Especificação do projeto 1 **API sockets.**
Redes de computadores 2020.1
Prof. Filipe

Introdução

Este projeto avaliará o domínio inicial da API socket. A linguagem a ser utilizada no trabalho é C. O trabalho é, no máximo, em dupla.

1 Visão geral

Você criará uma comunicação entre uma origem e um destino. Ela será mediada por um comutador intermediário. Comutador é um nome para a classe de dispositivos responsáveis por comunicar hospedeiros, ou seja, estabelecer uma comunicação indireta. Tanto switches, quanto roteadores são comutadores. O cenário da comunicação é apresentado na figura abaixo.

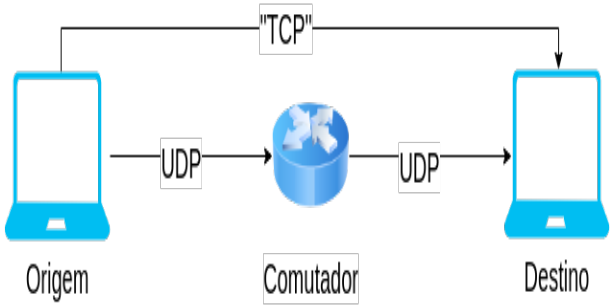

Figura 1: Cenário.

Perceba que você terá de criar três programas: origem, comutador e destino. A origem enviará os dados para o comutador, que repassará ao destino. Para isso, você emulará uma conexão TCP sobre uma comunicação UDP. Ou seja, você replicará as funcionalidades do TCP, tanto para cliente, quanto para servidor. Porém, na hora que send() e recv() forem chamados, eles chamarão as funcionalidades de comunicação UDP, sento() e recvfrom().

2 Criação do socket

Em cada programa haverá a criação de sockets. Porém, na origem e no destino a criação será uma chamada à função meuSocket(). Você programará essa função que emulará toda a criação de um socket TCP. A assinatura da função meusocket é: int meuSocket(bool destino).

2.1 Origem

Suponha que você tenha chamado a função meuSocket() com parâmetro falso, você criou o socket para o cliente comunicar com o servidor. O que essa função fará é a criação de um socket UDP. Essa criação é parecida com o que já vimos em sala, a diferença está no componente ai_socktype da struct addrinfo que passaremos por parâmetro à função getaddrinfo(), o valor de ai_socktype deve ser SOCK_DGRAM.

Após o retorno de getaddrinfo ocorrer, procederemos para a criação do socket fazendo uma chamada à função socket(), igual a que já fizemos em aula. O retorno dela deve ser um identificador de arquivo que corresponde ao socket UDP criado.

Toda comunicação com o destino do dado será via esse socket criado. Ele enviará o dado para o comutador, que receberá e repassará ao destino. Caso o destino envie algo para a origem, esse socket será utilizado para recepção do que será enviado pelo destino.

2.2 Comutador

No caso do comutador, você não precisa implementar meuSocket(). Você usará dois socket UDP, um para comunicação com origem e outro para comunicação com destino. O socket que fará a comunicação do comutador com o destino é a réplica do socket criado pela origem na subseção anterior. A diferença é a porta usada na comunicação, que deve ser a porta de comunicação com o destino.

Já o socket a ser utilizado para comunicação com origem é um pouco diferente. Você usará a função getaddrinfo com primeiro parâmetro igual a NULL. Isso deve-se ao fato que o comutador esperará os dados da origem, então ele será um servidor. O segundo parâmetro deve ser a porta de comunicação com a origem.

2.3 Destino

Suponha que você tenha chamado a função meuSocket() com parâmetro true, você criou o socket no destino. A função criará um socket real, que replica o comportamento do segundo socket do comutador. Você usará a função getaddrinfo com primeiro parâmetro igual a NULL. Isso deve-se ao fato que o destino esperará os dados do comutador, então ele será um servidor. O segundo parâmetro deve ser a porta de comunicação com o comutador.

3 Reserva de recursos no SO

Vimos que depois que o socket é criado, o sistema operacional precisa ser avisado para reservar ao socket o endereço de comunicação (o par: ip, porta). Ele faz isso quando nosso programa chama a função bind(). Você deve executar essa função após a execução da chamada a socket(). Porém, ela deve ser chamada somente nos casos de criação dos sockets servidores.

