Desenvolvimento de uma aplicação cliente servidor em nodeJS

Igor Mateus Giacobe, Thales Eduardo Muller, Eduardo Brum Meurer, João Carlos Ilha Baierle

Departamento de Computação

Curso de Engenharia de Computação

Universidade de Santa Cruz do Sul (UNISC)

Av. Independência, 2293 - 96815-900 – Santa Cruz do Sul, RS

igorg,joaobaierle,ebmeurer,thalesem(@mx2.unisc.br)

Abstract. *This article develops the theoretical basis of an application project of client server utilizing sockets in node JS*

Resumo. *Este artigo desenvolve a base teórica do projeto de aplicação cliente servidor utilizando sockets em node JS.*

1. Introdução e Contexto

TCP e UDP são protocolos da camada de transporte e, quando precisamos de confiabilidade no transporte do dado, usamos o protocolo *IP* associado ao *TCP* (que garante a entrega das informações). Quando priorizamos mais velocidade e menos controle, associamos o protocolo *IP* ao *UDP* (tráfego de voz e vídeo são bons exemplos onde o UDP teria boa aplicabilidade, ademais, perdendo um ou outro pacote, não interfere totalmente no todo, permanecendo inteligível).

Muito do que fazemos no dia a dia faz uso de sockets. O nosso navegador utiliza sockets para requisitar as páginas; quando acessamos o nosso servidor pelo protocolo de aplicação SSH também estamos abrindo e utilizando um socket. O Socket provê a comunicação entre duas pontas (fonte e destino) – também conhecido como two-way communication – entre dois processos que estejam na mesma máquina (Unix Socket) ou na rede (TCP/IP Sockets). Na rede, a representação de um socket se dá por ip:porta, por exemplo: *127.0.0.1:4477* (IPv4). Um socket que usa rede é um Socket TCP/IP.

1.1. Sockets

Existem quatro tipos de sockets: *Stream Sockets*, *Datagram Sockets*, *Raw Sockets* e *Sequenced Packet Sockets* sendo que os dois primeiros são os mais comuns e utilizados.

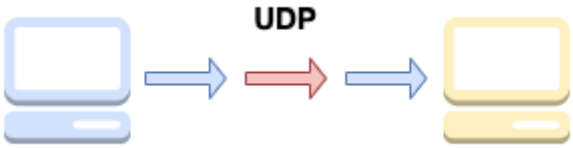
- **Stream Sockets (SOCK_STREAM)**:Quando necessitamos de uma troca confiável de informações, isto é, quando é necessária a confirmação de recebimento da mensagem enviada, devemos utilizar o protocolo **TCP** (Transmission Control Protocol). Este protocolo estabelece uma conexão entre dois pontos interligados. Por exemplo, uma mensagem enviada de um host (o termo host representa uma máquina conectada na rede) 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). 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 muito o processo de leitura e envio de dados pela rede.

- **Datagram Sockets (SOCK_DGRAM)**: Os datagram sockets são mensagens que podem ser enviadas pela rede quando não existe a necessidade de confirmação de entrega, de tempo de entrega e nem mesmo garantia de conteúdo. Datagramas são úteis em aplicações que não necessitam do estabelecimento de uma conexão para o envio da mensagem. Um bom exemplo do seu uso é o envio de mensagens em broadcast para clientes de uma rede (o servidor pode enviar um datagrama para todos os clientes avisando que irá reiniciar, por exemplo).

Observe que a comunicação no TCP se dá nas duas pontas:

- Orientado à conexão (só transmite dados se uma conexão for estabelecida depois de um *Three-way Handshake*);
- É Full-duplex, ou seja, permite que as duas máquinas envolvidas transmitam e recebam ao mesmo tempo;
- Garante a entrega, sequência (os dados são entregues de forma ordenada), não duplicação e não corrompimento;
- Automaticamente divide as informações em pequenos pacotes;
- Garante equilíbrio no envio dos dados (para não causar “sobrecarga” na comunicação)

No UDP, a comunicação se dá em uma ponta e, se algum segmento falha, isso é ignorado e o fluxo continua:



- Diferente do TCP ele **não** é orientado à conexão;

- Não é confiável como o TCP, ele não garante a entrega completa dos dados;
- É preciso que dividamos manualmente os dados em datagramas (entidades de dados);
- Não garante a sequência da entrega, portanto, os dados podem chegar em uma ordem aleatória;

2.Funcionamento

Utilizando a linguagem javascript desenvolvemos a aplicação que funciona desse jeito:

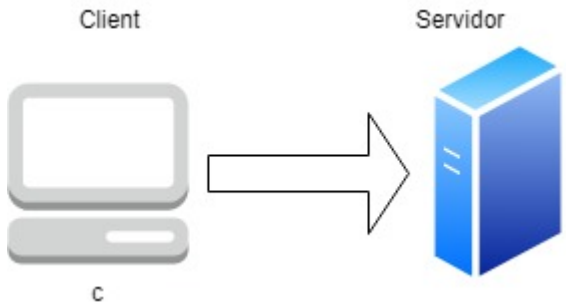


Figura 3: Arquitetura Cliente Servidor

Ao ser iniciado o servidor fica ouvindo na porta 8899 á espera de conexões de clientes;

- O cliente solicita uma conexão ao servidor;
- O servidor exibe uma mensagem na tela com o endereço IP do cliente conectado;
- O servidor aceita a con exão e envia um objeto Date ao cliente;
- O cliente recebe o objeto do servidor e faz o cast necessário, em seguida exibe na tela as informações de data;
- O servidor encerra a conexão.

3. Conclusão

Este trabalho possibilitou compreender o funcionamento de sockets na comunicação TCP e UDP. Ficou claro que sockets estão presentes em praticamente todos tipos de aplicações que utilizamos diariamente e embora sejam abstraídos da visão do usuário, as suas diversas maneiras de implementação são essenciais para o administrador manter o controle sobre as conexões cliente e servidor, assim como foi simulado neste projeto.

4. Referências

TREINAWEB - Uma introdução a TCP, UDP e Sockets, disponível em:

<https://www.treinaweb.com.br/blog/uma-introducao-a-tcp-udp-e-sockets/> Acessado em novembro 2019