

Universidade Paulista – UNIP

Ciência da Computação

**APS: Atividade Prática Supervisionada**

**Sockets de Berkeley**

|  |  |
| --- | --- |
| *Daniela Ferreira* | *D394BC-7* |
| *Leonardo Henrique Machado* | *D41186-1* |
| *Lucas Merlin de Oliveira* | *D22398-4* |
| *Maycon Ravyer de A. Brasil* | *D2602E-0* |
| *Paulo Henrique Paiva da Silva* | *D4122I-8* |

**São Paulo**

**2019**

**ÍNDICE**

# 

# **INTRODUÇÃO**

Atualmente pode-se dizer que mais de 4 bilhões de pessoas ao redor do mundo estão conectadas à rede de internet de algum modo, seja através de computadores, tablets e o mais comum, os smartphones. Pode-se chegar à conclusão de que havendo a conexão em rede de internet, em algum momento será necessária estabelecer alguma forma de comunicação, para envio e recebimento de dados e informações, tal como mensagens instantâneas, e-mails, etc.

Para que tudo isso ocorra de forma eficiente, foram criadas diversas estruturas e aplicações para que os mais diversos dispositivos existentes possam estabelecer conexões entre si.

Uma das aplicações pioneiras de comunicação em rede foi o Socket de Berkeley, que é considerada a base para a maioria das interfaces entre protocolos de internet existentes.

1. **SOCKETS**

Os sockets estabelecem uma comunicação por meio de protocolos de transportes, que por sua vez fazem parte da estrutura de camadas de rede, no qual são controladas pelo Sistema Operacional. Segundo o professor universitário de ciência da computação da Universidade de Massachusetts, James Kurose, define que *“socket é a interface entre a camada de aplicação e a de transporte dentro de uma máquina.”.*

Ao longo dos anos, conforme a demanda por acesso à internet aumentou, foram necessárias a criação de novas estruturas que pudessem suportar o tráfego de dados com segurança e integridade (sem perda ou alterações de dados). Alguns computadores foram dedicados apenas para prover serviços na rede, ao passo que os demais acessariam tais serviços. Essa estrutura hoje é conhecida como Cliente-Servidor. Como exemplo, pode-se citar um site: Ele provê serviços e recursos, e um usuário, através de um navegador, acessa esses recursos, logo, um site seria o servidor, e o navegador o cliente. No caso o site aguarda conexões de entrada e requisições por parte de um cliente, e o navegador, que é o cliente, solicita a conexão para o servidor realizar alguma requisição. Logo, pode-se concluir que todas as redes que utilizam servidores são do tipo Cliente-Servidor.

É importante saber e lembrar que: não é o computador em si que difere quem é o cliente e quem é o servidor, mas sim da forma como o programa executado utiliza os sockets.

Dentre as informações transmitidas na camada de transporte via socket, estão especificados alguns detalhes, como por exemplo o remetente da informação, o destinatário, onde os dados deverão ser armazenados, tipo de dado, ou mesmo detectar se trata de um cliente ou servidor.

Existem dois tipos de serviços de transporte via socket: O confiável, que é orientado a cadeia de bytes e é efetuado pelo protocolo TCP e o não confiável, que são datagramas implementados pelo protocolo UDP.

Todos os sockets possuem um endereço único na internet, este endereço é formado pelo IP e pelo número da porta. Devido à dimensão da internet, é inviável para a máquina saber o endereço de todas as outras. Como solução, foi criado o protocolo DNS (Domain Name Service). Este protocolo possui como função efetuar a tradução dos nomes ou endereços de alto nível das máquinas para o seu respectivo número IP. Assim, ao informar o endereço de um socket de um servidor, não é informado diretamente o seu número IP, mas um nome mais fácil de se recordar, assim o DNS efetua a tradução para o endereço IP do destinatário.

1. **BERKELEY**

O BSD (Berkeley Software Distribution) é um código fonte desenvolvido pela área de pesquisa da AT&T e distribuído pela Universidade da Califórnia, que originalmente era uma extensão do UNIX (Primeiro Sistema Operacional a introduzir o conceito de suporte a multitarefas e multiusuários, considerado o pai dos Sistemas Operacionais). Foi a primeira extensão do Unix a ter bibliotecas que suportam os protocolos de internet, possibilitando a comunicação em rede.

Grande parte dos SO’s atuais utilizam conceitos do UNIX, incluindo as interfaces de protocolos TCP/IP para comunicação em rede utilizados atualmente.

