PÓS-GRADUAÇÃO ALFA

FACULDADES ALFA ALVES ARIA

A MELHOR ESCOLA DE NEGÓCIOS DO CENTRO-OESTE

Diego Américo Guedes

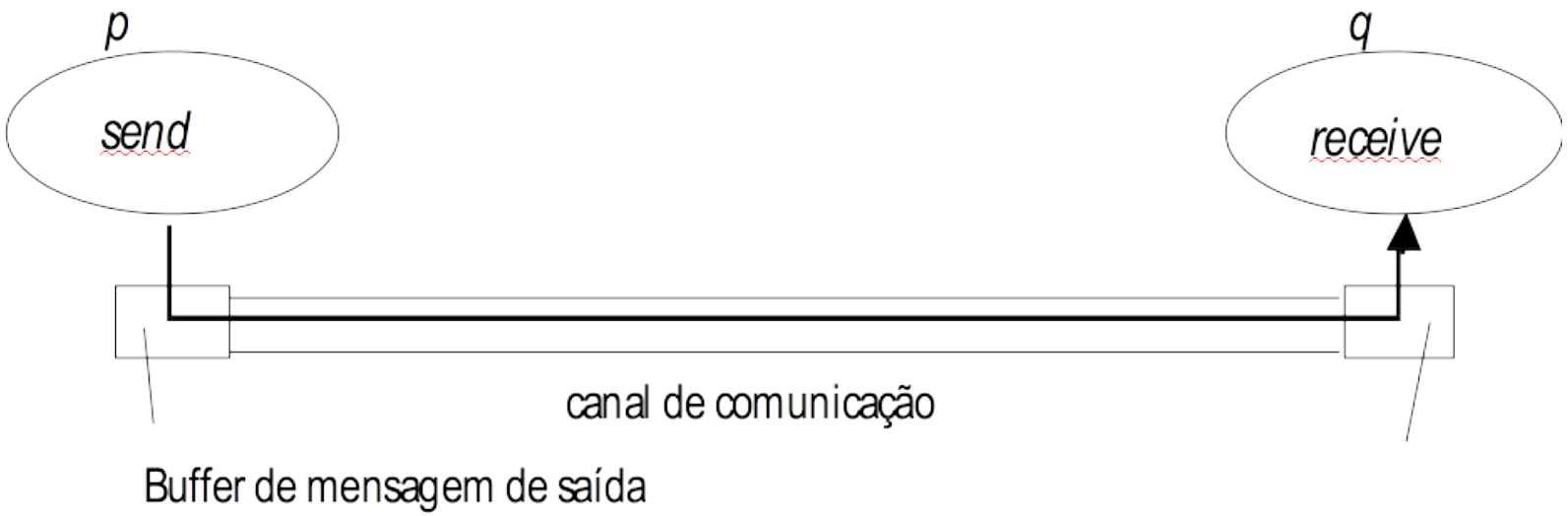
PROGRAMAÇÃO COM FRAMEWORKS E COMPONENTES

GRUPO JOSÉ ALVES

Apresentação

- Bacharel em Ciências da Computação, UFG (2010)
- Mestrado em Ciências da Computação, UFG (2013)
- Campeão das regionais da Maratona de Programação em 2008 e 2009
- Trabalhei em projetos regionais, nacionais e mundiais de pesquisa na área de Ciências da Computação
- Fui professor substituto na UFG por 2 anos (2013-2014)
- Fui professor Adjunto na Faculdade Senac por 2 anos (2015-2016)
- P&D goGeo (2014)
- Analista de Sistemas na Saneago (Desde 2014)

Primitivas de Comunicação



Primitivas de Comunicação

- Primitivas
  - send: usada para enviar mensagem
  - receive: usada para receber uma mensagem
- Semântica
  - Comunicação síncrona
    - Os processos remetente e destino são sincronizados a cada mensagem.
    - send e receive são operações bloqueantes
  - Comunicação assíncrona
    - send não bloqueante
    - receive bloqueante/não bloqueante

