ТЕМА 14. ОСНОВНЫЕ ВИДЫ КОММУНИКАЦИЙ РАСПРЕДЕЛЕННЫХ СИСТЕМ. УДАЛЕННЫЙ ВЫЗОВ ПРОЦЕДУР

МОТИВАЦИЯ ИСПОЛЬЗОВАНИЯ RPC

- Использовать низкоуровневый обмен сообщениями на основе функций встроенных в язык программирования (примитивов) *send* и *receive*. primitives.
- Стремление к обеспечению прозрачности на уровне доступа (*access transparency)*.
  - Сокрытие различий в представлениях данных на машинах, в понимании процессов обмена сообщениями и т.п. Другими словами - сделать RPC прозрачным: вызывающей процедуре не требуется знать, что вызываемая процедура находится на другой машине, и наоборот.
- Упрощение программирования процесса обмена информацией за счет использования методов подобных тем, которые применяются при обмене через разделяемую память при локальном вызове процедур.
- Впервые идея RPC была предложена Birrell and Nelson (1984)

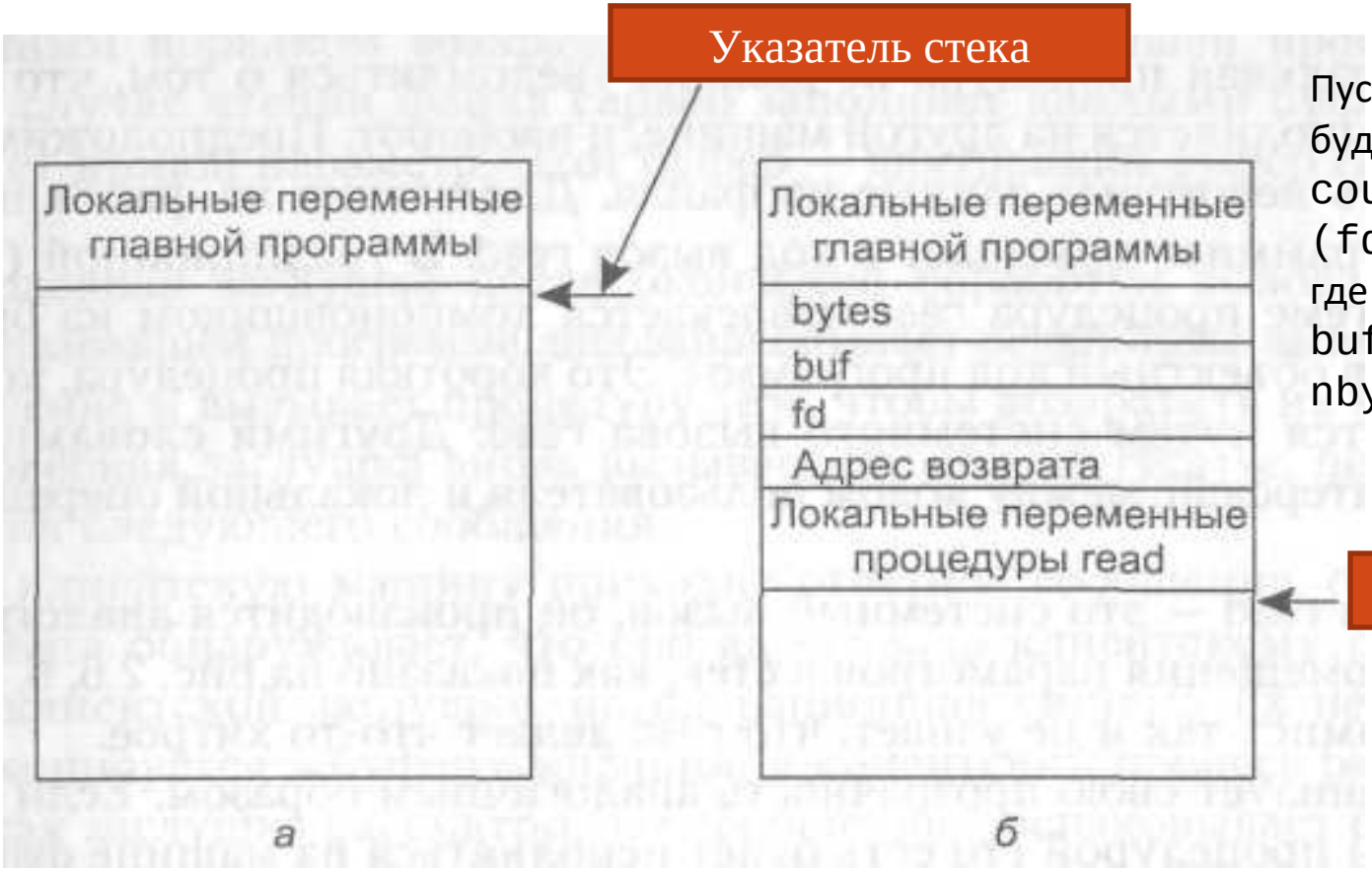
МОДЕЛЬ REMOTE PROCEDURE CALL (RPC)

- Использование высокоуровневых интерфейсов сетевых коммуникаций, предоставляемых ОС.
- Основан **на модели** вызова **локальных процедур** в рамках одного процесса.
- Запрос клиента: оформляется как **вызов процедуры** выполнения функции на сервере.
- Ответ сервера: оформляется как **возвращение результата** выполнения вызванной функции.

ТРАДИЦИОННЫЙ (ЛОКАЛЬНЫЙ) ВЫЗОВ ПРОЦЕДУРЫ. ИНИЦИАЦИЯ

- Инициируется вызовом процессом функции или процедуры.
- Вызывающий процесс приостанавливается (suspended) до тех пор пока выполнение вызываемой процедуры не будет завершено.
- Аргументы и адрес возврата помещаются в стек процесса.
- Локальные переменные для использования в вызываемой процедуре также помещаются в стек.

ПЕРЕДАЧА ПАРАМЕТРОВ ПРИ ЛОКАЛЬНОМ ВЫЗОВЕ ПРОЦЕДУРЫ



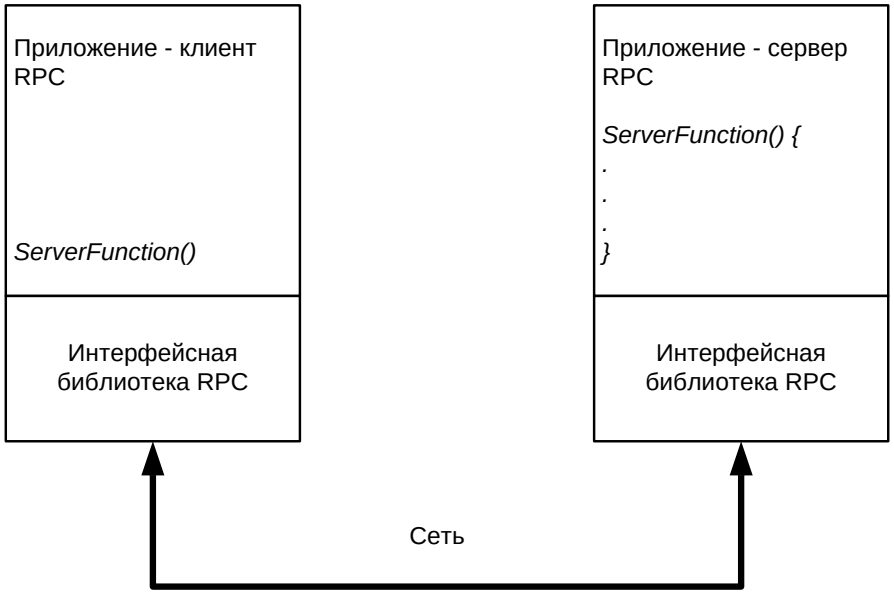
Пусть это, например, будет системный вызов `count=read (fd,buf,nbytes);` где `fd` - целое число, `buf` - массив символов, `nbytes` - целое число.

```
count = read(fd, buf, bytes);
```

