Redes de computadores 1 – Trabalho 1

Leonardo Becker de Oliveira – 20211779

Mensageiro Local

Introdução

A comunicação via raw socket, permite que dados sejam trocados entre duas máquinas através de um cabo de ethernet. Essa transmissão ocorre na camada de enlace seja ele no modelo OSI ou TCP/IP, fazendo com que mensagens sejam enviadas em mais baixo nível, sendo necessário o tratamento dos dados através de pacotes, onde cada parte do pacote é definido pelo pelo próprio desenvolvedor. Com isso, nesse trabalho será o desenvolvimento de um código em linguagem C que tem por objetivo permitir a troca de mensagens entre um cliente e um servidos utilizando raw socket.

Pacotes

O pacote de envio entre dois dispositivos, foi realizando o seguinte padrão:

-Marcador de início: O marcador de início tem por objetivo indicar que um pacote irá ser transmitido, ou seja, sua principal função é preparar o dispositivo para receber o pacote, cabendo a quem envia, inserir esta sequência de bits no início do pacote.

-Tipo: Nesse trabalho, permite-se que sejam enviados tanto mensagens quanto imagens, com isso, é necessário distinguir qual dos dois tipos de dado que está sendo enviado, além disso, o campo tipo também é responsável por indicar o pacote inicial e final de uma transmissão de dados e se o pacote foi bem recebido ou não (ACK ou NACK).

-Sequência: Trabalhado em 4 bits, a sequência tem por objetivo organizar o envio dos pacotes, quando se aplica o métodos de janela deslizante, permitindo que apenas 16 pacotes sejam enviados por vez, em sequência e de forma cíclica (quando chega no bit 1111 ele retorna ao 0000).

-Tamanho: O tamanho indica o tamanho do campo de dados que será enviado, com isso o recebedor sabe que qualquer outro tipo de dado que apareça no meio da sequência, são de fato parte da sequência, e não algum dado referente a outro campo (exemplo marcador de início de sequência).

-Dados: Os dados são justamente o que o dispositivo deseja enviar, ou seja, todos os campos anteriores servem como preparação do pacote, para os dados de fato, que é a mensagem/arquivo enviado pelo dispositivo, aparecer somente nessa parte do pacote.

-CRC-8: Esse byte é destinado ao resultado da verificação cíclica de redundância dos bytes reservados aos dados. Sua função é garantir que o dispositivo recebeu os dados corretamente, em que caso veia a divergir o resultado da operação de verificação entre os dois campos, o NACK é gerado, sendo solicitado o reenvio deste pacote.

Funcionamento

O programa funciona em um loop infinito, sendo quebrado somente quando o usuário deseja encerrar o programa, ou quando algum outro erro é encontrado.

Para cada repetição deste loop, é dada a possibilidade ao usuário enviar um mensagem ou receber uma mensagem de outro dispositivo.

Código determinante de leitura ou escrita:

```
state = terminalEntrada();
    // Recebe possivelmente algo
  if (state == NADA_RECEBIDO)
    state = NADA_RECEBIDO;
  // Recebe um pacote e verifica seu tipo
  if (packdevolve.tipo == 0x1D)
    state = RECEBE_MENSAGEM;
```

Lógica responsável por organizar os dados para envio:

```
if ((state == ENVIA_MENSAGEM) || (state == ENVIA_ARQUIVO)){
        if (state == ENVIA_MENSAGEM){
          // Código para leitura de arquivo de texto
 }

        if (state == ENVIA_ARQUIVO){
          //Código para leitura de arquivo de mídia
}
}
```

Caso a opção de escrever uma mensagem seja selecionado, é chamada a função que recria o terminal vim, permitindo assim que a mensagem seja escrita. Os dados escritos no buffer, são salvos em um arquivo de nome genérico “msg”, para que na função principal ele volte a ser lido. Se os dados escritos no arquivo sejam um comando (encerrar ou enviar mídia), a função principal faz a leitura da imagem no formato leitura de bits, caso contrário faz a leitura em modo normal.

