Protocolo de Ligação de Dados

Relatório do 1º trabalho laboratorial

Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Grupo xx:
Francisco Rodrigues - 201305627l
João Nogueira - 201303882
Marta Lopes - 201208067

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

30 de Outubro de 2015

Conteúdo

1 Sumário

Este relatório tem como objetivo explicar o primeiro projeto, realizado para esta unidade curricular, denominado "Protocolo de Ligação de Dados". Este projeto consiste no envio de informação de um computador para outro, através do uso de porta de série. Foram assim implementados programas para ler e escrever a informação a ser enviada.

O projeto foi finalizado com sucesso, sendo que os dados foram enviados e recebidos de forma correcta. Foram também incluídos a prevenção e correção de erros ao longo da transmissão, restabelecendo a transmissão quando os erros acontecem.

2 Introdução

Iremos então descrever o nosso trabalho, que consiste em implementar um protocolo de ligação de dados, de acordo com a especificação descrita no guião, de uma forma mais teórica e pormenorizada para assim poderem ser avaliados certos aspectos que não seriam possíveis avaliar durante a apresentação na aula. O ambiente de desenvolvimento utilizado foi em PC's com *Linux*, a linguagem de programação foi C e as portas de série existentes realizavam comunicação assíncrona.

O protocolo de ligação de dados pretende assim fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um cabo de série. As funções utilizadas serão a de criação e sincronismo de tramas que irão organizar os dados a ser enviados (*framing*), a do estabelecimento/-conclusão da ligação, a numeração de tramas, o controlo de fluxo, a confirmação de envio sem erros e o controlo dos erros que poderão ser criados por *timeouts*, tramas fora da sequência esperada ou retransmissões.

O nosso relatório terá então as **secções principais** seguintes:

- **Arquitetura**: especificação dos blocos funcionais e da *interface*;
- **Estrutura do código**: API's, principais estruturas de dados, principais funções e a sua relação com a arquitetura;
- **Casos de uso principais**: identificar os principais aspectos, abordando as sequências de chamadas de funções;
- **Protocolo de ligação lógica**: identificar os principais aspectos funcionais da *linkLayer*, descrevendo a estratégia de implementação;
- **Protocolo de aplicação**: identificar os principais aspectos funcionais da *applicationLayer*, descrevendo a estratégia de implementação;
- **Validação**: testes efetuados ao programa com apresentação de resultados;
- **Elementos de valorização**: identificação dos elementos implementados, descrevendo a estratégia de implementação.

3 Arquitetura

3.1 *Application Layer* e *Link Layer*

O projeto está organizado em duas *layers* que vão ser responsáveis pela correta funcionalidade do mesmo. A *layer* que vai servir para a aplicação tem os ficheiros *applicationLayer.c* e *applicationLayer.h* e a ligação lógica está representada nos ficheiros *linkLayer.c* e *linkLayer.h* com um ficheiro *.h auxiliar, o *linkLayerAux.h*.

A **camada de ligação lógica**, contém várias funções, servindo estas para a configuração e abertura da porta de série, o envio de comandos, envio e receção de mensagens e para a realização dos processos de *stuffing* e *destuffing*.

A **camada de aplicação**, que vai depender da camada de ligação lógica utilizando algumas das suas funções, disponibilizará funções onde serão especificados o envio e receção de pacotes de dados, o envio e receção de pacotes de controlo e finalmente a leitura e a escrita do ficheiro a enviar.

3.2 *Interface*

A interface na linha de comandos está implementada nos ficheiros *cli.c* e *cli.h* que têm as funções que vão permitir a escolha de valores de alguns parâmetros referentes à transferência do ficheiro, existindo limites impostos de acordo com o guião. Os parâmetros a serem especificados pelo utilizador serão a *baudrate*, o tamanho máximo para a mensagem, o número de *timeouts* máximo no caso de falha de comunicação e a duração de cada *timeout*. Além disso, o utilizador terá também de dizer se será o transmissor ou o recetor e qual é a sua porta de série. Depois de o utilizador especificar cada um destes valores a aplicação vai iniciar, sendo que terá uma informação inicial mostrando os parâmetros com a escolha feita pelo utilizador.

