Trabalho DESAFIO (OPCIONAL) - ENTREGA 06/05/2015 - IMPRETERIVELMENTE

1. Objetivo

Implementar um aplicativo cliente-servidor que realize a transferência de arquivos entre duas máquinas distintas empregando o protocolo hipotético VSFTP (*Very Simple File Transfer Protocol*). As mensagens do protocolo VSFTP deverão ser enviadas (encapsuladas) em um quadro HDLC (*High-level Data Link Control*). Como é necessário que essa comunicação ocorra de forma confiável, isso é, com a garantia de que todos os dados enviados serão entregues na ordem em que foram emitidos, sem duplicação, nem perdas, o HDLC oferecerá para o VSFTP um serviço de controle de fluxo e controle de erro baseado em *Go-back* N. A arquitetura do serviço segue um modelo cliente-servidor.

A implementação deverá ser obrigatoriamente feita usando a linguagem C (não C++) e executar em ambientes GNU/Linux.

2. Descrição do *Very Simple File Transfer Protocol* (VSFTP)

O VSFTP é um protocolo de aplicação para a transferência de arquivos entre duas máquinas. Qualquer transferência de arquivos utilizando o protocolo VSFTP é iniciada por uma mensagem de requisição de leitura ou de escrita de um arquivo, enviado de uma máquina (o cliente VSFTP) para outra (o servidor VSFTP). Considera-se que os arquivos a serem enviados estão armazenados no diretório corrente a partir do qual os programas cliente e servidor VSFTP são executados.

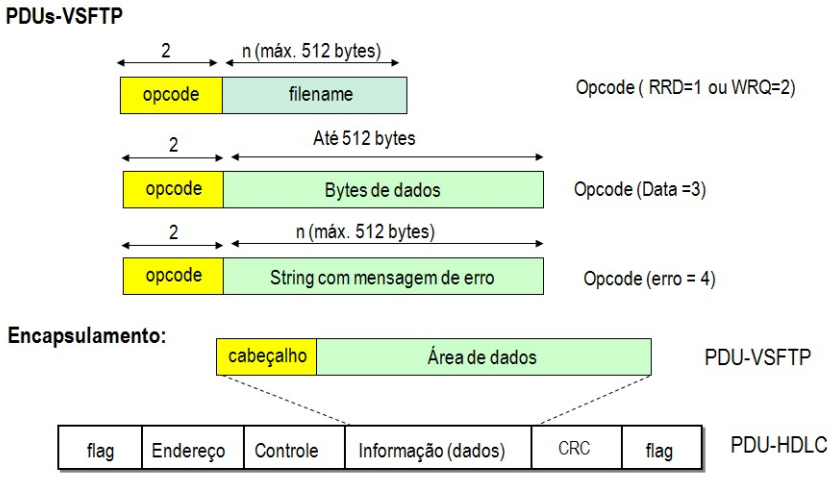

FIGURE 1 – PDU do protocolo VSFTP

O VSFTP tem três tipos de mensagem : requisição, dados e erro. As PDUs (*Protocol Data Unit*) do VSFTP são fornecidas na figura 1. Inicialmente a transferência é iniciada pelo envio, por parte do cliente VSFTP, de uma mensagem de requisição com o código *Read Request* (RRQ=1), ou com o código *Write Request* (WRQ=2),seguido de uma *string*. A *string* fornece o nome do arquivo a ser lido do servidor VSFTP (requisição RRQ) ou a ser transferido (escrito) no servidor VSFTP (requisição WRQ). Logo após o envio da mensagem de requisição (RRD ou WRQ) é iniciado o envio dos bytes que compõem o arquivo em mensagens de dados. As mensagens de dados tem o código *Data* (DATA=3) e possuem uma área de dados de zero a 512 bytes. Todas as mensagens de dados, a

exceção da última, possuem 512 bytes na área de dados. Dessa forma, uma mensagem com a área de dados menor que 512 bytes indica o término da transferência do arquivo.
Em caso de qualquer tipo de erro, por exemplo, arquivo a ser lido não existente, uma mensagem de erro com uma *string* indicativa do erro ocorrido deverá ser enviado. A mensagem de erro deverá ser impressa na tela de quem a recebeu e o programa deverá terminar. As mensagens e situações de erros são livres, isso é, você pode definir as que achar necessário. Entretanto, o formato da mensagem de erro deverá seguir o formato da PDU definida para essa finalidade.
As mensagens VSFTP são encapsuladas pela camada inferior, nesse caso, uma camada baseada em um quadro HDLC, que implementa um controle de fluxo e controle de erro necessários a confiabilidade da transmissão de dados.

