Universidade Regional de Blumenau - FURB
Centro de Ciências Exatas e Naturais - CCEN
Departamento de Sistemas e Computação - DSC

SISTEMAS DISTRIBUÍDOS

MARCOS RODRIGO MOMO
momo@furb.br

Blumenau, setembro de 2024.

Aulas anteriores

- Conceitos
- Características
- Tipos de aplicação
- Propriedades críticas de um SD
- Elementos básicos de um SD
- Tipos de SD
- **Arquitetura de SD**
- **Arquitetura versus Middleware**

Unidade 2 – Plano de Ensino

- Comunicação entre processos distribuídos
  - Características de comunicação
  - Tipos de comunicação
  - Comunicação cliente-servidor
  - RPC
  - Sistemas de arquivos distribuídos

Camadas *do Middleware* SD

RMI/RPC
Socket
Web services
Microserviços
Mensageria

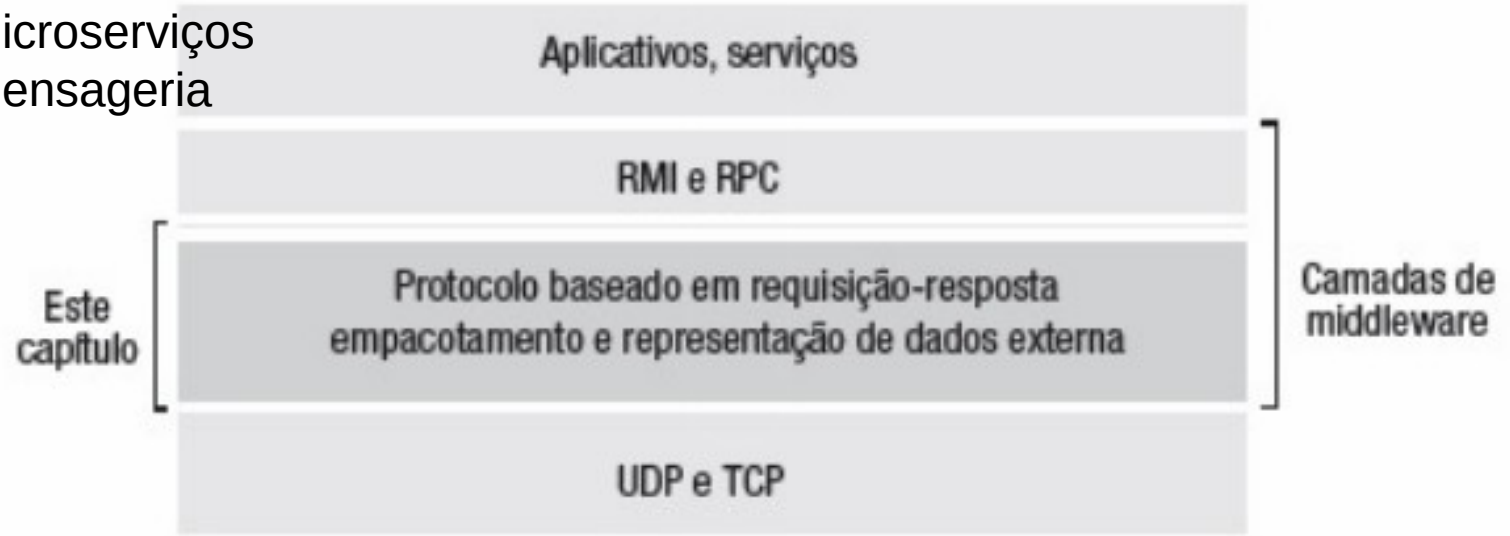


Figura 4.1 Camadas de *middleware*.

Comunicação por fluxo TCP

- A API deste protocolo fornece a abstração de um fluxo de bytes no qual podem ser lidos *(receive)* e escritos *(send)*
- As características da rede são ocultas pela abstração de fluxo *(stream)*

Comunicação por fluxo TCP - Características

- Tamanho das mensagens: o aplicativo escolhe o volume de dados que vai ser enviado ou recebido em um fluxo
- Mensagens perdidas: o protocolo TCP usa um esquema de confirmação
- Controle de fluxo: o protocolo TCP tenta combinar a velocidade dos processos que leêm e escrevem no fluxo
- Duplicação e ordenamento de mensagens: identificadores de mensagens são associados em cada datagrama IP
- Destino de mensagem: dois processos que estão em comunicação estabelecem uma conexão antes de poderem se comunicar por meio de um fluxo

