# Sistemas Distribuídos
# RPC

**Disciplina:** Sistemas Distribuídos
**Prof.:** Edmar Roberto Santana de Rezende

**Faculdade de Engenharia de Computação**
**Centro de Ciências Exatas, Ambientais e de Tecnologias**
**Pontifícia Universidade Católica de Campinas**

# Remote Procedure Call

## Dynamic Binding

- Como o cliente localiza o servidor?

1. inserir o endereço do servidor no código do cliente
   - ☹ abordagem extremamente inflexível
   - → se o servidor for movido, replicado ou mudar de endereço todos os clientes deverão ser recompilados

2. usar **dynamic binding**

# Remote Procedure Call

## Dynamic Binding

- Especificação de um servidor:

```
#include <header.h>

specification of file_server, version 3.1:

    long read (in char name[MAX_PATH], out char buf[BUF_SIZE],
            in long bytes, in long position);
    long write (in char name[MAX_PATH], in char buf[BUF_SIZE],
            in long bytes, in long position);
    int create (in char[MAX_PATH], in int mode);
    int delete (in char[MAX_PATH]);

end;
```

# Remote Procedure Call

## Dynamic Binding

- Para cada procedimento:
  - o tipo dos parâmetros é informado
  - cada parâmetro é especificado como:
    - **in**, **out** ou **in out**

    → a direção é dada em relação ao servidor

    - **in:** é enviado do cliente para o servidor
    - **out:** é enviado do servidor para o cliente
    - **in out:** é enviado do cliente para o servidor, modificado e enviado de volta para o cliente (cópia e restauração)

      → tipicamente usado para passar ponteiros como parâmetro quando o servidor precisa ler e modificar os dados apontados

  - a direção é crucial para que os stubs cliente e servidor saibam que parâmetros enviar

# Remote Procedure Call

## Dynamic Binding

- Principal uso da especificação:
  - servir de entrada para o gerador de stubs
    - gera os stubs cliente e servidor
    - ambos são colocados nas bibliotecas apropriadas
  - Quando o programa cliente chama algum procedimento definido pela especificação:

    → o procedimento do stub cliente correspondente é ligado a este binário
  - De maneira análoga, quando o programa servidor é compilado:

    → o stub servidor é ligado a ele também

# Remote Procedure Call

## Dynamic Binding

- Quando o servidor inicia sua execução:
  - o servidor exporta sua interface
    - envia uma mensagem para um programa chamado **binder**

      → para tornar sua existência conhecida
    - processo conhecido como **registro do servidor**

      → informa seu nome, sua versão, um identificador único (32 bits) e um **handle** (usado para localizá-lo)
- Como o cliente localiza o servidor?
  - quando o cliente chama um procedimento remoto **read**
    - o stub cliente envia uma mensagem para o **binder**

      → pede para **importar** a versão 3.1 da interface **file_server**

# Remote Procedure Call

## Dynamic Binding

- Se não houver servidor exportando esta interface:
  - a chamada do procedimento **read** falha
- Para que informar a versão?
  - o **binder** pode garantir que interfaces obsoletas resultarão em falha

    → falha na localização do servidor

    → evita que parâmetros sejam interpretados incorretamente
- Vantagem: flexibilidade
  - múltiplos servidores podem suportar a mesma interface

  → permite que o **binder** espalhe os clientes aleatoriamente
- Desvantagem: overhead
  - o binder pode se tornar um gargalo

# Remote Procedure Call

## Semântica RPC na presença de falhas

- Na chamada de um procedimento remoto podem ocorrer 5 tipos de falhas possíveis:

1. o cliente não consegue localizar o servidor
2. mensagem de requisição perdida
3. mensagem de resposta perdida
4. o servidor cai após receber uma requisição
5. o cliente falha após enviar uma requisição

# Remote Procedure Call

## Semântica RPC na presença de falhas

1. *O cliente não consegue localizar o servidor*
   - pode ocorrer devido a dois motivos:
     - o servidor caiu
     - o cliente e o servidor estão executando versões incompatíveis de protocolo, impedindo a sua comunicação