- Camada de transporte
  - fornece uma comunicação fim-a-fim
  - Os processos comunicantes são identificados por um par <IP, porta>
  - Uma porta
    - Especifica um valor inteiro
    - Tem exatamente um destino
    - Pode ter vários remetentes

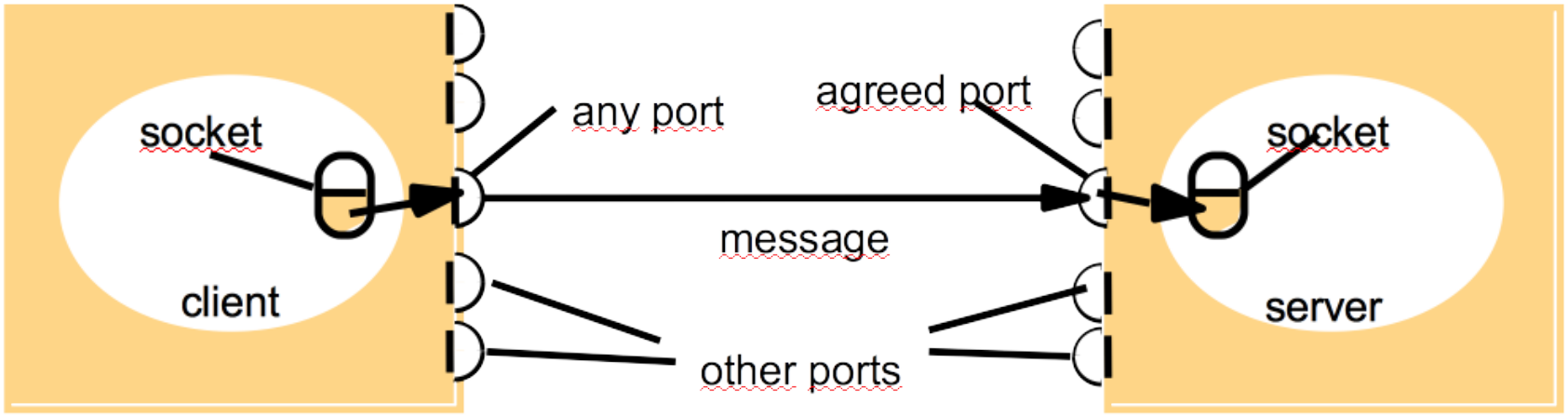
  Um processo pode usar várias portas

Sockets

Socket

- A forma mais elementar para implementar uma comunicação entre processos é através de sockets (soquetes).
- A comunicação entre processos consiste em transmitir uma mensagem entre um socket de um processo e um socket do outro processo.
- A API de socket é provida pelo sistema operacional (Free BSD, Linux, Windows, Mac OS)
- A API de socket provê acesso aos serviços de transporte (TCP e UDP)

