Лекция 18:
Удаленный вызов процедур
Remote Procedure Call, RPC

Алексей Линёв
Александр Мощук
Кирилл Погорельский

somes slides are adapted from the OS course at the University of Washington

Клиент-серверное взаимодействие

- Наиболее распространенной моделью структурирования распределенных вычислений является клиент-серверная парадигма
  - сервер – это программа (или набор программ), которая предлагает услуги другим программам
    - например, сервер файлов, сервер печати, web-сервер, почтовый сервер,…
    - сервер/сервис может располагаться на нескольких узлах (на кластере)
      - часто сами узлы тоже называют серверами
      - например, web-сервер выполняется на серверe – компьютере Dell
  - клиент – это программа, использующая сервис
    - клиент связывается с сервером
      - находит его, устанавливает с ним сетевое соединение
    - затем клиент посылает запрос (с данными) для выполнения обработки, а сервер посылает ответ (с результатами обработки)
      - например, web-клиент посылает запрос "GET", сервер отсылает ему web-страницу
- TCP/IP – механизм передачи, но какая модель взаимодействия используется на более высоком уровне?

Сообщения

- Вначале разработчики "вручную" организовывали передачу сообщений – запросов и ответов
  - сообщение – это поток байтов – кодов операций и аргументов
- Недостатки
  - необходимо заботится о форматах сообщений
  - необходимо упаковывать данные в сообщения, а впоследствии извлекать их из сообщения
  - серверы должны разбирать служебную информацию сообщений и доставлять их обработчикам
  - сообщения часто приходят асинхронно
    - после отправки сообщения, что вы будете делать до прихода ответа?
  - использование сообщений – не самая естественная модель в программировании

Вызовы процедур

- Вызовы процедур – естественный способ взаимодействия нескольких модулей в рамках одной программы
  - все языки программирования поддерживают вызовы процедур
  - семантика четко определена и предельно понятна
  - программисты привычны к их использованию
- "Сервер" (вызываемая процедура) экспортирует API
- "Клиент" (вызывающая процедура) вызывает API процедуры сервера
- Линковщик связывает их друг с другом

Пример вызова процедур

Программа-клиент:

```
…
sum = server->Add(3,4);
…
```

API сервера:

```
int Add(int x, int y);
```

Программа-сервер:

```
int Add(int x, int y) {
    return x + y;
}
```

- Если сервер будет просто библиотекой, "Add" будет локальным вызовом процедуры

Удаленный вызов процедур (Remote Procedure Call, RPC)

- Использует традиционные синтаксис и семантику вызова процедур при взаимодействии через сеть
- Наиболее используемый механизм при удаленном взаимодействии в клиент-серверных системах
- Используется как операционными системами, так и прикладными программами
  - NFS реализована в виде набора RPC
  - HTTP в большой степени тоже RPC
  - DCOM, CORBA, Java RMI,… – это все RPC-системы
- Возможно, когда-нибудь вам тоже придется разрабатывать приложение, использующее сетевое взаимодействие
  - и вы сможете использовать для организации такого взаимодействия модели и механизмы RPC

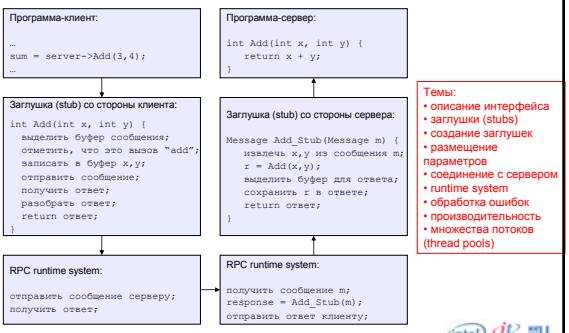
RPC

- Использование вызова процедур в качестве модели распределенного (удаленного) взаимодействия
  - серверы экспортируют набор процедур, которые могут быть вызваны прикладными программами
    - очень похоже на API библиотеки или объявление класса
  - клиенты выполняют локальный вызов процедуры, как будто они непосредственно слинкованы с сервером
    - в реальности, механизм удаленного вызова процедур преобразует вызов в обмен сообщениями с сервером
    - ***практически прозрачно для разработчика!***

RPC: вопросы/проблемы

- Имеется масса вопросов и проблем
  - как сделать так, чтобы сделать удаленную природу вызова прозрачной для программиста?
    - и является ли это хорошей идеей?
  - какова семантика процесса передачи параметров?
    - что будет, если мы попытаемся передать параметр по ссылке?
  - как мы подключаемся (находим и соединяемся) к серверу?
  - как работать в гетерогенной среде?
    - архитектура, ОС, язык программирования,…
  - как сделать механизм RPC быстрым?