4 Início da escuta

No TCP, a chamada listen() inicia a escuta de conexões por um servidor. No nosso caso, você implementará uma função meuListen() que imprimirá no terminal a mensagem "Iniciando escuta". Ela fará apenas isso, já que em sockets UDP não estabelecemos conexão, não há listen(). Perceba que só há implementação dela no destino.

5 Requisição de conexão

Para simular um estabelecimento de conexão, a origem enviará pelo socket UDP criado a string "SYN". Ela viajará via UDP para o comutador. O comutador receberá essa string e repassará ao destino manipulando os sockets criados para comunicação. O envio dessa string deve ocorrer quando o cliente executar a função meuConnect().

Durante sua execução, ela deve imprimir no terminal: “Pedindo conexão.”, ao enviar “SYN” ao destino. Caso a conexão tenha sido estabelecida, ou seja, tenha recebido a string “SYNACK” do destino, ela deve imprimir: “Conexão estabelecida.”.

6 Aceitação de conexão

Para simular um estabelecimento de conexão, o destino deve implementar a função meuAccept(), que escuta pelo socket UDP, logo após a chamada a meuListen(), a recepção da mensagem “SYN” da origem. Ou seja, ela imprimirá a mensagem “Esperando conexão.” e ao receber a string “SYN” imprimirá “Conexão pedida” e enviará a string “SYNACK”, imprimindo no terminal a mensagem “Conexão aceita.”.

7 Envio de dados

Depois da conexão estabelecida, a origem pode enviar dados ao destino. O dado é simples, uma string “TESTE”. Esse envio será através da função meuSend(), que chamará a função de envio do UDP, sendto(). meuSend() deve ter três parâmetros: o socket, a mensagem e o tamanho da mensagem. Já a função sendto necessita de seis. Portanto, você terá de armazenar em variáveis os outros parâmetros necessários à comunicação, para que possa utilizar em sendto.

O destino, ao receber a string “TESTE” deve responder com uma string “TESTADO”. Ele deve usar também a sua implementação de meuSend().

8 Recepção de dados

O destino deve receber os dados através da chamada de função meuRecv(). Ela deverá ter três parâmetros: socket, o buffer e o tamanho do buffer. Para realizar a recepção via UDP, ela terá de chamar a função recvfrom, que tem 6 parâmetros. A implementação de meuRecv() deve contemplar esses parâmetros necessários.

O destino, após receber “TESTE” com meuRecv, deve enviar “TESTADO” com meuSend() à origem. A origem deve, após enviar “TESTE” ao destino, esperar pela recepção de “TESTADO” chamando sua própria implementação de meuRecv().

9 Papel do comutador no envio e recepção de dados

O comutador não deve implementar qualquer função de envio e recepção de dados. Isso é para os hospedeiros. Ele deve limitar-se a receber dados por um socket com recvfrom e enviar por outro com sendto.

10 Evolução da comunicação

Inicie com a execução do destino, depois inicie o comutador e por fim a origem. Essa sequência deve ser seguida porque o destino espera conexão do comutador, que ao ser executado estabelece conexão com o destino e depois espera conexão da origem. A origem é executada e conecta-se ao comutador.

Após isso, o destino ficará pronto para “aceitar” conexões. A origem iniciará o procedimento de conexão enviando a mensagem “SYN” ao destino pelo socket UDP que a conecta ao comutador. O comutador receberá por um socket via recvfrom essa string e repassará com o outro socket via sendto ao destino. Já o

destino receberá com recvfrom, chamado dentro de meuAccept() e enviará "SYNACK" à origem, passando pelo comutador. O comutador receberá pelo socket que comunica com o destino "SYNACK", utilizando recvfrom, e enviará pelo socket que comunica com origem utilizando sendto(). Após essa troca inicial a conexão foi estabelecida.

Com a conexão estabelecida, a origem envia via meuSend() a mensagem "TESTE", que é recebida pelo comutador via socket UDP, através da chamada recvfrom(), e envia para o destino utilizando o outro socket de comunicação UDP via sendto(). Ao chegar no destino, "TESTE" será recebida com a chamada meuRecv(), que chamará recvfrom para realizar a recepção. Daí, o destino envia "TESTADO" com sendto() para a origem, e faz caminho e operações semelhantes até que seja recebido pela origem com a chamada a meuRecv() dela.

11 Parâmetros da comunicação

Os parâmetros abaixo devem ser utilizados para criação dos sockets:

- Porta do comutador para comunicação com origem: 5001.
- Porta do destino para comunicação com comutador: 5002.
- Endereço: 127.0.0.1 ou localhost.