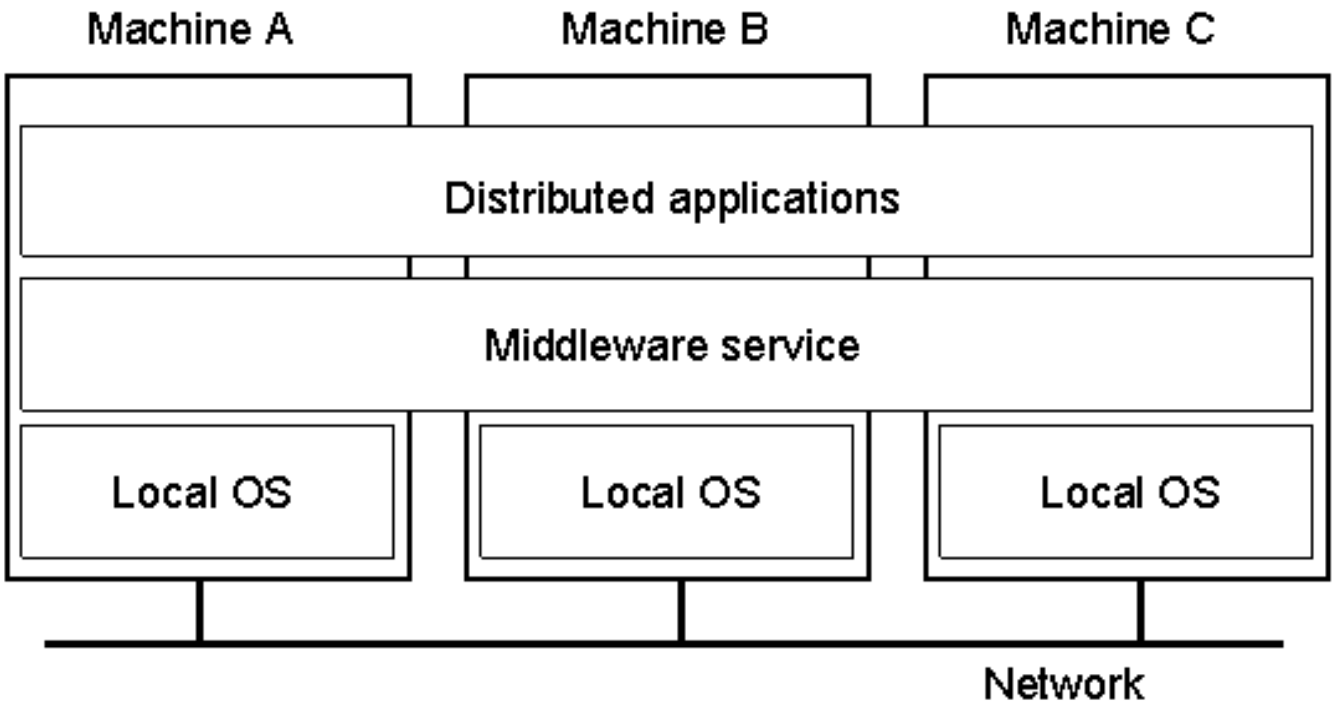
Socket



- - Todo socket está vinculado a uma porta e a um endereço IP
- - Há 2^{16} números de portas disponíveis
- - Os processos podem usar o mesmo socket para ler e enviar mensagens
- - Cada socket é associado a um protocolo de transporte (UDP ou TCP)

Tecnologias para middleware: RPC

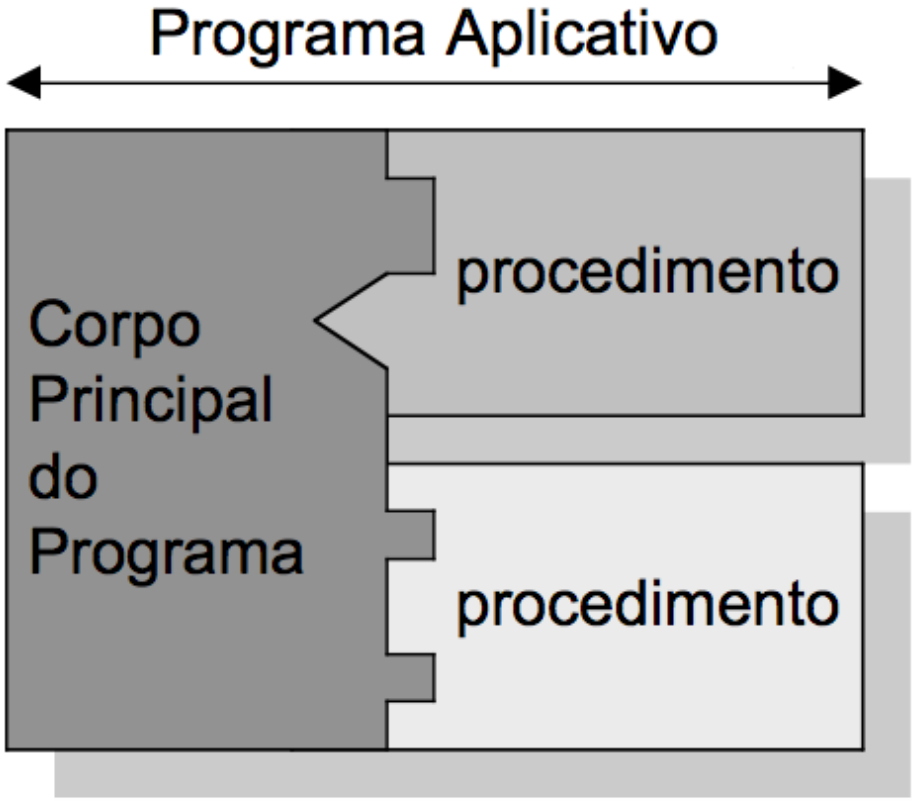
Middleware


Machine A
Machine B
Machine C
Distributed applications
Middleware service
Local OS
Local OS
Local OS
Network