Comunicação por fluxo TCP – características

- O protocolo TCP pressupõe que quando dois processos estão estabelecendo uma conexáo, um deles desempenha um papel de cliente e o outro de servidor
- O papel do cliente
  - Criação de um soquete vinculado a qualquer porta
  - Pedido de *connect* solicitando uma conexão a um servidor a uma determinada porta

Comunicação por fluxo TCP – características

- O papel do servidor
  - Envolve a criação de um soquete de escuta
  - Vincula uma porta de serviço
  - O soquete de escuta mantém uma fila de pedidos da conexão recebidos

API Java para fluxos TCP

- A interface Java para fluxo TCP é fornecida pelas classes:
  - *ServerSocket*
  - *Socket*

ServerSocket

- Usada por um servidor para criar um soquete em uma porta de serviço para receber requisitções de *connect* de clientes
- Através do método *Accept* recupera um pedido *connect* da fila
- O resultado da execução de *accept* é uma instância de *Socket* – permite dar acesso aos fluxos de comunicação com o cliente

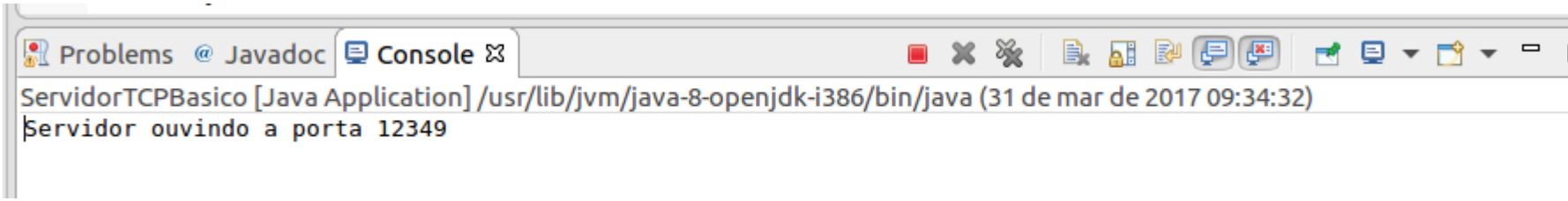
Socket

- Esta classe é usada pelos dois processos de uma conexão
- O cliente usa um construtor para criar uma soquete
  - Nome de *host DNS*
  - Porta do servidor
- O construtor cria um soquete associado a uma porta local e conecta com o computador remoto e com o número de porta especificado

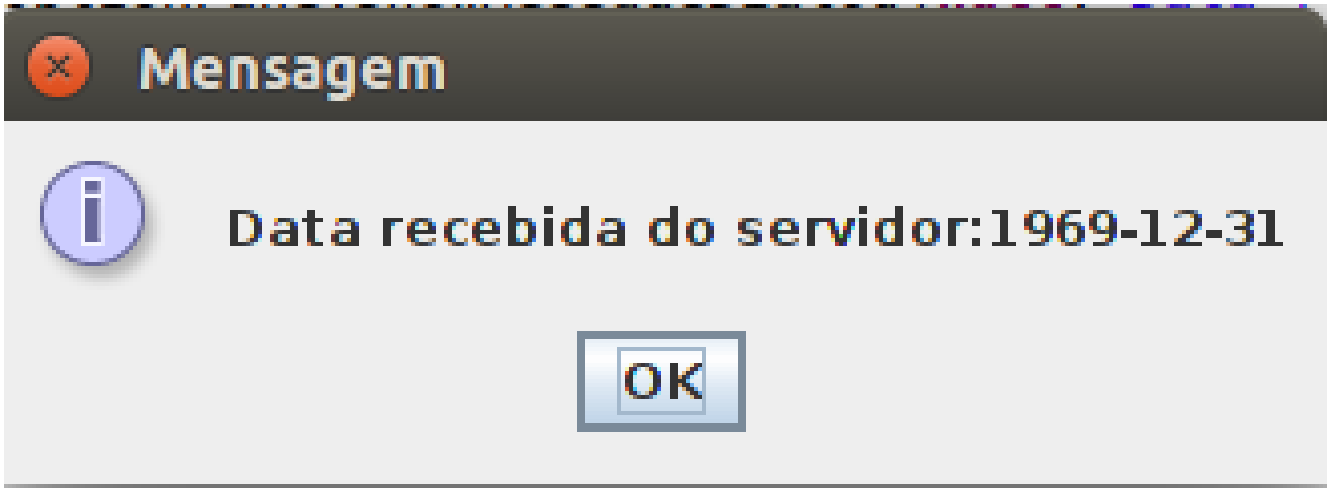
Socket

- A classe *Socket* fornece métodos:
  - *GetInputStream* e *getOutputStream* para acessar os dois fluxos associados a um soquete
  - *DataInputStream* e *DataOutputStream* permitem representações binárias de tipos dados primitivos serem lidos e escritos de forma independente de máquina

