Universidade de Brasília
Departamento de Ciência da Computação
Redes de Computadores – Turma B
Profa. Priscila Solís Barreto

Projeto 02

Data de Entrega: 17 de Maio de 2021, até 23:55pm
Grupos de 03 alunos

1. Objetivo

Neste laboratório, o objetivo é desenvolver o código na camada de transporte e envio e recebimento para implementar um protocolo de transferência de dados confiável e simples. O laboratório trabalhará com duas versões: a) Stop and Wait e (b) Go-Back-N. O código deverá ser executado em um ambiente de hardware / software simulado.

2. Descrição

2.1 Código a ser desenvolvido

O código deve ser desenvolvido para a entidade emissora (A) e a entidade receptora (B). Apenas a transferência unidirecional de dados (de A para B) é necessária. Obviamente, o lado B terá que enviar pacotes para A para reconhecer (positiva ou negativamente) o recebimento dos dados. O código deve ser implemento na forma dos procedimentos descritos abaixo. Esses procedimentos serão chamados (e chamarão) procedimentos que emulam um ambiente de rede. A estrutura geral do ambiente é mostrada na Figura 1.

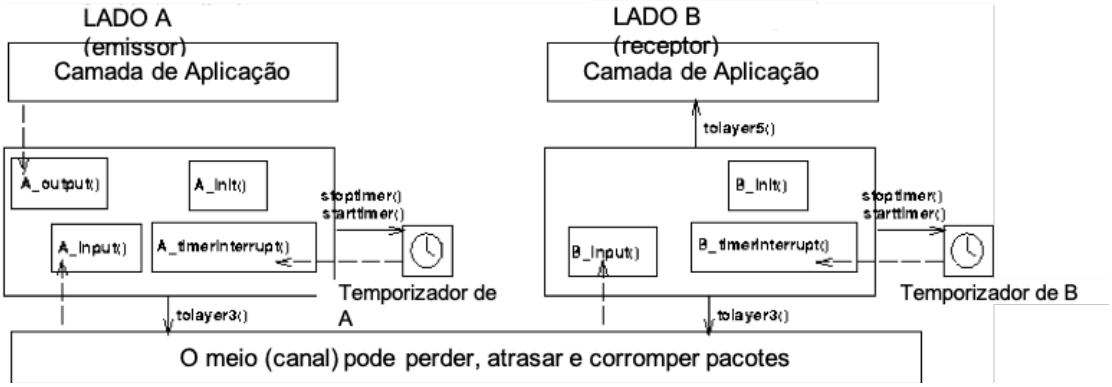

Figura 1. Estrutura do Ambiente Emulado

A unidade de dados passada entre as camadas superiores e seus protocolos é uma mensagem, que é declarada como:

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

Essa declaração e todas as outras estruturas de dados e rotinas do emulador, bem como as rotinas de `stub` (ou seja, aquelas que você deve desenvolver) estão no arquivo prog2.c, descrito posteriormente. Assim, o emissor receberá dados em blocos de 20 bytes da camada de aplicação; o receptor deve entregar pedaços de 20 bytes de dados recebidos corretamente para a camada de aplicação no lado B. A unidade de dados passada entre suas rotinas e a camada de rede é o pacote, que é declarada como:

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

As rotinas preencherão o campo de `payload` a partir dos dados da mensagem transmitidos da camada de

aplicação. Os outros campos de pacote serão usados pelos protocolos para garantir uma entrega confiável, como foi apresentado nas vídeo aulas e exercícios da disciplina. Esses procedimentos em um ambiente real fariam parte do sistema operacional e seriam chamados por outros procedimentos no sistema operacional.

As rotinas que devem ser desenvolvidas no projeto são detalhadas a seguir.