4 Estrutura do código

4.1 Application Layer

A camada da aplicação está implementada nos ficheiro *applicationLayer.c* e *applicationLayer.h* e é representada por uma estrutura na qual se encontram um *char* que representa o modo do programa (Receptor/Transmissor) e um inteiro chamado debug que permite fazer o *debugging* através de prints ao longo do código.

```
1 typedef struct {
2   char status;
3
4   int debug;
5 } applicationLayer;
```

Struct linkLayer

4.2 Link Layer

É nos ficheiros *linkLayer.c*, *linkLayer.h* e *linkLayerAux.h* que está implementada a camada de ligação de dados representada por uma estrutura na qual se encontram:

- A porta de série;
- O Baud Rate;
- O número de sequência;
- O timeout;
- O número de tentativas;
- O tamanho máximo de cada pacote;
- Estrutura de estatísticas.

```
1  typedef struct {
2    int fd;
3    char port[20];
4    int baudRate;
5    unsigned int sequenceNumber;
6    unsigned int timeout;
7    unsigned int numTransmissions;
8    int packSize;
9
10   char status;
11   int debug;
12
13   Statistics * stat;
14 } linkLayer;
```

Struct linkLayer

5 Casos de uso principais

(identificação; sequências de chamada de funções)

6 Protocolo de ligação lógica

O protocolo de ligação lógica está implementado na camada *linkLayer*, camada da qual depende a camada *applicationLayer*.

6.1 Principais aspectos funcionais

- Configurar a porta de série como é pretendido;
- Repor a configuração da porta de série como originalmente após a transferência dos dados pretendidos;
- Estabelecer a ligação de dados utilizando a porta de série;
- Enviar comandos;
- Enviar/receber mensagens;
- Processo de *Stuffing* e *Destuffing* dos *packets* recebidos da camada *applicationLayer*.

6.2 Funções implementadas na *linkLayer*

6.2.1 ll_open

Esta função é responsável por configurar a porta de série com as opções pretendidas (utilizando, por exemplo, o *baudRate* escolhido pelo utilizador) e por guardar numa variável que passa por argumento à função *configure*. Depois de configurar a porta de série, do lado do emissor envia o comando SET e aguarda a resposta UA do recetor que, ao ser recebida termina a função. Utiliza o alarme para controlar os *timeouts*. Do lado do recetor aguarda a recepção do comando SET e, ao recebê-lo responde com o comando UA.

6.2.2 ll_close

Do lado do receptor, esta função começa por aguardar a recepção do comando DISC, respondendo com o mesmo comando. Após o envio deste comando, e imediatamente antes de reestabelecer as configurações originais da porta de série aguarda a recepção do comando UA por parte do emissor. Do lado do emissor, esta função começa por enviar o comando DISC e aguarda pela resposta com o mesmo comando por parte do receptor (implementando o alarme para manter controlo dos timeouts). Após a correta recepção de DISC evia o último comando do programa, comando UA. Termina por fazer *resetConfiguration*, função na qual é feito um *sleep* de um segundo por forma a garantir que as configurações originais da porta de série não são restabelecidas antes de que toda a informação tenha sido passada.

6.2.3 llwrite

Esta função começa por alocar memória num *buffer* no qual a informação será organizada (antes do processo de *stuffing*). Após alocar a memória necessária começa por atribuir os valores de FLAG, A, C e BCC de acordo com o número de sequência da trama. A partir de BCC é copiada para o buffer toda a informação recebida por argumento desta função da *applicationLayer* correspondente ao *packet* a enviar. Antes de terminar o preenchimento do *buffer* coloca o BCC2 ('ou' exclusivo dos

octetos do *packet*) e a FLAG para terminar. Depois de ter a trama preenchida e antes de começar o processo de envio, envia o *buffer* para a função *stuff* que retorna um *buffer* com a trama pronta para envio. Envia a trama através da porta de série e aguarda a resposta por parte do recetor, o que vai determinar o caminho a seguir pela função, que pode variar entre terminar a função pois houve a recepção de RR, ou re-enviar a mesma trama por ter sido rejeitada. A mesma trama pode ser re-enviada por ocorrência de *timeout*, o que também é controlado nesta função através da implementação de alarmes. Esta função pode também terminar pela ocorrência de *timeouts* maior do que o número máximo definido previamente pelo utilizador.

