УДК 004.738

ВЗАИМОДЕЙСТВИЕ КЛИЕНТА И СЕРВЕРА В ИНТЕРНЕТ-СЛУЖБЕ ОБРАБОТКИ УДАЛЕННЫХ ПАКЕТОВ ФАЙЛОВ

Р.Э. Асратян

Рассмотрены принципы организации сетевого взаимодействия в Интернет-службе RFPS (службе удаленных пакетов наборов данных), предназначенной для поддержки распределенных систем управления. Акцент сделан на задаче восстановления взаимодействия после случайного или намеренного разрыва сетевого соединения без потери результатов обработки. Описано ее решение в службе RFPS, основанное на использовании удаленного пакета наборов данных в качестве основы для создания стабильного контекста взаимодействия, не разрушающегося в результате сетевых сбоев.

Ключевые слова: распределенная система управления, Интернет-служба, сетевое взаимодействие.

ВВЕДЕНИЕ

Появление Интернета дало новый импульс разработкам территориально-распределенных систем управления, пользующихся Интернетом для организации взаимодействий между удаленными компонентами. Вполне естественно, что разработчики распределенных систем традиционно опираются на Интернет-технологии (сетевые службы и протоколы), составляющие основу Всемирной сети (HTTP, SOAP, RPC, RMI и т. п.) [1—3].

Общая проблема всех онлайновых Интернет-служб, основанных на семействе протоколов TCP/IP [2, 4], состоит в их неустойчивости к сетевым сбоям. Причина коренится в общей для них логике взаимодействия между клиентом и сервером, согласно которой исходные данные информационного запроса и результаты его обработки не связывает между собой ничто, кроме того, что они передаются по одному и тому двустороннему TCP-каналу в рамках одного и того же сетевого соединения. Другими словами, сетевое соединение играет роль «контекста» взаимодействия — при его разрушении результаты обработки запроса оказываются потерянными для клиента.

По-видимому, при разработке этих служб и протоколов предполагалось, что клиентское приложение должно в этом случае или попросту повторить запрос заново или принять собственные меры, чтобы сохранить результаты запроса на сервере и впоследствии каким-то образом получить к ним доступ. Понятно, что такой подход приводит к появлению множества «частных решений» этой проблемы, привязанных к особенностям и логике работы конкретных систем. В наиболее распространенных сетевых технологиях (таких, как технология Web-сервисов [5] или новая технология WCF [6]) акцент сделан скорее на достижении максимальной производительности и совершенствовании пользовательского интерфейса, чем на защите от сетевых сбоев.

Безусловно, распределенные системы управления предъявляют более высокие требования к устойчивости взаимодействия, чем, например, Web-форум или сетевая служба новостей, так как цена потери данных может оказаться весьма высокой. Эти требования еще более возрастают в системах, в которых один информационный запрос может обрабатываться не в одном сервере, а в цепочке серверов (т. е. в системах, опирающихся на архитектуру «клиент — сервер — сервер —...— сервер»). Если организация системы предполагает возможность вовлечения в обработку одного запроса сразу нескольких серверов, то проблема защиты от сетевых сбоев не может быть решена на уровне прикладных программ, так как последние, как правило, не участвуют в организации межсерверных взаимодействий. Другими словами, в этом случае данная проблема должна быть решена на уровне самой Интернет-службы, обеспечивающей взаимодействие между сетевыми узлами. В данной работе мы рассмотрим принципы организации

таких взаимодействий в Интернет-службе RFPS (Remote File Packets Service), предназначенной для сетевой поддержки распределенных систем управления [7, 8]. В § 1 приводятся краткие сведения об этой технологии (более подробную информацию о целях создания RFPS, принципах его организации, преимуществах и недостатках в сравнении с другими сетевыми технологиями можно найти в работе [9]).

1. КРАТКИЕ СВЕДЕНИЯ О RFPS

Служба RFPS представляет собой сетевую технологию, специально ориентированную на организацию информационных взаимодействий в распределенных системах и приложениях. Ее главные особенности:

— наличие средств межсерверного взаимодействия и межсерверной маршрутизации данных;

— возможность обработки клиентского запроса в цепочке серверов;

— устойчивость к сетевым сбоям.

