Sistemas Distribuídos

RPC x RMI

Edeyson Andrade Gomes

www.edeyson.com.br

Roteiro da Aula

- Chamada Remota a Procedimento
  - Definição
  - Passagem de Parâmetros
  - STUBS
  - Semântica de Falhas

RPC

Chamada Remota a Procedimento

RPC

- Encapsula a troca de mensagens em um conceito usual da programação centralizada

- Conceito simples

- Problemas:
  - Passagem de parâmetros e resultados entre espaços de endereçamento de memória diferentes
  - Formato dos dados em diferentes plataformas
  - Falhas nas máquinas e na comunicação

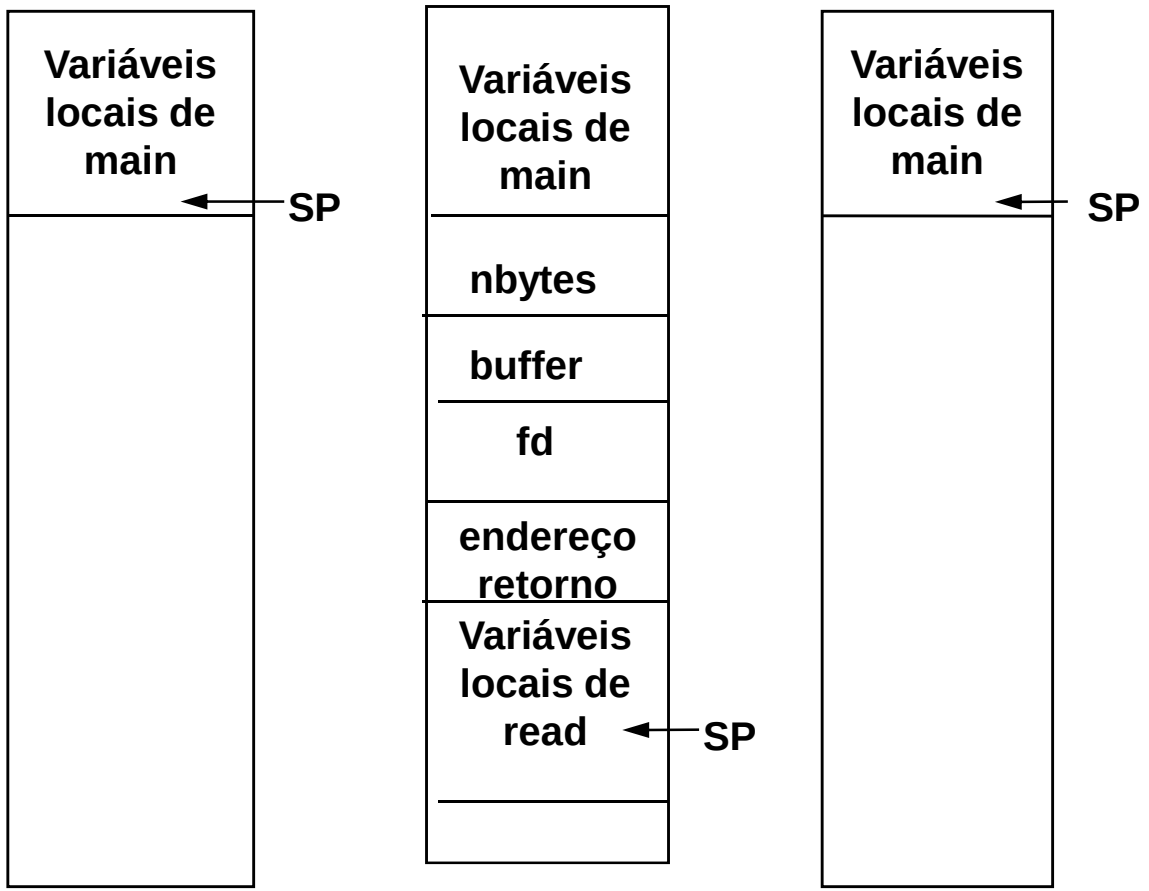
RPC

- É uma técnica muito usada na implementação de sistemas operacionais distribuídos
- Funcionamento de uma chamada de procedimento convencional (local):

  read(fd, buffer, nbytes)

  Ordem inversa dos parâmetros na pilha.