Передача параметров при локальном вызове процедуры:

- Стек до вызова функции `read` (а).
- Стек во время работы вызванной процедуры (б)

ФУНКЦИОНИРОВАНИЕ RPC



- Приложения RPC похожи на другие структурированные приложения:
  - у них есть основная программа, которая для выполнения специфических задач вызывает процедуры или библиотеки процедур.
- Отличие приложений RPC от обычных программ в том, что некоторые библиотеки процедур в приложениях RPC выполняются на удаленных компьютерах, а некоторые — на локальном.
- Для приложения RPC все процедуры кажутся локальными.

•Функционирует приложение RPC следующим образом. В процессе своей работы оно вызывает как локальные процедуры, так и процедуры, отсутствующие на локальной машине.

•Для обработки последнего случая приложение связывается с локальной DLL, которая содержит интерфейсные процедуры (**stub procedures**) для всех удаленных процедур. В простой программе интерфейсные процедуры статически связываются с приложением, но в компоненте большего размера они включаются в отдельные DLL.

•В DCOM обычно применяется последний метод. **Интерфейсная процедура** имеет то же имя и тот же интерфейс, что и удаленная процедура, но вместо выполнения соответствующей операции она просто преобразует переданные ей параметры для передачи по сети — такой процесс называется *маршалингом* (marshaling).

•Маршалинг заключается в упорядочении параметров и их упаковке в определенном формате.

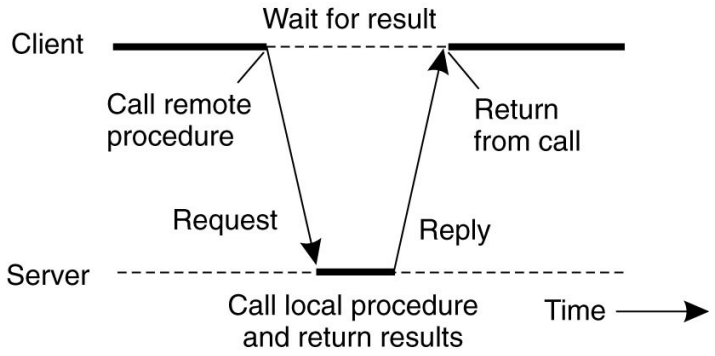
•Далее интерфейсная процедура вызывает процедуры библиотеки RPC периода выполнения, и они находят компьютер, на котором расположены удаленные процедуры, определяют используемые этим компьютером механизмы транспорта и посылают запрос при помощи локального программного обеспечения сетевого транспорта. Когда удаленный сервер получает запрос RPC, он выполняет обратное преобразование параметров

ЛОКАЛЬНЫЙ ВЫЗОВ ПРОЦЕДУР. ВЫПОЛНЕНИЕ

- Управление передается вызываемой функции (процедуре)
- Вызванная функция выполняется и по завершении работы возвращает результаты либо через параметры (стек), либо сохраняет их в регистрах CPU.
- Выполняется извлечение данных из стека.
- Процесс вызывавший процедуру продолжает свою работу.

ОСОБЕННОСТИ REMOTE PROCEDURE CALLS

- Основные операции RPC выполняются параллельно с выполнением процесса вызвавшем RPC.
- Процесс вызвавший RPC выполняет удаленный вызов при этом его **работа приостанавливается** до завершения выполнения вызываемой функции и возвращения результатов.
- Параметры передаются на машину, на которой процедура будет исполняться.
- Когда выполнене вызываемой процедуры будет **завершено**, результаты передаются назад на машину источник вызова, после чего клиентский процесс **продолжит** свою работу.



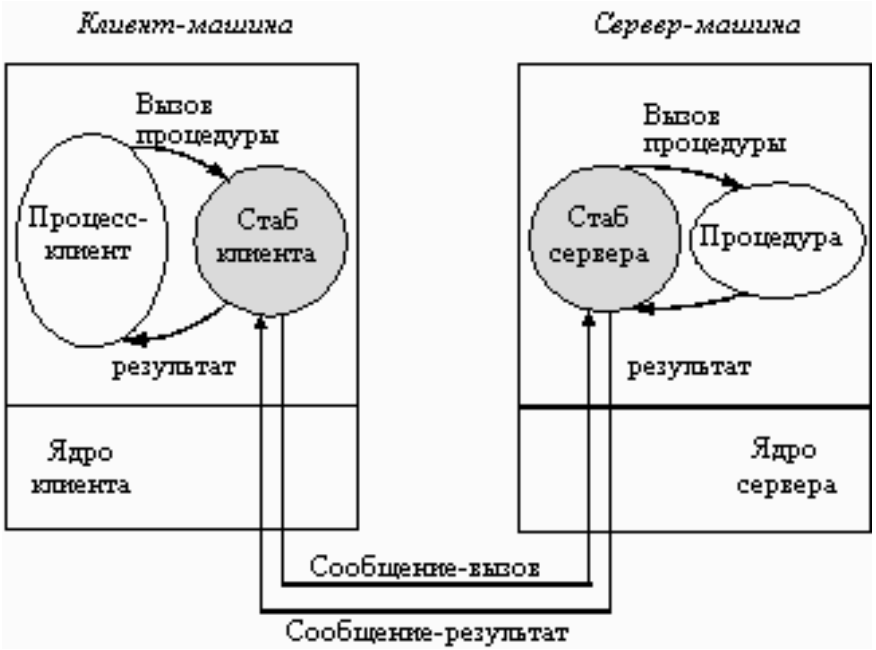
RPC И МОДЕЛЬ КЛИЕНТ-СЕРВЕР

- RPC лежит в основе большинства систем клиент-сервер.
- Клиенты формулируют запросы к серверам в виде вызовов процедур.
- Прозрачность доступа обеспечивает реализация механизма RPC
- Существующие реализации RPC ?

РЕАЛИЗАЦИЯ ПРОЗРАЧНОСТИ: ИСПОЛЬЗОВАНИЕ ПРОГРАММНЫХ ЗАГЛУШКЕ (STUBS)

- Программные заглушки создаются для каждого вызова RPC (по одной для каждого RPC)
- Передача данных серверной процедуре. Для каждого RPC управление потоком передаваемых данных выполняется с помощью:
  - Приложение клиент управляет обменом данных с клиентской заглушкой (client-side stub)
  - Клиентская заглушка (client stub) управляет передачей параметров серверной заглушке (server stub).
  - Серверная заглушка (server stub) обеспечивает передачу данных серверной процедуре (server procedure).
- Возврат данных от серверной процедуры. В этом случае управление распределяется следующим образом:
  - Серверная процедура передает результаты выполнения серверной заглушке.
  - Серверная заглушка отсылает результаты по сети клиентской заглушке.
  - Клиентская заглушка передает результаты клиентскому приложению.