Семантика RFPS основана на понятии удаленного пакета наборов данных — поименованного контейнера информации, способного перемещаться по сети от сервера к серверу и аккумулировать результаты обработки на разных сетевых узлах. Основные формы обработки включают в себя:

— создание (открытие) пакета на обслуживающем сервере;

— заполнение его данными;

— применение к нему одного или нескольких обработчиков (программ, активизируемых по вызову);

— передачу пакета в очереди на обработку к агентам (постоянно активным программам, работающим на отдельных узлах);

— временное перемещение пакета на обработку в другой сервер и продолжение обработки на другом сервере (может выполняться рекурсивно в цепочке серверов);

—считывание результатов обработки из пакета по сети;

— закрытие (уничтожение) пакета.

Перечисленные формы обработки можно комбинировать в разных сочетаниях. Создав удаленный пакет на каком-либо сервере, RFPS, клиент может последовательно (или параллельно) применить к нему несколько обработчиков и (или) последовательно поставить его в очередь на обработку к нескольким агентам, и (или) переместить его на другой RFPS-сервер для продолжения обработки. При этом пакет данных в RFPS создает устойчивый «контекст» взаимодействия, позволяющий продолжить обработку даже после случайного (или намеренного) разрыва сетевого соединения.

Все компоненты, с которыми имеет дело сервер RFPS (клиенты, агенты, обработчики и серверы), должны быть предварительно зарегистрированы в его конфигурационных данных с указанием необ-