     → Soluções:
     - uso de um código de retorno de função
       - ☹ pode não fazer sentido am alguns casos
     - através do uso de *tratamento de exceções*:
       - ☺ torna o código mais claro e fácil de dar manutenção
       - ☹ nem todas as linguagens prevêem o tratamento de exceção
       - ☹ acaba com a "ilusão" de que procedimentos remotos não diferem dos locais

# Remote Procedure Call

## Semântica RPC na presença de falhas

2. *Mensagem de requisição perdida:*
   - a mensagem de requisição do cliente para o servidor pode ser perdida
   - o kernel deve controlar:
     - através de timeout
     - ao detectar a perda

       → reenviar a mensagem

# Remote Procedure Call

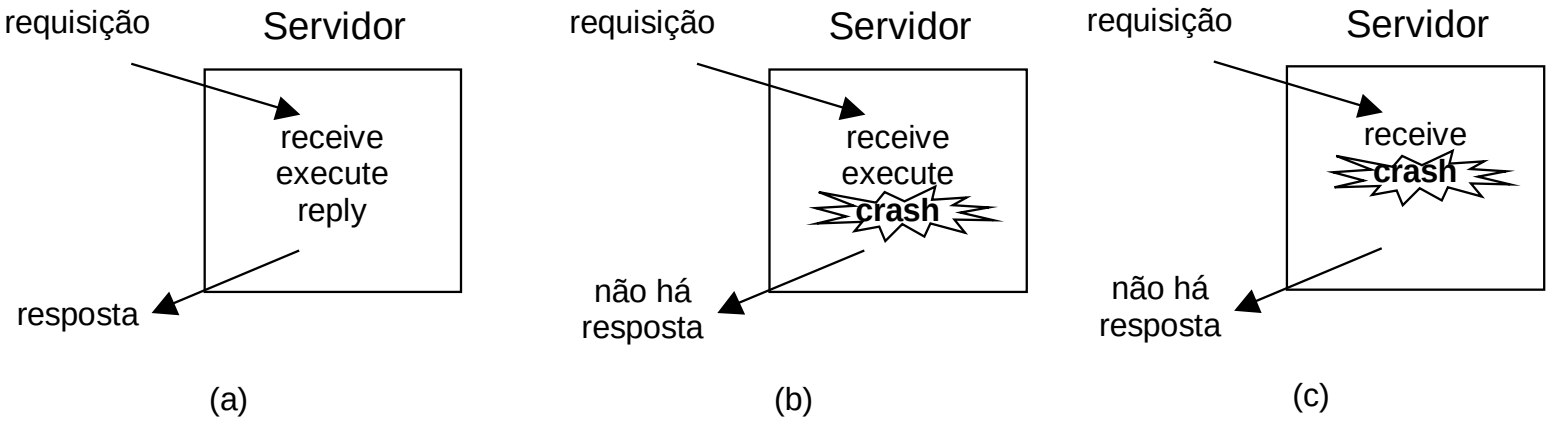
## Semântica RPC na presença de falhas

3. *A mensagem de resposta perdida:*
   - a mensagem de resposta do servidor para o cliente pode ser perdida
   - caso é um pouco mais difícil de tratar
     - uso de *timeout* no kernel do cliente não pode determinar o porquê da ausência de resposta:
       1) a mensagem de resposta pode ter sido perdida, ou
       2) a mensagem de requisição foi perdida ou apenas um servidor muito lento
   - o reenvio de requisição pode causar erros caso o procedimento remoto não seja idempotente
     - **procedimento idempotente** (*idempotent*): é aquele cuja execução não causa efeito colateral. **Ex:** ler um determinado bloco de um arquivo.
   - Solução:
     - numerar as requisições
       → permite que o servidor identifique e ignore uma retransmissão

