Especificação do projeto 2 **Implementando um protocolo de transporte confiável**

Redes de Computadores 2019.1

Prof. Filipe

Visão geral

Neste laboratório, você escreverá o código de nível de transporte de envio e recepção, implementando um simples protocolo de transferência de dados confiável com bit alternante e Go-Back-N. O laboratório irá diferir um pouco do que seria exigido numa situação do mundo real, e todo código deve ser implementado no arquivo trabalho_2.c

Rotinas que você irá escrever

Os procedimentos que você escreverá são para a entidade de envio (A) e para a entidade de recepção (B). Apenas transferência unidirecional de dados (de A para B) é exigida. Naturalmente, o lado B deverá enviar pacotes ao lado A para confirmar a recepção do dado. Suas rotinas devem ser implementadas na forma dos procedimentos descritos abaixo. Esses procedimentos serão chamados por uma aplicação que você escreverá, para transferir um arquivo da A para B.

A unidade de dados que trafega entre a aplicação e seus protocolos é uma mensagem, que é declarada como:

```
struct msg {
  char data[20];
};
```

Então, a aplicação A enviará blocos de 20bytes para seu protocolo enviar para o lado receptor, e lá, seu protocolo deve passar para a aplicação B.

A unidade de dados que trafega entre suas rotinas e a camada de rede é o pacote, que é declarado como:

```
struct pkt {
  int seqnum;
  int acknum;
  int checksum;
  char payload[20];
};
```

Suas rotinas preencherão o payload com os dados que receberão da aplicação, os outros campos serão utilizados pelo seu protocolo para assegurar a entrega confiável.

As rotinas que você escreverá estão detalhadas abaixo. Conforme dito acima, tais procedimentos no mundo real seriam parte do sistema operacional e chamados por outros procedimentos no sistema operational.

A_output(message) , onde message é a estrutura do tipo msg, contendo dados que serão enviados para o lado B. Esta rotina será chamada sempre que a camada superior do lado (A) tenha uma mensagem a ser enviada. Este é o trabalho do seu protocolo para assegurar que os dados em tal mensagem sejam entregues em ordem, e corretamente, para a camada superior do lado destinatário.

A_input(packet) , onde packet é a estrutura do tipo pkt. Esta rotina será chamada sempre que um pacote enviado pelo lado B (exemplo: como resultado de um tolayer3() feito por um procedimento do lado B) chega ao lado A. packet é o pacote (possivelmente corrompido) enviado pelo lado B.

A_timeinterrupt() , esta rotina será chamada quando o temporizador de A expirar (gerando uma interrupção no temporizador). Você provavelmente usará esta rotina para controlar a retransmissão dos pacotes. Veja starttimer() e stoptimer() abaixo para ver como o temporizador é acionado e interrompido.

A_init() , esta rotina será chamada apenas uma vez, antes de qualquer outra rotina do lado A ser chamada. Ela pode ser usada para fazer qualquer inicialização requerida.

B_input(packet) , onde packet é a estrutura do tipo pkt. Esta rotina será chamada sempre que um pacote enviado pelo lado A (exemplo: como resultado de um tolayer3() feito por um procedimento do lado A) chegar ao lado A. packet é o pacote (possivelmente corrompido) enviado pelo lado A.

B_init() , esta rotina será chamada apenas uma vez, antes de qualquer outra rotina do lado B ser chamada. Ela pode ser usada para fazer qualquer inicialização requerida.

Interfaces de software

Os procedimentos descritos acima são aqueles que você irá escrever. As rotinas a seguir foram escritas e podem ser chamadas pelas suas rotinas:

starttimer(calling_entity, increment) , em que calling_entity será "0" (para acionar o temporizador do lado A) ou "1" (para acionar o temporizador do lado B), e increment é um valor flutuante que indica a quantidade de tempo que se passou antes de o temporizador ser interrompido. O temporizador de A dever ser acionado (ou interrompido) apenas pelas rotinas do lado A, similarmente para o temporizador do lado B. Para dar uma idéia do valor apropriado de incremento a ser usado, um pacote enviado dentro de uma rede leva em média 5 unidades de tempo para chegar ao outro lado quando não há outras mensagens no meio.

stoptimer(calling_entity) , onde calling_entity será "0" (para acionar o temporizador do lado A) ou "1" (para acionar o temporizador do lado B).

tolayer3(calling_entity,packet) , onde calling_entity será "0" (para envio do lado A) ou "1" (para envio do lado B), e packet é a estrutura do tipo pkt. Ao chamar esta rotina, um pacote será enviado pela rede, destinado a outra entidade.

tolayer5(calling_entity,packet) , onde calling_entity será "0" (entrega do lado A para a camada 5) ou "1" (entrega do lado B para a camada 5), e message é a estrutura do tipo msg. Em transferência de dados unidirecional, você poderia apenas ser chamado com calling_entity igual a "1" (entrega para o lado B). Ao chamar esta rotina, os dados passarão para a aplicação.