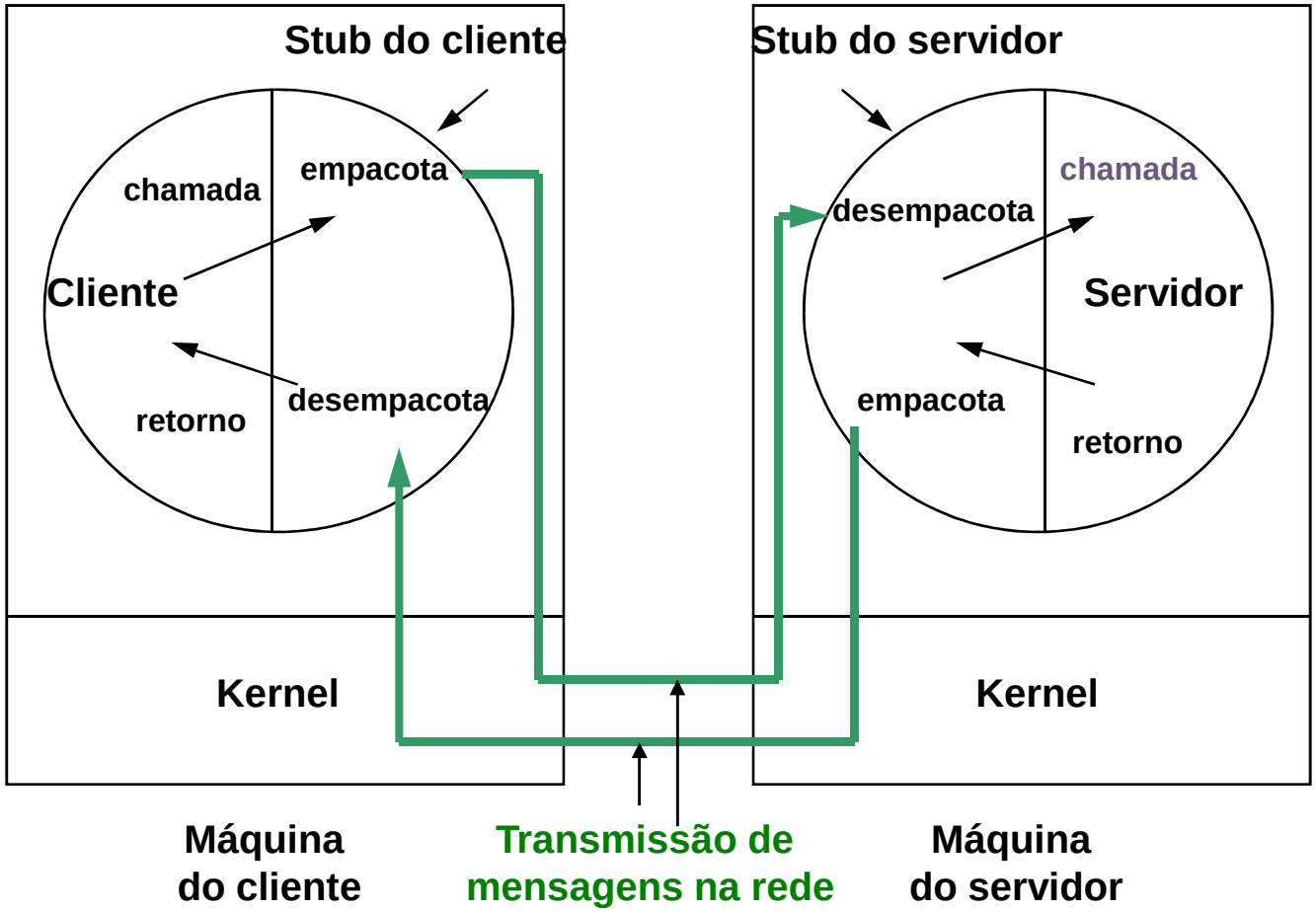
RPC


Variáveis locais de main
SP
Variáveis locais de main
nbytes
buffer
fd
endereço retorno
Variáveis locais de read
SP
Variáveis locais de main
SP

