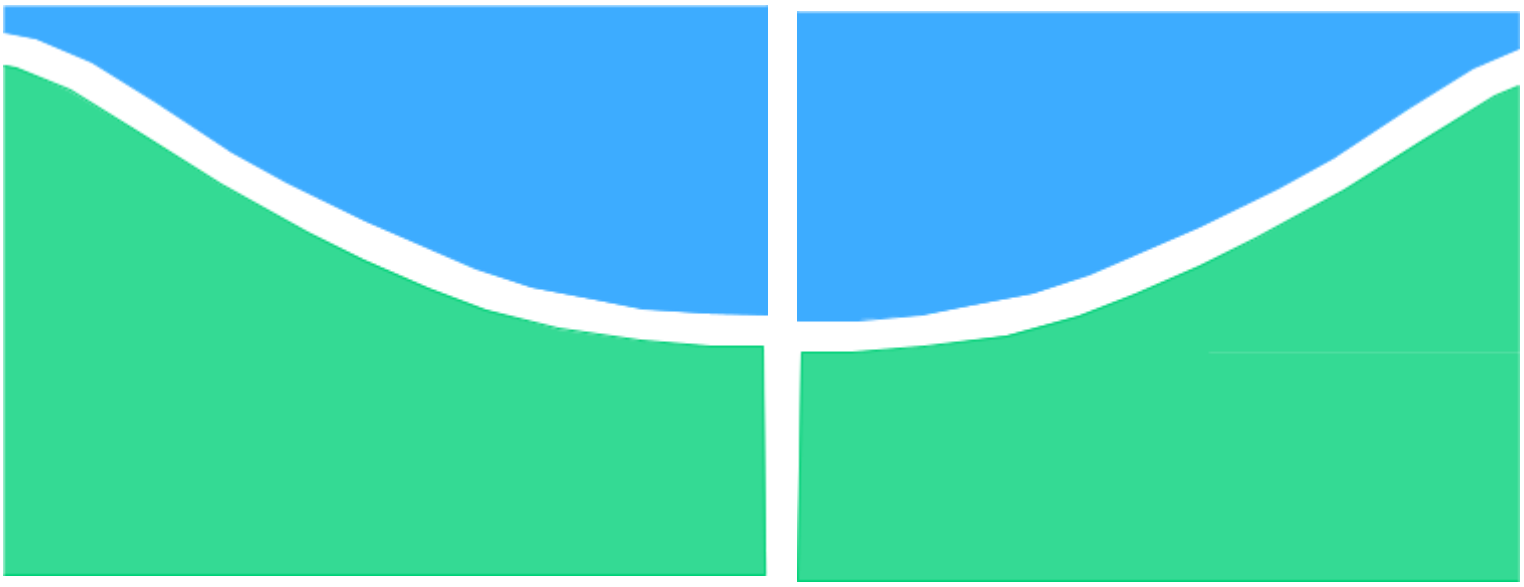


Universidade de Brasília - UnB
Faculdade UnB Gama - FGA

PROGRAMAÇÃO PARA SISTEMAS PARALELOS E DISTRIBUÍDOS

Integrantes:
Jackes da Fonseca - 190030291

Turma: 2021.2

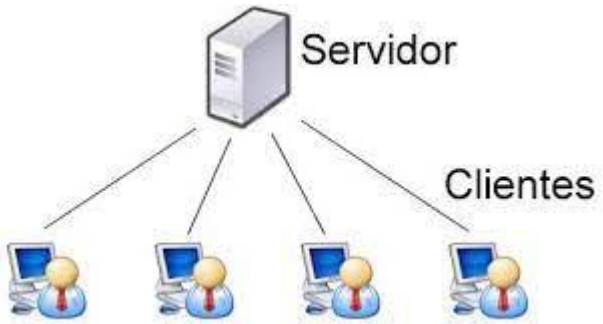
Brasília, DF
2019

Introdução

Sistemas distribuídos em geral são baseados na troca de mensagens entre processos. Dentre os mecanismos de troca disponíveis, as Chamadas de Procedimento Remoto ou RPC (Remote Procedure Call´s) são consideradas até o momento como um middleware básico para a implementação de boa parte dos requisitos de Sistemas Distribuídos (escalabilidade, transparência etc.). Exemplos de aplicações distribuídas:

