INSTITUTO FEDERAL
Santa Catarina

Projeto 1:

um protocolo de comunicação

Curso: Engenharia de Telecomunicações
Disciplina: PTC29008 – Projeto de Protocolos
Professor: Marcelo Sobral

Alunos
Marina Souza
Renan Rodolfo da Silva

12 de maio de 2019

1 Introdução

O sistema simula um protocolo para a camada de enlace, cujo o objetivo é possibilitar a troca de informações ponto a ponto através da rede, fazendo o uso de um transceiver RF APC 220 conectado na porta USB de cada máquina. Como requisito deve se basear no protocolo PPP e deve oferecer uma tratativa para possíveis erros em quadros, controle de sequência, garantia de entrega e controle de sessão.

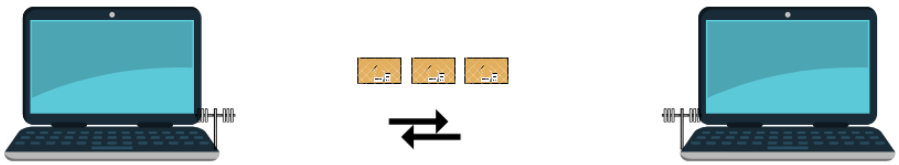

Figura 1: Diagrama do sistema

1.1 Descrição do sistema

Para administrar a geração dos eventos gerados pelo sistema, utilizamos a classe poller implementada pelo professor. A classe poller basicamente monitora os eventos adicionados pelo Envia.py e recebe.py e assim que ocorre o evento adicionado (entrada de texto via teclado ou recebimento de pacotes via interface tun) ele envia a informação para o método do objeto correspondente.

A primeira camada do protocolo implementada foi o enquadramento (Framming.py), essa camada é responsável por aguardar o recebimento dos bits da camada física, validá-los e notificar a camada superior (ARQ) quando receber um quadro completo ou descartar os bits recebidos caso ocorra um timeout. No projeto ela foi desenvolvida no formato de uma classe e nomeada como “Framming.py” . A classe após receber um evento identificado pelo poller, invocará o método handlefsm() que implementa em um ciclo de estados que irá alternar entre estado-ocioso, estado-receptivo e estado-escape. Enquanto permanece no estado receptivo, o framming itera os bits que recebe pela porta serial em busca do byte delimitador “7E” que indica o inicio e o fim de um quadro. Quando é identificado o primeiro byte delimitador, o sistema armazena os bytes seguintes em um buffer até identificar o segundo para então verificar seu valor de CRC e enviá-lo

à camada seguinte pelo método notifyLayer(). Caso “estoure” o tempo delimitado como timeout, antes que o framming receba seu segundo byte delimitador, o sistema volta para o estado ocioso e descartar o buffer armazenado. O terceiro estado dessa classe, estado-escape tratará o caso de a mensagem conter um byte delimitador no corpo da mensagem transmitida, neste caso, o byte deve ser precedido do caractere “r” e aplicada a operação lógica xor 20H entre os bytes.

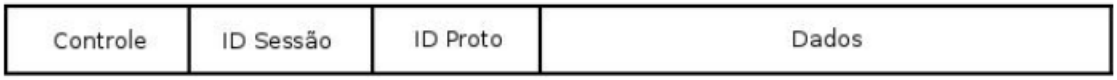

Figura 2: Formato do quadro na camada ARQ

A classe ARQ irá analisar se o quadro recebido da camada inferior é uma mensagem de confirmação (ACK) ou é um quadro contendo a informação ”útil”. Caso seja identificada uma mensagem de confirmação, ou seja se o primeiro bit do quadro for 0x80 (ACK para o número de sequência 0) ou 0x88 (ACK para o número de sequência 1), entende-se que a mensagem enviada foi recebida corretamente no destinatário. O número de sequência é aplicado para que o sistema identifique que tipo de mensagem ele deve receber, ou seja, se o transmissor envia um dado com o número de sequência 0 ele deve aguardar um ACK que também possua o número de sequência 0 e esse valor alterna entre as duas aplicações durante a comunicação. Se o quadro recebido for identificado como uma mensagem de dado, essa camada irá enviar a informação para a camada superior.