# Remote Procedure Call

## Semântica RPC na presença de falhas

4. *O servidor cai após receber uma requisição:*
   - também está relacionado a semântica idempotente ou não idempotente de um procedimento

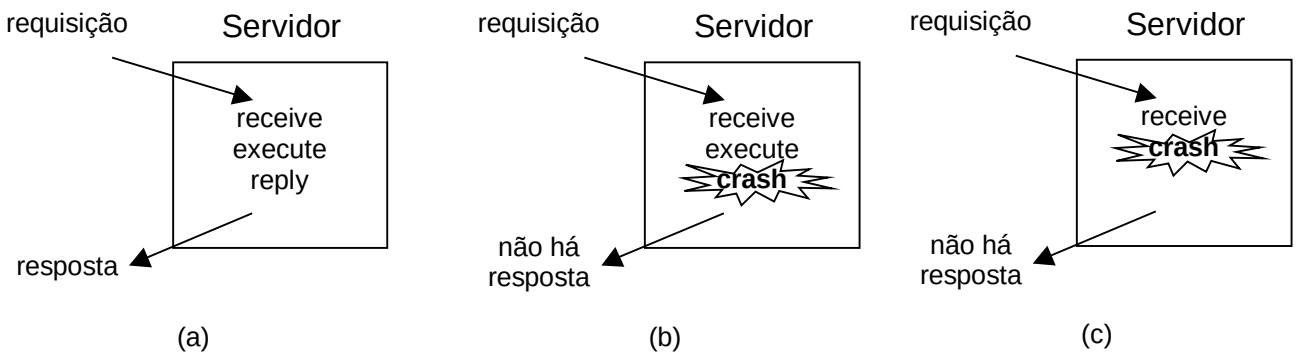


# Remote Procedure Call

## Semântica RPC na presença de falhas

(b) houve a execução da requisição, mas o resultado não será enviado ao cliente

(c) a requisição não chegou a ser processada.




- Logo:
  - no caso (b) a requisição só poderá ser reexecutada se o procedimento for idempotente
  - no caso (c) a reexecução seria permitida sem restrições

# Remote Procedure Call

## Semântica RPC na presença de falhas

- *Implementações RPC tratam a possibilidade de falha do servidor através da definição de uma das três possíveis **semânticas** de execução de procedimento remoto:*
  - pelo menos uma vez (*at least once semantics*):
    - o procedimento vai ser executado pelo menos uma vez, mas possivelmente mais de uma vez
    - → o cliente espera que o servidor volte a funcionar enviando nova requisição
  - no máximo uma vez (*at most once semantics*):
    - o procedimento será executado uma única vez ou nenhuma
    - → no caso de erro, o cliente desiste e reporta erro
  - exatamente uma vez (*exactly once semantics*):
    - garante a execução do procedimento uma única vez
    - → teoricamente seria a melhor semântica, porém difícil de implementar

# Remote Procedure Call

## Semântica RPC na presença de falhas

1. *O cliente falha após enviar uma requisição*
   - a computação estará ativa no servidor mas não existirá nenhum processo aguardando o resultado

     → **processo** ou **computação órfã**

   - dois tipos de problemas:
     - desperdício de CPU
     - processo cliente ao retornar pode receber uma mensagem não esperada enviada ao processo órfão

# Remote Procedure Call

## Semântica RPC na presença de falhas

- Soluções:
  1. extermínio (*extermination*):
     - o cliente mantém log das requisições (em disco)
     - quando falhar, ao reiniciar, elimina todos os processos órfãos
  2. reencarnação (*reincarnation*):
     - quando o cliente reinicia, envia mensagem broadcast declarando o início de uma nova época
     - computações remotas serão finalizadas
  3. reencarnação gentil (*gentle reincarnation*):
     - idem à solução anterior, porém só mata as requisições remotas quando não estiver ativo
  4. expiração (*expiration*):
     - o servidor tem um tempo T para executar
     - se precisar de mais tempo tem que enviar requisição