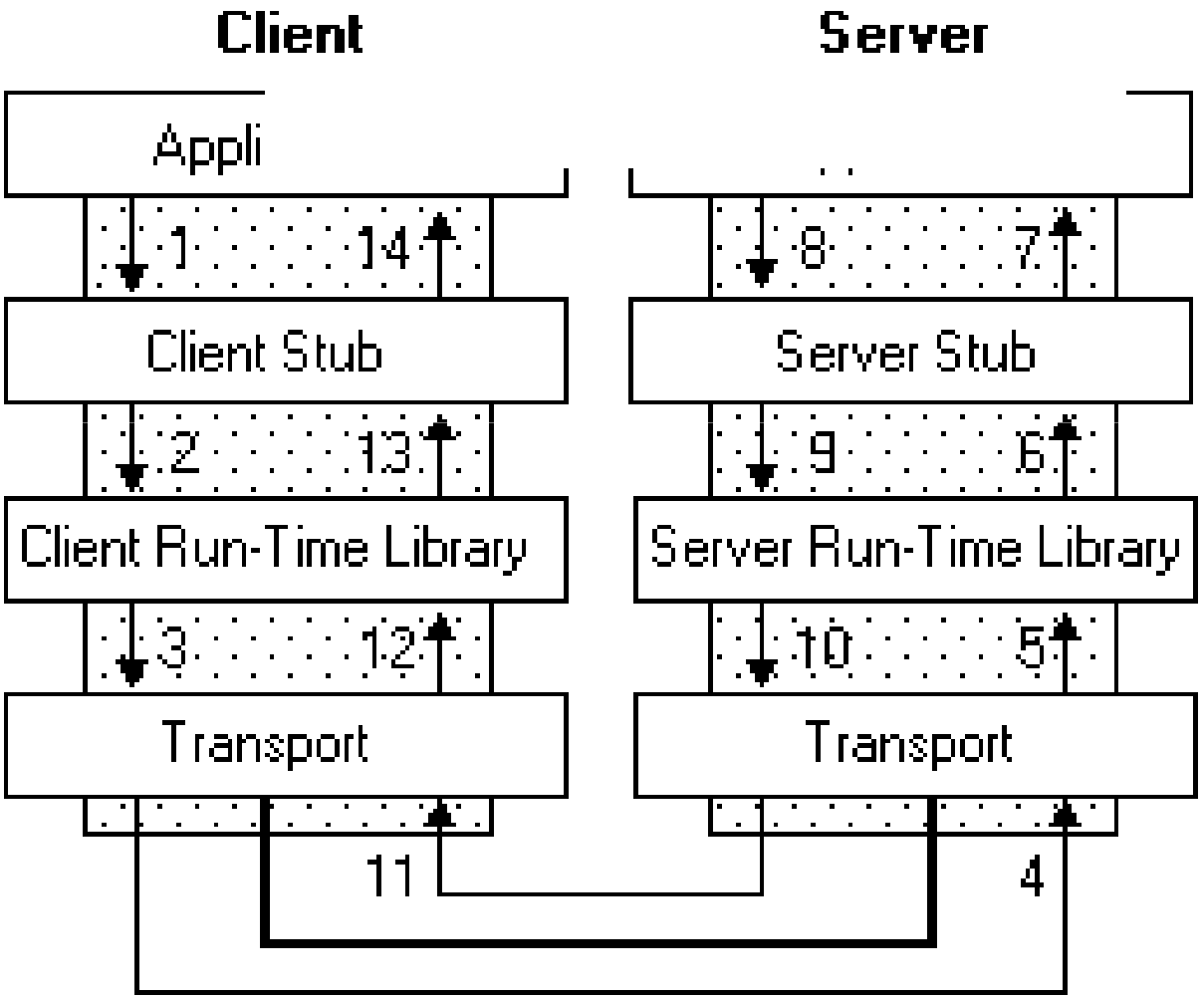
RPC

- Funcionamento de RPC
  - Como fazer uma RPC transparente?
  - Marshaling
    - Empacotamento de Parâmetros

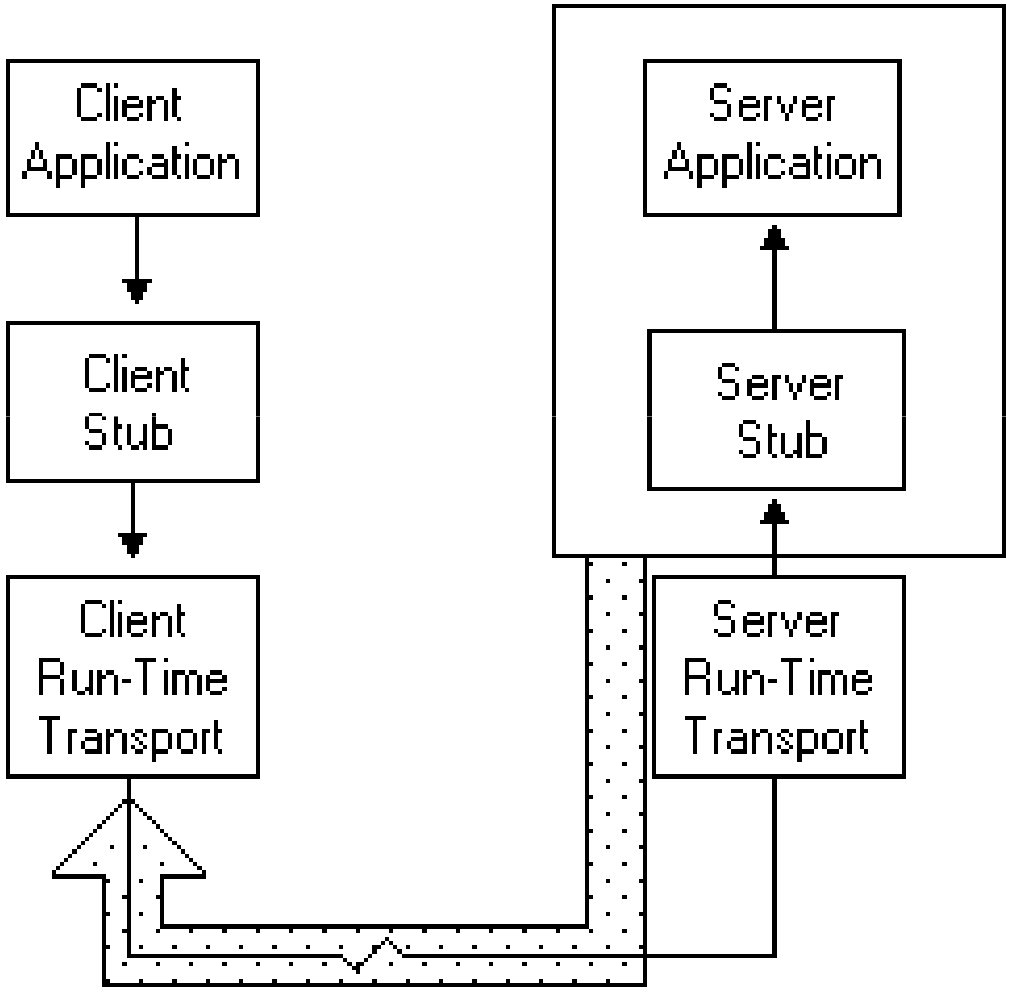
RPC


Stub do cliente
Stub do servidor
chamada
empacota
Cliente
desempacota
retorno
chamada
desempacota
Servidor
empacota
retorno
Kernel
Kernel
Máquina
do cliente
Transmissão de
mensagens na rede
Máquina
do servidor

RPC


Client
Server
Appli
1
14
8
7
Client Stub
Server Stub
2
13
9
6
Client Run-Time Library
Server Run-Time Library
3
12
10
5
Transport
Transport
11
4

