UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA

**Trabalho DESAFIO (OPCIONAL) - ENTREGA 01/10/2013 - IMPRETERIVELMENTE**

## 1. Objetivo

Implementar um aplicativo cliente-servidor que realize a transferência de um arquivo do cliente para o servidor. O protocolo a ser usado pela aplicação é uma versão simplificada do HDLC (*High-level Data Link Control*).

## 2. Especificação

Implementar em C (não em C++) um aplicativo que realize um protocolo baseado no HDLC respeitando as seguintes condições :

1. Executar em ambientes GNU/Linux ;
2. Protocolo GO BACK N considerando 3 bits para numeração de sequência. O programa deve gerar um número aleatório entre 1 e 7, indicando a quantidade máxima de quadros da janela a serem enviados antes de esperar por uma confirmação positiva. O próximo envio só deve ocorrer depois que for recebido a confirmação positiva ($RRn$) desta sequência. Se houver menos quadros na janela do que a quantidade máxima gerada, o programa enviará todos os quadros da janela.
3. Frame Check Sequence calculado como CRC16 (utilizar o código fornecido na página Web da disciplina) ;
4. Geração artificial e aleatória de erros de transmissão. A aleatoriedade deve ser implementada com auxílio das funções `srand()` e `rand()` (ou similares) para obter números randômicos entre 0 e 99. Esses números, no momento de envio de um quadro, deverão ser interpretados da seguinte forma :
   - Faixa 0-9 : um erro de CRC deve ser provocado. Esse erro pode ser obtido alterando um ou mais bits quaisquer da área de dados após o cálculo do CRC. (sugestão : complementar um byte qualquer da área de dados)
   - Faixa 10-99 : nada deve ser feito.

   O transmissor (cliente) deverá fornecer um *log* com os quadros de informação e de controle enviados e recebidos. Deve ser identificado os quadros que foram enviados com erro de CRC (ver seção 5).
5. O receptor (servidor) deve enviar aleatoriamente quadros RNR $n$ ao invés de quadros RR $n$. A aleatoriedade deve ser implementada com auxílio das funções `srand()` e `rand()` (ou similares) para obter números randômicos entre 0 e 99. Se o número randômico ficar na faixa de 0 a 9, o RNR $n$ deve enviado no lugar do RR $n$. Após o envio do RNR $n$ o receptor deve aguardar 2 segundos e enviar um RR $n$ para liberar o transmissor (cliente) a continuar a enviar dados. Esse esquema pode gerar uma situação de $deadlock$ (ver seção 5) que **não** precisa ser considerada nesta implementação.
6. O controle de erro deverá ser de forma explícita usando apenas *time-out*, ou seja, não é preciso implementar a ação do quadro de controle REJ $n$), nem usar os bits de *poll* e *final* (P/F).
7. Não há necessidade de implementar *piggybacking* ;
8. Usar *byte stuffing*, mas empregar o *flag* delimitador de início e fim de quadro (01111110) ;
9. Time-out de 100 ms (aproximadamente) para reenvio de quadros ;
10. Respeitar imperativamente o formato de quadro fornecido na figura 1.
11. Um único programa deve implementar os comportamentos de cliente (transmissor) e servidor (receptor). O programa assumirá um ou outro comportamento de acordo com parâmetros passados pela linha de comando (ver seção 5).
12. Criar o canal lógico de comunicação entre duas máquinas empregando a interface de sockets em modo não conectado (UDP). Usar a porta UDP 64000. Na página da disciplina há um programa exemplo de uso de sockets UDP.