6.2.4 llread

A função *llread* começa por alocar memória para o *buffer* que vai ser recebido, entrando imediatamente a seguir num ciclo do ual apenas sai quando algo é lido da porta de série. Após a leitura o *buffer* lido é passado para a função responsável pelo processo de *destuffing* que retorna a trama de Informação recebida "descodificada". Os valores de BCC e de BCC2 são verificados por forma a garantir que a trama foi recebida sem erros, caso o tenha sido é interpretado o número de sequência e se tudo for o pretendido é enviado o comando RR, caso contrário é enviado o comando REJ, pedindo que a mesma seja enviada novamente.

7 Protocolo de aplicação

O protocolo de aplicação está implementado na camada applicationLayer, camada esta que depende, como dito anteriormente, da camada linkLayer.

7.1 Principais aspectos funcionais

- Envio e recepção de pacotes de dados;
- Envio e recepção de pacotes de controlo;
- Leitura do ficheiro a enviar;
- Escrita do ficheiro a enviar.

7.2 Funções implementadas na *applicationLayer*

A primeira função a ser implementada em *applicationLayer* foi a função **main** que chama e gere as diversas funções da *Interface* para o programa inicializar com as configurações desejadas pelo utilizador. Vai também chamar as funções *ll_open, ll_close, readFile e sendFile.*

De seguida, é implementada a função **initStat** que será responsável por inicializar os valores da *struct* que vai conter as estatísticas finais do programa.

É implementada também a função **fillLinkLayer** que inicializa a *struct linkLayer* onde estão atributos como *baudRate, timeout*, etc.

A próxima função implementada foi a **createFirstControlPacket** que vai gerir o primeiro pacote de controlo, é neste pacote que é enviado o tamanho do ficheiro a enviar bem como o nome do mesmo.

Foram também implementadas as funções **sendFile** e **readFile**. Na função *sendFile* vai ser lida toda a informação do ficheiro a enviar e guarda-la num *buffer* de chars, a partir do qual a informação será lida para o envio. Após a leitura chama a função responsável pelo envio do primeiro *control packet* e envia-o para o recetor, o *llwrite*. Depois do envio do primeiro *control packet* é calculado o número de pacotes necessários a enviar tendo em conta o tamanho máximo definido pelo utilizador. Os pacotes de dados vão sendo criados com a informação necessária e com os dados do ficheiro em si, sendo chamada a função *llwrite* para enviar os vários pacotes individualmente. Em *readFile* o ficheiro vai sendo recebido e escrito em disco. Cada pacote vai ser interpretado, depois de passar pela função *llread*, e vão ser tomadas medidas diferentes dependendo de que pacote for (primeiro, último ou um pacote de dados). Se for o primeiro *control packet* é guardada a informação do nome e tamanho do ficheiro. Se for o último é criado o ficheiro e toda a informação lida é escrita em disco. Se for um pacote de dados a informação é adicionada ao final do *buffer* de leitura.

8 Validação

Para verificar se a transferência do ficheiro *pinguim.gif* foi efetuada com sucesso decidimos colocar uma verificação de inicio e fim de envio, durante o envio do ficheiro adicionamos uma barra de progresso que iria sendo actualizada à medida que os packets iam sendo enviados.