A classe ARQ também implementa o controle de acesso ao meio, técnica necessária para evitar um conflito de pacotes que pode acontecer quando duas mensagens são transmitidas ao mesmo tempo. Para reduzir essa possibilidade, foi implementado um método de acesso baseado na técnica ALOHA, onde geramos um valor aleatório entre 1 e 7 (em segundos) para que o sistema aguarde, antes de enviar uma mensagem. Também cabe à camada ARQ encapsular as flags de ACK ou dado e a ID da sessão, quando a mensagem vem da camada superior, quando a aplicação gerar uma mensagem para transmissão via interface serial.

A camada superior à camada ARQ é a classe responsável pelo gerenciamento de sessão, nomeada no projeto como sessionmanage.py. Essa classe define seu estado atual de acordo com o número do dado de controle recebido pela mensagem.

Valor	Descrição
0	CR (Connect request)
1	CC (Connect confirm)
2	KR (Keep Alive Request)
3	KC (Keep Alive Confirm)
4	DR (Disconnect Request)
5	DC (Disconnect Confirm)
7	CA (Connect Acknowledge)

Figura 3: Numeração das mensagens de gerenciamento

Estados da classe de gerenciamento de sessão:

- CON Ao identificar que recebeu um pedido de conexão, o método mock adiciona o IDProtoSession ao payload, em seguida envia um quadro com o dado de controle 2 (Keep Alive Request) e atualiza o estado para CHECK.

- DISC: desconectado Estado em que o protocolo está desconectado, caso receba uma evento "start" da aplicação faz o encapsulamento(mock quadro + dado controle) que nesse caso, envia um CR, mudando para o estado Hand1, já que foi ele que teve a iniciativa de se conectar. Caso o Disc recebe um cr, ele irá encapsular o quadro e envia um cc e mudara para o estado Hand2, onde o outro lado tomou iniciativa de se conectar

- HAND1: em negociação para estabelecimento de conexão, quando tomou a iniciativa de estabelecê-la. Caso a gerência receba outro evento start nesse estado, quer dizer que os dois lados tomaram a iniciativa juntos de se conectar, passando então para o estado CON e ativando as camadas superiores para poderem enviar dados. Se receber um CC, entende que o outro lado recebeu o CR enviado e está disponível para se conectar, também ativa a camada superior e envia um reconhecimento de conexão(CA) para o outro lado se conectar. Caso receba um CR, permanecerá no estado Hand1 e enviará um cc para o outro lado. Quando estoura o timeout desse estado ele envia um CR novamente, caso atinja o limite de tentativas que foi estipulado(3) ele volta para o estado desconectado.

- HAND2: em negociação para estabelecimento de conexão, quando o outro lado tomou a iniciativa Caso o quadro recebido tenha o controle CA, ocorre o reconhecimento da conexão e máquina para o estado CON, ativando as camadas superiores e o timeout(). Caso receba um DR, passará para HALF2, enviará um DR e aguardará o recebimento de um DC ou timeout para desconectar. Ao estourar o tempo de timeout, o sistema apenas volta o estado DISC

- CHECK: aguardando resposta para Keep Alive Quando ocorre o timeout no estado CON, ele faz o checkinterval e passa para o estado CHECK onde faz a verificação da conexão, caso o outro lado encaminhe um Kc, ele volta para o estado Con, se não receber, desconecta.

- HALF2: em estado de terminação de enlace (half-close), quando o outro lado tomou a iniciativa Como sugerido, nesse estado ele está aguardando a resposta do outro lado para finalizar a conexão, caso receba um DR ele encaminha um DR e permanece no estado, caso receba um DC, vai para DISC.

- HALF1: em estado de terminação de enlace (half-close), quando tomou a iniciativa de terminá-lo Da mesma forma que o HALF2 aguarda a confirmação de desconexão, assim que recebe a mensagem envia um DC e volta para o DISC

As classes fakeapp e tun são responsáveis por controlar a entrada de texto via teclado e transceiver respectivamente. Ambas são monitoradas pelo poller e operam de forma paralela e assíncrona. A classe Layeruse foi utilizada para instanciar os objetos de cada classe e adicioná-los no poller para monitoramento dos eventos.

As aplicações Envia.py e recebe.py são responsáveis por especificar o endereço das portas seriais, endereços IPs e valor de timeout que serão utilizadas na comunicação pelo protocolo .

