

# Sistemas Distribuídos

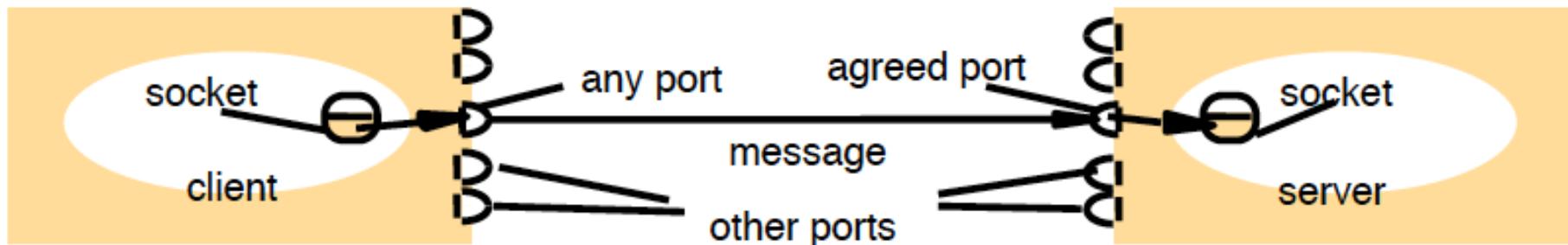
Comunicação entre Processos

Prof. Tales Viegas

<https://fb.com/ProfessorTalesViegas>

# Sockets

- ▶ Ideia surgida com o sistema Unix de Berkeley  
– BSD Unix)
- ▶ Abstração para representar a comunicação entre processos
- ▶ A comunicação entre dois processos consiste na transmissão de uma mensagem de um socket em um processo para um socket de outro



Internet address = 138.37.94.248

Internet address = 138.37.88.249

# Sockets

- ▶ Nos protocolos Internet, as mensagens são enviadas para um par:
  - Endereço Internet
  - Número de porta
- ▶ O socket de um processo tem de ser conectado a uma porta local para que possa começar a receber mensagens
- ▶ Uma vez criado, serve tanto para receber como para enviar mensagens

# Sockets

- ▶ O número de portas disponíveis para um computador é  $2^{16}$  (65536)
- ▶ Para receber mensagens, um processo pode usar várias portas simultaneamente, mas não pode compartilhar uma porta com outro processo diferente no mesmo computador
- ▶ Cada Socket é associado a um determinado protocolo, UDP ou TCP

# Sockets

```
public Cliente(){  
    ...  
    try {  
        Socket sc = new Socket("127.0.0.1", 2222);  
        System.out.println("Cliente cria Socket " + sc);  
    }  
    ...  
}  
  
public Servidor(){  
    ...  
    try {  
        ServerSocket ss = new ServerSocket(2222);  
        while(true){  
            Socket s = ss.accept();  
            System.out.println("Servidor - accept executado " + s);  
            System.out.println(s.getOutputStream());  
        }  
    }  
}
```

# Sockets

- ▶ Obter endereço IP de uma máquina em Java

- Classe InetAddress – representa os endereços IP

```
InetAddress ia = InetAddress.getByName("www.ulbra.br");  
System.out.println(ia.getHostAddress());
```

```
InetAddress ia = InetAddress.getByName("123.456.789.10");  
System.out.println(ia.getHostName());
```

# Protocolos de Rede

- ▶ Principais protocolos de rede atuais
- ▶ UDP – User Data Protocol
  - Protocolo sem conexão
  - Comunicação por “datagrams”
- ▶ TCP – Transfer Control Protocol
  - Protocolo com conexão
  - Comunicação por streams

# Comunicação UDP

- ▶ A comunicação entre dois processos é feita através dos métodos *send* e *receive*
- ▶ Um item de dados (“datagram”) é enviado por UDP sem confirmação (“acknowledgment) nem reenvio
- ▶ Qualquer processo que queira enviar ou receber uma mensagem tem que criar um socket com o IP da máquina local e o número de uma porta local.

# Comunicação UDP

- ▶ A porta do servidor terá de ser conhecida pelos processos clientes
- ▶ O cliente pode usar qualquer porta local para conectar o seu socket
- ▶ O processo que invocar o método *receive* (cliente ou servidor) recebe o IP e a porta do processo que enviou a mensagem, juntamente com os dados da mensagem

# Comunicação UDP

- ▶ Tamanho da mensagem
  - O receptor da mensagem tem que definir um array (buffer) com dimensão suficiente para os dados da mensagem
- ▶ O IP permite mensagens até  $2^{16}$  bytes
- ▶ Mensagens maiores que o buffer definido serão truncadas