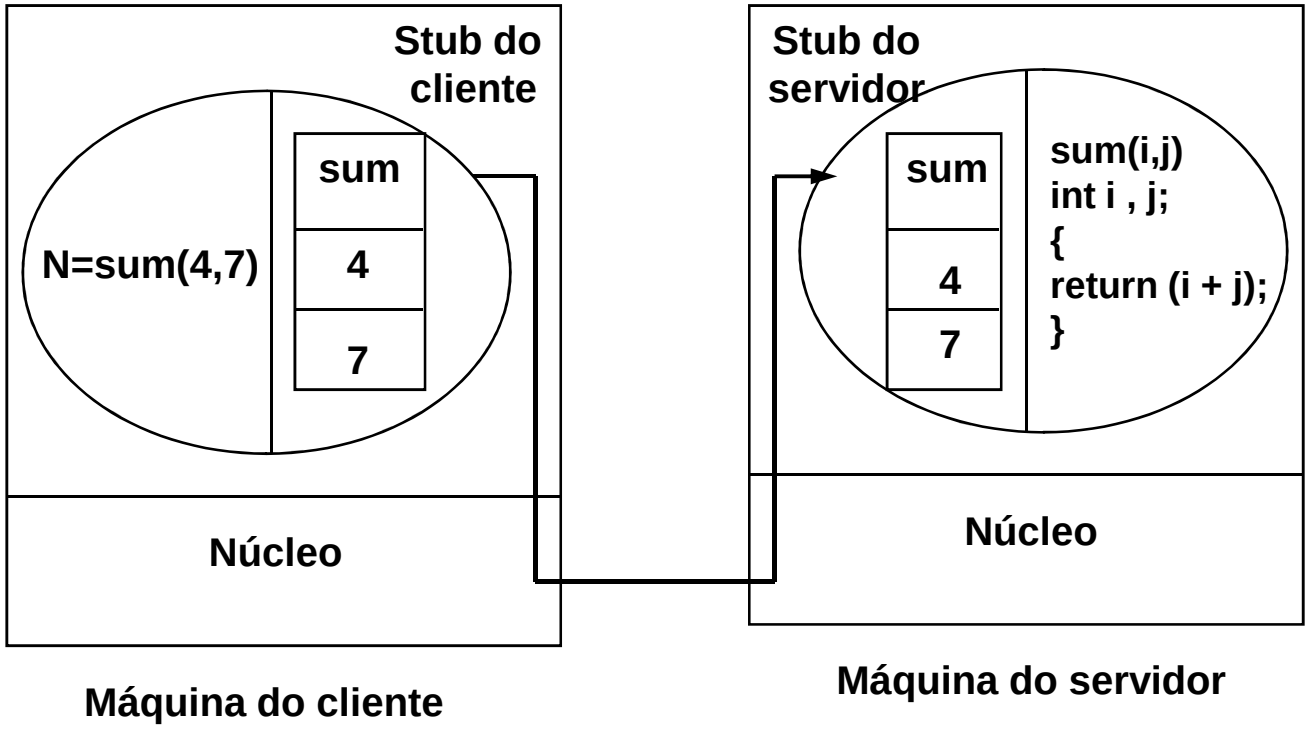
RPC


Client
Application
Client
Stub
Client
Run-Time
Transport
Server
Application
Server
Stub
Server
Run-Time
Transport

RPC

- Passagem de Parâmetros
  - Na chamada de procedimento convencional, 3 formas:
    - Por Valor
    - Referência
    - Cópia

RPC


Stub do cliente
N=sum(4,7)
sum
4
7
Núcleo
Máquina do cliente
Stub do servidor
sum
4
7
sum(i,j)
int i , j;
{
return (i + j);
}
Núcleo
Máquina do servidor

RPC

- Passagem de Parâmetros
  - Stubs fazem o marshaling de parâmetros
    - Empacotamento
    - Conversão de formatos
  - Representação de Dados
    - Arquiteturas diferentes:
      - Códigos de caracteres
        - ASCII, EBCDIC
      - Tamanho e representação dos dados
        - ordem dos bytes (Alta ou Baixa Ordem)
        - números em complemento de 1 ou 2

RPC

- Soluções:
  - Usar um formato intermediário (forma canônica)
    - Pode provocar conversões desnecessárias
    - Solução:
      - Indicar o formato usado no início da mensagem
        - Flag

RPC

- Passagem de Parâmetros por Referência (Ponteiros)
  - Uso de Cópia e Restauração
    - O Stub do cliente copia a variável apontada pelo parâmetro para a mensagem local
    - O Stub do servidor recebe e desempacota a mensagem

RPC

- Passagem de Parâmetros por Referência (Ponteiros)
  - Uso de Cópia e Restauração
    - O Stub do servidor recebe e desempacota a mensagem
      - Cria uma réplica da variável em seu espaço de endereçamento
      - Chama o servidor, passando como parâmetro um ponteiro para sua variável
      - Empacota o valor de sua variável e envia ao Stub do cliente
      - Ao receber a mensagem de resposta com a variável modificada, o Stub do cliente a restitui ao cliente

RPC

- Passagem de Parâmetros por Referência (Ponteiros)
  - Uso de Cópia e Restauração
    - Uma das cópias pode ser economizada quando o parâmetro é apenas uma entrada, ou apenas uma saída para o servidor
  - Resolução remota de ponteiros