1.2 Executando a aplicação

1. Alterar nos arquivos Envia.py e recebe.py os endereços das portas seriais, endereços IP.

2. Executar os arquivos em python3.

Obs.: A interface Tun só pode ser utilizada uma em cada placa de rede.

1.3 Executando as aplicações de teste

Para testar o funcionamento individual da camada ARQ foi adaptado o programa principal para segregar a chamada dos objetos. Para executar o teste, deve-se executar o arquivo Envia.py e recebe.py contidas dentro da pasta DemoARQ. Para testar e verificar como é feito o estufamento dos bytes e o cálculo do CRC pode-se executar o arquivo Demo_Framming_CRC.py onde demonstra esse funcionamento.

1.4 Documentação

ARQ:

Aloha(self) Foi definido o Aloha do tipo aleatório, onde cada Ack recebido irá fazer com que ocorra um Backoff() com tempo aleatório de 1 a 7, para evitar as colisões. Caso haja colisão, o quadro é descartado.

Fsmarq(self, recebefake) Recebe uma mensagem proveniente das camadas fake/Tun e encaminha para o Enquadramento com as inforamações de identificação de um quadro utilizando os métodos sendData0() e sendData1().

init(self, obj, timeout=5) Define a camada responsável pelo ARQ e Aloha, recebe como parâmetro o tempo de timeout eo objeto que será utilizado pelo "poller.py"para fazer o monitoramento dos eventos, que nesse caso será somente o Timeout, o objeto deverá ser None. As variáveis globais foram pré estabelecidos para o funcionamento do protocolo exceto o IDSession, IDProtosession e IDPretoARQ que foram definidos estaticamente por nós.
Funcionalidade de algumas variáveis:

- state : Responsável por definir o estado atual da máquina de estados
- enviadatebackoff: Caso receba Ack errado, após o backoff encaminha o Data espeficado pelo enviadatebackoff
- contreenvio: Limita a quantidade de tentativas de transmissão
- enviabot: Irá receber o objeto do tipo Enquadramento para enviar quadros para camada de baixo

getManager(self, obj) Recebe o objeto do tipo sessionmanage para ser utilizado no método notifyLayer(). Responsável por enviar esse objeto é o Envia.py e recebe.py ao iniciar

getTun(self, obj) Recebe um objeto do tipo Tun para notificar a aplicação quando receber um quadro com proto diferente de 255

getframming(self, obj) Recebe um objeto do tipo Enquadramento, responsável por enviar os quadros para camada de baixo. Quem envia esse objeto são Envia.py e recebe.py

handletimeout(self) Timeout que está sendo monitorado pelo "poller.py"assim que a camada não estiver recebendo nem transmitindo nada.

notifyLayer(self, date) Retorna para a camada superior, que nesse caso é o sessionmanage, o quadro que chegar no recebeserial() e tiver o IDProtoSession = 255.

recebeserial(self, date) Recebe os quadros vindos do Enquadramento, verifica se possui informação de controle para notificar a camada superior "sessionmanage"ou se é apenas informação da camada Fake/Tun date = 0 50 255 Info por exemplo Caso o date[1] seja diferente do estabelecido, o quadro é descartado Caso o date[2] seja 255 e tenha dado de controle, será notificado para sessionmanage

sendACK0(self) Envia para o Enquadramento um quadro contendo a informação do ACK0 mais as informações de identificação da sessão.

ACK0 = 0X80h = 128d

sendACK1(self) Envia para o Enquadramento um quadro contendo a informação do ACK1 mais as informações de identificação da sessão

ACK1 = 0X88h = 136d

sendData0(self) Envia para o Enquadramento um quadro contendo a informação de Data0 mais as informações de identificação da sessão junto com a mensagem/informação a ser transmitda.

Data0 = 0x00 = 0d

sendData1(self) Envia para o Enquadramento um quadro contendo a informação de Data1 mais as informações de identificação da sessão junto com a mensagem/informação a ser transmitda.

Data1 = 0x08 = 8d

Framming:

CloseConection(self) Utilizado para fechar a conexão da serial utilizada.

init(self, dev)
Para contrui-lá é necessário passar como parâmetro o timeout desejado e o objeto contendo as informações da serial para ser utilizado Ex.:serial.Serial('/dev/pts/5', 9600, timeout=5)
Funcionalidade de algumas variáveis.