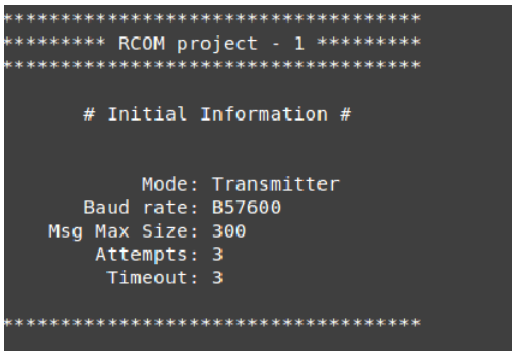

Figura 1: Informações iniciais - Modo Transmissor

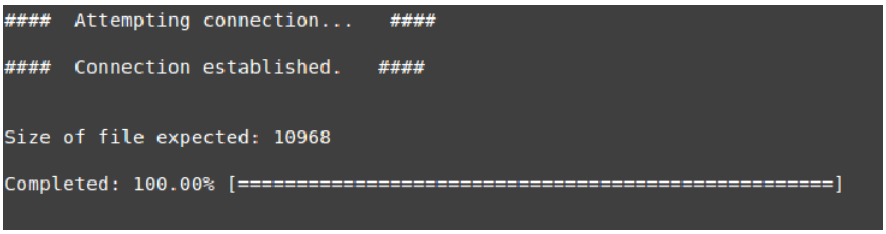

Figura 2: Inicio do envio e barra de progresso - Modo Transmissor

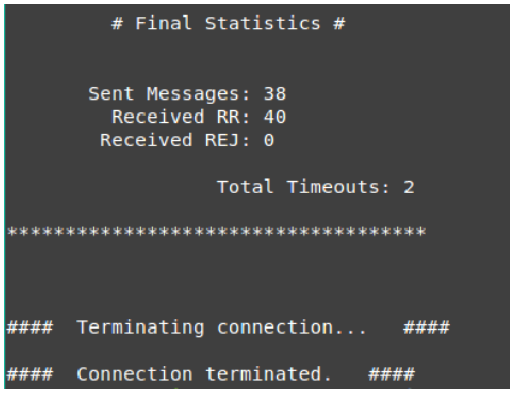

Figura 3: Fim do envio e estatisticas - Modo Transmissor

Adicionamos também o modo *debug* em que iriam ser feitos mais *prints* à medida que o ficheiro

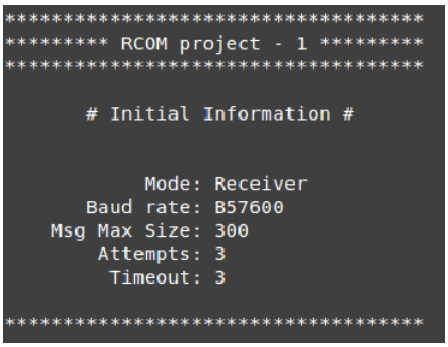

Figura 4: Informações iniciais - Modo Recetor

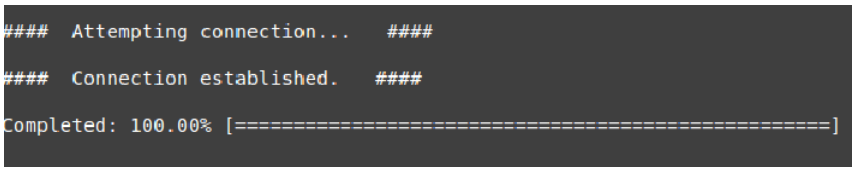

Figura 5: Inicio do envio e barra de progresso - Modo Recetor

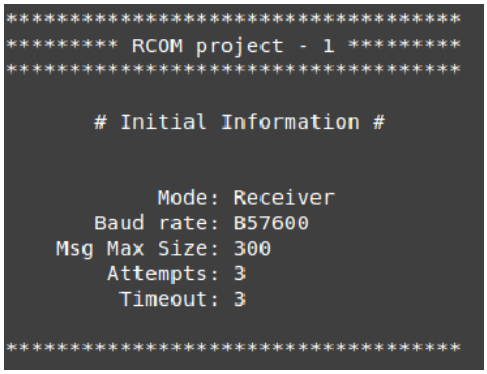

Figura 6: Fim do envio e estatisticas - Modo Recetor

ia sendo enviado, sendo também possível verificar quando houvesse *timeouts* ou envio/receção de REJ's.