3. Descrição do HDLC (*High-level Data Link Control*) adaptado

O HDLC (*High-level Data Link Control*) é um protocolo síncrono, orientado a bit, que emprega um quadro de tamanho variável delimitado pelo caracter especial 01111110_2 (0x7E) e utiliza como mecanismo de detecção de erro o cálculo de CRC. O quadro HDLC a ser considerado é fornecido na figura 2. Para este trabalho serão consideradas algumas modificações em relação ao HDLC original, a saber :

Endereço (8 bits) : usar sempre o endereço de *broadcast*, isso é, o valor binário 11111111_2 (0x0FF).

Controle (8 bits) : O campo de controle é formado por um único byte. Esse campo é utilizado para o sequenciamento de quadros (necessário ao controle de fluxo e erro) e na codificação dos quadros de controle (RR, RNR e REJ). Serão considerados apenas dois tipos de quadros : informação e supervisão. O controle de fluxo e de erro a ser implementado considera um algoritmo de janela deslizante com número de sequência em 3 bits (bits N(S)).

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

Frame Check Sequence - FCS (16 bits) : O campo *Frame Check Sequence* corresponde ao cálculo do CRC-16 (dois bytes).

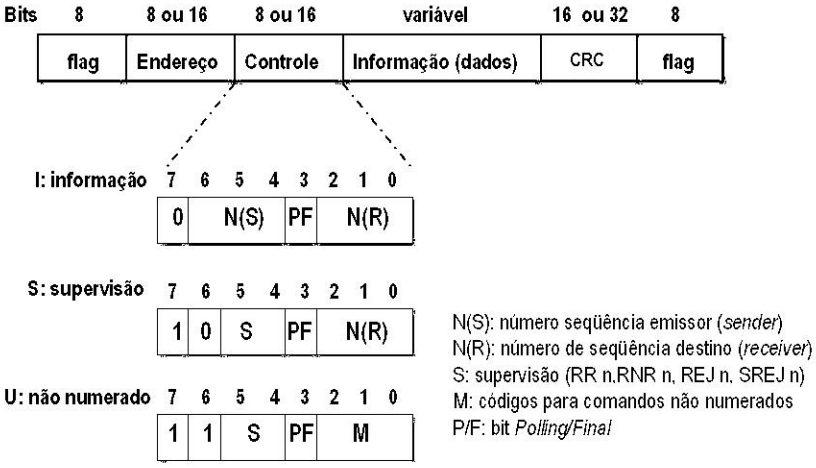

FIGURE 2 – Formato do quadro HDLC

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=1,S_4=0 correspondem a ação Receive Not Ready (RNR) e quadros com bits S_5=0,S_4=1 correspondem a quadros REJ. Nesta implementação **não são usados** quadros SREJ (S_5=1,S_4=1). 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)). Neste trabalho, os bits do campo N(R) dos quadros de informação não são usados, pois, na transferência de arquivos via VSFTP, 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).

O cálculo do CRC deverá ser feito empregando o *método da tabela*, fornecido no Moodle, junto com esta especificação. Nesse método, o transmissor realiza o cálculo do CRC considerando os bytes dos campos de endereço, de controle e da área de dados (se houver), o valor obtido é então binariamente negado (complemento de 1). Esse é o CRC do transmissor. No receptor, 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 é implementado - que é um padrão existente - o valor calculado pelo receptor deverá ser sempre a constante 0x0F0B8 para uma transmissão SEM ERROS.

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_2 (0x7E) como marca de início e fim. Assim, para fornecer transparência de dados, isso 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*) como segue.

Depois do cálculo do 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 resultantes o CRC é calculado. Dessa forma, o CRC na recepção considera os bytes dos campos de endereço, de controle, da área de dados (se houver), e o CRC calculado pelo transmissor.

O controle de fluxo e de erro é baseado em janela deslizante usando uma estratégia Go-back N com números de sequência de três bits (janela de 7 quadros) com a retransmissão de quadros com erro baseado em *time-out* e pelo envio de quadros REJ. O valor a ser considerado para o *time-out* é de aproximadamente 100 ms.

