INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Comunicação em Sistemas Distribuídos

IPC vs RPC vs RMI

Aluno: Artur Rocha Lapot (15011640)

INSTITUTO DE INFORMÁTICA
UFG

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Comunicação em Sistemas Distribuídos

Comunicação é essencial para a operação coordenada de sistemas distribuídos. As três técnicas principais são:

- **IPC** (Inter-Process Communication)
- **RPC** (Remote Procedure Call)
- **RMI** (Remote Method Invocation)

IPC – Comunicação entre Processos

A **Comunicação entre Processos (IPC)** ocorre quando dois ou mais processos, executando em um mesmo sistema operacional ou em máquinas próximas, trocam informações entre si por meio de mecanismos locais.

Etapas do processo:

1. **Criação dos processos:**
   - Cada processo executa em um ambiente protegido (espaço de memória isolado).
2. **Estabelecimento do mecanismo de comunicação:**
   - O sistema operacional disponibiliza diferentes métodos (ex: pipes, filas de mensagem, memória compartilhada, sockets locais).
3. **Troca de mensagens:**
   - O primeiro processo escreve dados em uma área comum ou canal (pipe ou memória compartilhada).
   - O segundo processo lê esses dados diretamente da mesma área ou canal.
4. **Controle da comunicação:**
   - Uso de mecanismos como semáforos ou mutex para evitar condições de corrida.
   - Garantia de integridade dos dados durante a transferência.

Resultado: Comunicação rápida, eficiente e de baixa latência, adequada para processos no mesmo sistema.

RPC – Remote Procedure Call

O **Remote Procedure Call (RPC)** permite que um programa execute procedimentos (funções) em máquinas remotas, como se fossem locais. É um modelo procedural, que abstrai detalhes da rede para os desenvolvedores.

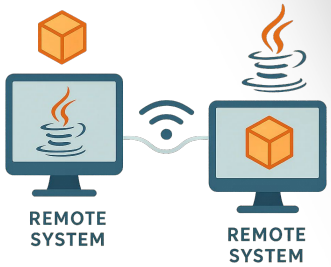
Etapas do processo:

1. **Definição da interface:**
   - Usando IDL (Interface Definition Language), definem-se claramente os procedimentos remotos disponíveis, parâmetros e valores de retorno.
2. **Chamada do procedimento remoto:**
   - O processo cliente chama uma função, sem perceber que ela é executada remotamente.
3. **Serialização dos parâmetros:**
   - O cliente transforma os dados dos parâmetros da função para um formato adequado para transmissão pela rede.
4. **Envio pela rede:**
   - Os dados serializados são enviados para o servidor remoto através do protocolo escolhido (TCP/IP).
5. **Recepção no servidor:**
   - O servidor recebe, desserializa os parâmetros e executa o procedimento solicitado.
6. **Execução e retorno dos resultados:**
   - O servidor executa o procedimento localmente e retorna o resultado serializado ao cliente.
7. **Recepção do resultado pelo cliente:**
   - O cliente recebe o resultado, desserializa, e continua sua execução normalmente.

Resultado: O RPC permite simplicidade de programação e interoperabilidade entre sistemas heterogêneos, porém com maior latência em comparação ao IPC local.

RMI – Remote Method Invocation

O **Remote Method Invocation (RMI)** é uma implementação específica do RPC no paradigma orientado a objetos (principalmente em Java), permitindo a invocação de métodos em objetos remotos como se fossem locais. Embora seja fortemente popularizado pelo Java, o conceito básico pode ser implementado em diversas linguagens orientadas a objetos.

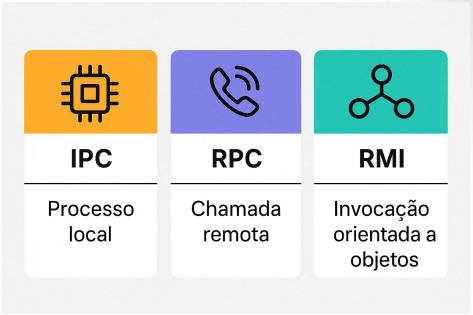


Etapas do processo:

1. **Criação e registro do objeto remoto:**
   - Um objeto Java é criado e disponibilizado para acesso remoto pelo servidor.
   - O objeto é registrado em um registro RMI, onde fica disponível por nome (lookup).
2. **Obtenção da referência ao objeto remoto:**
   - O cliente obtém uma referência ao objeto remoto consultando o registro RMI através do nome do objeto.
3. **Chamada do método remoto:**
   - O cliente chama um método diretamente no objeto remoto, utilizando a referência obtida.
4. **Serialização dos parâmetros (Marshall):**
   - Os parâmetros da chamada são automaticamente serializados pelo sistema RMI para serem transmitidos pela rede.
5. **Envio da chamada remota pela rede:**
   - A invocação do método remoto e os parâmetros serializados são enviados ao servidor.
6. **Desserialização e execução do método (Unmarshall):**
   - O servidor recebe a solicitação, desserializa os parâmetros, e executa o método no objeto remoto.
7. **Serialização e retorno dos resultados:**
   - Após a execução, o resultado (ou exceção) é serializado e enviado de volta ao cliente.
8. **Recepção dos resultados pelo cliente:**
   - O cliente recebe e desserializa os resultados, tratando-os como se fossem retornos normais de um método local.

Resultado: O RMI oferece simplicidade e transparência para desenvolvedores Java, porém limita-se ao ecossistema Java.

IPC vs RPC vs RMI

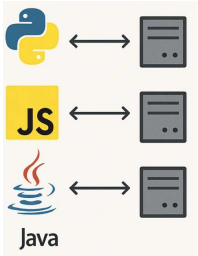


Critério	IPC	RPC	RMI
Tipo de uso	Comunicação entre processos locais	Procedimentos remotos	Métodos remotos em objetos
Comunicação	Local, eficiente e com baixa latência	Rede, maior latência	Rede, com serialização de objetos
Dependência	Sistema operacional local	Interfaces definidas por IDL	Java e JVM
Complexidade	Baixa	Média	Média-alta
Exemplos de uso	Pipes, memória compartilhada, sockets locais	APIs REST, gRPC, JSON-RPC	Java RMI, .NET Remoting, Pyro (Python), CORBA

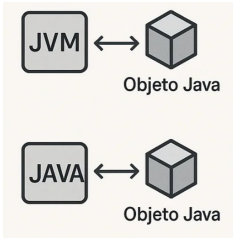
Cenários de Aplicação Ideal

- **IPC**: Comunicação eficiente entre processos em um mesmo computador (ex: troca de mensagens entre threads ou processos locais que compartilham recursos).

- **RPC:** Sistemas distribuídos heterogêneos (diferentes linguagens/plataformas), integração entre diferentes sistemas remotos (ex: integrações entre sistemas distribuídos em diferentes tecnologias, como serviços web, APIs REST e gRPC).



- **RMI:** Aplicações distribuídas fortemente orientadas a objetos, onde é importante invocar métodos em objetos remotos de maneira transparente e natural para o desenvolvedor. (ex: Java RMI, .NET Remoting, CORBA ou Pyro (Python)).



Vantagens e Desvantagens em Resumo

Técnica	Vantagens	Desvantagens
IPC	Alta desempenho, comunicação rápida e simples para processos locais	Limitado ao mesmo sistema operacional ou máquinas próximas
RPC	Independente da plataforma, permite comunicação entre sistemas diferentes (heterogêneos), facilita integração remota	Maior complexidade na implementação, latência mais alta, necessidade de tratar falhas
RMI	Forte suporte à orientação a objetos, invocação remota transparente e simples para desenvolvedores	Geralmente restrito a ambientes orientados a objetos específicos, maior sobrecarga devido à serialização de objetos

Conclusão

- **IPC**: Comunicação altamente eficiente e rápida, ideal para processos locais ou comunicação dentro do mesmo sistema.

- **RPC:** Excelente opção para integrar sistemas diferentes, permitindo comunicação remota de procedimentos de forma independente da linguagem ou plataforma utilizada.

- **RMI:** Abordagem orientada a objetos recomendada para sistemas distribuídos que exigem invocação remota transparente de métodos em objetos. Implementações populares incluem Java RMI, CORBA, .NET Remoting e Pyro para Python.

Cada técnica atende a necessidades específicas, e a escolha ideal depende das características desejadas para o sistema distribuído em desenvolvimento.