно или несколько диагностических сообщений в стандартный вывод.

Перед запуском обработчика RFPP-сервер обеспечивает выборку всех наборов данных, содержащихся в обрабатываемом пакете и занесение их в рабочий каталог в виде файлов. Соответственно, после завершения программы все файлы из рабочего каталога помещаются в пакет. В управляющую информацию пакета будут помещены диагностические сообщения программы. С помощью соответствующих методов класса Rfppclnt клиент может получить и результаты обработки, и диагностические сообщения.

С помощью одного вызова метода Process (или Start) клиент может запустить не один, а целую последовательность обработчиков. Для этого он должен задать список их имен в значении соответствующего аргумента

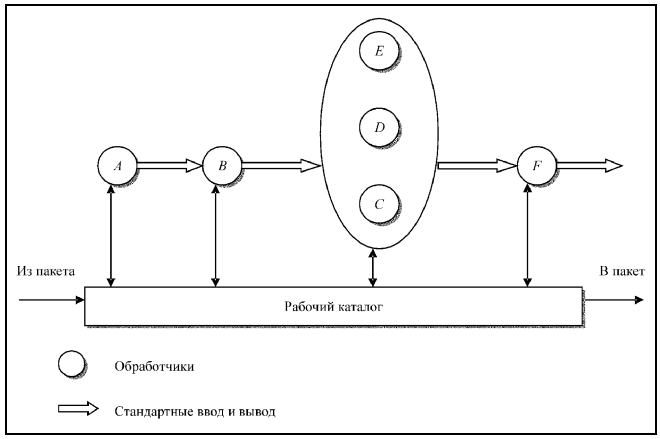

Рис. 1. Организация вызовов для списка обработчиков

метода. Имена в списке должны быть отделены друг от друга с помощью точки с запятой или с помощью запятой. В первом случае процесс выполнения обработчика (вернее, соответствующего ему исполняемого модуля) будет запущен только после завершения всех предшествующих обработчиков, а во втором — немедленно. Другими словами, обработчики, имена которых разделены запятой, запускаются как параллельные процессы (разумеется, их логика обработки должна допускать параллельное выполнение).

Независимо от способа запуска все обработчики получают строку параметров, переданную клиентом, в качестве первого параметра командной строки. На стандартный ввод каждого процесса подаются диагностические сообщения предшественников. После завершения работы всех обработчиков файлы, находящиеся в рабочем каталоге, и порожденные диагностические сообщения помещаются в пакет. На все время обработки пакет блокируется и становится недоступным для клиента.

Поясним эти правила на следующем примере. Пусть список обработчиков имеет следующий вид: "*A*; *B*; *C*, *D*, *E*; *F*". Организация вызовов в этом случае проиллюстрирована на рис. 1. Важно отметить, что обработчики *C*, *D* и *E* выполняются параллельно.

Список обработчиков может быть задан не только в аргументах методов Process и Start, но и при определении нового обработчика. В определении можно задать список нескольких исполняемых модулей, построенный с помощью вышеуказанных разделителей. Обращение к такому обработчику повлечет за собой вызов всех перечисленных исполняемых модулей по вышеописанным правилам. В этом же списке могут быть размещены не только спецификации исполняемых модулей, но и имена других зарегистрированных обработчиков, заключенные в угловые скобки (чтобы отличить их от исполняемых модулей). Эффективный (т. е. действительно обрабатываемый) список модулей будет построен путем рекурсивного замещения имен обработчиков их определениями.

В аргументе метода Process (или Start) или в определении вызываемого обработчика может содержаться не имя обработчика, а имя клиента или агента. Обработка таких ситуаций связана с очередями пакетов.

4. ОЧЕРЕДИ ПАКЕТОВ

Очереди пакетов представляют собой механизм обмена информацией между клиентами, а также между клиентами и агентами.

Как уже отмечалось, каждый пакет может находиться либо в рабочей области, либо в очереди. Пакеты создаются, обрабатываются и уничтожаются в рабочей области. Пакеты в очереди находятся в состоянии ожидания обработки.

Каждый клиент или агент имеет собственную рабочую область и собственную очередь. В классе Rfppclnt имеются методы, позволяющие клиентам и агентам перемещать пакеты из рабочих областей в очереди и наоборот.

Если в аргументе метода Process (или Start) или в определении вызываемого обработчика содержится не имя обработчика, а имя клиента или агента, то это имя обрабатывается особым образом:

- независимо от предшествующего разделителя выполняется ожидание завершения всех предыдущих обработчиков (если они есть);
- если обрабатываемое имя является именем клиента, то создается копия обрабатываемого пакета (с другим идентификатором, но точно повторяющая текущее состояние исходного пакета); эта копия помещается в конец очереди пакетов этого клиента, после чего сервер немедленно продолжает обработку последующих элементов списка (если они есть);
- если обрабатываемое имя является именем агента, то обрабатываемый пакет извлекается из рабочей области и помещается в конец очереди пакетов этого агента. К управляющим данным пакета добавляется информация возврата, организованная в форме стека. В этот стек помещается имя клиента, выполнившего метод Process (или Start). Обработка пакета прекращается до тех пор, пока агент не вернет обработанный пакет в рабочую область клиента или же до истечения предельного времени ожидания. Это время ожидания (в секундах) может быть указано сразу после имени агента в скобках (например, “zipagent(100)”).