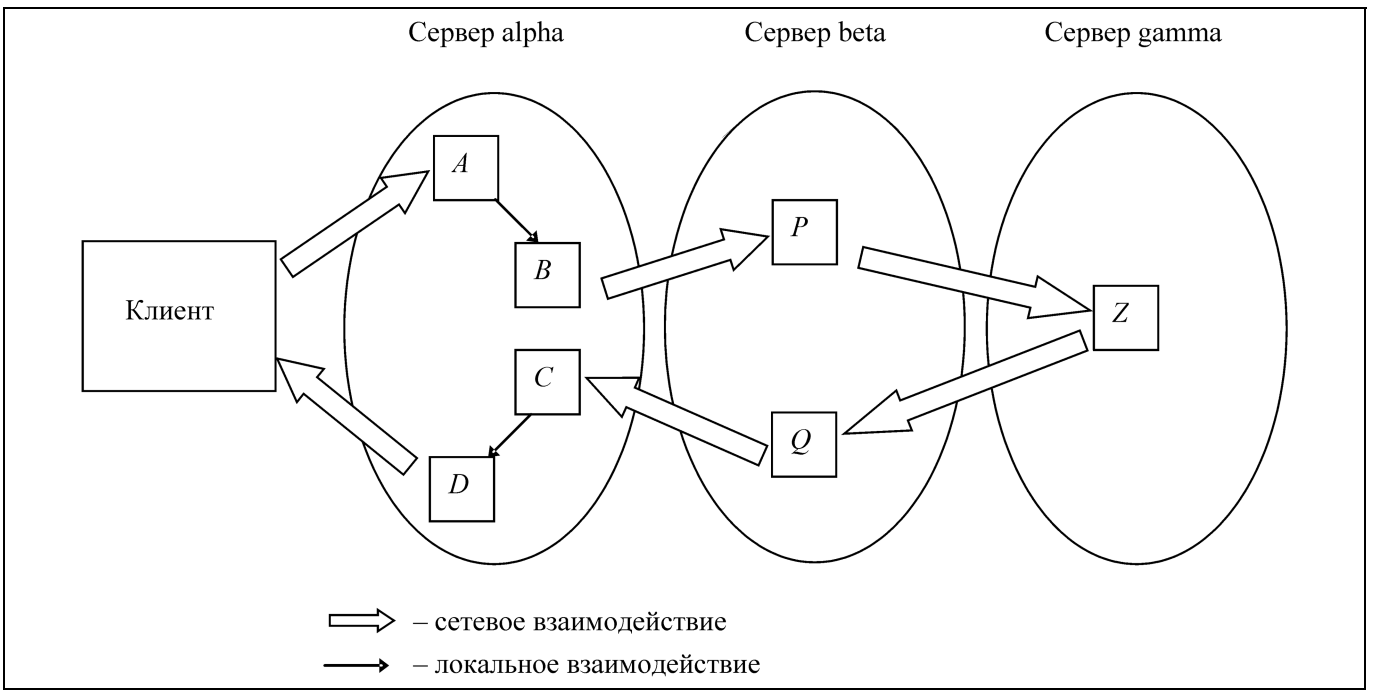

Рис. 1. Обработка пакета несколькими обработчиками на разных серверах

ходимых для работы характеристик (регистрационное имя и пароль клиента или агента, имя исполняемого модуля обработчика, Интернет-имя или IP-адрес сервера и т. п.). На рис. 1 проиллюстрирована обработка пакета в цепочке серверов с именами alpha, beta и gamma с помощью одного вызова клиентского метода «Process», имеющего два параметра: идентификатор пакета и командная строка, задающая последовательность вызовов обработчиков:

```
Client.Process(mypacket, «A, B, beta$(P, gamma$Z,
Q), C, D»).
```

Предполагается, что клиент соединен с сервером alpha, на котором создан обрабатываемый пакет (имя пакета сохранено в переменной mypacket). С помощью стрелок отображена последовательность вызова обработчиков, которые обозначены квадратиками. Выполнение вызова включает в себя:

— передачу пакета на обработку обработчикам *A* и *B* на сервере alpha;

— перемещение пакета на сервер beta и применение к нему обработчика *P*;

— перемещение пакета на сервер gamma, применение к нему обработчика *Z* и возврат пакета на сервер beta;

— применение к пакету обработчика *Q* на сервере beta и возврат пакета на сервер alpha;

— применение к пакету обработчиков *C* и *D* на сервере alpha.

Предполагается, что имена alpha, beta и gamma — регистрационные имена серверов, а *A*, *B*, *C*, ... — регистрационные имена обработчиков на соответствующих серверах.

2. ОРГАНИЗАЦИЯ ВЗАИМОДЕЙСТВИЯ МЕЖДУ КЛИЕНТОМ И СЕРВЕРОМ

Рассмотрим следующий фрагмент кода на языке C++, иллюстрирующий обращение к удаленному серверу со стороны клиента.

```
RFPSClient Rfppc; //Объект класса RFPSClient
char PackId[9]; //Строка для хранения
//идентификатора пакета
Rfppc.Config("192.168.1.77", 8126,"alibaba","sezam");
//Параметры соединения
Rfppc.Connect(); //Соединение с RFPS-сервером
Rfppc.Open(PackId); //Открытие пакета
Занесение файлов в пакет
Rfppc.SendFile(PackId,"c:\\out","file1.zip");
Rfppc.Process(PackId,"unzip,beta$trans(\"win-koi\"),
zip",300); //Обработка пакета
while(Rfppc.GetFile(PackId, "c:\\in", "*.*")==0);
//Считывание файлов из пакета
Rfppc.Close(PackId); //Закрытие пакета
Rfppc.Disconnect(); //Закрытие соединения
```

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

Четвертая строка устанавливает TCP-соединение с сервером (на сервере немедленно создается отдельный процесс для обслуживания этого соединения), а пятая — открывает на сервере новый пакет: в результате выполнения метода Open строка PackId окажется заполненной уникальным (для конкретного сервера) идентификатором открытого пакета. Все последующие методы будут пользоваться этим идентификатором.