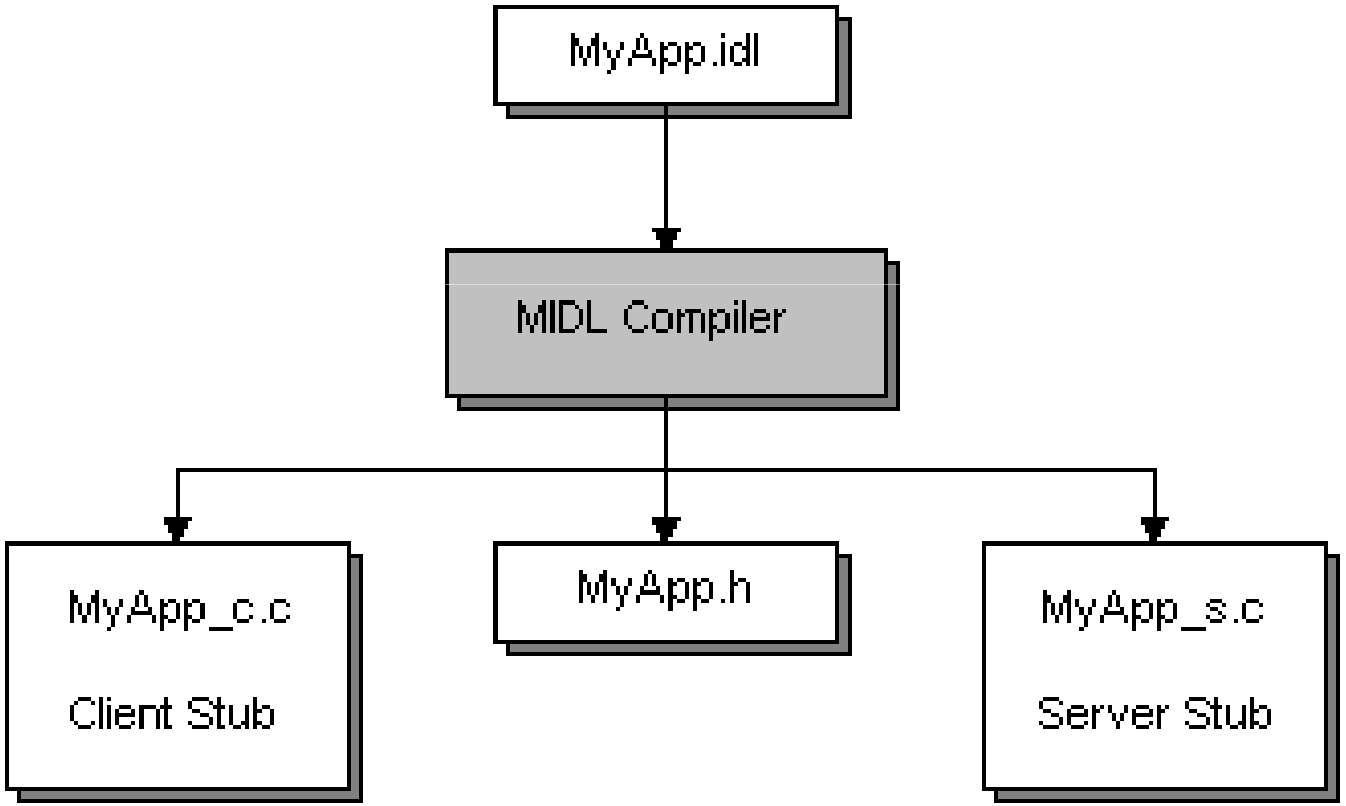
STUBs

- Os Stubs (do cliente e do servidor) são códigos que encapsulam a comunicação entre processos e são gerados automaticamente a partir de uma especificação
  - IDL
    - Linguagem de Definição de Interface
    - Independe de plataforma ou linguagem

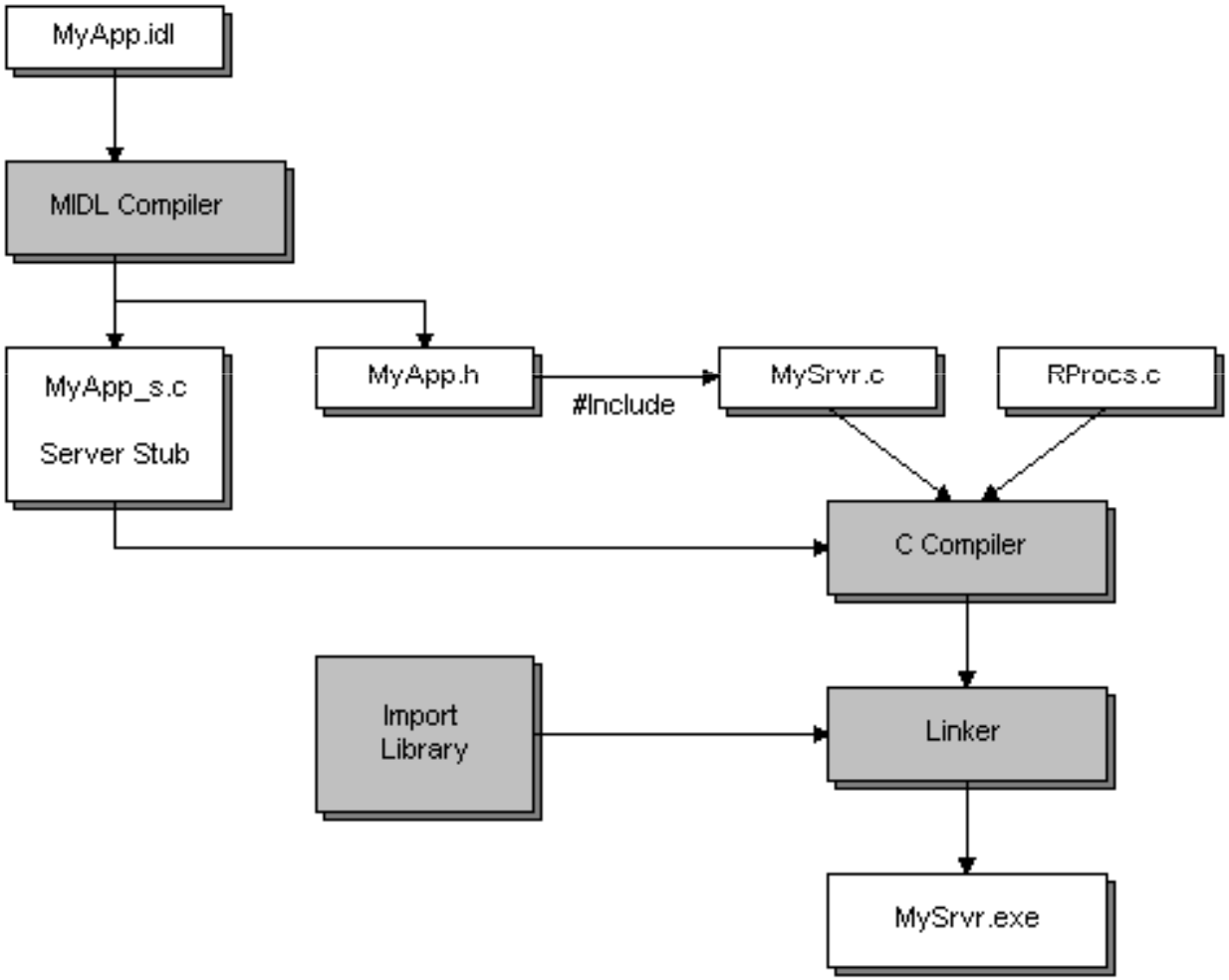
STUBs

```
[ uuid (12345678-1234-1234-1234-123456789ABC),
  version(1.0)
]
interface OperacoesH  {
    long soma([in] int a, [in] int b);
    long mult([in] int a, [in] int b);
    void maiuscula([in, out, string] char * s);
}
```