O ambiente de rede simulado

Uma chamada ao procedimento tolayer3() envia pacotes para o meio (exemplo: dentro da camada de redes). Seus procedimentos A_input() e B_input() são chamados quando um pacote deve ser entregue do meio para a sua camada de protocolo.

O meio é capaz de corromper e perder pacotes. Ele não irá reordenar os pacotes. Quando você compilar seus procedimentos com os que são fornecidos aqui e executar o programa resultante, será necessário especificar os valores de acordo com o ambiente de rede simulado:

Número de mensagens para simular . O emulador (e suas rotinas) irá parar quando esse número de mensagens tiver sido passado para a camada 5, não importando se todas as mensagens foram, ou não, entregues corretamente. Então, você não precisa se preocupar com mensagens não entregues ou não confirmadas que ainda estiverem no seu remetente quando o emulador parar. Note que se você ajustar esse valor para "1", seu programa irá terminar imediatamente, antes de as mensagens serem entregues ao outro lado. Logo, esse valor deve ser sempre maior do que "1".

Perdas . Você deverá especificar a probabilidade de perda de pacotes. Um valor de $0,1$ significa que um em cada dez pacotes (na média) será perdido.

Corrompimento . Você deverá especificar a probabilidade de perda de pacotes. Um valor de $0,2$ significa que um em cada cinco pacotes (na média) será corrompido. Note que os campos de conteúdo da carga útil, seqüência, ack, ou checksum podem ser corrompidos. Seu checksum deverá então incluir os campos de dados, seqüência e ack.

Tracing . Ajustar um valor de tracing de "1" ou "2" imprimirá informações úteis sobre o que está acontecendo dentro da emulação (exemplo: o que ocorre com pacotes e temporizadores). Um valor de tracing de "0" desliga esta opção. Um valor maior do que "2" exibirá todos os tipos de mensagens especiais que são próprias para a depuração do meu emulador. O valor "2" pode ser útil para você fazer o debug do seu código. Mantenha em mente que implementações reais não provêem tais informações sobre o que está acontecendo com seus pacotes.

Média de tempo entre mensagens do remetente . Você pode ajustar este valor para qualquer valor positivo diferente de zero. Note que quanto menor o valor escolhido, mais rápido os pacotes chegarão ao seu remetente.

Versão bit alternante

Você escreverá os procedimentos A_output(), A_input(), A_timerinterrupt(), A_init(), B_input() e B_init() que, juntos, implementarão uma transferência pare-e-espere (exemplo: o protocolo bit alternante referido como rdt3.0 no livro Kurose) unidirecional de dados do lado A para o lado B.

Escolha um valor bem alto para a média de tempo entre mensagens da camada 5 do remetente, assim ele nunca será chamado enquanto ainda tiver pendências, isto é, mensagens não confirmadas que ele esteja tentando enviar ao destinatário. Sugiro que se escolha um valor de 1.000. Realize também uma verificação em seu remetente para ter certeza de que quando A_output() é chamado, não haja nenhuma mensagem em trânsito. Se houver, você pode simplesmente ignorar (descartar) os dados que estão sendo passados para a rotina A_output().

Não deixe de ler as "dicas úteis" para este laboratório logo após a descrição da versão Go_Back-N.

Go-Back-N deste trabalho

Você escreverá os procedimentos A_output(), A_input(), A_timerinterrupt(), A_init(), B_input() e B_init() e juntamente implementará uma transferência unidirecional de dados do lado A para o lado B, com tamanho de janela igual a 8. Seu protocolo deverá usar mensagens ACK e NACK. Consulte a versão protocolo bit alternante acima para informações sobre como obter e emulador de rede.