Шестая строка обеспечивает передачу на сервер файла из каталога «c:\out». Можно представлять себе, что файл помещается по сети в открытый пакет. Седьмая строка инициирует обработку путем вызова метода Process. В качестве аргумента передается командная строка, обеспечивающая последовательный вызов трех обработчиков. Причем первый и последний (зарегистрированные под именами «unzip» и «zip») выполняются непосредственно на обслуживающем сервере, а второй («trans») — на сервере с именем «beta». Обработчику «trans» передается строковый параметр «win-koi», который становится доступным обработчику как параметр командной строки. Наборы данных пакета становятся доступны обработчику как файлы в текущем каталоге. Последний параметр метода Process — размер таймаута (в секундах), ограничивающий предельное время обработки.

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

Разумеется, из данного фрагмента кода намеренно удалены операторы проверки успеха выполнения методов и обработки исключений. Важно подчеркнуть, что идентификатор открытого пакета является обязательным параметром всех последующих методов, связанных с его обработкой.

Взаимодействие клиента и сервера на уровне TCP-соединения организуется по следующим правилам.

Сервер любой службы, основанной на семействе протоколов TCP/IP, постоянно занят «прослушиванием» входящих TCP-соединений на выделенном для него порте [4]. Зафиксировав входящее

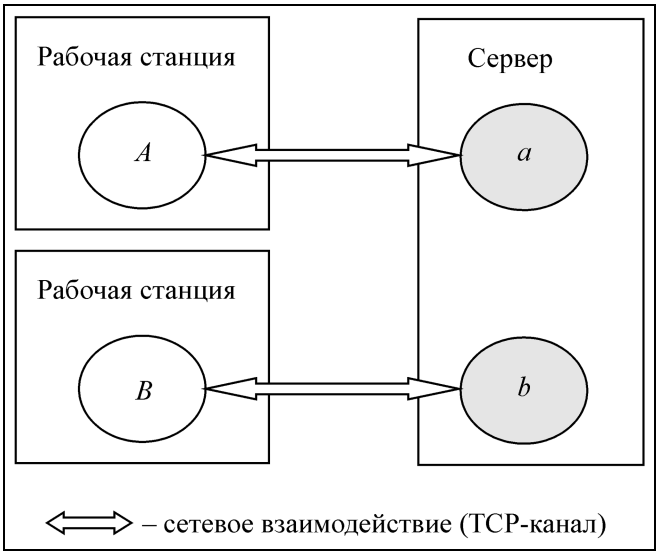

Рис. 2. Клиентские и обслуживающие процессы

соединение, сервер порождает отдельный обслуживающий процесс для обработки входящего запроса, создает двунаправленный TCP-канал между клиентским процессом (далее просто клиентом) и обслуживающим процессом и немедленно возвращается к прослушиванию входящих соединений. (Говоря более строго, для обработки запроса обычно порождается параллельная программная нить — program thread, разделяющая адресное пространство сервера, но в дальнейшем мы будем для краткости пользоваться термином «процесс».) Вся содержательная обработка запроса (получение кода операции, параметров и входных данных от клиента по TCP-каналу, собственно обработка запроса и возврат результатов клиенту) производится в обслуживающем процессе. (Например, в случае RFPS именно обслуживающие процессы обеспечивают создание и уничтожение пакетов наборов данных, вызов обработчиков, организацию обращения к удаленным серверам и т. п.) Данные правила проиллюстрированы на рис. 2, где клиентские процессы отображены белыми кружками и обозначены прописными буквами, а обслуживающие — темными кружками и строчными буквами.

Передав код операции и параметры обращения (и, возможно, данные для обработки) в созданный TCP-канал, клиент приступает к чтению результатов обработки из того же TCP-канала. Если результаты еще не готовы, клиент вновь переводится в неактивное состояние до появления результатов в TCP-канале. Если в процессе обработки произойдет неожиданное разрушение TCP-канала, операция чтения данных из канала вернет аномальный код завершения. Обрабатывающий процесс на сервере будет продолжать свою работу до того момента, когда он попытается записать в канал результаты обработки. В результате он также получит аномальный код завершения операции записи в канал, и ему не останется ничего другого, как просто завершить свою работу. Другими словами, в этом случае результаты обработки могут оказаться потерянными, если не принять специальных мер по их сохранению и обеспечению доступа к ним.