- buffer: Onde será armazenado os quadros transmitods
- bit: Onde é armazenado o que é lido da serial
- quadro: Apenas informa quando ocorre o recebimento de um quadro completo
- fcs: Recebe objeto do tipo CRC16 para efetuar o cálculo.
- temp: Utilizado para fazer o estufamento dos bytes recebidos

enviabyte(self, dado) Recebe um quadro, faz o cálculo do CRC16 da informação contida no quadro, faz o estufamento e logo em seguida escreve na serial, contendo as identificações do quadro + informação com CRC.

handle(self) Evento onde o poller.py fica monitorando a serial que foi passada no construtor assim que lê um byte, joga para máquina de estado handlefsm() para enquadrar.

handlefsm(self, octeto)

Essa máquina possui 3 estatos: **Ocioso**, **Recp** e **Escape**. Quando ocorre evento na serial, recebe um byte no octeto, assim que chegar um " "máquina identifica que se iniciou um quadro, passa para o estado Recp e análisa os próximos octetos, quando receber " "ele faz o cálculo do CRC16, se estiver sem nenhum erro, notifica a camada ARQ e volta para ocioso. Escape ocorre quando se chega algum octero "7D".

notifyLayer(self, date) Notifica o ARQ quando acaba o Enquadramento e não ocorre nenhum erro no CRC16

vemtop(self, obj) Recebe um objeto do tipo ARQ para utilizar o método recebeserial() do mesmo. Responsável por enviar esse objeto é Envia.py e recebe.py ao iniciar.

sessionManage:

Váriaveis globais:

- cr: conection request
- cc: conection confirm
- Kr: Keep Alive Request
- kc: Keep Alive confirm
- dr: desconection request
- dc: Desconection Confirm
- ca: Connect Acknowledge
- start: Usado para iniciar enviando um cr
- IDSession: Identifica o número da sessão estabelecida
- IDProtosession: Identifica que um quadro é para o Gerenciamento de Sessão
- IDProtoARQ: Identifica que que é um quadro para o ARQ

GetFake(self, obj) Recebe um objeto da camada Fake utilizada para ter interação com o teclado e pode ativar/desativar o gerenciamento de eventos do poller sobre ele.

GetTun(self, obj) Recebe um objeto da camada Tun utilizada para Ativar/Desativar a camada.

MockQuadro(self) Retorna um quadro padrão com valor de ID Proto utilizado para enviar mensagens de controle para o Gerenciamento de Sessão.

Recebearq(self, quadro) Recebe um quadro da camada ARQ, que contenha IDProtosession com valor 255 e joga para a máquina de estado fazer o processamento.

A máquina possui os seguintes estados:

- DISC: desconectado
- HAND1: em negociação para estabelecimento de conexão, quando tomou a iniciativa de estabelecê-la
- HAND2: em negociação para estabelecimento de conexão, quando o outro lado tomou a iniciativa
- CON: conectado
- CHECK: aguardando resposta para Keep Alive
- HALF1: em estado de terminação de enlace (half-close), quando tomou a iniciativa de terminá-lo
- HALF2: em estado de terminação de enlace (half-close), quando o outro lado tomou a iniciativa

SendtoArq(self, quadro) Recebe um quadro e encaminha para o enviabyte do Enquadramento, para jogar na serial

StartConection(self) Quando um lado chamada esse método, ele joga para o Recebearq() um valor 9, que a máquina identifica como start e envia um cr para tentar iniciar uma conexão.

init(self, obj, timeout=5) Recebe como parâmetro None como objeto e um valor de Timeout desejado, que por padrão está 5.

- Auxcon: Utilizado quando os dois lados tem a iniciativa de estabelecer uma conexão e estão no estado Hand1
- OnOFFTOP: Utilizado para ativar e desativar a camada superior
- quadrocontrole: Recebe o quadro que deverá ser enviado para o Gerenciamento de Sessão.

getArq(self, obj) Recebe um objeto do tipo Enquadramento ao iniciar do sistema (Envia.py e recebe.py)

handletimeout(self) Chamada pelo poller.py quando ocorre um evento de timeout.