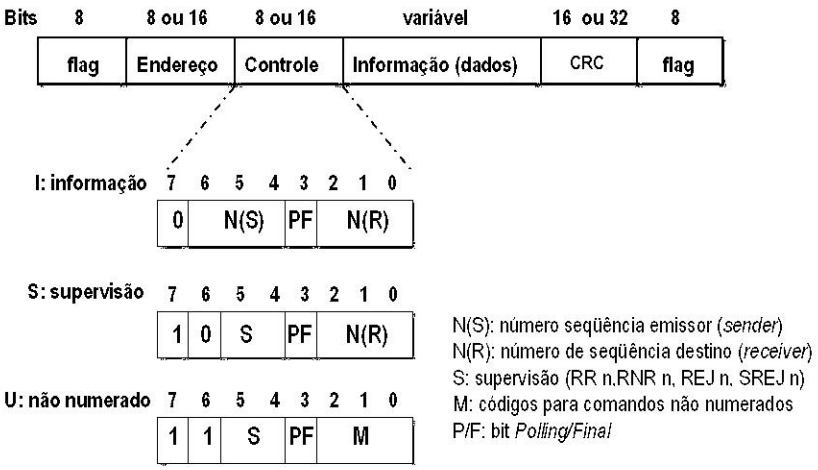


FIGURE 1 – Formato do quadro

### 3. Descrição dos campos do quadro

**flag (8 bits) :** delimitador de início e final do quadro. Corresponde a sequência binária 01111110 (0x7E).

**Campo de endereço (8 bits) :** usar sempre o endereço de *broadcast*, isto é, o valor binário 11111111 (0x0FF).

**Campo de controle (8 bits) :** O campo de controle é formado por um único byte. Esse campo é utilizado para o sequenciamento de quadros (necessário ao Go back-N), na codificação de controle (RR, RNR, REJ e SREJ) e na configuração do enlace (quadros U). Existem três tipos de quadros : informação, supervisão e quadros U. A figura 1 fornece a interpretação dos bits de controle. O protocolo a ser implementado considera um algoritmo de janela deslizante com número de sequência em 3 bits (bits N(S)).

Os vários tipos de quadros de supervisão são diferenciados pelo valor dos bits S : quadros com bits $S_5$=0,$S_4$=0 correspondem a quadros de *acknowledgement* positivo que indicam o próximo quadro a ser recebido (Receive Ready - RR) ; quadros com bits $S_5$=0,$S_4$=1 correspondem a ação Receive Not Ready (RNR). Nesta implementação **não são usados** os quadros REJ e SREJ. O campo N(R) indica o número de sequência do quadro. Os quadros de informação (I) são usados para o envio/recebimento de dados e possuem campos para o envio de número de sequência dos quadros (campo N(S)) e para *piggybacking* (campo N(R)).

No caso específico da implementação proposta neste trabalho alguns bits perdem seus significados :

- Os bits N(R) dos quadros de informação não são usados, pois não há situação em que o *piggybacking* possa ocorrer. Eles devem ser mantidos em zero. IMPORTANTE : apenas nos quadros de informação os bits N(R) são zerados.
- Os quadros do tipo REJ não serão gerados, pois o controle de erros será feito com base no *time-out*.
- Os quadros U (não-numerados) são usados para o estabelecimento e encerramento da conexão. No protocolo HDLC original existem 32 tipos diferentes de quadros U, mas neste trabalho serão usados apenas dois para encerrar a transferência do arquivo. A saber :

| Comando | Código Byte controle | Ação |
|---|---|---|
| DISC | 11000010 (0x0c2) | Disconnect = Solicitação de encerramento de conexão |
| UA | 11001110 (0x0ce) | Ununbered ACK = Confirmação (ACK) de um quadro U qualquer |

**Informação (variável) :** Os quadros possuem uma quantidade variável de bytes na área de dados (entre zero e 250 bytes). Em quadros de supervisão não há área de dados.

**Frame Check Sequence - FCS (16 bits) :** O campo *Frame Check Sequence* corresponde ao cálculo de um CRC 16 bits (dois bytes). O FCS é calculado considerando-se todos os bytes dos campos de endereço, de controle e da área de dados (se houver).