Рассмотрим как решается эта задача в RFPS на примере выполнения метода Process. В описанную схему взаимодействия клиента и сервера добавляется новая функциональность.

- При создании нового пакета в базе данных сервера RFPS создается новая запись — дескриптор пакета. В этой записи поддерживается идентификатор и управляющая информация пакета: имя создавшего клиента и дата создания, текущее состояние (обрабатывается или свободен), код завершения и системное диагностическое сообщение последней операции обработки, трасса (история) перемещения и обработки пакета, имена файлов, входящих в пакет и т. п. Эта запись удаляется только при закрытии пакета клиентом (или процедурой «сборки мусора», автоматически уничтожающей все слишком давно не обрабатываемые пакеты). Собственно наборы данных пакета размещаются в файловой системе сервера.

- При выполнении операции Process в памяти сервера RFPS создается дескриптор сеанса обработки, который содержит идентификатор обрабатывающего процесса в серверной ОС, время начала обработки, идентификатор пакета и несколько системных компонентов для синхронизации процесса обработки с другими процессами сервера (в реализации для платформы Win32 используются дескрипторы событий, а в реализации для UNIX — условные переменные). Сервер PFPS ведет учет всех открытых пакетов, сеансов и процессов таким образом, что каждый дескриптор сеанса и обслуживающий процесс ассоциируется с переданным ему (через параметры запроса) идентификатором пакета. В данном случае эта ассоциация означает, что по идентификатору пакета можно быстро определить адрес дескриптора сеанса обработки. (Разумеется, все это реализуется с помощью довольно сложной структуры данных сервера RFPS, включающей в себя связанные между собой дескрипторы сеансов, обслуживающих процессов и пакетов, но мы не будем останавливаться на описании этой структуры.)

- После завершения обработки каждого запроса обслуживающий процесс помещает код завершения обработки и системное диагностическое сообщение в дескриптор обрабатываемого пакета

и, кроме того, пытается передать их клиенту по TCP-каналу. Независимо от успеха этой передачи (соединение с клиентом может оказаться разорванным) обслуживающий процесс посылает сигнал активизации всем процессам, которые ждут окончания обработки пакета (если они имеются) и немедленно заканчивает работу.

• Если при издании запроса процесс-клиент обнаруживает, что соединение с сервером разрушено, он автоматически восстанавливает его (с порождением нового процесса на сервере). Если разрыв соединения произошел в период ожидания кода завершения от обслуживающего процесса, процесс-клиент немедленно издает новый специальный запрос к серверу RFPS: «Ожидание окончания обработки» (с помощью специального клиентского метода Wait) и возвращается в состояние ожидания кода завершения. (Это поведение заложено в клиентские методы класса RFPSClient и реализуется автоматически).

• Запрос «Ожидание окончания обработки» порождает новый процесс на сервере RFPS — процесс ожидания — и новый TCP-канал. Этот процесс прежде всего проверяет состояние пакета по его дескриптору. Если обработка пакета уже завершена, то процесс ожидания попросту считывает его код завершения и диагностическое сообщение из дескриптора сеанса пакета, передает этот код процессу-клиенту по TCP-каналу и заканчивает свою работу. В противном случае он определяет адрес дескриптора сеанса обработки и переходит в состояние ожидания завершения обработки пакета. Эта операция может итеративно повторяться несколько раз: если процесс ожидания также потеряет связь с процессом клиентом, то будет порожден новый процесс ожидания и т. д. На рис. 3 изображена ситуация, когда в рамках сеанса обработки порождено несколько процессов (обозначены серыми кружками), один из которых (*c*) — обслуживающий, а остальные (*w*) — процессы ожидания.