Diagrama de classes

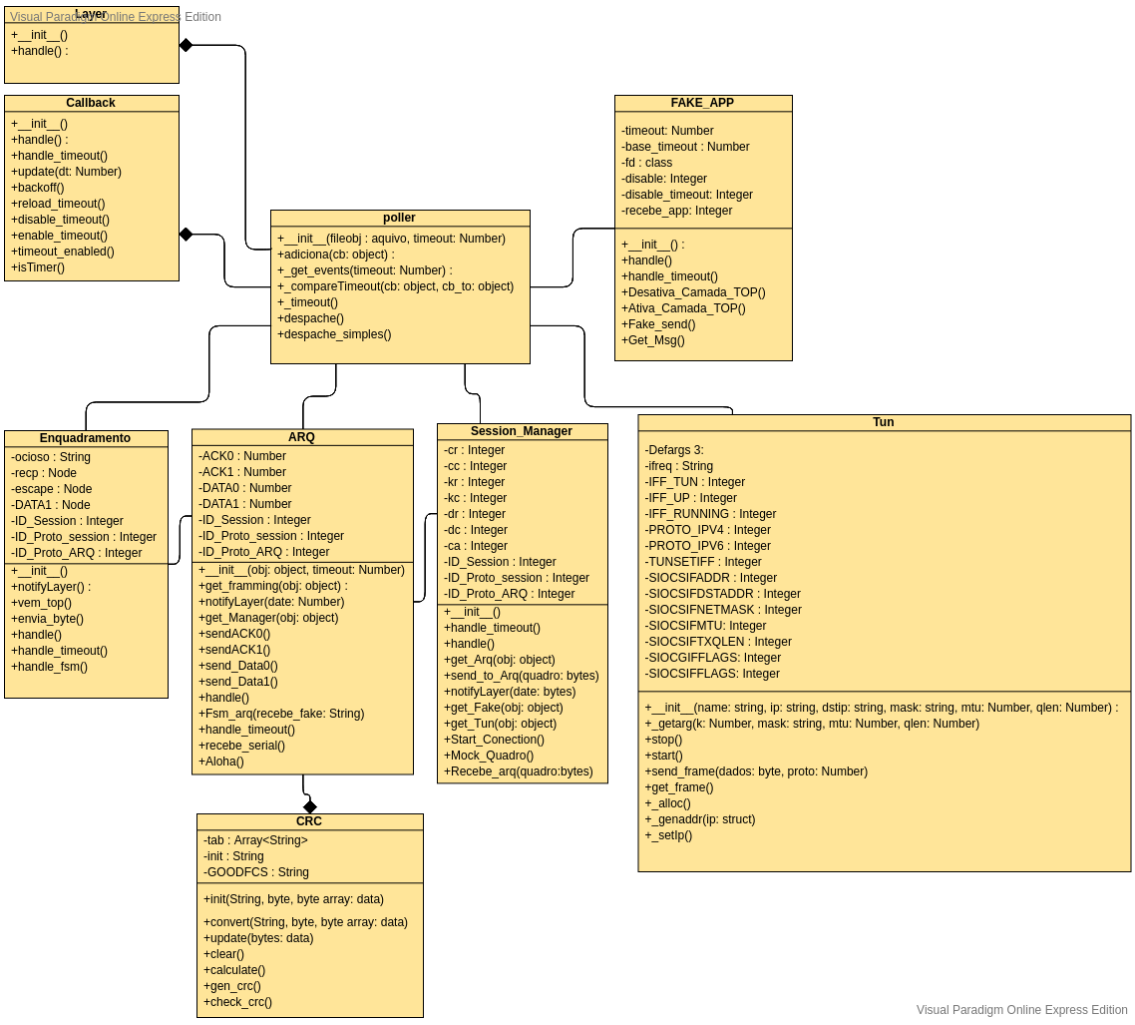

Figura 4: Diagrama de Classes

1.5 Conclusão

A implementação desse protocolo com metódo de agregação de camadas possibilitou visualizar a composição do quadro desde a recepção dos bits serializados até a separação da mensagem útil que será recebida pela aplicação. Essa divisão em camadas permitiu visualizar técnicas como garantia de entrega e correção de erros que são aplicadas em protocolos reais funcionando praticamente da mesma forma nesse projeto simulado e também evidenciou importância da padronização dos quadros e mensagens trocadas para interpretação por qualquer aplicação envolvida no sistema.