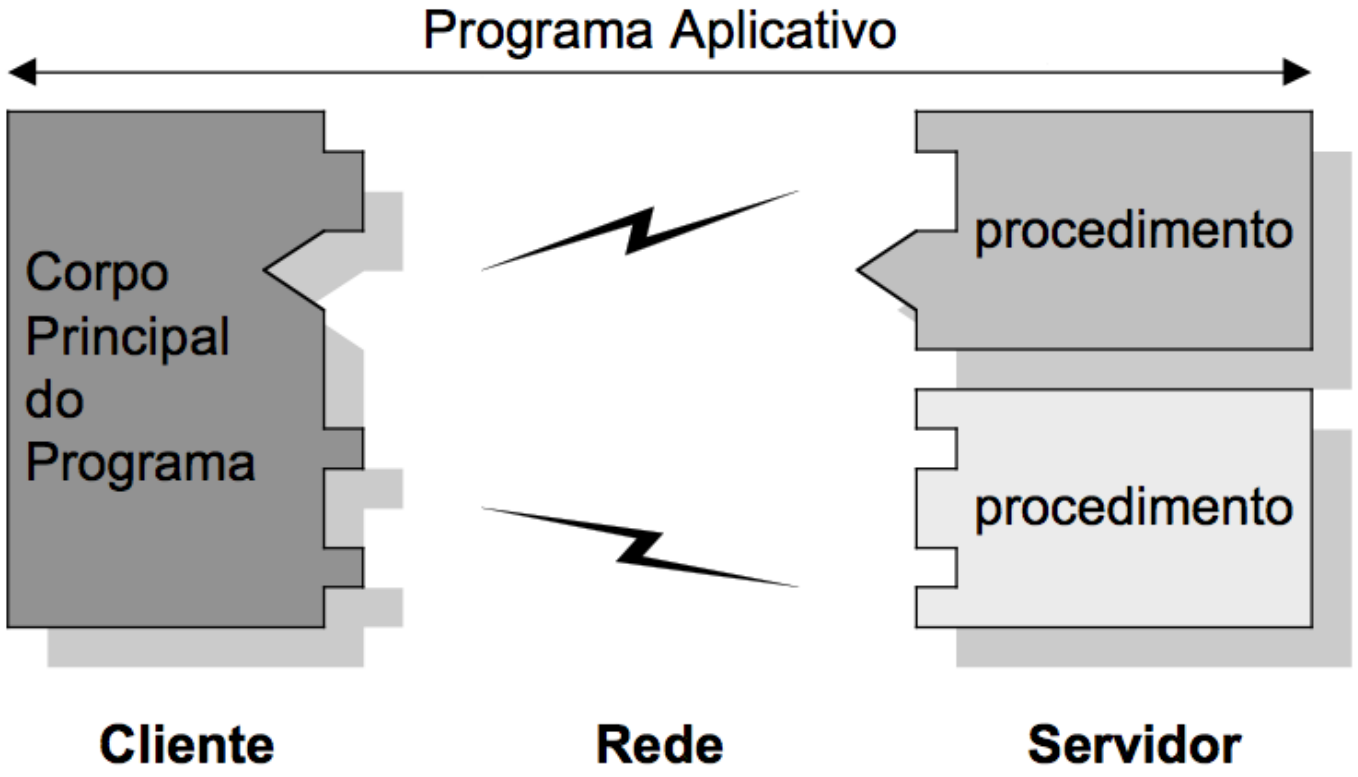
O que há de errado com Socket?