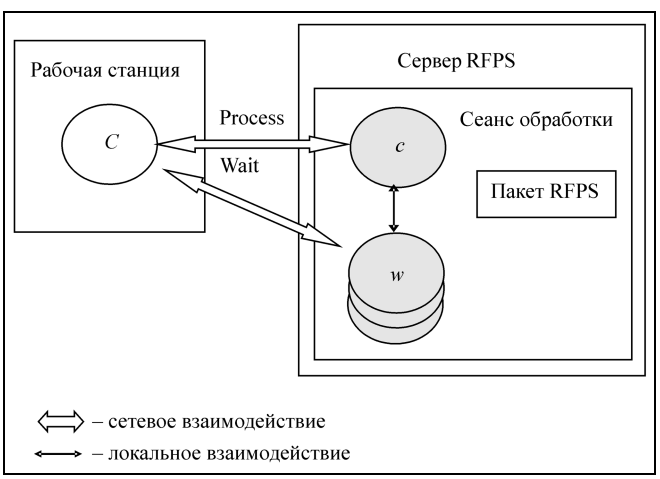

Рис. 3. Обслуживающий процесс и процессы ожидания

• После завершения обработки обслуживающий процесс проверяет, имеются ли процессы ожидания, связанные с тем же сеансом связи. Если есть, он посылает сигнал активизации всем этим процессам и заканчивает свою работу. Тот из процессов ожидания, который имеет связь с процессом-клиентом, передает ему код завершения обслуживающего процесса по TCP-каналу, после чего все процессы ожидания, порожденные в рамках данного сеанса обработки, также заканчивают свою работу.

3. ОРГАНИЗАЦИЯ ВЗАИМОДЕЙСТВИЯ МЕЖДУ АГЕНТОМ И СЕРВЕРОМ

Агенты, как и обработчики, предназначены для обработки пакетов. Главное отличие агентов от обработчиков заключается в следующем:

— агенты представляют собой постоянно активные или периодически запускаемые программы (а не запускаются «по вызову»);

— агенты могут работать на отдельной рабочей станции и взаимодействовать с RFPS-сервером по сети (в этом отношении агенты подобны клиентам);

— агенты получают данные для обработки из входной очереди пакетов.

Отправка пакета на обработку агенту осуществляется с помощью метода Process: если в командной строке имеется имя агента, то обрабатываемый пакет переносится во входную очередь этого агента, а выполнение метода Process задерживается до возврата пакета из очереди.

Как и программа клиента, программа агента должна связываться с обслуживающим сервером по сети (фактически, агентов можно рассматривать как клиентов, наделенных дополнительной функциональностью). Основу программы агента должен составлять цикл последовательной выборки пакетов из входной очереди (с помощью специальных методов GetCount и Select) и их обработки. Обработка каждого отдельного пакета включает в себя сетевой обмен данными между рабочей станцией агента и RFPS-сервером (например, с помощью методов GetFile и SendFile). Подчеркнем, что метод Select обеспечивает получение лишь идентификатора пакета, но не данных. В ка-

честве примера рассмотрим «скелет» исходного кода программы агента на языке C++:

```
RFPSClient Rfppc; //Объект класса RFPSClient
int QueLen; //Переменная для числа пакетов
//в очереди
char PackId[9]; //Строка для хранения
//идентификатора пакета
Rfppc.Config("192.168.1.77", 8126,"agent_007",
"sezam"); //Параметры соединения
Rfppc.Connect(); //Соединение с RFPS-сервером
while(1) //Начало «вечного» цикла обслуживания
//очереди
{
   Rfppc.GetCount(QueLen, 180); //Ожидание
   //пакетов в очереди (до 3 мин)
   If(QueLen<=0) continue; //Если их нет — снова
   //ожидание
   if(Rfppc.Select(1,PackId)==0) //Получение ид.
   //первого пакета в очереди
   {
       while(Rfppc.GetFile(PackId, "c:\\inp", "*.*")==0);
       //Считывание файлов из пакета
       . . . //Обработка
       Rfppc.Return(PackId); //Возвращение пакета
       //клиенту-отправителю
   }
}
```

Как видно из примера, основу программы составляет вечный цикл (предложение while), в котором осуществляется:

— ожидание появления пакетов в очереди;

— последовательная выборка всех пакетов из очереди в рабочую область агента (с получением идентификатора каждого пакета);

—считывание файлов из каждого пакета;

— обработка данных пакета (ее исходный текст не приводится в примере);

— возврат пакета клиенту-отправителю (с автоматической активизацией его обслуживающего процесса).