Finalizado a leitura, o envio dos pacotes segue o método de janela deslizante, onde no máximo 16 pacotes são enviados por vez, e para cada “ack” ou “nack” recebido durante o envio, a função atualiza ponteiro é chamada, para que mantenha a janela de acordo com o tamanho preestabelecido. Além disso, cabe a esta parte preencher os pacotes para que contenha todas as informações descritas no tópico pacotes. Segue lógica a seguir:

Lógica para envio de pacotes:

```
while (1)
        {
                // Atualiza os ponteiros
          if (atualiza)
          {
            atualizaPonteiros(&ack, &pontsinal, &pontsinalant, &pontlocal);
            atualiza = 0;
          }
          if ((pontlocal - pontsinal) < TAM_JANELA)
          {
            // Enquanto não tiver enviado todos os pacotes, ele continua enpacotando de acordo com o pontLocal
            if (pontlocal < sequencia)
            {
                // Preenchimento de pacote com os dados, seguido por um leitura, e adiciona-se 1 ao ponteiro local, inidicando que mais um pacote foi enviado
```

```
                int bytes_sent = sendto(soquete, &pacote, sizeof(pacote), 0, (struct sockaddr *)&endereco, sizeof(endereco));

                pontlocal++;
            }
            // Envia pacote que sinaliza o fim da sequência e sai do while
            if ((pontlocal == sequencia) && (ack == 1))
            {
                //Preenche o pacote com o tipo final, e envia o mesmo

        int bytes_sent = sendto(soquete, &pacote, sizeof(pacote), 0, (struct sockaddr *)&endereco, sizeof(endereco));
            }
          }
          // Recebe mensagem de ack ou nack
                int msg = recvfrom(soquete, &packdevolve, sizeof(packdevolve), 0, (struct sockaddr *)&sndr_addr, &addr_len);
          analisaDevolucao(packdevolve, pontsinalant, &pontsinal, &ack);
        }
    }
```

Caso entre em modo de leitura de mensagem, é necessário verificar primeiro o tipo do pacote enviando, em que nele estará descrito se é uma mensagem ou arquivo, sabendo disso, o recebedor tem por objetivo ler os próximos pacotes salvando os dados da maneira correta. Segue lógica a seguir:

Lógica para recebimento de pacotes:

```
if (state == RECEBE_MENSAGEM)
    {
        // Enquanto o pacote for diferente do final
      while (pacote.tipo != 0x0F)
      {
        // Recebe a mensagem
        msg = recvfrom(soquete, &pacote, sizeof(pacote), 0, (struct sockaddr *)&sndr_addr, &addr_len);
        else
        {
                // Deixa de acordo com o tipo da transferência final
          if ((pacote.tipo == 0x01) || (pacote.tipo == 0x10))
          {

            if (pacote.tipo == 0x01)
              stateAuxiliar = RECEBE_MENSAGEM;
            if (pacote.tipo == 0x10)
              stateAuxiliar = RECEBE_ARQUIVO;
          }
                // Retorna um pacote com ACK ou NACK e sua sequência
            int bytes_sent = sendto(soquete, &packdevolve, sizeof(packdevolve), 0, (struct sockaddr *)&endereco, sizeof(endereco));
        if (stateAuxiliar == RECEBE_ARQUIVO)
          // Preenche o arquivo final de acordo com o seu tipo arquivo
        if (stateAuxiliar == RECEBE_MENSAGEM)
        {
          //Preenche o arquivo final com o seu tipo mensagem
        }
    }
```

Trabalho Completo

Acesse https://github.com/LeonardooBecker/redesComputadores1 para obter acesso ao desenvolvimento do código e suas respetivas funções

Conclusão

Conclui-se portanto, que para uma boa comunicação entre dois dispositivos, é necessário seguir uma sequência de passos, a fim de que ambos consigam trabalhar em sincronia, permitindo assim a troca de mensagem, sem que nenhum se sobressaia. Além disso, é importante aplicar as funções de detecção de erro e janelas deslizantes corretamente, pois ambas são responsáveis pelo prosseguimento do envio da forma correta e ágil.