RPC: пример использования



RPC: модель

- Сервер задает интерфейс сервиса, используя язык описания интерфейсов (interface definition language, IDL)
  - с помощью IDL объявляются имена, типы и параметры для всех видимых для клиента процедур сервера
    - пример: ASN.1 в ISO/OSI Reference Model
    - пример: Sun XDR (external data representation)
- "Редактор связей" ("stub compiler") обрабатывает объявления XDR и генерирует по две заглушки для каждой процедуры сервера
  - разработчик сервера реализует сервисные процедуры и линкует их с заглушками стороны сервера (server-side stubs)
  - разработчик клиента реализует клиентское приложение и линкует его с заглушками стороны клиента (client-side stubs)
  - в заглушках реализованы все детали взаимодействия между клиентом и сервером на основе использования подсистемы RPC времени исполнения (RPC runtime system)

RPC: заглушки (stubs)

- Заглушка стороны клиента – это процедура, которая с точки зрения клиента выглядит в точности так же, как доступная процедура сервера
  - она имеет такой же API, как и серверная реализация
  - заглушка стороны клиента в Java RMI называется просто "заглушка"
- Заглушка стороны сервера похожа на вызывающий сервер клиент
  - она выглядит просто как кусок кода, вызывающий процедуру сервера
  - заглушка стороны сервера в Java RMI называется "skeleton" или "skel"
- Программа-клиент думает, что она вызывает сервер
  - но вызывает заглушку стороны клиента
- Программа-сервер думает, что ее вызывает клиент
  - на самом деле ее вызывает заглушка стороны сервера
- Заглушки обмениваются сообщениями через подсистему RPC, делая механизм прозрачным для вышележащего кода

RPC: передача параметров и результата

- Передача параметров – это упаковка параметров вызова процедуры в блок сообщения
  - заглушки RPC вызывают для различных типов параметров специальные процедуры, позволяющие поместить параметр в сообщение или извлечь его из сообщения
    - заглушка стороны клиента помещает параметры в сообщение
    - заглушка стороны сервера извлекает параметры и использует их при вызове процедуры сервера
  - при завершении (return)
    - заглушка стороны сервера помещает возвращаемое значение
    - заглушка стороны клиента извлекает возвращаемое значение и передает его в программу-клиент

RPC: соединение с сервером (binding)

- Binding – это процесс установления клиентом соединения с сервером
  - сервер при запуске экспортирует интерфейс
    - регистрирует себя на сетевом сервере имен
    - уведомляет подсистему RPC о том, что он готов к обработке вызовов
  - клиент перед вызовом функций получает (импортирует) информацию о сервере
    - подсистема RPC использует сервер имен для поиска сервера и установления соединения
- Операции экспортирования и импортирования явно присутствуют в программах-клиентах и программах-серверах
  - небольшое нарушение принципа прозрачности

RPC: прозрачность

- Одна из задач механизма RPC – быть максимально прозрачным
  - сделать удаленный вызов процедур максимально похожим на локальный вызов процедур
  - мы уже увидели, что процесс соединения с сервером не является полностью прозрачным
- Что еще нарушает прозрачность?
  - сбои: существует гораздо больше причин выхода из строя удаленных узлов или сети, чем причин невыполнения локального вызова процедур
    - разрыв физического соединения, зависание или поломка сервера,…
    - необходима дополнительная обработка сбоев
    - сервер может выйти из строя независимо от клиента
      - "сбой по причине физического разрыва" – большая проблема для распределенных систем
      - если в процессе обработки RPC произошел сбой, что вызывается на стороне сервера?
  - производительность: удаленное взаимодействие по определению медленнее, чем локальное взаимодействие
    - если при использовании RPC вы не будете думать об их производительности, ваша программа может работать ужасно медленно

RPC: thread pools (множества потоков)

- Что произойдет, если два клиентских потока (или две программы-клиента) одновременно выполнят вызов одной и той же процедуры через RPC?
  - в идеале, на сервере запустятся два отдельных потока
  - в этом случае, подсистема RPC на сервере должна при приходе сообщения создать или выделить потоки для выполнения заглушки стороны сервера
    - есть ли ограничение на максимальное число потоков?
    - если есть, изменит ли такое ограничение семантику?
    - если нет, что будет при одновременном вызове одной и той же RPC-процедуры миллионом клиентов?