Terça, 24/02/2009

Programação

Tutorial de Sockets - Parte I

Por: Frederico Perim

Extraído de artigos publicados por Brian Beej Hall

O que é um Socket?

Você já deve ter ouvido falar sobre Sockets e talvez esteja imaginando do que se trata exatamente.

Bem, resumindo: através de Sockets nos comunicamos com outros programas usando discritores de arquivos Unix padrão. O que? Tudo bem. Você deve ter ouvido algum guru em Unix dizer,"Tudo no Unix é um arquivo!". O que esta pessoa esta querendo dizer, na verdade, é o fato de que quando sistemas Unix realizam qualquer tipo de E/S(Entrada/Saída), eles o fazem lendo ou escrevendo através de um descritor de arquivo.Um descritor de arquivo é simplesmente um inteiro associado a um arquivo aberto.Mas(e aqui está o detalhe), este arquivo pode ser uma conexão de rede,um FIFO, um pipe, um terminal, um arquivo no disco, ou qualquer outra coisa.Tudo no Unix(Linux) é um arquivo!.Assim quando você quiser se comunicar com outro programa na internet você vai usar um descritor de arquivo.Apartir dai, você realiza chamadas de socket send() e recv().

"Mas,peraí!", você deve estar pensando. "Se é um descritor de arquivo, porque não usar as chamadas read() e write() para se comunicar através de um Socket?"a resposta é,"Você pode!"a resposta completa é,"Você pode, mas send() e recv() oferecem melhor controle sobre a transmissão de dados".

"Tá bom, e daí?"Que tal isto: existem vários tipos de Sockets.Tem o DARPA(Sockets de Internet), Sockets Unix, CCITT X.25 e provavelmente outros dependendo da sua versão Unix. Este artigo abordará somente o primeiro: Sockets de Internet.

Dois Tipos de Sockets

O que é isto? Existem dois tipos de Sockets de Internet? Sim. Bem,na verdade não.Tem mais, mas não quero assustá-lo. Irei abordar apenas dois tipos.O primeiro é o "Stream Sockets", o outro "Datagram Sockets", que apartir de agora serão chamados de "SOCK_STREAM" e "SOCK_DGRAM", respectivamente. Os "Datagram Sockets" também são conhecidos como sockets sem conexão, embora você possa usar a chamada connect().

Os "Stream Sockets" são fluxos de comunicação confiáveis. Se você enviar 2 itens na ordem "1,2", eles irão chegar chegar na ordem "1,2" no outro extremo do link.Eles também são livres de erros.Qualquer erro que você encontrar é apenas ilusão da sua mente :-).

O que realmente utiliza um "Stream Socket"? Você já deve ter ouvido falar no programa Telnet.Pois bem, todos os caracters que você digita precisam chegar na mesma ordem em que você os digitou logo este aplicativo consegue isto através de "Stream Sockets".Além disso os browsers, que utilizam o protocolo HTTP, usam "Stream Sockets" para receber páginas.

Como os "Stream Sockets" conseguem este alto grau de qualidade de transmissão de dados? Eles utilizam um protocolo chamado "Protocolo de Controle de Transmissão", mais conhecido como "TCP". TCP assegura que os dados chegarão sequencialmente e livres de erros. Você deve ter ouvido falar que TCP é a melhor metade do TCP/IP, onde

IP significa Protocolo de Internet. IP cuida basicamente do roteamento e não é responsável pela integridade dos dados.

Legal. E os "Datagram Sockets"? Porque são conhecidos como sem conexão? Porque não são confiáveis? Bem, alguns fatos: Se você enviar um datagrama, ele pode chegar. Ele pode chegar fora de ordem. Se chegar , os dados contidos no pacote estarão livres de erros.

"Datagram Sockets" também usam o IP para roteamento, mas eles não utilizam TCP, e sim o "UDP".

Porque são sem conexão? Bem , basicamente devido ao fato de você não precisar manter uma conexão aberta como os "Stream Sockets". Você constrói um pacote, anexa um cabeçalho IP com informações de destino, e envia. Não precisa haver conexão. Ele são mais utilizados para transferências pacote por pacote de informações. Alguns aplicativos que utilizam UDP: tftp, bootp,etc.

"Chega!", você diz. "Como estes programas funcionam se um pacote pode se perder na rota?"Bem, meu caro amigo, cada um tem se próprio protocolo acima do UDP. Por exemplo, o protocolo tftp diz que para cada pacote que é enviado, o receptor tem que enviar um pacote de volta que diz, "Eu recebi!"(um pacote "ACK") Se o emissor do pacote original não receber uma resposta, digamos, em cinco segundos , ele irá retransmitir o pacote até que receba um "ACK". Este procedimento é muito importante quando você for implementar aplicações que utilizam "SOCK_DGRAM".

Alguma Teoria de Rede

Já que mencionei algumas camadas de protocolos, é hora de falar como as redes realmente funcionam, e mostrar alguns exemplos de como pacotes SOCK_DGRAM são construídos. Você pode pular esta parte se não estiver interessado.

É hora de aprender sobre Encapsulamento de Dados. Isto é muito importante. Basicamente, é isto: um pacote é criado, o pacote é empacotado("encapsulado") em um cabeçalho pelo primeiro protocolo (digamos, o protocolo TFTP), então tudo (cabeçalho TFTP incluído) é empacotado novamente pelo próximo protocolo (por exemplo, UDP), novamente pelo próximo (IP) , e então finalmente pelo último protocolo no hardware(camada física) , por exemplo Ethernet.

Quando outro computador receber o pacote, o hardware retira o cabeçalho Ethernet, o kernel retira os cabeçalhos IP e UDP, e o programa TFTP retira o cabeçalho TFTP, e finalmente se tem os dados.

Agora finalmente posso falar sobre o temível Modelo de Camadas de Rede. Este modelo descreve um sistema de funcionalidade de rede que tem muitas vantagens sobre outros modelos. Por exemplo, você pode escrever programas de sockets que não se importam como os dados são fisicamente transmitidos (serial, thin ETHERNET,AUI, seja lá o que for) porque programas nos níveis mais baixos tomam conta disto pra você.

O hardware de rede e a topologia são transparentes para o programador.

Bem sem mais enrolação, irei apresentar o modelo completo:

- Camada de Aplicação (Application Layer)
- Camada de Transporte (TCP,UDP)
- Camada de Internet (IP)
- Camada de Acesso de Rede (Ethernet, ATM)

Nesse momento você provavelmente deve estar imaginando como estas camadas correspondem ao encapsulamento dos dados.

Veja quanto trabalho para construir um simples pacote? E você ainda tem que digitar os cabeçalhos do pacote usando "cat"!. Brincadeirinha. Tudo que você tem que fazer para utilizar "Sockets Streams" é enviar (send()) os dados. Com relação aos "Sockets Datagram" é empacotar os pacotes num método de sua escolha e enviar(sendto()) os dados. O kernel constrói a Camada de Transporte e a Camada de Internet pra você e o hardware cuida das camadas mais baixas.

Bem aqui termina nossa breve introdução em teoria de redes. AH, esqueci de mencionar alguma coisa sobre roteamento: nada! Isso mesmo , não irei falar sobre isso. O roteador retira o cabeçalho IP, consulta a tabela de roteamento , etc,etc.

Próximo»

Enviar para um amigo Imprimir Índice de artigos