# Comunicação UDP

- ▶ A operação *send* não é bloqueante
- ▶ O processo retorna do *send* assim que a mensagem é enviada
- ▶ No destino, a mensagem é colocada na fila do socket respectivo
- ▶ Se nenhum processo estiver ligado ao socket, a mensagem é descartada

# Comunicação UDP

- ▶ A operação *receive* é bloqueante
- ▶ O processo que executa o *receive* bloqueia até que consiga ler a mensagem para o buffer do processo
- ▶ Enquanto espera por uma mensagem, o processo pode criar uma nova thread para executar outras tarefas
- ▶ Um timeout pode ser associado ao socket, findo o qual o receive desbloqueia

# Comunicação UDP

- ▶ Modelo de Falhas
- ▶ Falhas por omissão: A mensagem não chega porque
  - Buffer cheio local ou remotamente
  - Erro de conteúdo – checksum error
- ▶ Falha de ordenação: As mensagens chegam fora da ordem

# Classe DatagramSocket em Java

- ▶ Permite criar um socket na máquina local para o processo corrente
- ▶ Construtor sem argumentos, usa a primeira porta disponível
- ▶ Construtor com argumentos especifica-se o número da porta
- ▶ Se a porta está sendo utilizada, é gerada uma exceção do tipo SocketException

# Classe DatagramPacket em Java

- ▶ Ao instanciar um DatagramPacket para enviar uma mensagem, usar o construtor com os parâmetros:
  - Um array de bytes que contém a mensagem
  - O comprimento da mensagem
  - O endereço IP do Socket destino (objeto do tipo InetAddress)
  - Número da porta no Socket destino

# Classe DatagramPacket em Java

- ▶ Ao instanciar um DatagramPacket para receber uma mensagem, usar o construtor com os parâmetros:
  - Referência de um buffer de memória para onde a mensagem será transferida
  - O comprimento deste buffer
- ▶ A mensagem é colocada neste objeto do tipo DatagramPacket
- ▶ Para extrair os dados desta mensagem, utiliza-se o método getData() da classe DatagramPacket
- ▶ Os métodos getPort() e getAddress() devolvem o número da porta e o IP do processo emissor, respectivamente.

# Comunicação via TCP

- ▶ Utilização da abstração *Stream* para ler/escrever dados
- ▶ A aplicação é quem decide quantos bytes serão enviados ou lidos do stream, sem a preocupação com o tamanho dos pacotes
- ▶ O protocolo TCP utiliza um esquema de confirmação de recepção de mensagens e, se necessário, retransmite a mesma.

# Comunicação via TCP

- ▶ O TCP tenta “uniformizar” as velocidades dos processos que lêem/escrevem em um stream.
  - Se quem escreve é muito mais rápido de quem lê, então o processo que escreve é bloqueado até que o outro processo leia dados suficientes
- ▶ Identificadores de mensagens são associados com cada pacote de dados, permitindo ao receptor detectar e rejeitar mensagens duplicadas ou reordenar mensagens fora de ordem

# Comunicação via TCP

- ▶ Um par de processos estabelece uma conexão antes de poderem se comunicar através de stream.
  - A partir desta ligação, podem se comunicar sem ter de indicar IP/Porta

# Cliente TCP

- ▶ Cria um objeto do tipo Socket, que tenta estabelecer uma ligação com uma porta do servidor em uma máquina remota, especificando IP/Porta desta máquina

# Servidor TCP

- ▶ Cria um objeto do tipo “listening” Socket, associado a uma das portas livres do servidor.
- ▶ O processo fica bloqueado até que receba um pedido de conexão nesta porta
- ▶ Ao chegar, o servidor aceita a conexão, instanciando um novo Socket que, assim como o Socket do cliente, associa 2 streams de dados (uma para envio e outra para recebimento de dados)

# Modelo de Falhas

- ▶ Checksum para detectar e rejeitar pacotes corrompidos
- ▶ Timeouts e retransmissão para lidar com pacotes perdidos
- ▶ Número de sequência para detectar e rejeitar pacotes duplicados

# Modelo de Falhas

- ▶ Se uma mensagem não chega porque o sistema está congestionado, ele não recebe a confirmação da recepção da mensagem até que a conexão seja cancelada após um certo tempo (timeout)
- ▶ A mensagem não é retransmitida, os processos participantes ficam sem saber o que aconteceu
  - Falha na rede?
  - Falha em outro processo?