A partir daí, tais sockets ficaram conhecidos como *Sockets de Berkeley.*

1. **PROGRAMAÇÃO DOS SOCKETS**
   1. **Programação de aplicações TCP**

O primeiro passo é contatar o servidor, e para isso o processo servidor já deve estar executando o programa, antes mesmo de já ter criado o socket (porta) que aceita a conexão do cliente. O cliente cria um socket TCP local ao contatar o servidor, o endereço IP e a porta do processo. Quando o servidor é contatado, é criado um socket para se comunicar com o cliente permitindo a liberação do socket de “boas vindas” para que possa ser contatado por outro clientes.

Do ponto de vista da aplicação, a conexão TCP é um fluxo contínuo de dados, a mensagem é fragmentada em pacotes e não há duplicação. A conexão é ponto-a-ponto, ou seja, um remetente e um destinatário.

No início do programa deverá ser importado as bibliotecas que contém as classes utilizadas em uma aplicação com Socket (java.net.\*), e as classes de recepção de informação do teclado, Socket do cliente ou servidor (java.io.\*).

Para exemplificar a ilustração mencionada acima, iremos demonstrar e explicar as linhas de códigos utilizando a linguagem Java. Iremos iniciar a aplicação no lado Servidor.

* import java.io.\*;
* import java.net.\*;

Agora efetuaremos a criação de um objeto *welcomeSocket* que é o socket do lado servidor, o parâmetro *numero\_da\_porta* deverá ser substituído pelo número da porta que a aplicação cliente usará na conexão.

* ServerSocket welcomeSocket = new
* ServerSocket(numero\_da\_porta);

Normalmente o programa servidor fica executando por tempo indeterminado, portanto colocamos o restante das instruções dentro de um loop infinito.

* while(true) {

...

}

A instrução a seguir cria um objeto *connectionSocket* do tipo Socket quando o cliente conectar no servidor. O TCP ficará responsável em criar uma conexão virtual direta entre os sockets servidor/cliente de forma em que todos os bytes serão entregues ao servidor na ordem correta.

* Socket connectionSocket = welcomeSocket.accept();

No caso do envio de um objeto do tipo String do cliente utilizamos as seguintes instruções.

* BufferedReader infoDoCliente = new BufferedReader(new InputStreamReader(connectionSocket.getInputStream());
* String mensagemDoCliente = infoDoCliente.readLine();

Depois de efetuar o processamento das informações recebidas do cliente queremos enviar outro objeto (*mensagem\_para\_cliente)* do tipo String de volta para o cliente confirmando o recebimento.

* DataOutputStream infoParaCliente = new DataOutputStream(connectionSocket.getOutputStream());
* infoParaCliente.writeBytes(mensagem\_para\_cliente);

Este conjunto de comando encerra o programa do lado Servidor, agora iremos analisar o programa cliente. Como o programa do lado Servidor, o código inicia efetuando a importação das bibliotecas que contém as classes de sockets e envio de informações.

* import java.io.\*;
* import java.net.\*;

Inicialmente criaremos o socket que conectará com o servidor, o primeiro parâmetro informado deve ser o nome do servidor, por exemplo, 127.0.0.1 se a aplicação servidor estiver executando no mesmo computador, o segundo parâmetro é o número da porta que é informado ao socket servidor.

* Socket clientSocket = new Socket("nome\_do\_servidor", numero\_da\_porta);

Após criar o socket cliente deve-se criar os objetos de cadeia que serão ligados ao socket. O objeto *infoParaServidor* será a cadeia que efetuará o envio das informações para o servidor e o objeto *infoDoServidor* será o receptor das informações do servidor.

* DataOutputStream infoParaServidor = new DataOutputStream(clientSocket.getOutputStream());
* BufferedReader infoDoServidor = new BufferedReader(new InputStreamReader(client.getInputStream()));

Utilizemos o mesmo caso do servidor e enviaremos *(mensagem\_do\_cliente)* e receberemos *(mensagem\_para\_cliente)* dois objetos do tipo String.

* infoParaServidor.writeBytes(mensagem\_do\_cliente);
* String mensagem\_para\_cliente = infoDoServidor.readLine();

Agora podemos fechar o socket e a conexão TCP.