ЗАГЛУШКИ КЛИЕНТА И СЕРВЕРА



- Специальная версия функции read, называемая *клиентской заглушкой {client stub}*

- Серверная заглушка эквивалентна клиентской, но работает на стороне сервера.

РАБОТА КЛИЕНТСКОЙ ЗАГЛУШКИ

- Когда приложение выполняет вызов RPC клиентская заглушка выполняет:
  - Создает сообщение содержащее параметры и обращается к локальной ОС с вызовом функции (*send)* для отправки сообщения.
  - Процесс подготовки параметров в виде пакета передаваемого по сети называется маршалингом параметров (**parameter marshalling**).
  - Выполняет вызов процедуры *receive( )* для ожидания ответа (блокирующая функция receive).

ДЕЙСТВИЯ ВЫПОЛНЯЕМЫЕ OS (OS LAYER ACTIONS)

- Клиентская ОС отправляет сообщение к удаленной машине по сети.
- Удаленная ОС передает принятое сообщение серверной заглушке.

ДЕЙСТВИЯ ВЫПОЛНЯЕМЫЕ СЕРВЕРНОЙ ЗАГЛУШКОЙ

- Распаковка параметров, выполнение вызова серверной процедуры.
- После выполнения процедуры сервером, полученный результат поступает серверной заглушке. Серверная заглушка пакует результат в ответное сообщение.
- Выполняет вызов ОС для отправки сообщения на клиентскую машину.

ДЕЙСТВИЯ ВЫПОЛНЯЕМЫЕ СЕРВЕРНОЙ И КЛИЕНТСКОЙ ОС

- Серверная ОС отсылает сообщение – ответ клиенту.
- Клиентская ОС принимает сообщение – ответ и передает его клиентской заглушке.

ДЕЙСТВИЯ ВЫПОЛНЯЕМЫЕ КЛИЕНТСКОЙ ЗАГЛУШКОЙ ПРИ ПОЛУЧЕНИИ ОТВЕТА

- Клиентская заглушка распаковывает ответное сообщение и передает результат клиентскому приложению, используя один из нормальных механизм передачи результатов от функций:
  - Либо напрямую, в виде значений,
  - Либо через параметры вызова функции.

РАБОТА RPC

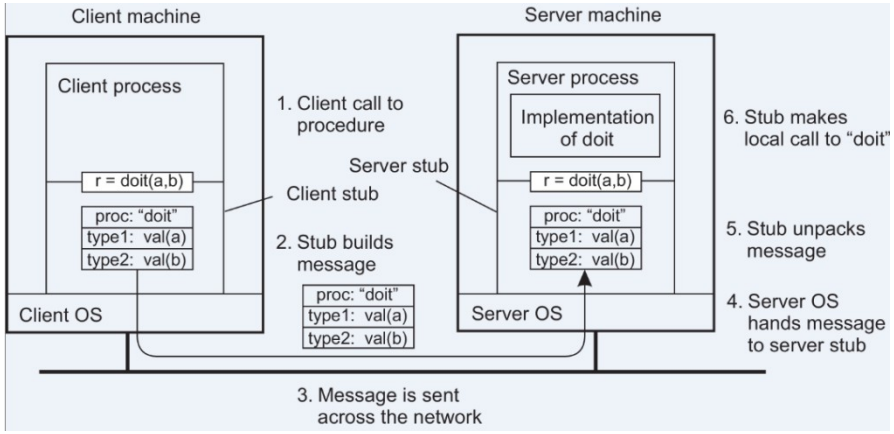
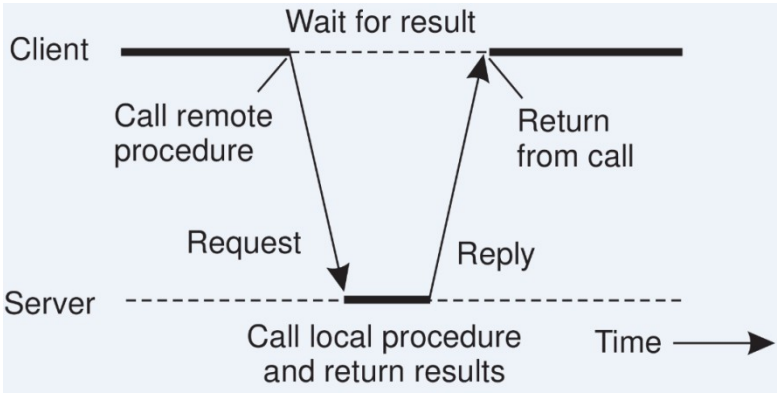


СХЕМА ОБМЕНА МЕЖДУ СЕРВЕРОМ И КЛИЕНТОМ RPC



При удаленном вызове процедур происходят следующие действия:

1. Процедура клиента обычным образом вызывает клиентскую заглушку.
2. Клиентская заглушка создает сообщение и вызывает локальную операцион ную систему.
3. Операционная система клиента пересылает сообщение удаленной операцион ной системе.
4. Удаленная операционная система передает сообщение серверной заглушке.
5. Серверная заглушка извлекает из сообщения параметры и вызывает сервер.
6. Сервер выполняет вызов и возвращает результаты заглушке.
7. Серверная заглушка запаковывает результаты в сообщение и вызывает свою локальную операционную систему.
8. Операционная система сервера пересылает сообщение операционной системе клиента.
9. Операционная система клиента принимает сообщение и передает его клиентской заглушке.
10. Заглушка извлекает результаты из сообщения и передает их клиенту.

