Comunicação entre dois computadores através de RAW Socket

Antonio Carlos Salzvedel Furtado Junior e Tiago Rodrigo Kepe
GRR20080946,GRR20084630

1 de dezembro de 2010

Universidade Federal do Paraná
CI058 Redes I Bacharelado em Ciência da Computação
For: Luiz Carlos Pessoa Albini

---

1 Funcionamento

Nossa rede funciona no esquema cliente-servidor. A máquina que estiver executando o servidor ficará escutando a rede, esperando por mensagens do cliente. Quando o cliente for executado, uma espécie de Shell estará disponível para o usuário. Todos os comandos definidos na especificação estarão disponíveis, são eles:

- ls [opções]
- cd
- lls [opções]
- lcd
- put
- get

2 Implementação

O envio de mensagens não é feito com janelas deslizantes, como definido na especificação. A nossa implementação baseia-se no "Para e espera".

Tratamos o caso em que uma mensagem é esperada, e por algum motivo ela não foi recebida imediatamente, seja por retirada do cabo ou outro motivo. A máquina que quiser receber a mensagem, seja cliente

ou servidor, deverá esperar 16 vezes por um determinado tempo (TIMEOUT). Para cada uma das tentativas falhas de recebimento, ela deverá enviar um Nack. No final da 16ª vez ela deve desistir de receber a mensagem. Quem estiver enviando, deve tentar enviar até receber um Ack de sua mensagem. O recebimento de resposta segue o mesmo modelo citado anteriormente, exceto que nenhum Nack é enviado, já que você entraria em um ciclo de recebimento de respostas de respostas.

O Timeout é verificado por uma função. Ela usa a função do sistema chamada select. Esta função monitora o descritor do nosso socket por um determinado período. Ela retorna positivo se alguma mensagem foi recebida durante a constante TIMEOUT definida, mesmo que seja lixo, e falso se durante este período nada foi recebido.

O campo tipo de mensagem abrange todos os tipos definidos na especificação. Além disso foram definidos tipos específicos para mensagens de erro, por motivo de simplicidade de implementação, são eles:

- TYPE_E1 : diretório inexistente;
- TYPE_E2 : falta de permissão;
- TYPE_E3 : espaço insuficiente;
- TYPE_E4 : arquivo inexistente.

Realizamos a paridade vertical de 16 bits. A abordagem de solução foi criar dois campos de 8 bits, chamados parity, ele é um vetor de char com duas posições. A string dos campos que a paridade considera é dividida em duas colunas, uma de posições pares e outra para posições ímpares. A posição 0 do vetor parity é reponsável por calcular a paridade da primeira coluna, enquanto a posição 1 do vetor cuida da segunda. Cada posição deste vetor fará um xor bit a bit de sua respectiva coluna. No final é feito um último xor entre as duas posições do vetor parity, para que a paridade de 16 bits caiba em um campo de 8 bits.

String: 00 11 00 11 01 01 10 00 10 10 10 10 11 10 11 00

P0 P1
00 11 00 11 | 01 01 10 00
10 10 10 10 | 11 10 11 00

3 Classes

Realizamos nosso trabalho em C++, então dividimos funções específicas para cada uma de nossas classes, são elas:

- Message: Esta classe é responsável por definir todos os detalhes de nossas mensagens. Ela possui funções para montagem de mensagem, gerador e conferidor de paridades e deve retornar campos específicos da mensagem.
- Socket: Ela é reponsável por abrir o socket de acordo com regras do RAW SOCKET. Ele possui funções simples para envio e recebimento de mensagens simples.

- Control: Esta classe cuida de detalhes de maior abstração da rede. Ela usa a classe Socket para enviar e receber múltiplas mensagens, ela também cuida de detalhes como o TIMEOUT da rede e da seqüêncialização de mensagens. Os detalhes do para e espera são definidos aqui também.
- Cliente: Ela define o comportamento do cliente. Basicamente, é criado um shell e funções relacionadas aos comandos de especificação. Ela usa a classe Control para cuidar do resto.
- Servidor: Assim como o cliente, ela define o comportamento do servidor. Também usa a classe Control. A sua diferença é não possuir um shell, e sim uma função para escutar na rede.