`A_output (message)`, onde mensagem é uma estrutura do tipo `msg`, contendo dados a serem enviados para o lado B. Essa rotina será chamada sempre que a camada superior do lado de envio (A) tiver uma mensagem para enviar. É função do seu protocolo garantir que os dados em tal mensagem sejam entregues em ordem e corretamente à camada superior do lado receptor.
`A_input (packet)`, onde pacote é uma estrutura do tipo `pkt`. Esta rotina será chamada sempre que um pacote enviado do lado B (isto é, como resultado de um `tolayer3()` sendo feito por um procedimento do lado B) chega ao lado A. Este é o pacote (possivelmente corrompido) enviado do lado B.
`A_timerinterrupt()`, rotina que será chamada quando o temporizador de A expirar (gerando assim uma interrupção do temporizador). Essa rotina será usada para controlar a retransmissão de pacotes. Veja as rotinas `starttimer ()` e `stoptimer ()` sobre como o cronômetro é iniciado e parado.
`A_init()`, rotina será chamada uma vez, antes que qualquer uma das outras rotinas do lado A. Server para fazer qualquer inicialização necessária.
`B_input(packet)`, em que `packet` é uma estrutura do tipo `pkt`. Esta rotina será chamada sempre que um pacote enviado do lado A (isto é, como resultado de um `tolayer3()` sendo feito por um procedimento do lado A) chega ao lado B. Este é o pacote (possivelmente corrompido) enviado do lado A.
`B_init()`, será chamada uma vez, antes que qualquer uma de outras rotinas do lado B. Pode ser usada para fazer qualquer inicialização necessária.

Este projeto pode ser desenvolvido em qualquer máquina com suporte para C. Não são utilizados recursos do UNIX ou LINUX. O código de apoio (`prog2.c`, melhor descrito na seção 2.2 e 2.4) pode ser copiado para qualquer máquina e sistema operacional.

2.2 Rotinas de Apoio

Estão disponíveis as *procedures* seguintes no código básico `prog2.c`, que podem ser chamadas pelos procedimentos/funções descritos no item 2.1. O `prog2.c` básico está disponível na plataforma Moodle da disciplina.

`starttimer (call_entity, incremento)`, em que `call_entity` é 0 (para iniciar o cronômetro do lado A) ou 1 (para iniciar o cronômetro do lado B), e incremento é um valor flutuante que indica a quantidade de tempo que passará antes que o cronômetro interrompa . O cronômetro de A só deve ser iniciado (ou interrompido) pelas rotinas do lado A, e da mesma forma para o cronômetro do lado B. Para se ter uma ideia do valor de incremento apropriado a ser usado: um pacote enviado para a rede leva em média 5 unidades de tempo para chegar ao outro lado quando não há outras mensagens no meio.
`stoptimer (call_entity)`, em que `calling_entity` é 0 (para parar o cronômetro do lado A) ou 1 (para parar o cronômetro do lado B).
`tolayer3 (call_entity, packet)`, em que `calling_entity` é 0 (para o envio do lado A) ou 1 (para o envio do lado B), e `packet` é uma estrutura do tipo `pkt`. A chamada desta rotina fará com que o pacote seja enviado para a rede, com destino à outra entidade.
`tolayer5 (call_entity, message)`, em que `calling_entity` é 0 (para entrega do lado A à camada de aplicação) ou 1 (para entrega do lado B à camada de aplicação), e a mensagem é uma estrutura do tipo `msg`. Com a transferência de dados unidirecional, você só chamaria isso com `call_entity` igual a 1 (entrega para o lado B). Chamar essa rotina fará com que os dados sejam passados para a camada de aplicação.

2.3 Simulação do Ambiente

Uma chamada ao procedimento `tolayer3 ()` envia pacotes do canal para a camada de rede. Os procedimentos `A_input()` e `B_input()` são chamados quando um pacote deve ser entregue do canal para sua camada de protocolo. O canal é capaz de corromper e perder pacotes e não reordenará os pacotes.

Ao compilar o código desenvolvido com as rotinas disponíveis e ao executar o programa resultante, será solicitado que sejam especificados os valores relativos ao ambiente de rede simulado, a saber:

- Número de mensagens a simular: a execução deve parar assim que este número de mensagens for transmitido

da camada de aplicação, independentemente de se todas as mensagens foram ou não entregues corretamente. Portanto, não há necessidade de se preocupar com mensagens não entregues no remetente quando o emulador parar. Observe que se esse valor for 1, o programa será encerrado imediatamente, antes que a mensagem seja entregue ao outro lado. Portanto, esse valor deve ser sempre maior que 1.

- Probabilidade de Perda: especificar uma probabilidade de perda de pacote. Um valor de 0,1 (10%) significaria que um em cada dez pacotes (em média) é perdido.
- Probabilidade de Pacotes Corrompidos: especificar uma probabilidade, por exemplo um valor de 0,2 significa que um em cada cinco pacotes (em média) está corrompido. Observe que o conteúdo dos campos de carga útil, sequência, ack ou soma de verificação pode ser corrompido. A verificação deve, portanto, incluir os campos de dados, sequência e ack.
- Rastreamento: um valor de rastreamento de 1 ou 2 imprimirá informações úteis sobre o que está acontecendo dentro da emulação (por exemplo, o que está acontecendo com pacotes e temporizadores). Um valor de rastreamento de 0 desativará isso. Um valor de rastreamento maior que 2 exibirá todos os tipos de mensagens estranhas que são para fins de depuração do emulador. Um valor de rastreamento de 2 pode ser útil para depurar o código.
- Tempo médio entre mensagens da camada de aplicação ao remetente: definir esae valor positivo e diferente de zero. Quanto menor o valor escolhido, mais rápido os pacotes chegarão ao remetente.