É altamente recomendável que você implemente primeiro o laboratório mais fácil (bit alternante) e então estenda seu código para implementar o mais difícil (Go-Back-N). Acredite, isso não será perda de tempo! No entanto, algumas novas considerações para o seu código Go-Back-N (que não se aplicam ao protocolo bit alternante) são:

A_output(message) , onde message é uma estrutura do tipo msg contendo dados para serem enviados ao lado B.

Agora sua rotina A_output() será chamada algumas vezes quando houver pendências, mensagens não confirmadas no meio – implicando a necessidade de você ter um buffer para mensagens múltiplas em seu remetente. Você também precisará do buffer devido à natureza do Go-Back-N: algumas vezes seu remetente será chamado mas não será capaz de enviar a nova mensagem porque ela cai fora da janela.

Então você deverá se preocupar em armazenar um número arbitrário de mensagens. Você pode ter um valor finito para o número máximo de buffers disponíveis em seu remetente (para 50 mensagens) e pode simplesmente abortar (desistir e sair) se todos os 50 buffers estiverem em uso de uma vez (Nota: usando os valores acima, isso nunca deverá acontecer). No mundo real, naturalmente, deveria haver uma solução mais elegante para o problema de buffer finito.

A_timerinterrupt() Esta rotina será chamada quando o temporizador de A expirar (gerando uma interrupção de temporizador). Lembre que você possui apenas um temporizador, e pode ter muitas pendências e pacotes não confirmados no meio; então, pense um pouco em como usar este único temporizador.

Consulte a versão protocolo bit alternante para uma descrição geral do que você precisa ter em mãos. Você pode querer ter uma saída para uma execução que seja grande o bastante de modo que pelo menos 20 mensagens sejam transferidas com sucesso do remetente ao destinatário (exemplo: o remetente recebe o ACK para estas mensagens), uma probabilidade de perda de $0,2$, e uma probabilidade de corrupção de $0,2$, e um nível de trace de 2, e um tempo médio entre chegadas de 10.

Dicas úteis

Soma de verificação . Você pode usar qualquer solução de soma de verificação que quiser. Lembre que o número de seqüência e o campo ack também podem ser corrompidos. Sugerimos uma soma de verificação como o do TCP, que consiste na soma dos valores (inteiro) dos campos de seqüência e do campo ack adicionada à soma caracter por caracter do campo carga útil do pacote (exemplo: trate cada caracter como se ele fosse um inteiro de 8 bits e apenas adicione-os juntos).

Note que qualquer "estado" compartilhado entre suas rotinas deve estar na forma de variáveis globais. Note também que qualquer informação que seus procedimentos precisam salvar de uma invocação para a próxima também deve ser uma variável global (ou estática). Por exemplo, suas

rotinas precisam manter uma cópia de um pacote para uma possível retransmissão. Nesse caso, seria provavelmente uma boa idéia uma estrutura de dados ser uma variável global no seu código. Note, no entanto, que se uma das suas variáveis é usada pelo seu lado remetente, essa variável não deve ser acessada pela entidade do lado destinatário, pois, no mundo real, a comunicação entre entidades conectadas apenas por um canal de comunicação não compartilha variáveis.

Há uma variável global flutuante chamada time, que você pode acessar de dentro do seu código para auxiliá-lo com diagnósticos de mensagens.

COMECE SIMPLES . Ajuste as probabilidades de perda e corrupção para zero e teste suas rotinas. Melhor ainda, projete e implemente seus procedimentos para o caso de nenhuma perda ou corrupção. Primeiro, faça-as funcionar; então trate o caso de uma dessas probabilidades não ser zero e, finalmente, de ambas não serem zero.

O emulador gera perda de pacotes e erros usando um gerador de números aleatórios. Nossa experiência passada é que geradores de números aleatórios podem variar bastante de uma máquina para outra. Você pode precisar modificar o código do gerador de números aleatórios no emulador que estamos fornecendo. Nossas rotinas do emulador possuem um teste para ver se o gerador de números aleatórios em sua máquina funcionará com o nosso código. Se você obtiver uma mensagem de erro é provável que o gerador de números aleatórios em sua máquina seja diferente daquele que este emulador espera. Favor verificar a rotina jimsrand() no código do emulador. Desculpe. Então você saberá que precisa olhar como os números aleatórios são gerados na rotina jimsrand(); veja os comentários nessa rotina.

Este trabalho é do livro. O autor disponibiliza uma página com perguntas e respostas comuns dos estudantes da cadeira que ele ministra. Para acessar: `http://gaia.cs.umass.edu/kurose/transport/programming_assignment_QA.htm`.