O cálculo de CRC foi originalmente concebido para ser implementado em hardware através de uma sequência de portas XOR. Nesse caso, os dados são enviados serialmente e, a medida que são recebidos, o CRC é calculado. Com esse princípio de funcionamento, o receptor não tem condições de determinar o final do quadro até que o flag delimitador de final de quadro seja recebido e, por consequência, o CRC do quadro acaba sendo considerado no cálculo do CRC por parte do receptor. Portanto, transmissor e receptor realizam o cálculo de CRC sobre dados (bytes) diferentes.

O transmissor, como descrito anteriormente, realiza o cálculo do CRC considerando os bytes dos campos de endereço, de controle e da área de dados, o valor obtido é então binariamente negado (complemento de 1). Esse é o CRC do transmissor (cliente). No receptor (servidor), o cálculo do CRC é feito considerando os bytes dos campos de endereço, de controle, da área de dados e o CRC calculado pelo transmissor. Da forma como o CRC foi implementado - que é um padrão existente - o valor calculado pelo receptor deverá ser sempre a constante 0x0f0b8 para uma transmissão SEM ERROS.

Para o trabalho, usar o método de tabela para cálculo do CRC. O código que implementa esse método é fornecido na página da disciplina. **USE-O !**

**Byte stuffing :** O protocolo HDLC original é síncrono orientado a bit, mas neste trabalho ele será implementado na forma síncrono orientado a byte. Um quadro é delimitado pelo caracter 01111110 (0x7E) como marca de início e fim. Para fornecer transparência de dados, isto é, impedir que esse caracter ocorra na área de dados provocando o fim prematuro do quadro deve ser usada a técnica de *byte stuffing*. O procedimento de *byte stuffing* é baseado na inserção de um caracter especial (*escape*). Esse procedimento é descrito a seguir.

Depois do cálculo de CRC, o transmissor examina o quadro a ser enviado e troca cada ocorrência do valor 0x7E (com exceção dos delimitadores de início e fim de quadros) por uma sequência de dois bytes correspondendo ao caracter de *escape* (0x7D) seguido do valor obtido pela operação "ou exclusivo" entre o valor original do byte a ser *stuffed* e a constante 0x20. Dessa forma, as seguintes transformações são possíveis :

- 0x7E é codificado pela sequência 0x7D 0x5E
- 0x7D é codificado pela sequência 0x7D 0x5D

O destinatário ao receber um quadro, varre os dados procurando pelas ocorrências das sequências 0x7D 0x5E e 0x7D 0x5D, substituindo-as, respectivamente, por 0x7E e 0x7D.

IMPORTANTE : O *byte stuffing* é feito APÓS o cálculo do CRC, ou seja, o quadro é montado, o CRC é calculado e **apenas no momento do envio de dados** é que ocorre o *byte stuffing*. Na recepção, os bytes introduzidos pelo *byte stuffing* são retirados, e após, sobre os bytes restantes o CRC é calculado. Dessa forma, o CRC na recepção considera os bytes dos campos de endereço, de controle, da área de dados, e o CRC calculado pelo transmissor.

**4. Estabelecimento e encerramento da conexão**

O HDLC possui três fases : inicialização, transferência de dados e encerramento. Na inicialização, é enviado um quadro U com o comando *setmode*. Esse quadro faz com seja iniciada uma conexão de troca de dados usando uma janela de tamanho negociado. Em resposta ao comando *setmode*, é enviado um quadro UA. Havendo erros nessa etapa, a recuperação é feita por *timeout*.Após estabelecida a conexão, inicia a fase de transferência de dados, que é caracterizada pelo envio de quadros de informação (quadros I) e do recebimento de quadros de supervisão (S). Os quadros de supervisão são do tipo RR $n$, RNR $n$ e REJ $n$. Neste trabalho, NÃO é necessário realizar a fase de inicialização. O cliente já iniciará a sua transmissão enviando os quadros de Informação (dados) com o contéudo do arquivo.