Существует некоторая аналогия между очередью пакетов и почтовым ящиком электронной почты. В данном примере отражено их важное различие, связанное с возможностью синхронизации программы агента с событием появления пакетов во входной очереди с помощью метода GetCount. Данный метод возвращает число пакетов в очереди (в первом параметре), если она не пуста, или переводит программу агента в состояние ожидания в противном случае. Ожидающая программа будет автоматически активизирована после появления пакетов в очереди или после истечения таймаута (второй параметр).

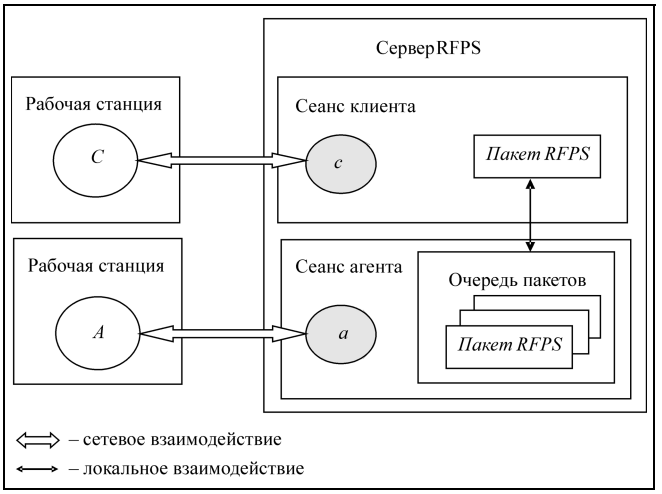

Рис. 4. Обработка пакетов агентом

Важно подчеркнуть, что агент — активное звено, а сервер — пассивное. Агент периодически обращается к обслуживающему серверу для выборки пакетов из собственной (поддерживаемой на сервере) входной очереди. Если агент потерял работоспособность, то входная очередь перестанет обслуживаться. По истечении заданного клиентом таймаута (размер таймаута является опциональным параметром метода Process) обслуживающий клиента процесс будет уничтожен на сервере, а клиент, обратившийся к агенту, получит сообщение о таймауте операции.

Как и в метод Process, методы GetCount и Select опираются на устойчивый контекст сеанса связи, но в данном случае он связывается не с пакетом, а с очередью пакетов. При обработке этих методов нет необходимости в «процессах ожидания» — в случае разрыва сетевого соединения обращение к серверу попросту повторяется. На рис. 4 проиллюстрирована передача пакета RFPS в очередь агента и обратно. Белыми кружками обозначены процессы клиента и агента (*C* и *A* соответственно), а темными — обслуживающие процессы на сервере RFPS.

Наконец, несколько слов о выполнении методов SendFile и GetFile, предназначенных для обменов данными между пакетом RFPS и файловой системой клиента или агента (и их аналогов SendMem и GetMem, предназначенных для обменов между пакетом и оперативной памятью). В случае разрыва сетевого соединения при передаче набора данных по сети данные методы обеспечивают автоматическое восстановление соединения и продолжение обмена с «точки разрыва», а не с начала набора данных. (При этом применяется примерно та же технология восстановления обмена,

что и в протоколе FTP [1, 2], которая основана на передаче дополнительного параметра — смещения в наборе данных — от узла-получателя в узел-отправитель.)

ЗАКЛЮЧЕНИЕ

Семантика RFPS построена таким образом, чтобы все основные сетевые операции, связанные с передачей и удаленной обработкой данных, выполнялись в рамках стабильного контекста, независимого от сетевого соединения. В зависимости от операции, этот контекст связывается или с удаленным пакетом наборов данных, или с очередью пакетов. Это свойство RFPS избавляет прикладного программиста от необходимости разрабатывать собственные средства защиты от сетевых сбоев, что особенно важно в аспекте организации сложных межсерверных взаимодействий.