- A troca de mensagem entre os componentes do sistema é feita através de uma interface de entrada e saída
- Essa, no entanto, não é a forma como os componentes de programas não distribuídos interagem.
- A **chamada de procedimento** é o modelo de interface padrão de sistemas não distribuídos

Chamada Local



Chamada Remota



Chamada Local

- Como é implementada?
  - Empilha os parâmetros da chamada
  - Empilha o endereço do procedimento chamado
  - Procedimento desempilha os parâmetros
  - Executa seu trabalho e coloca o valor de retorno em um registrador
  - Retorna para o endereço no topo da pilha

Chamada Remota de Procedimento (RPC)

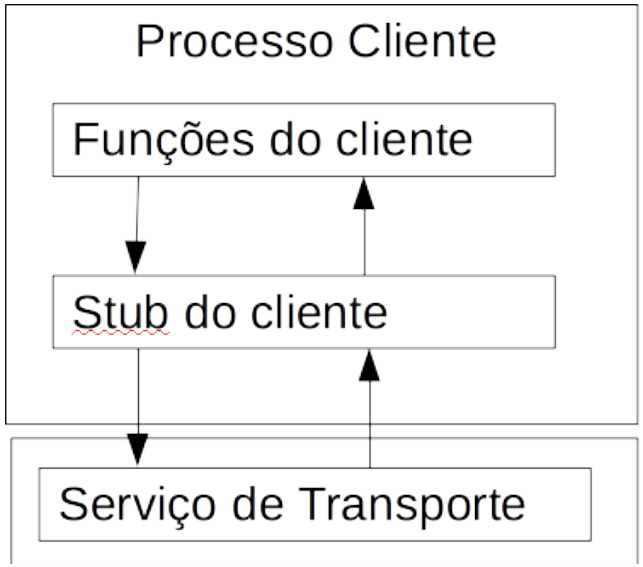
- Criada em 1984 com o objetivo de tornar a computação distribuída parecida com a computação centralizada
- Permite que processos possam invocar procedimentos/funções de outros processos (numa mesma máquina ou não)

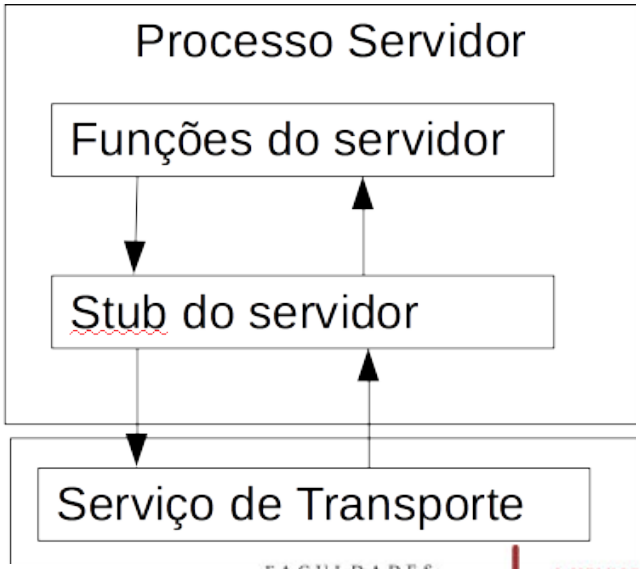
Chamada Remota de Procedimento (RPC)

- Chamada Remota
  - Mesma semântica da chamada local:
    - chamador bloqueia até que o código chamado termine
  - Simula uma chamada local com procedimentos locais e socket
  - É uma construção de linguagem (language level construct) e não de sistema operacional

Chamada Remota de Procedimento (RPC)

- Como é implementada?
  - Funções *stub* são geradas localmente, em cada ponto da comunicação
  - Uma função *stub* se parece com a função que se deseja chamar, mas contem código para enviar e receber mensagens pela rede





Chamada Remota de Procedimento (RPC)