ПЕРЕДАЧА ЗНАЧЕНИЙ ПАРАМЕТРОВ

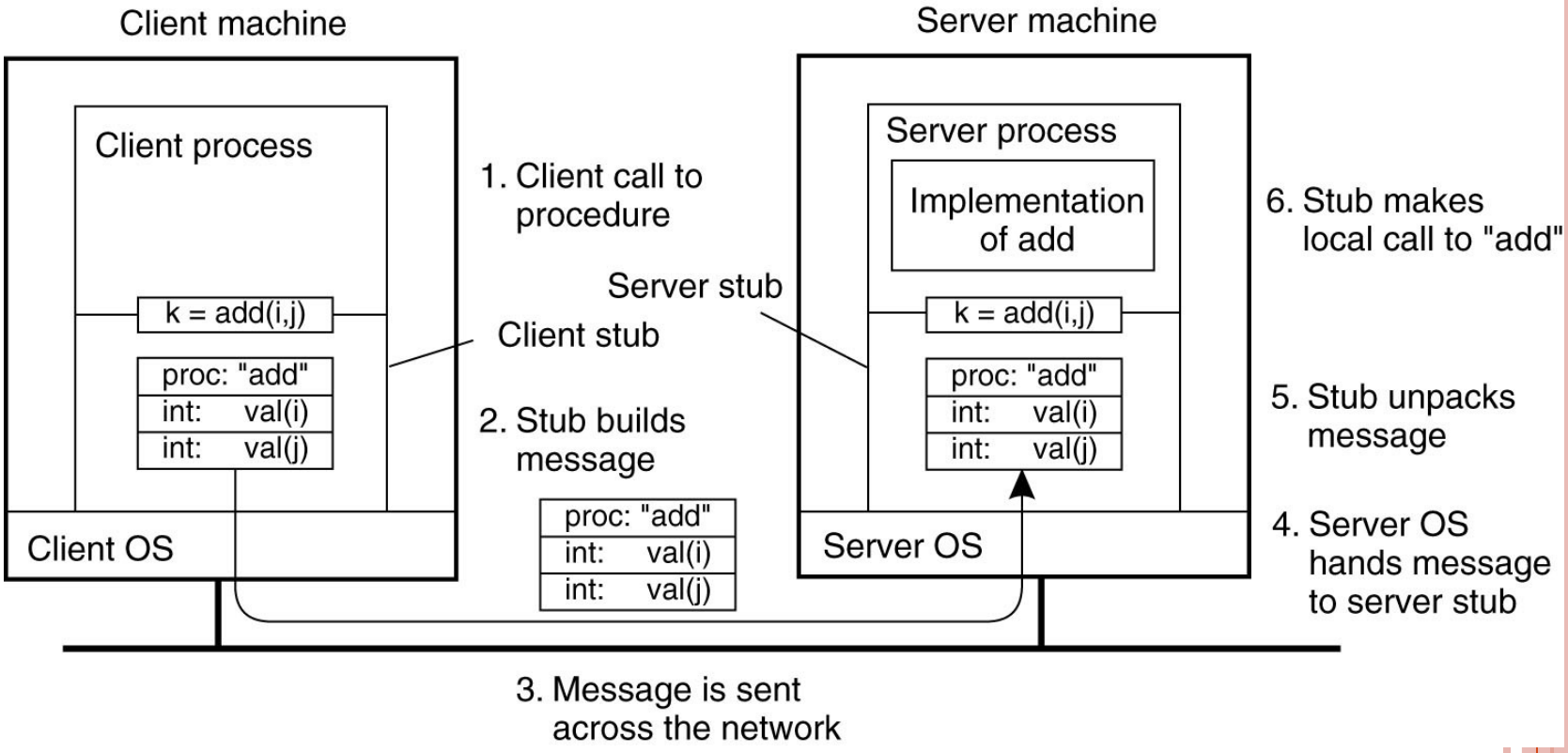
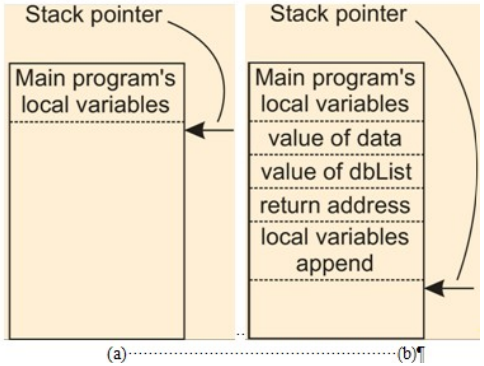


Figure 4-7. The steps involved in a doing a remote computation through RPC.

ПРОБЛЕМЫ ОРГАНИЗАЦИИ ВЗАИМОДЕЙСТВИЯ RPC (1). ПЕРЕДАЧА ПАРАМЕТРОВ

1. Основная функция клиентской заглушки является прием параметров и их передача серверной стороне. Следовательно необходимо согласовать один из возможных способов передачи – приема параметров. Например возможны варианты передачи параметров:
   - по значению **(copy-by-value)**,
   - по ссылке **(copy-by-reference)**,
   - путем копирования/восстановления **(call-by-copy/restore)**.
2. Согласование форматов передаваемых сообщений и видов представления простых типов данных;



(a) Передача параметра при вызове локальной процедуры: стек перед вызовом для добавления.
(b) Стек, пока вызываемая процедура активна

ПРОБЛЕМЫ ПЕРЕДАЧИ ПАРАМЕТРОВ

- Выбор способа передачи параметров - по значению (call-by-value), по ссылке (call-by-reference) или с помощью копирования/восстановления (call-by-copy/restore).
  - Передача параметров по значению (Call-by-value): выполняется в рамках того же процесса в котором выполняется вызов RPC. Значения параметров помещаются в стек процесса и действуют как локальные переменные.
  - Передача параметров по ссылке (Call-by-reference): выполняется в рамках того же процесса в котором выполняется вызов RPC. Указатель на структуру параметров помещается в стек процесса.
  - существует еще один механизм передачи параметров, хотя он не используется в большинстве языков программирования, он называется "вызов копированием/восстановлением" (call-by-copy/restore). Переменная копируется в стек вызывающей стороной, как при вызове по значению, а затем копируется обратно после вызова, перезаписывая исходное значение вызывающей стороны
- Решение о том, какой механизм передачи параметров использовать, обычно **принимается разработчиками языка** и является **фиксированным** свойством языка. Иногда это зависит от передаваемого типа данных.
- Выбор представления данных.
- Выбор сетевого протокола для сетевого взаимодействия между клиентом и сервером.

ПЕРЕДАЧА ПАРАМЕТРОВ ПО ЗНАЧЕНИЮ

- В этом случае значения параметров вызова удаленной процедуры могут быть помещены непосредственно в сообщение – запрос и доставлены напрямую вызываемой процедуре, если …
  - В обеих машинах используются одни и теже способы внутреннего представления данных (кодировка символов, представления чисел и т.п.)
  - в машинах используются одинаковые способы адресации байт в машинном слове (остроконечное - little endian или тупоконечное big endian).

ПЕРЕДАЧА ПАРАМЕТРОВ ПО ССЫЛКЕ

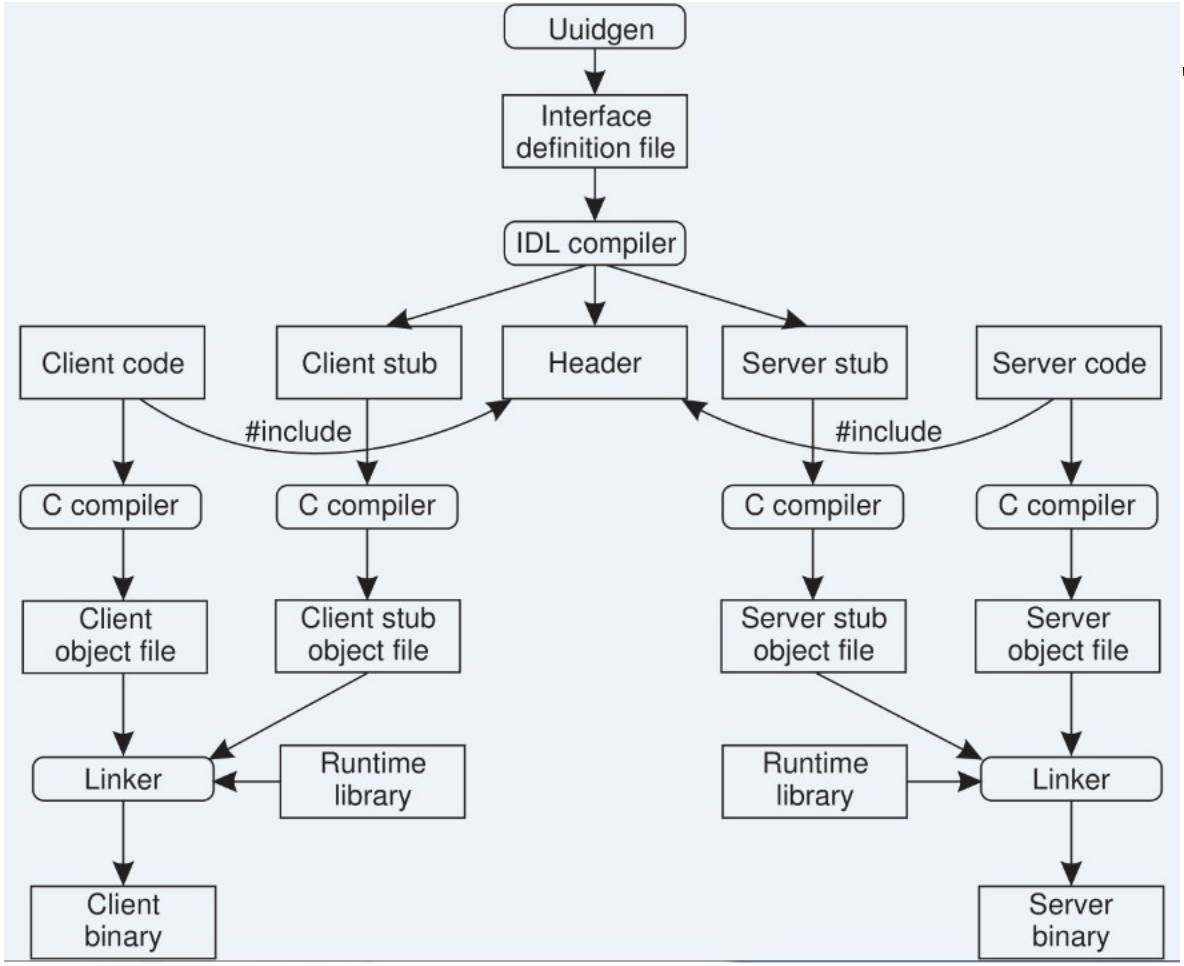
- Предполагает передачу ссылки на обыкновенный массив:
  - Передается ссылка на массив с помощью указателя.
  - Функция использует указатель для прямого обращения к массиву значений параметров, расположенному в адресном пространстве вызываемой функции.
- Указатель = машинный адрес; есть опасность, что для удаленной машины будет указывать на несоответствующий сегмент памяти.
- Решение: массив значений копируется в сообщение-запрос; затем сохраняется в адресном пространстве серверной заглушки. Серверный процесс использует указатель для доступа к сохраненному массиву параметров.