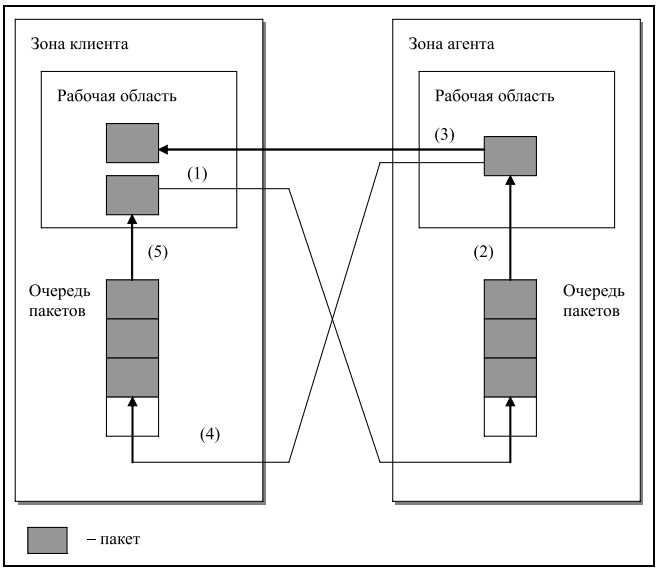

Рис. 2. Обработка очередей пакетов

С помощью метода Select класса Rfppclnt программа клиента или агента может извлечь любой пакет из своей очереди в свою рабочую область. В качестве аргумента методу задается порядковый номер пакета в очереди, а возвращает он идентификатор извлеченного пакета (здесь имеется определенная аналогия с обработкой входящей почты по протоколу POP3). Имея пакет в своей рабочей области и “зная” его идентификатор, клиент или агент могут работать с ним точно так же, как со своими собственными пакетами (включая передачу его на обработку другому агенту).

В распоряжении агента имеется специальный метод Return, позволяющий вернуть пакет в рабочую область того клиента (или агента), от которого он был получен. Обработка пакета агентом завершается вызовом этого метода.

Механизм обработки очередей проиллюстрирован на рис. 2, где отдельные фазы обработки обозначены номерами.

(1) В результате выполнения метода Process (или Start) обрабатываемый пакет переносится из рабочей области клиента в очередь агента.

(2) В результате выполнения программой агента метода Select пакет переносится из очереди в рабочую область агента.

(3) После обработки находящихся в пакете данных программа агента вызывает метод Return, в результате чего пакет возвращается в рабочую область клиента.

Важно подчеркнуть, что RFPP-сервер не управляет запуском агентов (в отличие от обработчиков). Поддержка процессов постоянно активных или периодически запускаемых агентов осуществляется общесистемными средствами.

На рис. 2 проиллюстрирован также альтернативный механизм взаимодействия, основанный на возврате обработанного пакета не в рабочую область, а в очередь клиента. Он инициируется в случае, если вызов агента осуществляется в форме “имя_агента(q)”. В этом случае в очередь агента помещается копия обрабатываемого пакета, а обработка собственно пакета продолжается без какого-либо ожидания. При выполнении программой агента метода Return обработанная копия помещается в очередь клиента (4). Предполагается, что клиент получит результаты обработки в процессе “разбора” своей очереди (5).

5. ОРГАНИЗАЦИЯ АСИНХРОННОЙ ОБРАБОТКИ

Метод Start инициирует создание на сервере одного или нескольких процессов-обработчиков пакета. Сервер извещает клиента об успешном начале обработки, после чего метод Start немедленно завершается. Метод Wait инициирует создание на сервере специального “следящего” процесса (Wait-процесса). Его функция — периодическая проверка состояния пакета в течение заданного интервала времени с целью определения момента окончания обработки (рис. 3).

Выполнение Wait-процесса завершается по одному из двух событий:

- окончание обработки пакета;
- завершение заданного интервала ожидания.

В первом случае из пакета извлекается системное диагностическое сообщение (о нормальном или аномаль-

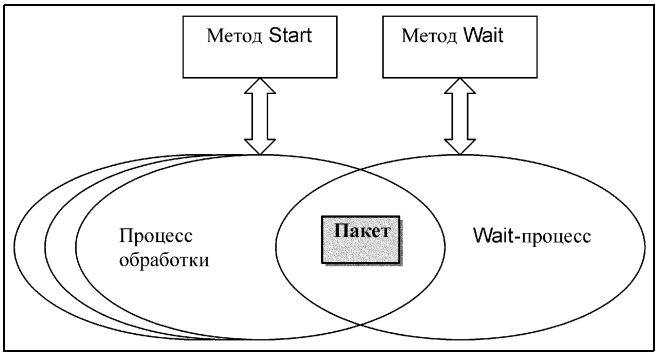

Рис. 3. Организация ожидания завершения обработки

ном завершении обработки), которое посылается по сети клиенту. Во втором — клиенту посылается диагностическое сообщение о "тайм-ауте". В любом случае выполнение метода Wait у клиента немедленно завершается.