- Etapas
  - O cliente chama um procedimento local, a função *stub*.
  - A função *stub* do cliente empacota os argumentos e constrói a mensagem a ser enviada pela rede (*marshalling*)
  - A função *stub* faz uma chamada ao sistema (socket) para enviar a mensagem para o outro processo
  - A função *stub* do servidor recebe a mensagem e desempacota os parâmetros (*unmarshalling*)
  - A função *stub* do servidor chama o procedimento local
  - A função *stub* do servidor empacota o resultado e faz uma chamada ao sistema para enviar o resultado para o outro processo
  - A função *stub* do cliente recebe a mensagem do servidor, desempacota e retorna para o cliente.
  - O cliente continua sua execução

- Tipos de passagem de parâmetros
  - Passagem por valor
    - Valores dos parâmetros são copiados para a mensagem que sai do cliente
  - Passagem por referência
    - Não faz sentido copiar endereços em chamadas remotas. Por que?

- Tipos de passagem de parâmetros
  - Passagem por valor
    - Valores dos parâmetros são copiados para a mensagem que sai do cliente
  - Passagem por referência
    - Não faz sentido copiar endereços em chamadas remotas. Por que?
    - A alternativa é usar parâmetros *inout*: os valores dos parâmetros são copiados para a mensagem que sai do cliente e copiados de volta da mensagem que chega no cliente

- Representação de dados
  - Representação externa de dados: um protocolo ou conjunto de regras para representar tipos de dados

Questões relacionadas a RPC

- Representação de dados
  - Representação externa de dados: um protocolo ou conjunto de regras para representar tipos de dados
- Que tipo de protocolo de transporte usar
  - Muitas implementações de RPC permitem o uso apenas de TCP

- O que acontece se uma falha ocorrer
  - Na chamada remota existe maior oportunidade para falhas (transmissão, cliente, servidor)
  - Esse tipo de falha não ocorre em chamadas locais
  - Portanto, as falhas quebram a transparência da chamada remota.
  - O cliente que faz uma chamada remota deve estar preparado para tratar exceções dessa natureza

Questões relacionadas a RPC

- Qual a semântica de uma chamada remota?
  - A semântica de uma chamada local é simples: o procedimento chamado é executado exatamente uma vez.
  - E na chamada remota?
    - Semântica de invocação talvez: Surge quando não há medidas de tolerância a falhas. Não há retransmissão nem filtragem de duplicata
    - Semântica de invocação pelo menos uma vez: Surge quando há retransmissão mas não há filtragem de duplicata.
    - Semântica no máximo uma vez: Surge quando há retransmissão de requisição, filtragem de duplicata e retransmissão de resposta.