4. Simulação de situações de erros

Para exercitar as situações de recuperação no caso de erro, ou perda de quadros, a implementação deverá contemplar a geração artificial e aleatória de erros de transmissão (realizado no lado que envia o arquivo). A aleatoriedade deve ser implementada com auxílio das funções `srand()` e `rand()` (ou similares) para obter números randômicos entre 0 e 99. Para a **geração artificial de erros**, 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 no quadro e o mesmo é enviado normalmente.

O cliente VSFTP deverá fornecer um *log* com os quadros de informação enviados e dos quadros de controle recebidos. Deve ser identificado os quadros que foram enviados com erro de CRC (ver seção 7).

Geração aleatória de quadros RNR n e RR n (realizado pelo lado que recebe o arquivo) : 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 uma confirmação positiva (RNR ou RR), deverão ser interpretados da seguinte forma :

- Faixa 0-19 : envia um quadro RNR n invés do quadro RR n.
- Faixa 20-99 : se o número aleatório gerado for ímpar, a confirmação positiva RR n NÃO DEVE ser enviada. Caso contrário, número par, então o RR n é enviado normalmente. Esse procedimento provoca dois comportamentos : (i) simula a perda de quadros de confirmação positiva ; (ii) exercita o efeito acumulativo dos quadros de confirmação positiva.

Após o envio de um quadro RNR n o receptor deve aguardar DOIS segundos e enviar o RR n apropriado para liberar o transmissor a continuar a enviar dados. Nessa situação, o envio do quadro RR n não está sujeito a regra "par-impar" de envio descrita anteriormente, ou seja, o RR n será sempre enviado após os 2 segundos de espera.

5. Simulando um fio virtual

Para facilitar a implementação, e mesmo os testes, será usado como meio de comunicação um *fio virtual* implementado através de um programa cliente-servidor em UDP. O código de base para esse fio virtual é fornecido no Moodle junto com esta especificação. Basicamente, esse código-exemplo é um servidor de ECO. Ele permite que um cliente UDP envie um mensagem como uma *string* para um servidor UDP e, esse, ao receber a mensagem, a reenvie de volta para o cliente. Adapte-o a sua necessidade !

Dessa forma, o quadro HDLC é encapsulado por um datagrama UDP e pode ser enviado pela Internet. Para isso, é necessário definir uma porta para o servidor UDP. A porta a ser usada para é a 64000.

6. Executando o programa

O primeiro programa a ser disparado é o servidor VSFTP. Para isso deve-se empregar a seguinte linha de comando em uma janela da máquina servidora :

```
%vsftpserver
```

Somente após ter sido inicializado o servidor é que o cliente deve ser disparado. No lado cliente, o argumento da opção *-h* indica o endereço IP da máquina destino (servidor), ou seja, a máquina que receberá ou enviará o arquivo *filename* passado como parâmetro. A opção *-l* ativa a geração de um arquivo de histórico (*log*). As opções *-l, -h* e *-f* são utilizadas apenas pelo cliente VSFTP. A opção *-f* indica o nome (*filename*) do arquivo a ser enviado ou recebido :

```
%vsftpclient -l -h IP [-f filename]
```

O arquivo a ser enviado/recebido (*filename*) pode ser tanto texto como binário. Para as requisições de escrita (WRQ), o cliente VSFTP deve abrir o arquivo e enviá-lo em q quadros para o servidor VSFTP até atingir o EOF (*End Of File*). O servidor VSFTP deverá gerar um arquivo com os dados recebidos. Da mesma forma, para as requisições do tipo leitura (RRQ), o servidor VSFTP deverá abrir o arquivo e enviá-lo em q quadros ao cliente VSFTP até atingir o EOF. Nesse caso, é o cliente VSFTP que deverá gerar o arquivo com os dados recebidos. O cliente VSFTP 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 na última linha do *log* exemplo acima, seja enviado e descartado por erro. Nesse caso, a máquina cliente considera que o servidor ainda não está pronto para receber novos quadros, pois este havia enviado um RNR 1 (penúltima linha). 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 dessa 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 em janelas diferentes (um será o processo transmissor e outro, o receptor). Para o teste final do programa deverá ser usado duas máquinas distintas.

7. 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*.

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()*.

8. 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 do dia 06 de MAIO de 2015 por e-mail para $asc@inf.ufrgs.br$. 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 ! ! !
   (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. O programa 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. O programa 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. O programa 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 centenas 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 8 - item 4c).