- WWW
- ICQ
- IRC
- Morpheus/Kazza, etc
- Sistemas bancários
- Sistemas de gerenciamento de redes de telecomunicações, transmissão de energia, etc.
- Sistemas de informação de grandes empresas

De um modo geral, pode-se dizer que as chamadas de procedimento remoto são idênticas às chamadas de procedimento local, com a exceção de que as funções chamadas ficam residentes em hosts distintos. O cliente faz uma solicitação ao servidor e este, por sua vez, responde ao cliente processando e retornando as informações necessárias.



O objetivo da biblioteca RPC é permitir ao programador uma forma de escrever o seu código de forma similar ao método adotado para a programação convencional. Para isso, a estrutura RPC define um esquema de encapsulamento de todas as funções associadas à conexão remota num pedaço de código chamado de STUB.

Soluções

Utilizando 1 worker

1. Deve-se começar criando um arquivo .x que irá conter o IDF

```
$ touch find.x
$ vim find.x
```

2. Nele deverá ser estruturado a interface de comunicação entre cliente e servidor

```
struct operandos
{
	float vet[10];
};
program PROG
{
	version VERSAO
	{
		float BIGGER(operandos) = 1;
		float SMALLER(operandos) = 2;
	} = 100;
} = 55555555;
```

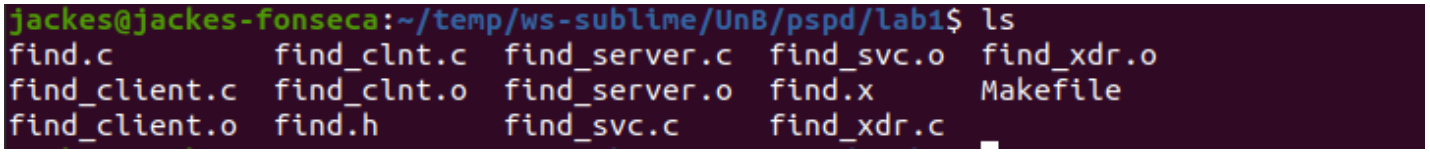
3. Uma vez gerado esse arquivo, pode-se aplicar a ferramenta rpcgen para que os arquivos-fonte em linguagem C dos lados cliente e servidor sejam gerados.

```
$ rpcgen -a find.x
```

4. Dentre os arquivos criados temos

- Arquivo com as definições que deverão estar inclusas nos códigos cliente e servidor
- Arquivo contendo o esqueleto do programa principal do lado cliente
- Arquivo contendo o stub do cliente
- Arquivo que contém as funções xdr necessárias para a conversão dos parâmetros a serem passados entre hosts
- Arquivo que contém o programa principal do lado servidor.
- Arquivo que contém o esqueleto das rotinas a serem chamadas no lado servidor
- Arquivo Makefile.arquivo que deve ser renomeado para Makefile. Contém as diretivas de compilação para a ferramenta make

5. Altere os arquivos _server.c e _client.c com a lógica necessária



6. Mover o arquivo Makefile.find para Makefile

$ mv Makefile.find Makefile

7. Compilar todos os arquivos para gerar o executável _server e _client

$ make

8. Em um terminal, execute o servidor

$ ./find_server

9. Abra outro terminal e execute o cliente informado o IP do host

$ ./find_client 172.17.0.1

Utilizando 2 workers

Como um servidor atende um cliente por vez, nessa situação o problema seria dividido em duas partes onde o cliente iria passar para cada worker uma parte do problema a ser resolvido e devolvam ao cliente que, por sua vez, irá agrupar as informações recebidas.

Conclusão

Os objetivos principais de um middleware são a intergação de sistemaas heterogêneos e a intermediação entre as aplicações e os sistema operacional. Para que estes objetivos sejam alcançados, um middleware deve fornecer serviços que atendam ao domínio de aplicações para o qual foi construído, sendo importante que esse serviço tenha sua base em uma das API's ou protocolos padrões.