* clientSocket.close();
  1. **Programação de aplicações UDP**

Os sockets UDP são mais rápidos que TCP e são mais simples, porém menos confiáveis. Em UDP não é necessário efetuar a abertura de uma conexão, deste modo a comunicação ocorre apenas com o envio da mensagem sem a necessidade de um canal pré-estabelecido entre os processos. Uma mensagem é um datagrama que é composto de um remetente (sender), um receptor (receiver) e a mensagem (content). No datagrama é informado o IP do destino e a porta de conexão, além do conjunto de bytes enviados.

Com a criação do pacote, ele é colocado na rede através do socket e o receptor deverá efetuar a abertura do pacote para retirar as informações. Como o UDP não possui conexão, não há nenhuma garantia que o pacote realmente chegará ao seu destino.

Segue abaixo um exemplo de programa de socket com UDP codificado em Java. Iremos novamente iniciar pelo servidor.

* DatagramSocket serverSocket = new
* DatagramSocket(9876);

Ao contrário do TCP, efetuaremos a criação de um objeto do tipo *DatagramSocket* e não *ServerSocket.* Todos dados enviados e recebidos pelo servidor passarão pelo socket do tipo *DatagramSocket*, não se faz necessário criar um objeto para cada nova conexão. Todas as conexões dos clientes passarão pelo único socket *serverSocket*, isto não é um problema, pois a cada recebimento de pacote, o servidor envia uma resposta, ficando assim livre novamente, pois como mencionado inicialmente não existe conexão entre ambos. O socket fica na escuta pela porta 9876.

* while (true) { ... }

Loop infinito, onde o servidor fica esperando o recebimento de pacotes.

* DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
* serverSocket.receive(receivePacket);

A alocação de memória onde serão salvos os dados de entrada recebidos pelo servidor fica responsável pelo objeto *receivePacket*.

* String sentence = new String(receivePacket.getData());
* InetAddress IPAdress = receivePacket.getAddress();
* int port = receivePacket.getPort();

As linhas acima desmontam os pacotes retirando as informações necessárias. O objeto *sentence* recebe os dados. *IPAdress* é responsável por armazenar o número de IP do cliente para posteriormente enviar a mensagem de resposta juntamente com o número da porta *port,* após o processamento do pacote.

Do lado cliente possuímos a criação do socket no qual será enviado o pacote.

* DatagramSocket clientSocket = new DatagramSocket();

Percebe-se que a única diferença entre o cliente e o servidor é que no servidor é necessário informar o número da porta. Este comando não efetua a criação de conexão e muito menos contata o servidor, e por este motivo não se faz necessário especificar o endereço do servidor e sua porta.

* InetAddress IPAddress = InetAddress.getByName(“hostname”);

Acima criamos um objeto para armazenar o IP do servidor, *“hostname”* é o nome (apelido) do servidor. O método *getByName()* efetua uma consulta no servidor DNS para descobrir qual o número de IP do servidor.

Os dois vetores abaixo armazenam os dados a serem enviados e recebidos.

* byte[] sendData new byte[1024];
* byte[] receiveData = new byte[1024];

A seguir cria-se o pacote que será enviado, nele é gravado os dados em si (sendData), o tamanho dos dados (sendData.length), o número de IP do servidor (IPAdress), e o número da porta que o servidor está “ouvindo” (9876).

* DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAdress, 9876);

A próxima linha efetua o envio do pacote através do método *send* do socket *clientSocket*. Após enviado, o cliente fica no aguardo da resposta do servidor.

* clientSocket.send(sendPacket);

Quando o pacote é recebido ele é armazenado em *receivePacket.*

* clientSocket.receive(receivePacket);

1. **BIBLIOGRAFIA**

<https://pt.wikibooks.org/wiki/Redes_de_computadores/Programa%C3%A7%C3%A3o_com_sockets#Programa%C3%A7%C3%A3o_de_sockets_com_UDP>

https://pplware.sapo.pt/tutoriais/networking/redes-sabe-o-que-sao-sockets-de-comunicacao-parte-i/

<https://www.tecmundo.com.br/internet/982-o-que-e-cliente-servidor-.htm>

<https://pt.wikipedia.org/wiki/Berkeley_Software_Distribution#Soquetes_Berkeley>

<https://www.freebsd.org/doc/pt_BR.ISO8859-1/articles/explaining-bsd/article.html>

<https://www.tecmundo.com.br/macos/10556-unix-o-pai-de-todos-os-sistemas-operacionais.htm>

<https://en.wikipedia.org/wiki/Jim_Kurose>