Servidor



Cliente



Soquetes

- As duas formas de comunicação UDP e TCP usam os conceitos de soquetes
- A comunicação entre os processos consistem em transmitir uma mensagem entre um soquete de um processo a um soquete de outro processo
- Para que o um processo receba uma mensagem, seu soquete deve estar vinculado a uma porta local e a um numero IP do computador que é executado
- O processo pode usar o mesmo soquete para enviar e receber mensagens
- Cada computador tem 2exp16 numero de portas disponíveis para serem usados pelos processos para envio o recepção de mensagens

Soquetes

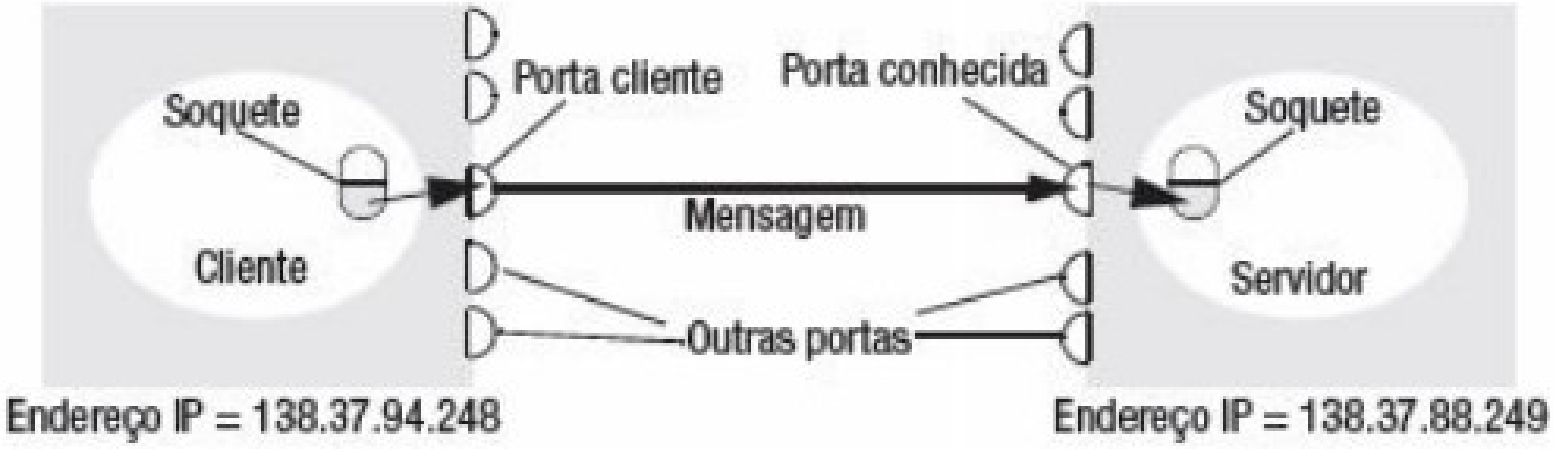


Figura 4.2 Soquetes e portas.

Comunicação por fluxo TCP

- A API deste protocolo fornece a abstração de um fluxo de bytes no qual podem ser lidos *(receive)* e escritos *(send)*
- As características da rede são ocultas pela abstração de fluxo *(stream)*

Comunicação por fluxo TCP - Características

- Tamanho das mensagens: o aplicativo escolhe o volume de dados que vai ser enviado ou recebido em um fluxo
- Mensagens perdidas: o protocolo TCP usa um esquema de confirmação
- Controle de fluxo: o protocolo TCP tenta combinar a velocidade dos processos que leêm e escrevem no fluxo
- Duplicação e ordenamento de mensagens: identificadores de mensagens são associados em cada datagrama IP
- Destino de mensagem: dois processos que estão em comunicação estabelecem uma conexão antes de poderem se comunicar por meio de um fluxo

Comunicação por fluxo TCP – características

- O protocolo TCP pressupõe que quando dois processos estão estabelecendo uma conexáo, um deles desempenha um papel de cliente e o outro de servidor
- O papel do cliente
  - Criação de um soquete vinculado a qualquer porta
  - Pedido de *connect* solicitando uma conexão a um servidor a uma determinada porta

Comunicação por fluxo TCP – características

- O papel do servidor
  - Envolve a criação de um soquete de escuta
  - Vincula uma porta de serviço
  - O soquete de escuta mantém uma fila de pedidos da conexão recebidos