ПРОБЛЕМЫ RPC: СОГЛАСОВАНИЕ ИНТЕРФЕЙСОВ

4. Согласование интерфейсов обращения к заглушкам клиента и сервера;

Для упрощения работы интерфейсы часто описываются с использованием *языка определения интерфейсов (Interface Definition Language, IDL).*

ПРОБЛЕМЫ RPC. ГЕНЕРАЦИЯ КОДОВ ЗАГЛУШЕК



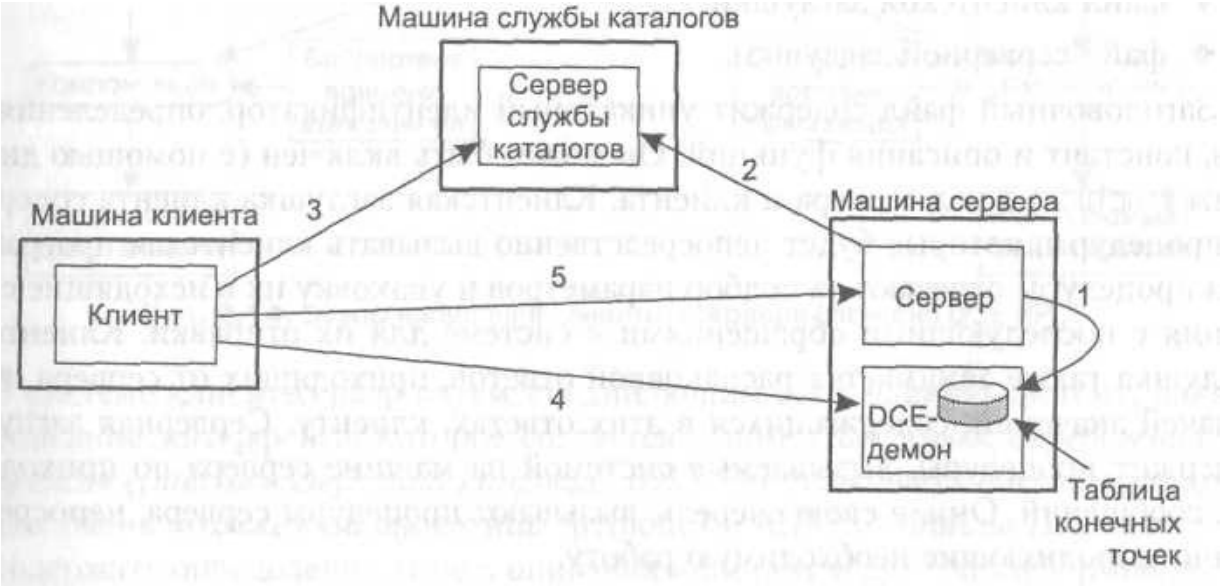
5. Обеспечение генерации кодов заглушек клиента и сервера;

Интерфейс, определенный на чем-то вроде IDL, компилируется затем в заглушки клиента и сервера, а также в соответствующие интерфейсы времени компиляции и времени выполнения.

ПРОБЛЕМЫ RPC

Решение проблемы сетевой идентификации серверов и механизмы их поиска клиентами;

Связывание (привязка клиента к серверу)



Как показано на рисунке, привязка выполняется в несколько этапов.

1.Регистрация конечной точки.

2.Регистрация службы.

3.Поиск сервера службы каталогов.

4.Опрос конечной точки.

5.Выполнение вызова RPC.

ДРУГИЕ ТРУДНОСТИ

- Клиент и сервер должны согласовать:
  - Формат сообщений, которыми они будут обмениваться (Message format).
  - Формат сложных структур данных (Format of complex data structures).
  - Транспортный протокол (TCP или UDP).

ПРОБЛЕМЫ RPC. СОГЛАСОВАНИЕ СЕТЕВЫХ ПРОТОКОЛОВ

Например, стороны могут решить использовать транспортный протокол с соединениями, такой как TCP/IP.

Альтернативой ему будет ненадежная служба дейтаграмм, в этом случае клиент и сервер должны включить реализацию схемы контроля ошибок в RPC.

На практике возможны различные варианты.