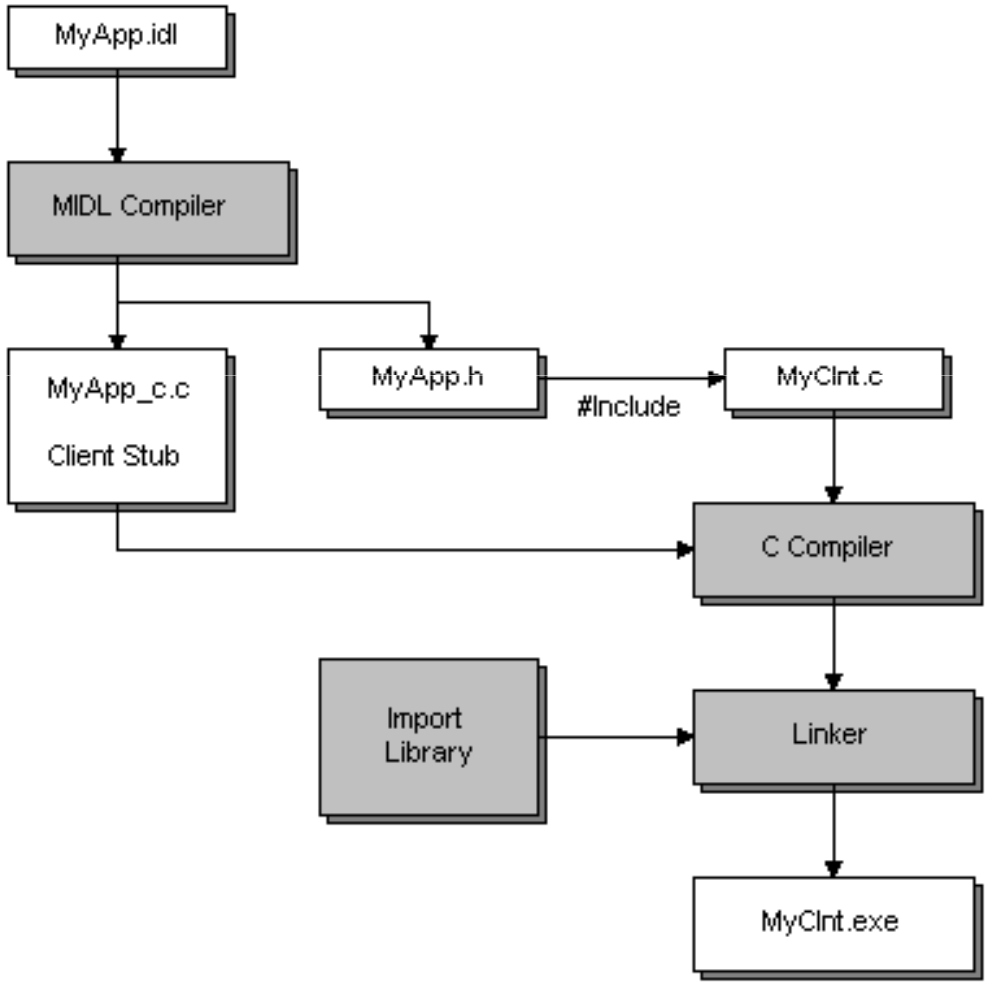
MIDL


MyApp.idl
MIDL Compiler
MyApp_c.c
Client Stub
MyApp.h
MyApp_s.c
Server Stub

MIDL


MyApp.idl
MIDL Compiler
MyApp_s.c
Server Stub
MyApp.h
#Include
MySrvr.c
RProcs.c
C Compiler
Import
Library
Linker
MySrvr.exe

MIDL


MyApp.idl
MIDL Compiler
MyApp_c.c
Client Stub
MyApp.h
#Include
MyClnt.c
C Compiler
Import
Library
Linker
MyClnt.exe