9 Elementos de valorização

Selecção de parâmetros pelo utilizador

Ao executar o programa irá inicializar uma interface onde o utilizador vai escolher se será o transmissor ou o recetor, qual a porta de série que irá usar, *baud rate*, tamanho máximo do campo de informação de tramas I, número máximo de retransmissões e intervalo de *timeout*.

Implementação de REJ

```
if (ll->debug == TRUE) {
      printf("\nBcc2-2: 0x%x", bcc2);
    }

    if ((*(*buffer + 3) != (*(*buffer + 1) ^ *(*buffer + 2)))  (*(*buffer + sizeOfInfoRead - 2) != bcc2)) {
        if ((*ll).sequenceNumber == 0) {

            if (ll->debug == TRUE) {
              printf("\n****\n2nd ERROR receiving 1, wanted 0, sending REJ 0\n****");
            }
            ll->stat->numSentREJ++;
            writeMsg(&(ll->fd), A_1, C_REJ_0);
            free(buffer_2);
            return -5;
        } else if ((*ll).sequenceNumber == 1) {
            if (ll->debug == TRUE) {
              printf("\n****\n2nd ERROR receiving 0, wanted 1, sending REJ 1\n****");
            }
            ll->stat->numSentREJ++;
            writeMsg(&(ll->fd), A_1, C_REJ_1);
            free(buffer_2);
            return -4;
        }
    }
    else if ((*ll).sequenceNumber == 0 && *(*buffer + 2) == C_0) {
        writeMsg(&(ll->fd), A_1, C_RR_1);
        ll->stat->numSentRR++;
        (*ll).sequenceNumber = 1;
    } else if ((*ll).sequenceNumber == 1 && *(*buffer + 2) == C_1) {
        writeMsg(&(ll->fd), A_1, C_RR_0);
        ll->stat->numSentRR++;
        (*ll).sequenceNumber = 0;
    } else if ((*ll).sequenceNumber == 0 && *(*buffer + 2) == C_1) {
        writeMsg(&(ll->fd), A_1, C_REJ_0);
        if (ll->debug == TRUE) {
          printf("\n****\nERROR receiving 1, wanted 0, sending REJ 0\n****");
        }
        ll->stat->numSentREJ++;
        free(buffer_2);
        return -2;
    } else if ((*ll).sequenceNumber == 1 && *(*buffer + 2) == C_0) {
        writeMsg(&(ll->fd), A_1, C_REJ_1);
        if (ll->debug == TRUE) {
          printf("\n****\nERROR receiving 0, wanted 1, sending REJ 1\n****");
        }
        ll->stat->numSentREJ++;
        free(buffer_2);
        return -3;
    }
```

Envio de REJ aquando do erro do tipo BCC2

Verificação da integridade dos dados pela Aplicação

O programa verifica o tamanho do ficheiro recebido.

Registo de Ocorrências

É registado na *struct* de estatisticas o número de tramas I (re)transmitidas/recebidas, número de ocorrências de *timeouts* e número de REJ enviados/recebidos.

10 Conclusões

Achamos que o objetivo principal para este projecto foi alcançado com sucesso. Durante os primeiros dias não conseguiamos interpretar bem o guião de trabalho fornecido, mas em conjunto com outros grupos tiramos as dúvidas existentes e a partir daí todos os pontos do guião foram bem compreendidos.

Podemos dizer que terminamos com um trabalho bem organizado, em duas camadas, como foi pedido em que a *applicationLayer* vai depender da *linkLayer*. Recorremos ao uso de alarmes

A realização deste projecto ajudou-nos a entender melhor os conceitos desta unidade curricular e a aprofunda conhecimentos no que toca a comunicação em redes a partir do uso de portas de série.

11 Anexos