# Serialização de Estruturas de Dados

- ▶ Tanto o processo local como o processo remoto manipulam estruturas de dados locais
- ▶ Para a transmissão de dados em uma mensagem, é necessário serializar estes dados para uma sequência de bytes
- ▶ Do outro lado, os dados devem ser reestruturados de forma a representarem a informação original, mesmo que a arquitetura da máquina do processo receptor seja diferente da do emissor

# Exemplo de diferença de formatos

- ▶ Valores inteiros podem ser representados com o bit mais significativo em primeiro lugar (mainframe IBM) ou com o bit significativo no fim (processadores Intel)
- ▶ Valores reais
  - Formato IEEE574 (Intel)
  - Formato BDC (mainframe IBM)
- ▶ Valores caracter
  - 1 char = 1 byte (UNIX)
  - 1 char = 2 bytes (Unicode)
- ▶ A heterogeneidade do Hardware obriga a utilização de formatos neutros de serialização

# Formas para trocar valores

- ▶ Ter uma representação externa comum para os dois
  - Os valores são convertidos para uma representação externa e depois, no receptor, são convertidos para o formato do receptor
  - Se dois computadores são iguais, podemos omitir a conversão

# Formas para trocar valores

- ▶ Não ter a representação externa mas, junto com os dados, enviar a informação do formato utilizado
- ▶ Assim o receptor poderá converter os valores, se necessário

# Serialização

- ▶ Na implementação de RPC (“Remote Procedure Calling”) e de RMI (“Remote Method Invocation”), qualquer tipo de dados que possa ser passado como argumento ou devolvido como resultado deve poder ser serializado.
- ▶ Um padrão definido para a representação de estruturas de dados e dos tipos primitivos denomina-se “representação externa de dados” (External Data Representation)

# Serialização

- ▶ Os formatos podem ser binários (ex: Sun XDR, RFC 1832, CDR-Corba)
  - São compactos e eficientes em termos de processamento
- ▶ Ou podem ser baseados em texto
  - Podem ser custosos pelo parsing e pelo par de conversões (nativo-texto, texto-nativo)

# Serialização

- ▶ O processo de transformar os dados do seu formato interno para uma representação externa que possa ser transmitida em uma mensagem denomina-se “marshalling” (serialização)
- ▶ O processo de converter os dados da representação externa para o formato interno do receptor, reconstruindo as estruturas de dados, denomina-se “unmarshalling” (desserialização)
- ▶ O middleware é que realiza o processo de marhalling/unmarshalling

# Exemplo em Java

- ▶ Para que um objeto em Java possa ser serializado, é necessário que a classe dele implemente a interface Serializable
- ▶ Os objetos desta classe poderão ser utilizados para comunicação entre processos ou para serem armazenados
- ▶ Ao desserializar, pode acontecer do processo receptor não conhecer a que classe pertence o objeto.
- ▶ O nome da classe e um número de versão são adicionados na serialização

# Exemplo em Java

- ▶ Objetos que contenham referências a outros objetos
  - O objeto referenciado também é serializado
  - A cada referência é associado um *handle*
  - Em posteriores serializações do mesmo objeto, é usado o seu *handle* (economia de espaço)

# Exemplo em Java

```
class Pessoa implements Serializable{  
    private String nome;  
    private String sobrenome;  
    private int idade;  
  
    Pessoa(String nome, String endereço,  
           int idade){  
        this.nome = nome;  
        this.endereco = endereço;  
        this.idade = idade;  
    }  
    ...  
}
```

# Exemplo em Java

- ▶ Seja o objeto

- Pessoa p = new Pessoa("Tales", "Viegas", 38);

Person	8-bytes version number		h0
3	int ano	java.lang.String nome:	Java.lang.String sobrenome:
1934	5 Tales	6 Viegas	h1

classe, versão  
Número, tipo e nome  
dos atributos  
Valores dos atributos

h0 e h1 são handles

# Exemplo em Java

- ▶ Para serializar o objeto p, usa-se o método
  - void writeObject(Object) da classe ObjectOutputStream
- ▶ Para desserializar:
  - Object readObject(), da classe ObjectInputStream

# Referência para Objetos Remotos

- ▶ Uma referência para um objeto remoto é um identificador de um objeto válido em todo o âmbito de um sistema distribuído
- ▶ Seja um objeto que desejamos acessar:
  - A sua referência deve existir:
    - no processo local
    - na mensagem que enviamos ao objeto
    - no processo remoto que possui a instância do objeto cujo método queremos invocar

# Referência para Objetos Remotos

- ▶ Referências remotas devem ser geradas de modo a garantir unicidade no espaço/tempo
- ▶ Exemplo: concatenando:
  - o endereço IP
  - o número da porta do processo que contém o objeto
  - a data da criação
  - número sequencial do objeto

# Comunicação Cliente–Servidor

- ▶ É necessário um protocolo que, utilizando um mecanismo de transporte (ex: TCP ou UDP), permita a comunicação entre cliente e servidor
- ▶ Protocolo pedido–resposta (request–reply protocol)
- ▶ Usado pela maioria dos sistemas que suportam RPC e RMI

# RMI

- ▶ Isto é para a aula que vêm!