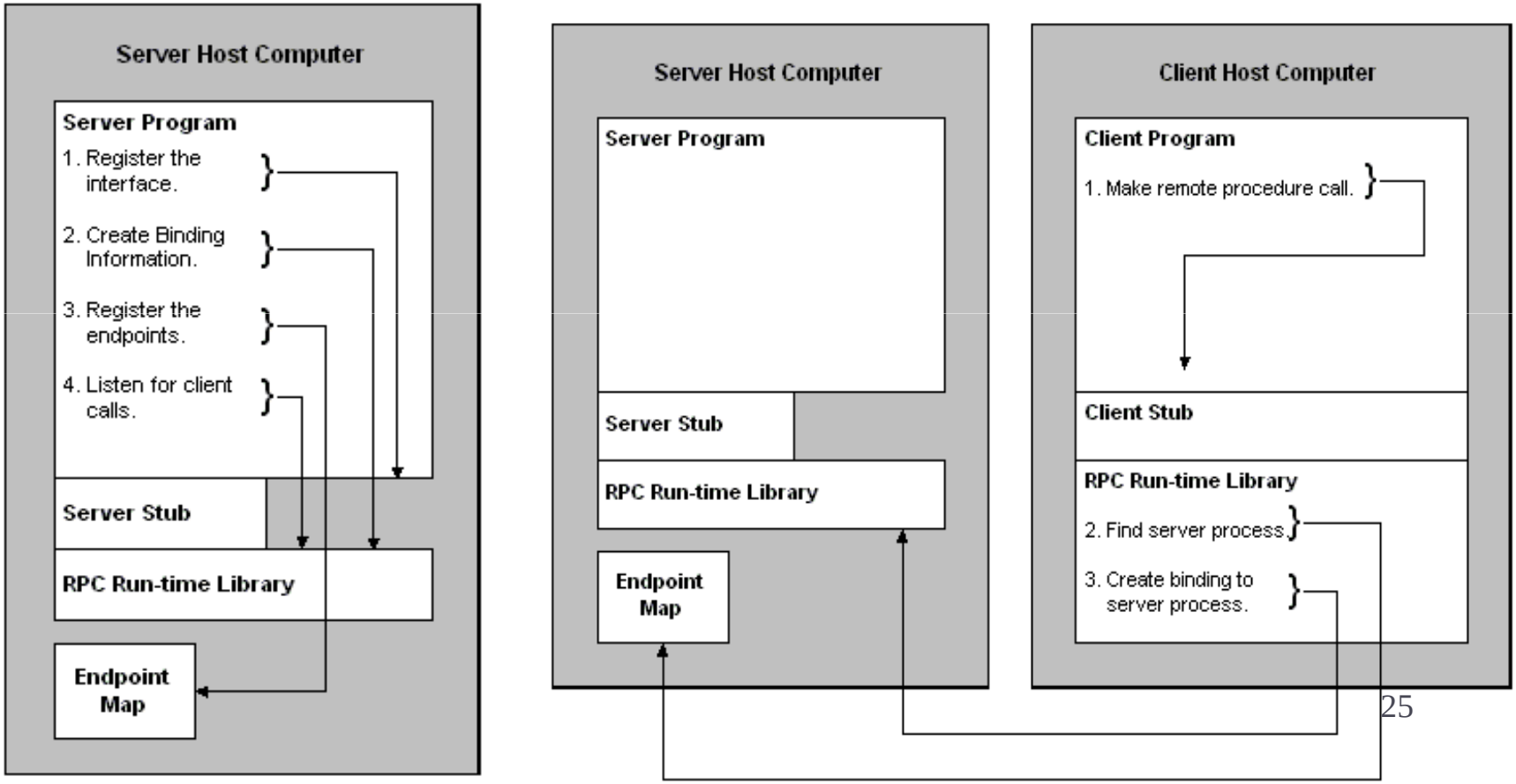
Localização do Servidor RPC

- Endereço explícito no código
  - Unicast
- Ligação Dinâmica
  - Um processo BINDER é responsável pela ligação
  - O servidor RPC exporta a sua interface para o BINDER
  - O cliente importa a versão do servidor através do BINDER
  - Bastante flexível

Localização do Servidor RPC

- Desvantagens da Ligação Dinâmica
  - Overhead extra para importar e exportar interfaces
  - BINDER pode se tornar um gargalo (componente centralizado) ou ponto de falha
  - Replicação do BINDER gera problemas de consistência

Localização do Servidor RPC



Falhas com RPC

- Semântica parecida com as LPC's quando não há falhas
  - Tipos de Falhas
    - O cliente não consegue localizar o servidor
    - A mensagem de ***request*** é perdida
    - A mensagem de ***reply*** é perdida
    - O servidor falha depois de receber um ***request***
    - O cliente falha depois de enviar um ***request***

Falhas com RPC

- Detecção de falha RPC
  - Exceção no sistema
    - Perda de transparência
      - Diferente de procedimento centralizado

Falhas com RPC

- O cliente não consegue localizar o servidor
  - O servidor pode ter falhado
    - 0x6ba
  - O cliente pode estar usando uma versão antiga da Interface, não disponível no servidor
    - UUID do cliente é diferente da UUID do servidor
    - 0x6b5
    - Erro no BINDER

Falhas com RPC

- A mensagem de REQUEST é perdida
  - *Kernel* usa *timeout* e retransmite a mensagem
  - Após um certo número de retransmissões mal sucedidas, o cliente assume que o servidor falhou, podendo tentar um outro servidor

Falhas com RPC

- A mensagem de *REPLY* é perdida
  - *Kernel* usa *timeout*
  - Quando o *timeout* expira, como o Cliente sabe o que aconteceu?
    - O REQUEST foi perdido?
    - O REPLY foi perdido?
    - O Servidor está lento?