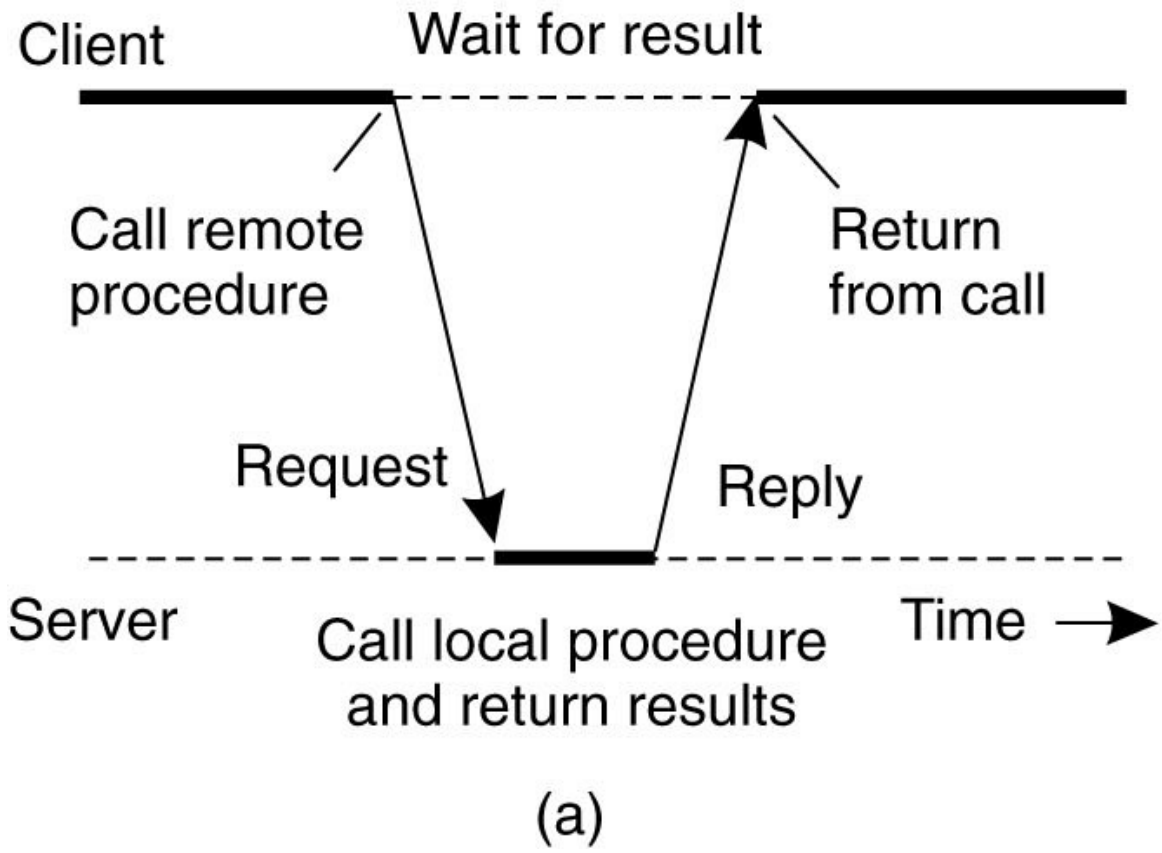
НАДЕЖНЫЙ ИЛИ НЕ НАДЕЖНЫЙ RPC

- Если RPC построен на использовании надежного транспортного протокола (например TCP), то будет обеспечиваться более надежная работа приложений.
- С другой стороны, при использовании более быстрого протокола, будет обеспечиваться более быстрая работа клиент-серверных систем.

АСИНХРОННЫЙ RPC

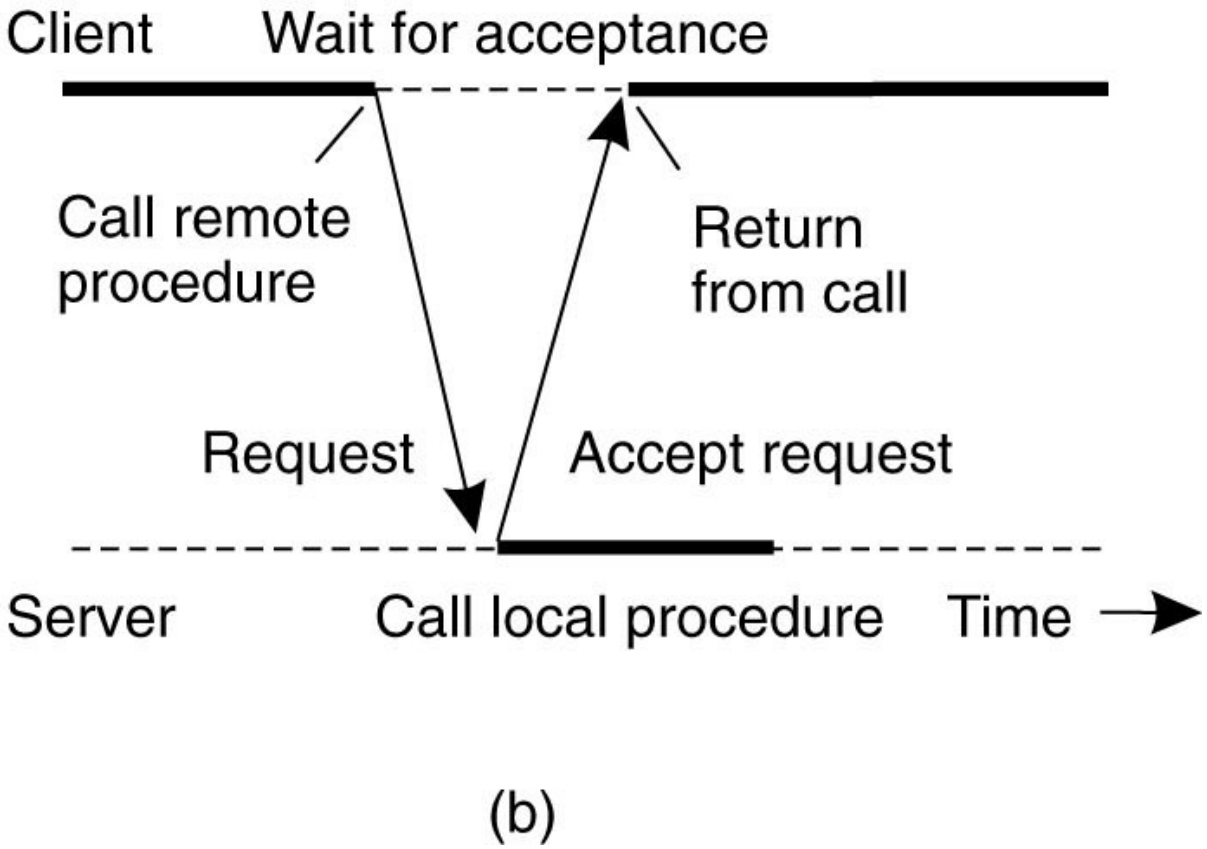
- Позволяет клиенту продолжать исполнение после выполнения вызова сервера, до завершения обработки запроса сервером и получения результата.
  - Может использоваться для выполнения действий не требующих ответа, например, обращения к принтерам, удаление файлов и т.п.
  - Также может использоваться в случаях, когда клиент желает выполнять некоторые операции не связанные с ответом на RPC. В этом случае увеличивается суммарная производительность машины клиента.
  - При этом остается проблема ненадежного асинхронного RPC.

SYNCHRONOUS RPC



- Взаимодействие между клиентом и сервером при традиционном (синхронном) RPC.