Quando o cliente acabar de enviar todos os bytes que compõem um arquivo, ele sinalizará isso ao servidor enviando um quadro U com o comando DISC. A resposta a um comando DISC é um quadro UA. (Observação : no verdadeiro HDLC o pedido de desconexão (encerramento) pode ser solicitado por qualquer um dos lados).

**5. Executando o programa**

O programa deve ser executado pela linha de comando empregando os seguintes argumentos :

```
%programa –l –h IP [–f filename]
```

O primeiro programa a ser disparado é o servidor. Somente após ter sido inicializado o servidor é que se deve disparar o cliente. No lado cliente, o argumento da opção *-h* indica o endereço IP da máquina destino (servidor), ou seja, a máquina que receberá o arquivo passado como argumento da opção *-f*. A opção *-l* ativa a geração de um arquivo de histórico (*log*).As opções *-l, -h* e *-f* são utilizadas apenas pelo cliente. A opção *-f* indica o nome (*filename*) do arquivo a ser enviado. A ausência das opções *-l, -h* e *-f* colocará o programa no modo "servidor" (receptor do arquivo). Exemplo :

```
Máquina servidor
%programa

Máquina cliente (supondo que o servidor seja IP 192.168.20.1)
%programa –l –h 192.168.20.1 –f texto.txt
```

O arquivo a ser enviado pode ser tanto texto como binário. O cliente (transmissor) deve abrir o arquivo e enviá-lo em $q$ quadros para o servidor (receptor) até atingir o EOF (*End Of File*). O servidor deverá gerar um arquivo com os dados recebidos. O final do envio do arquivo será sinalizado pelo pedido de desconexão enviado pelo cliente ao servidor. O cliente (transmissor) deve gerar um histórico ($log$) de todos os quadros enviados (tx) e recebidos (rx), discriminando seu tipo (informação ou supervisão) e o número de sequência, algo na forma :

```
....
(tx) I,4
(tx) I,5
(tx) I,6
(tx) ***estouro do time-out*** quadro 4
(tx) I,4
(tx) I,5
(tx) I,6
(rx) RR,7
(tx) I,7 -> gerado com erro de CRC
(tx) I,0
(rx) ***estouro do time-out*** quadro 7
(tx) I, 7
(tx) I, 0
(rx) RNR, 1
(rx) RR, 1
....
```

O *log* poderá ser escrito na tela ou em um arquivo. A geração do *log* em arquivo é sinalizado pelo emprego da opção *-l* na linha de comando. O arquivo de *log* terá sempre o nome *log.txt*.

**Situação de *deadlock* :** considere a hipótese em que o RR 1, ilustrado acima, seja enviado e descartado por erro. Nesse caso, a máquina cliente considera que o servidor ainda não está pronta para receber novos quadros, pois havia enviado um RNR 1. Por outro lado, o servidor, como enviou o RR 1, acha que a máquina cliente não tem mais quadros para enviar. Está configurado um *deadlock*. Os protocolos reais, para se proteger desta situação, criam um $timeout$ adicional denominado de *keep-alive*. Se esse *timeout* "estourar" uma máquina envia uma mensagem especial para outra para perguntar em qual situação a comunicação de encontra. NÃO é preciso implementar esse controle neste trabalho.

**Observação :** O programa pode ser usado e testado em apenas uma máquina usando o endereço IP 127.0.0.1 (*localhost*), ou o endereço IP da máquina, e disparando duas instâncias do mesmo programa (um será o processo transmissor e outro, o receptor). Para teste final do programa deverá ser usado duas máquinas distintas.

**6. Material suplementar de apoio**

- Stevens, Richard. *UNIX Network Programming*. Prentice-Hall. 1993
- Comer, Douglas. *Redes de Computadores e Internet*. Bookman.2000.
- Tanenbaum, A.S. *Redes de Computadores*. 4a edição. Editora Campus, 2003.

As duas primeiras referências (Stevens e Comer) são relativas ao uso de sockets UDP. O livro do Tanenbaum tem dicas de como implementar o protocolo Go-Back N e *time-out* (final da seção 3.4.2 ou pg. 237 da 4a edição).