6. МЕЖСЕРВЕРНЫЕ ВЗАИМОДЕЙСТВИЯ

Если для обработки пакета нужны обработчики сразу с нескольких RFPP-серверов, то возникает потребность в межсерверных взаимодействиях. Эта же потребность возникает, если необходим обмен данными между клиентами или агентами, зарегистрированными на разных RFPP-серверах. Другими словами, для этого два или более RFPP-сервера должны уметь взаимодействовать не только с собственными клиентами и агентами, но и друг с другом. Межсерверное взаимодействие основано на передаче пакетов от сервера к серверу (с аккумуляцией в них результатов обработки). При передачах пакета в его управляющей информации формируется "трасса", позволяющая отследить, на каких именно серверах он "побывал" и какие именно клиенты и агенты принимали участие в его обработке на каждом сервере. При инициации связи с удаленным сервером каждый RFPP-сервер выступает в роли RFPP-клиента.

Все доступные для взаимодействия удаленные серверы должны быть зарегистрированы. При регистрации удаленного сервера необходимо указать ряд его характеристик, основные из которых:

- регистрационное имя удаленного сервера;
- Интернет-имя или IP-адрес (и, возможно, порт) удаленного сервера;
- имя и пароль клиента, зарегистрированного на удаленном сервере, от имени которого наш сервер будет требовать обслуживания.

С точки зрения клиента межсерверное взаимодействие является частью обычной обработки, инициированной методом Process (или же парой методов Start и Wait). Это обращение выполняется в том случае, когда в списке имен вызываемых объектов (обработчиков, агентов и клиентов) обнаруживается специальная конструкция: "список удаленных объектов", имеющая следующий формат:

имя_сервера$(список_объектов)(тайм-аут),

где

- "имя_сервера" — регистрационное имя удаленного сервера;
- "список_объектов" — список имен обработчиков, агентов и клиентов на удаленном сервере (кроме имен, этот список может, в свою очередь, содержать "вложенные" списки удаленных объектов в качестве отдельных элементов);
- "таймаут" — число, отражающее максимальное время ожидания завершения обработки на удаленном сервере (в секундах).

Обработка списка удаленных объектов выполняется путем создания "пакета-клона" на удаленном сервере, переноса в него управляющей информации и наборов данных из обрабатываемого пакета и инициации обработки с помощью метода Start. После завершения обработки наборы данных и управляющая информация переносятся из пакета-клона в обрабатываемый пакет.

ЗАКЛЮЧЕНИЕ

Легко заметить, что RFPP содержит отдельные элементы таких сетевых служб, как FTP (протокол передачи файлов) или RPC (удаленный вызов процедур) [3]. Автор попытался показать, что объединение этих элементов в рамках общей концепции удаленных пакетов наборов данных дает "новое качество" и новые возможности обработки. Фактически, создание распределенного приложения или системы на базе RFPP сводится к разработке набора соответствующих обработчиков, клиентов и (или) агентов, использующих уже готовые средства взаимодействия.

Безусловно, работа сервера RFPP по созданию, поддержке и уничтожению удаленных пакетов можно отнести к "накладным расходам", связанным с реализацией протокола. Поэтому область его эффективного применения включает в себя взаимодействия, связанные с достаточно большими объемами передаваемой информации и с достаточно длительной обработкой (например, поиск информации в СУБД, удаленное формирование отчетов и т. п.). Наоборот, попытка создать на его основе что-то вроде DNS [3] вряд ли была бы оправданной.

В настоящее время RFPP-сервер реализован в двух вариантах: в форме сервиса для Windows2000 [4] и в форме "демона" для Unix FreeBSd [5]. В обоих вариантах пакеты и наборы данных организованы в файловой системе сервера. Клиентское ПО реализовано в форме библиотек функций: статической и динамической для Win32. Первые опыты применения RFPP показали его эффективность и удобство для разработчиков (в том числе для разработчиков, малознакомых с "сетевым" программированием).

ЛИТЕРАТУРА

1. *Асратян Р. Э., Волков А. Ф., Гаджиев А. М.* Автоматизированная система "Гибридная почта" // Труды ИПУ РАН. — 1999. — Т. IV. — С. 5—16.
2. *Фролов А. В., Фролов Г. В.* Глобальные сети компьютеров. — М.: Диалог-МИФИ, 1996. — 256 с.
3. *Паркер Т.* TCP/IP. Освой самостоятельно. — М.: Бином, 1997. — 448 с.
4. *Widows* 2000: Server и Professional / А. Г. Андреев, Е. Ю. Беззубов, М. М. Емельянов и др. — СПБ.: БХВ-Санкт-Петербург, 2001. — 1055 с.
5. *Келли-Бутл С.* Введение в Unix. — М.: ЛОРИ, 1995. — 596 с.

☎ *(095) 334-88-61*

E-mail: rea@L9.ipu.rssi.ru

❑