2.4 Versão STOP and Wait

Os procedimentos, `A_output ()`, `A_input ()`, `A_timerinterrupt ()`, `A_init ()`, `B_input()` e `B_init()` devem implementar o protocolo stop-and-wait (ou seja, o protocolo de bit alternado referenciado como rdt3.0 nas aulas), com transferência unidirecional de dados do lado A para o lado B e uso das mensagens ACK e NACK. É recomendável escolher um valor muito grande para o tempo médio entre as mensagens da camada de aplicação do remetente, de modo que o remetente nunca seja chamado enquanto ainda tiver uma mensagem pendente não confirmada que está tentando enviar ao destinatário. Sugere-se o valor de 1000. Também deve ser feita uma verificação no remetente para se certificar de que, quando `A_output ()` é chamado não há nenhuma mensagem em trânsito no momento. Se houver, pode-se simplesmente ignorar (descartar) os dados que estão sendo passados para a rotina `A_output()`.

O código desenvolvido deve ser integrado ao `prog2.c` em um novo arquivo chamado `prog2-nome-grupo.c`.

2.5 Versão Go-Back-N

Os procedimentos, `A_output()`, `A_input()`, `A_timerinterrupt ()`, `A_init()`, `B_input()` e `B_init()` irão implementar uma transferência unidirecional Go-Back-N de dados do lado A para o lado B, com um tamanho de janela 8. O protocolo deve usar mensagens ACK e NACK.
Recomenda-se fortemente que primeiro seja implementada a versão mais simples (stop and wait) para logo estender o código para o Go-Back-N. Algumas novas considerações para o Go-Back-N são:

- `A_output(message)`, em que `message` é uma estrutura do tipo `msg`, contendo dados a serem enviados para o lado B. `A_output()` as vezes será chamada quando houver mensagens pendentes e não confirmadas no meio - implicando que várias mensagens no remetente deverão ser armazenadas em buffer. Lembrar que há necessidade de um buffer no remetente devido a natureza do Go-Back-N: as vezes o remetente será chamado, mas não será capaz de enviar a nova mensagem porque ultrapassa o tamanho da janela. Uma forma de simplificar a implementação, é em vez de se preocupar com o armazenamento em buffer de um número arbitrário de mensagens, ter um número máximo e finito de buffers disponíveis no remetente (digamos, para 50 mensagens) e fazer com que o remetente simplesmente aborte (desista e saia) quando todos os 50 buffers estiverem em uso No mundo real, seria necessário chegar a uma solução mais elegante para o problema do buffer finito.
- `A_timerinterrupt()`, chamada quando o temporizador de A expirar (gerando assim uma interrupção do temporizador). Lembrar que há um único temporizador e pode ter muitos pacotes pendentes não confirmados no meio, então é importante pensar em uma forma inteligente de usar esse único temporizador.

3. Entrega do Relatório

Materiais a serem entregues:

- Relatório descritivo do projeto: formato habitual (arquivo pdf, com capa, introdução, descrição da solução e descrição dos testes realizados) com anexo do código implementado e documentado. É muito importante que o relatório detalhe

na descrição da solução os critérios utilizados para o desenvolvimento do código e como este foi integrado ao código básico disponível (`prog2.c`).

- Demonstração de execução (link para vídeo de até 8 minutos, mostrando a execução com os parâmetros utilizados)
  - Na demonstração, para facilitar a visualização e compreensão, os procedimentos desenvolvidos podem imprimir uma mensagem sempre que um evento ocorrer em seu remetente ou receptor (uma mensagem / chegada de pacote ou uma interrupção de cronômetro), bem como qualquer ação tomada em resposta.
  - O exemplo de execução deve ser ao menos até o ponto em que 10 mensagens sejam confirmadas (ACK'ed) corretamente no receptor, uma probabilidade de perda de 0,1 e uma probabilidade de corrupção de entre 01 e 0,3 e um nível de rastreamento de 2. Nos seus comentários no vídeo, explique como o seu protocolo se recuperou corretamente da perda e corrupção de pacotes.