Ainda, para implementar o *time-out* é necessário a criação de um mecanismo de interrupção. O mecanismo básico é a utilização de *signal* em ambientes UNIX. Entretanto é possível explorar as capacidades de *threads* para se implementar de forma mais fácil o *time-out*. Nesse caso, estude o comportamento da chamada *pthread_cond_timedwait()*.

**7. Datas e método de avaliação — LEIA com MUITA ATENÇÃO ! ! !**

1. O trabalho deve ser feito INDIVIDUALMENTE.
2. O trabalho deverá ser entregue até as 23 :59 :59 horas horas do dia 1 de OUTUBRO de 2013 por e-mail para $asc@inf.urgs.br$. Entregar os fontes e um Makefile para compilação e geração do executável.
3. A realização deste trabalho substitui **UMA** questão da prova teórica 1. Além disso, o trabalho vale até TRÊS pontos, sendo que as questões valerão DOIS pontos cada uma, ou seja, fazendo correta e adequadamente o trabalho você pode atingir ONZE pontos na prova teórica 1.
4. Parâmetros a serem usados na correção :
   (a) É obrigatório o uso do programa de CRC fornecido na página da disciplina. O não cumprimento dessa restrição implica em ZERO pontos no trabalho.
   (b) ZERO pontos : simplesmente enviar e receber dados usando o programa fornecido em UDP.
   (c) ZERO pontos : não implementar um Go-back N real. Cuidado para não fazer dele um *Stop-and-wait* disfarçado ! ! ! Leia atentamente o item 2 da seção 2 (especificação)
   (d) Até UM ponto : implementação do Go-back N de forma que funcione na ausência de qualquer tipo de erro. Para isso pode ser suprimido a geração randômica dos erros, o *time-out* e o uso do CRC. Deverá ser capaz de receber e enviar arquivos ASCII e binário.
   (e) Até DOIS pontos : implementação **completa** desta especificação, porém funcionando em apenas uma máquina, ou seja, os dois processos (transmissor e o receptor) estão na mesma máquina. Deverá ser capaz de receber e enviar arquivos ASCII e binário.
   (f) Até TRES pontos : implementação **completa** desta especificação, porém funcionando em máquinas distintas, ou seja, o transmissor e o receptor em máquinas diferentes. Nesse caso, o programa deve ser capaz de transmitir corretamente mesmo que o cabo de rede seja retirado e recolocado. Deverá ser capaz de receber e enviar arquivos ASCII e binário.
5. O professor da disciplina se reserva, caso necessário, o direito de solicitar uma demonstração do programa. A nota final será baseada nos parâmetros acima e no questionamento a questões de projeto e de implementação feitas ao aluno.

**Como testar se meu programa funciona ? (ou o quê o professor vai fazer...)**

O teste a ser feito disparará duas instâncias do programa em máquinas distintas e enviará arquivos de tamanhos variáveis (de poucos bytes até algumas dezenas de Kbytes). Será feita uma inspeção visual no código fonte para verificar se ele está fazendo o que realmente deve fazer (*byte stuffing*, geração de erros, etc). Para concorrer aos TRÊS pontos, o cliente (transmissor) e o servidor (receptor) serão disparados em máquinas distintas e durante a transferência dos dados o cabo de rede será retirado e recolocado algumas vezes para testar a recuperação de erros. Os arquivos que serão transferidos serão binários (executável do próprio programa) e arquivos ASCII.

**Importante * Importante * Importante * Importante * Importante* Importante * Importante * Importante**
Deve ser implementado um Goback N e não um *stop-and-wait* disfarçado, isso é, **NÃO** se deve enviar toda a janela de transmissão, esperar por um RR e repetir esse procedimento enquanto houver dados do arquivo a ser transmitido. Isso *não é* um GoBack N, portanto, não corresponde a implementação solicitada (ver seção 7 - item 4c).