Comunicação por fluxo TCP – características

- O par de soquetes (cliente e servidor) são conectados por dois fluxos, um em cada direção
  - Fluxo de entrada
  - Fluxo de saída
- Quando um aplicativo *encerra* um soquete (operação *close*), significa que ele não escreverá mais nenhum dado em seu fluxo de saída
- Os dados do seu *buffer* de saída são enviados para o outro lado do fluxo e colocados na fila de entrada do soquete de destino com uma indicação que o fluxo está desfeito

Emprego de TCP

- Muitos serviços frequentemente usados são executados em conexões TCP com números de portas reservados:
  - HTTP: hipertexto, comunicação entre navegadores e servidores web
  - FTP: protocolo de transferência de arquivos
  - Telnet: serviço de acesso a um computador remoto
  - SMTP: protocolo de transferência de correio eletrônico

API Java para fluxos TCP

- A interface Java para fluxo TCP é fornecida pelas classes:
  - *ServerSocket*
  - *Socket*

ServerSocket

- Usada por um servidor para criar um soquete em uma porta de serviço para receber requisitçóes de *connect* de clientes
- Através do método *Accept* recupera um pedido *connect* da fila
- O resultado da execução de *accept* é uma instância de *Socket* – permite dar acesso aos fluxos de comunicação com o cliente

Socket

- Esta classe é usada pelos dois processos de uma conexão
- O cliente usa um construtor para criar uma soquete
  - Nome de *host DNS*
  - Porta do servidor
- O construtor cria um soquete associado a uma porta local e conecta com o computador remoto e com o número de porta especificado

Socket

- A classe *Socket* fornece métodos:
  - *GetInputStream* e *getOutputStream* para acessar os dois fluxos associados a um soquete
  - *DataInputStream* e *DataOutputStream* permitem representações binárias de tipos dados primitivos serem lidos e escritos de forma independente de máquina

Características TCP

- Quando necessitamos de uma troca confiável de informações, isto é, quando é necessária a confirmação de recebimento da mensagem enviada
- TCP (T*ransmission Control Protocol*), estabelece uma conexão entre dois pontos interligados
- Por exemplo: uma mensagem enviada de um *host* a outro é confirmada pelo *host* receptor indicando o correto recebimento da mensagem
- Uma mensagem pode ser enviada em vários pacotes, o TCP cuida para que os pacotes recebidos sejam remontados no *host* de destino na ordem correta
- Caso algum pacote não tenha sido recebido, o TCP requisita novamente este pacote

Características TCP

- Somente após a montagem de todos os pacotes é que as informações ficam disponíveis para nossas aplicações
- A programação do TCP com *sockets* utiliza *streams*, o que simplifica o processo de leitura e envio de dados pela rede
- *Streams* são objetos Java que permitem obter dados de qualquer fonte de entrada, seja o teclado, um arquivo ou até mesmo um fluxo de bytes recebidos pela rede
- Isto torna a manipulação de dados da rede como se fossem arquivos, ao ler dados enviados é como se estivéssemos lendo um arquivo e ao enviar dados é como se estivéssemos gravando dados em um arquivo.

Aspectos importantes Servidor

- Implementando o Servidor
  - importar o pacote java.net
  - *import java.net.ServerSocket;*
  - *import java.net.Socket;*
  - instanciar um objeto do tipo ServerSocket
  - *ServerSocket server = new ServerSocket(12349);*
  - Este objeto vai atender pedidos via rede e em determinada porta, neste caso 12349.

Aspectos importantes Servidor

- Após Servidor receber uma conexão, um objeto
- do tipo *Socket* deve ser criado para manter a comunicação entre o cliente e o servidor.
- *Socket client = server.accept();*

- Este objeto vai tratar da comunicação com o cliente, assim que um pedido de conexão chegar ao servidor e a conexão for aceita

Aspectos importantes Servidor

- Comunicação por endereço IP
- A classe *InetAdress* permite obter informações sobre um computador conectado a rede
- getAddress(): Este método retorna um array de bytes contendo o endereço IP.