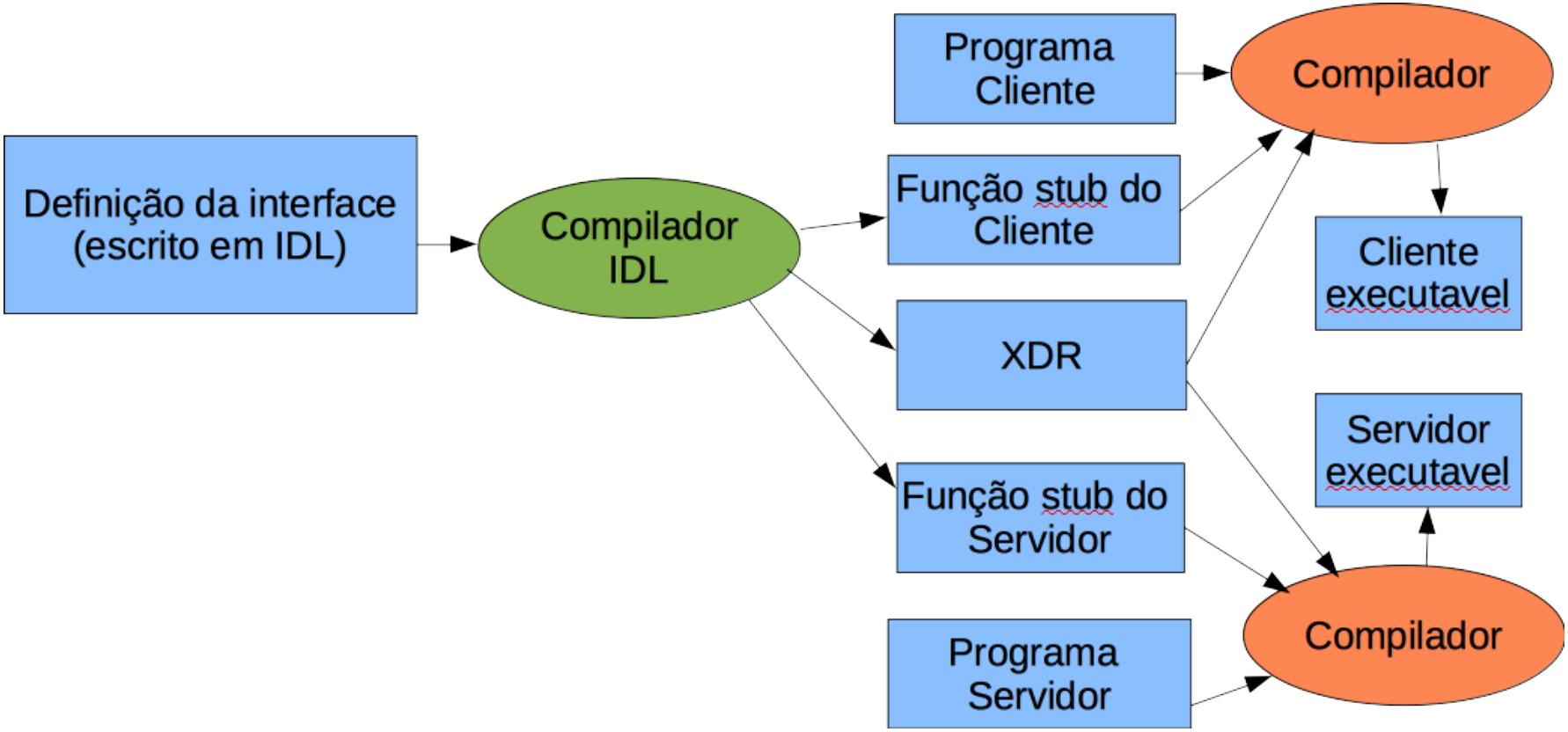
Questões relacionadas a RPC

- RMI Java
  - No máximo uma vez
- CORBA
  - No máximo uma vez
- RPC SUN
  - Pelo menos uma vez

- É possível programar com RPC usando a linguagem C
- Um compilador separado (a parte do compilador da linguagem) gera as funções *stub* do cliente e do servidor
- Esse compilador toma como entrada a definição do procedimento remoto, escrito em uma linguagem de definição de interface(IDL)

- Definição de Interface
  - Declaração contendo:
    - O protótipo dos procedimentos que podem ser chamados remotamente
    - Os parâmetros de entrada e retorno do procedimento

Programando com RPC



RPC – Estudo de Caso

- Sun RPC
  - Foi uma das primeiras bibliotecas de RPC
  - Compilador IDL: rpcgen
  - Pode usar UDP e TCP
  - Representação externa de dados: XDR (padrão IETF - Internet Engineering Task Force)
  - Programador
    - Escreve a definição dos procedimentos remotos
    - Implementa o programa cliente
    - Implementa o programa servidor

RPC – Estudo de Caso

- Suponha o seguinte programa idl (add.x)

```
struct intpair {
    int a;
    int b;
};
program ADD_PROG {
    version ADD_VERS {
        int ADD(intpair) = 1; //só aceita um parâmetro. Caso se deseja mais
                              //parâmetros, deve-se criar uma
                              //estrutura
    } = 1; //Número da versão
} = 999; //Número do programa
```

RPC – Estudo de Caso

- Compilando a idl com rpcgen, os seguintes arquivos são gerados
  - add.h : contém a definição do programa, da versão e as declarações dos procedimentos remotos
  - add_svc.c : código C que implementa o stub do servidor
  - add_clnt : código C que implementa o stub do cliente
  - add_xdr.c : contém as rotinas de marshalling e unmarshalling

Demonstração - add.x

Referências

- COULOURIS, George. Sistemas Distribuídos: Conceitos e Projetos. Bookman. 4ª edição. 2007