В статье не рассмотрена организация взаимодействия между серверами при обработке пакета в цепочке серверов. Отметим лишь, что оно построено примерно по тем же правилам, что и взаимодействие «клиент — сервер», так как при обращении к удаленному серверу вызывающий сервер выступает «в роли» клиента. И в этом случае контекст взаимодействия связывается с пакетом наборов данных, а не с сетевым соединением между серверами.

К недостаткам RFPS следует в первую очередь отнести относительно низкий уровень пользовательского интерфейса и отсутствие формализованных спецификаций обработчиков и агентов. В этом отношении он, безусловно, проигрывает, например, технологии Web-сервисов [5], опирающейся на формализованное описание удаленных процедур на языке WSDL и поддержанной в ряде распространенных систем программирования.

Одно из эффективных направлений применения RFPS состоит в создании «прозрачных» RFPS-туннелей между HTTP-клиентами и HTTP-серверами [10]. Суть подхода заключается во временной инкапсуляции HTTP-запроса и HTTP-ответа в пакет RFPS при прохождении данных через глобальную сеть в ситуации, когда HTTP-клиент и HTTP-сервер расположены в удаленных друг от друга частных сетях. Этот подход позволяет в значительной степени совместить преимущества технологии Web-сервисов (высокоуровневый пользовательский интерфейс) и преимущества RFPS (устойчивость к сетевым сбоям и наличие встроенных средств межсерверного взаимодействия и межсерверной маршрутизации данных). В процессе лабораторных экспериментов с RFPS, в частности, было проведено встраивание RFPS-туннелей в несколько ранее разработанных распределенных систем, построенных на основе технологии Web-сервисов. (Подход может быть применен и для систем, построенных на новой технологии WCF [6], при условии, что в качестве «транспорта» выбран HTTP.) Важно отметить, что ни в одном случае не понадобилась какая-либо программная адаптация этих систем, что подтверждает «прозрачность» туннелей для HTTP-клиентов и серверов.

ЛИТЕРАТУРА

1. *Фролов А.В., Фролов Г.В.* Глобальные сети компьютеров. — М.: Диалог-МИФИ, 1996. — 256 с.
2. *Шиндер Д.Л.* Основы компьютерных сетей. — СПб.: Вильямс, 2003. — 656 с.
3. *Гофф М.* Сетевые распределенные вычисления: достижения и проблемы. — М.: КУДИЦ-ОБРАЗ, 2005. — 320 с.
4. *Джамса К., Коуп К.* Программирование для Интернет в среде Windows. — СПб.: «Питер», 1996. — 659 с.
5. *Шапошников И.В.* Web-сервисы Microsoft .NET. — СПб.: БХВ-Петербург, 2002. — 336 с.
6. *Леве Д.* Создание служб WCF. — СПб: Питер, 2008. — 592 с.
7. *Асратян Р.Э.* Интернет-служба для поддержки распределенных информационно-управляющих систем // Проблемы управления. — 2005. — № 6. — С. 73—77.
8. *Асратян Р.Э.* Интернет-служба для поддержки межсерверных взаимодействий в распределенных информационных системах // Проблемы управления. — 2006. — № 5. — С. 58—62.
9. *Асратян Р.Э., Лебедев В.Н.* Средства информационного взаимодействия в современных распределенных гетерогенных системах. — М.: ЛЕНАНД, 2008. — 120 с.
10. *Асратян Р.Э.* Межсерверная маршрутизация HTTP/SOAP-взаимодействий в распределенных системах // Проблемы управления. — 2008. — № 5. — С. 57—61.

Статья представлена к публикации руководителем РРС В.Ю. Столбовым.

Асратян Рубен Эзрасович — канд. техн. наук,
вед. науч. сотрудник, Институт проблем управления
им. В.А. Трапезникова РАН, г. Москва,
☎ (495) 334-88-61, ✉ rea@ipu.ru.