  Para isso, o nome do host que se deseja obter o endereço IP é fornecido ao método getByName da classe InetAddress. Exemplo:

- *byte[] b= InetAddress.getByName("localhost").getAddress();*
- *System.out.println(b[0] + "." + b[1] + "." + b[2] + "." + b[3]);*

Aspectos importantes Servidor

- getHostAddress(): Este método retorna uma String contendo o endereço IP
- System.out.println("Endereço: " + InetAddress.getByName("localhost").getHostAddress());

- getHostName(): Dado um array de bytes contendo o endereço IP de um host, este método retorna uma String com o nome do host

byte[] addr = {127,0,0,1};

System.out.println(InetAddress.getByAddress(addr).getHostName());

Atividades de laboratório

- Vamos começar agora a trabalhar na prática com *sockets*.
- Primeiro vamos montar um servidor TCP que permite a seus clientes:
  - solicitarem a data e a hora atuais do servidor

Atividades de laboratório Funcionamento:

1. Ao ser iniciado o servidor fica ouvindo na porta 12349 a espera de conexões de clientes;
2. O cliente solicita uma conexão ao servidor;
3. O servidor exibe uma mensagem na tela com o endereço IP do cliente conectado;
4. O servidor aceita a conexão e envia um objeto Date ao cliente;
5. O cliente recebe o objeto do servidor e faz o *cast* necessário, em seguida exibe na tela as informações de data;
6. O servidor encerra a conexão.

Acesso ao Serviço Visa Payment

O acesso é feito utilizando um protocolo proprietário via socket TCP com estilo de comunicação *request-reply*. O IP e porta para conexão são:

IP: **127.0.0.1** Porta: **8900** Protocolo: **TCP**

Cartões da bandeira Visa **começam com o número 4 e possuem 16 dígitos**.

Exemplo de conversação correta:

Cliente (requisição)		Servidor (resposta)
Conexão TCP		**Visa Payment Service**
PAY	Solicita efetuação de pagamento	**OK**
4234567890123456	Nº do cartão	**OK**
ROQUE BEZERRA	Nome do cartão	**OK**
12/2025	Data de expiração do cartão (MM/yyyy)	**OK**
252.43	Valor da transação (em formato americano)	**OK**
COMMIT	Confirma a transação	**OK (1ª linha)** **Dados da transação* (2ª linha)**

*** Dados da transação:**
a3ba892b-0569-4ae4-9a36-3494406c2525:ROQUE BEZERRA:3456:12/2025:252.43:239.89
Composto por:

- Identificador da transação;
- Nome da pessoa;
- Últimos seis dígitos de cartão;
- Data de validade do cartão
- Valor da transação cobrada do cliente
- Valor da transação recebido pela empresa (descontado a taxa da Visa)

Código de erro

Para qualquer requisição, em caso de erro, o serviço irá retornar o texto "ERROR X", onde X é um dos possíveis códigos listados abaixo:

Código do Erro	Motivo
3000	Comando inválido
3001	Número de cartão de crédito inválido
3002	Cartão de crédito não é da bandeira Visa
3003	Nome da pessoa inválido
3004	Data de expiração do cartão de crédito inválida
3005	Valor da transação inválido
3006	Sistema indisponível

Acesso ao Serviço Mastercard Payment

O acesso é feito utilizando um protocolo proprietário via socket TCP com estilo de comunicação *request-reply*. O IP e porta para conexão são:
IP: **127.0.0.1** Porta: **8901** Protocolo: **TCP**

Cartões da bandeira Mastercard **possuem um total de 16 dígitos, com os dois primeiros entre 51 e 55 (inclusivos)**.

Exemplo de conversação correta:

Cliente (requisição)		Servidor (resposta)
Conexão TCP		**Mastercard Payment Service**
PAYMENT	Solicita efetuação de pagamento	**OK**
5348	Primeira parte do nº do cartão	**PROCEED**
3453	Segunda parte do nº do cartão	**PROCEED**
5489	Terceira parte do nº do cartão	**PROCEED**
3654	Quarta parte do nº do cartão	**PROCEED**
ROQUE BEZERRA	Nome do cartão	**PROCEED**
12/2025	Data de expiração do cartão (MM/yyyy)	**PROCEED**
252.43	Valor da transação (em formato americano)	**PROCEED**
COMMIT	Confirma a transação	**OK (1ª linha)** **Dados da transação* (2ª linha)**

*** Dados da transação: a3ba892b-0569-4ae4-9a36-3494406c2525:3456:252.43:244.86**
Composto por:

- Identificador da transação;
- Últimos seis dígitos de cartão;
- Valor da transação cobrada do cliente
- Valor da transação recebido pela empresa (descontado a taxa da Mastercard)

Código de erro

Para qualquer requisição, em caso de erro, o serviço irá retornar o texto "ERROR X", onde X é um dos possíveis códigos listados abaixo:

Código do Erro	Motivo
4000	Comando inválido
4001	Número de cartão de crédito inválido
4002	Cartão de crédito não é da bandeira Visa
4003	Nome da pessoa inválido
4004	Data de expiração do cartão de crédito inválida
4005	Valor da transação inválido
4006	Sistema indisponível