Falhas com RPC

- A mensagem de *REPLY* é perdida
  - Request não Idempotente
    - Ex.: Transferência monetária
  - Criar Idempotência aos Requests
    - Números de seqüência na mensagem
      - Controle de retransmissões
    - Indicador de retransmissão no cabeçalho

Falhas com RPC

- O Servidor Falha
  - Como saber se o servidor falhou antes ou depois de executar um REQUEST?
    - Recebe, Executa e FALHA
    - Recebe e Falha
    - Para o cliente o problema é não receber o REPLY

Falhas com RPC

- Existem 3 abordagens que levam a 3 semânticas diferentes:
  - At Least Once
    - Tentar até receber um REPLY
  - At Most Once
    - Termina indicando falha
  - Nenhuma Garantia

  - Nenhuma delas é muito atrativa
    - Exactly Once é normalmente impossível

Falhas com RPC

- O cliente falha depois de enviar um REQUEST
  - Pode gerar computações órfãs no servidor
  - Órfãos podem trazer problemas
    - Gasto desnecessário de CPU
    - A execução do REQUEST após a recuperação do cliente pode gerar problemas de inconsistência

Falhas com RPC

- O cliente falha depois de enviar um REQUEST
  - Solução por extermínio
    - O cliente mantém um *log* de *requests* enviados
    - Ao retornar de uma falha, elimina os órfãos
    - Os órfãos podem ter gerado mais órfãos em outras máquinas
    - Os órfãos podem ter bloqueado registros

Falhas com RPC

- O cliente falha depois de enviar um REQUEST
  - Solução por reencarnação
    - O cliente faz um broadcast com seu número atual de encarnação, toda vez que reinicia após uma falha.
    - Os servidores eliminam os órfãos
    - Um particionamento na rede durante o broadcast pode gerar órfãos sobreviventes
    - O cliente descarta as respostas provenientes desses órfãos

Falhas com RPC

- O cliente falha depois de enviar um REQUEST
  - Solução por reencarnação branda
    - Ao chegar uma mensagem de nova encarnação, um servidor tenta localizar o proprietário de cada processamento remoto
    - Se não conseguir, o servidor elimina o processo em questão

Falhas com RPC

- O cliente falha depois de enviar um REQUEST
  - Solução por expiração
    - Um tempo de validade T é associado aos processamentos remotos, em cada servidor
    - Quando o tempo expira, uma nova fatia de tempo deve ser requisitado para que o processo possa continuar a executar
    - Ao reiniciar de uma falha, o cliente espera por T, antes de ficar operacional

Problemas com RPC

- Problemas com RPC's
  - Comunicação envolve apenas dois participantes (ponto-a-ponto)
  - Resolve muitos problemas, mas não todos:
    - Servidor pode estar replicado em várias máquinas
    - Muitas vezes, para tolerar falhas, é conveniente que o cliente envie a mensagem de *request* para todos os servidores
    - Necessita de múltiplos RPC's

RMI

Remote Method Invocation

RMI

- Aplicação RMI
  - Servidor
  - Cliente

- Requisitos das Aplicações:
  - *Localizar Objetos Remotos*
    - Referências
    - Registro de Objetos Remotos
      - RMI's simple naming facility – rmiregistry
      - Ferramenta de registro e resolução de nomes
      - Referências remotas via parâmetro

RMI

- Requisitos das Aplicações:
  - *Localizar Objetos Remotos*
  - *Comunicação com objetos remotos*
    - Encapsulamento de detalhes
    - Comunicação remota
      - Similar a chamada padrão do Java
  - *Carregar bytecodes de classes para objetos passados remotamente*
    - Mecanismo de carga de código e transmissão de dados

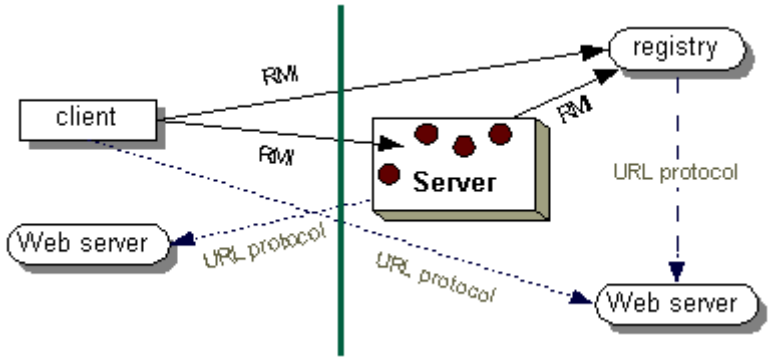
RMI

Referência remota via Registry.

Servidor chama o registry para associar (bind) um nome ao objeto remoto.

Cliente procura o objeto remoto por nome no registry do servidor e invoca um método.

Um servidor Web pode ser usado para transferência de bytecodes entre o cliente e o servidor.



RMI

- **Vantagens da Carga Dinâmica de Código**
  - Download de *bytecodes*
    - Classe não definida na VM do receptor.
    - RMI passa objetos por tipo real
      - Mantém comportamento
      - Introdução de tipos dinamicamente numa VM remota

RMI

- Interfaces, Métodos e Objetos
  - Objeto Remoto
    - Métodos invocados entre VM
    - Implementam interface ***Remote***
    - ***Remote Stub***

RMI

- ***Execução de Método Remoto***