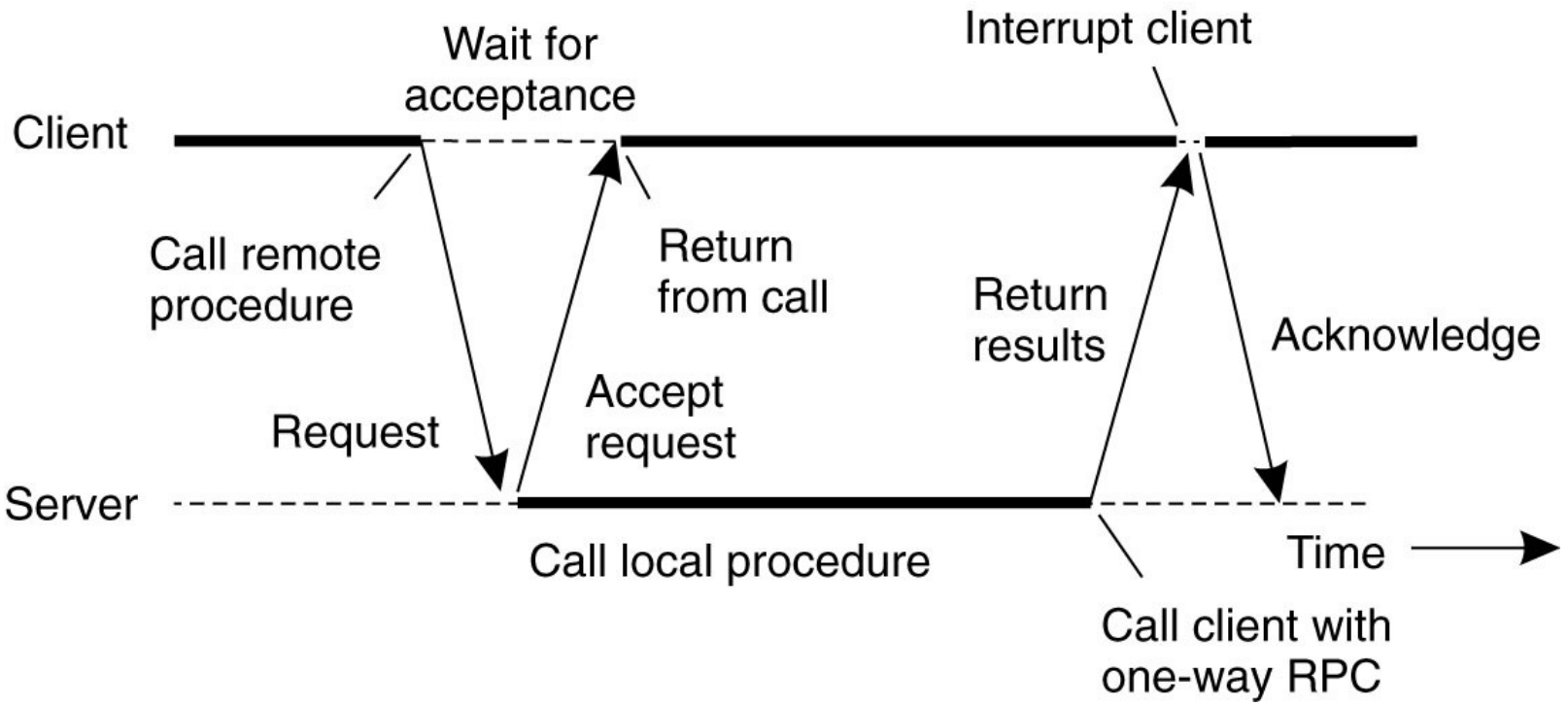
Asynchronous RPC



(b)

- Взаимодействие между клиентом и сервером при использовании асинхронного RPC.

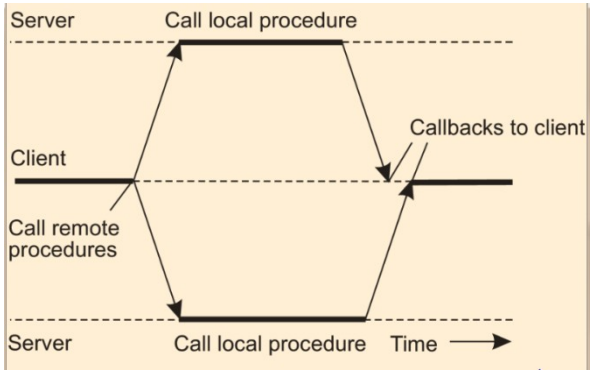
ASYNCHRONOUS RPC



- Случай когда клиент и сервер взоимодействуют друг с другом через асинхронные RPC.

МНОГОАДРЕСНЫЙ RPC (MULTICAST RPC)

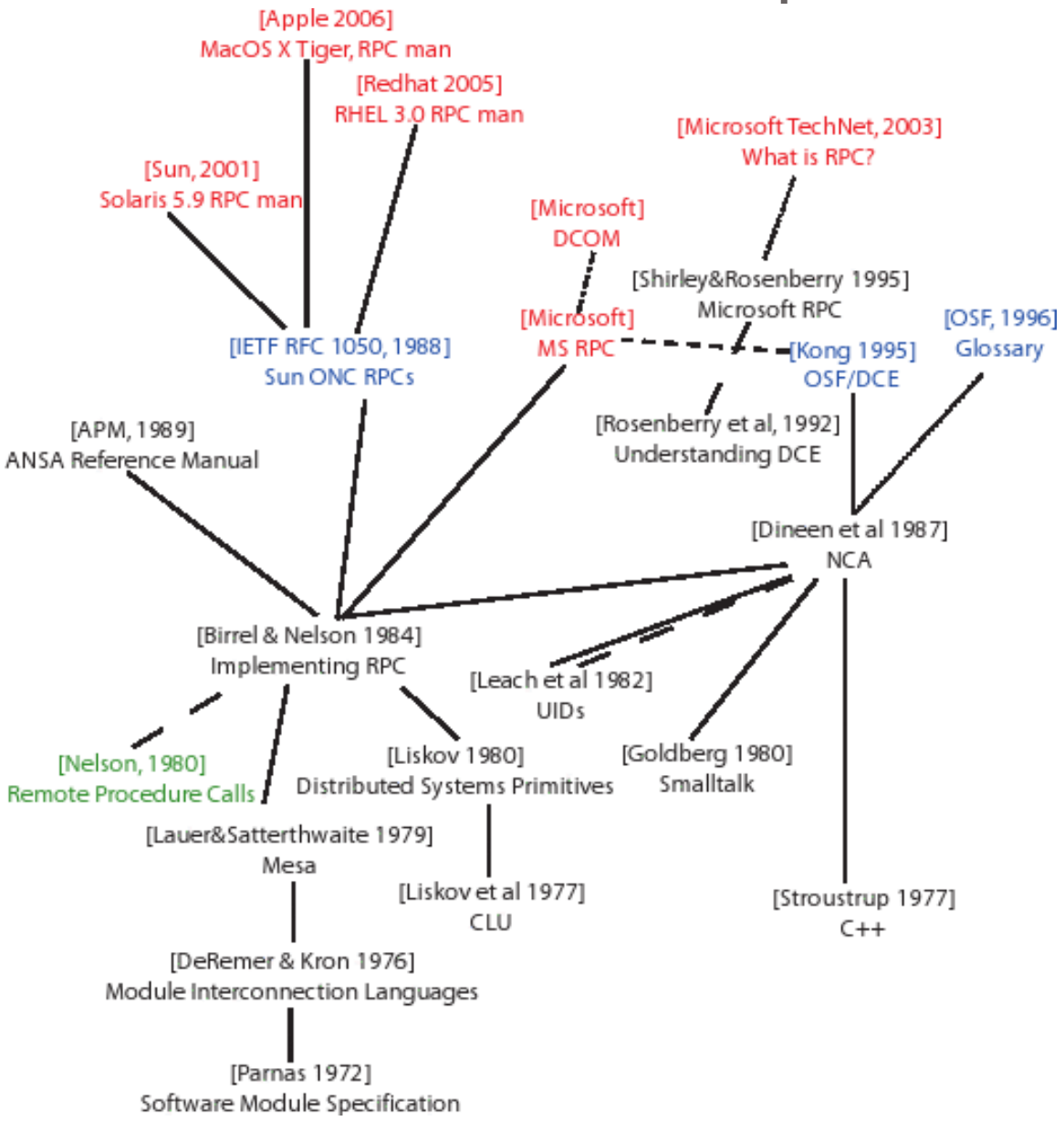
- **Многоадресный RPC** – это одновременная отправка запроса RPC группе серверов (т.е. по многим адресам). Используется в следующих случаях:
- Например, для повышения отказоустойчивости мы можем решить, чтобы все операции выполнялись также и резервным сервером, который может взять на себя управление при выходе из строя основного сервера. Клиенту достаточно получить только один ответ.
- Сбор результатов с нескольких серверов. В этом случае клиенту необходимо дождаться получения ответов от всех серверов.
- Детали многоадресного RPC скрывает заглушка на строне клиента. При этом клиентское приложение может не знать о том, что RPC фактически пересылается более чем на один сервер.



ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ RPC

- **Распределенные файловые системы**: обеспечивают доступ к файлам системы, в том числе и глобальный.
- **Служба каталогов**: отслеживает и управляет системными ресурсами (машины, принтеры, серверы и т.п.).
- **Служба безопасности**: ограничение доступа к ресурсам.
- **Распределенная служба времени**: пытается синхронизировать все часы в системе.

ПРИМЕРЫ РЕАЛИЗАЦИЙ RPC



- Sun ONC RPC
- DCE RPC
- XML RPC
- JSON RPC

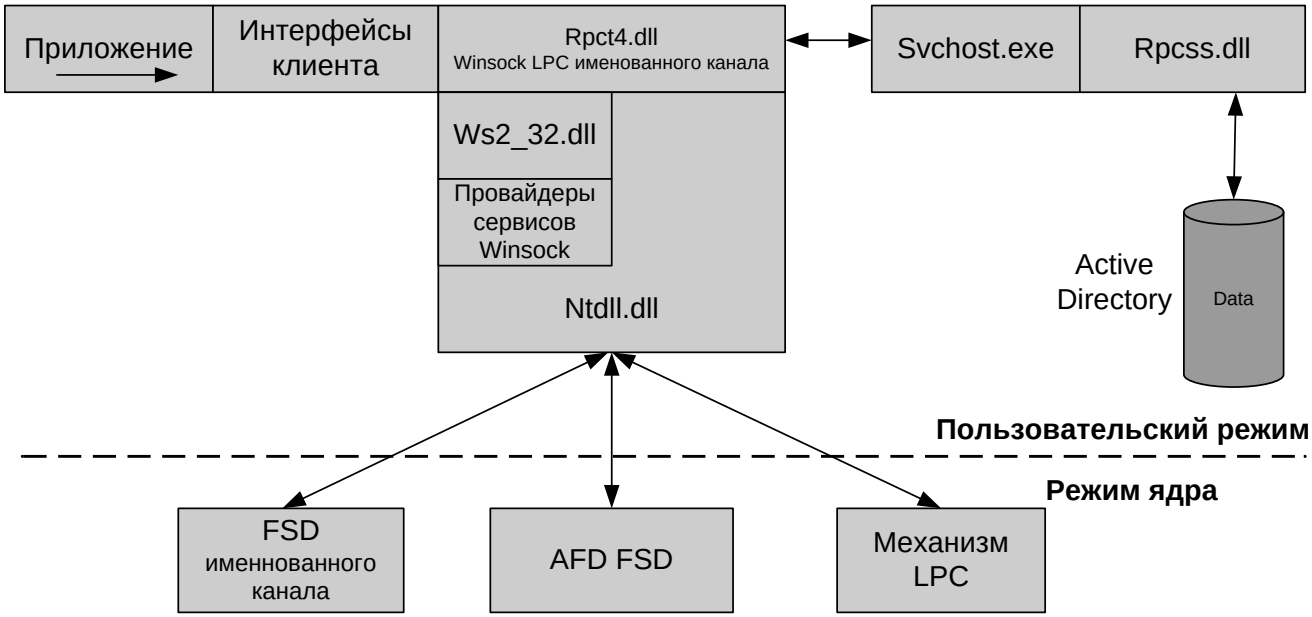
НАИБОЛЕЕ ПОПУЛЯРНЫЕ РЕАЛИЗАЦИИ RPC

- Sun RPC известный под названием Open Network Computing (ONC) RPC – широко применяемый стандарт RPC , особенно в ОС UNIX, Linux и других Unix-подобных системах (Oracle Solaris, FreeBSD, OpenBSD, MacOS X и др.).
- DCE RPC: Distributed Computing Environment
  - Разработана фондом открытого ПО Open Software Foundation (OSF) (не следует путать с Open Source ПО).
  - Адаптирован как стандарт фирмой Microsoft
  - Реализован в качестве надежного ПО промежуточного слоя во многих современных ОС.
    - Обеспечивает работу между операционными системами и пользовательскими приложениями.

REMOTE PROCEDURE CALL (RPC) КАК СТАНДАРТ СЕТЕВОГО ПРОГРАММИРОВАНИЯ

- RPC — стандарт сетевого программирования, разработанный в начале 80-х.
- Организация Open Software Foundation (теперь — The Open Group) сделала RPC частью стандарта OSF DCE (Distributed Computing Environment).
- Несмотря на наличие второго стандарта RPC, SunRPC, реализация RPC от Microsoft совместима со стандартом OSF DCE.
- RPC, опираясь на другие сетевые API (именованные каналы или Winsock), предоставляет альтернативную модель программирования, в какой-то мере скрывающую детали сетевого программирования от разработчика приложений.

РЕАЛИЗАЦИЯ RPC В ОС WINDOWS 2000/XP/2003



- Приложение на основе RPC связано с библиотекой RPC периода выполнения (\Windows\System32**Rpcrt4.dll**). Последняя предоставляет для интерфейсных RPC-функций приложений функции маршалинга, а также функции для приема и передачи упакованных данных. Библиотека RPC периода выполнения включает процедуры поддержки RPC-взаимодействия через сеть и разновидность RPC под названием *локальный RPC.*
- **Локальный RPC** позволяет двум процессам взаимодействовать в одной системе, при этом библиотека RPC в качестве сетевого API использует LPC в режиме ядра.
- Когда RPC-взаимодействие осуществляется между **удаленными системами**, библиотека RPC использует **API-функции Winsock**, именованного канала или Message Queuing.

ПРОБЛЕМЫ RPC: СВЯЗЫВАНИЕ

- **Связывание:** присвоение значений некоторым атрибутам службы RPC (например, назначение адресов идентификаторам). В разных стандартах используются разные схемы назначения атрибутов.
- **Sun RPC (ONC)** выполняет имеет исполняемую службу связывания, работающую на определенном порту на каждом компьютере (*the port mapper*).
- Клиенты находят нужный им сервис с помощью этой службы порт маппера.
- В DCE на машинах серверах работает **демон** хранящий таблицу пар <server, port #>. Также сервер должен зарегистрировать свой сетевой адрес в службе каталогов, работающем в сети.

RPC ВЫВОДЫ

- Основан на использовании простой модели вызова функций.
- Существующие реализации могут быть легко адаптированы для работы в рамках распределенных систем.
- Обеспечивает высокую степень прозрачности доступа к службам PC.

REMOTE METHOD INVOCATION (RMI)

- Основан на идеях используемых в RPC. Обеспечивает Java процессу, исполняемому на одной виртуальной Java машине вызывать процесс, работающий на другой виртуальной Java машине.
- Поддерживает создание распределенных Java систем.

СПАСИБО ЗА ВНИМАНИЕ !