Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465, Porto, Portugal

Protocolo de Ligação de Dados

MIEIC – Mestrado Integrado em Engenharia Informática e Computação

RCOM – Redes de Computadores

Grupo:

João Paulo Moreira Barbosa – up201406241

Miguel Lira Barbeitos Luís - up201405324

Miriam Cristiana Meireles Campos Gonçalves – up201403441

Ano letivo 2016/2017

Sumário

Este trabalho foi desenvolvido no âmbito da unidade curricular, do primeiro semestre do terceiro ano, Redes de Computadores (RCOM), tendo como objetivo desenvolver um “Protocolo de Ligação de Dados”, onde se pretende transferir ficheiros entre dois computadores ligados via porta de série.

No fim do desenvolvimento, o projeto foi bem-sucedido, sendo assim possível a transferência de ficheiros entre dois computadores, chegando os ficheiros ao computador de destino sem erros e perdas de informação.

Índice

1. Introdução

O desenvolvimento do projeto foi proposto no decorrer do terceiro ano do Mestrado Integrado em Engenharia Informática e Computação (MIEIC) no âmbito da unidade curricular de Redes de Computadores.

O objetivo principal deste trabalho consiste na implementação do Protocolo de Ligação de Dados, sendo este testado com transferência de dados entre dois computadores ligados por cabos de porta série. Utilizou-se a linguagem de programação C e o sistema operativo Debian durante o desenvolvimento do trabalho.

Neste relatório será possível obter informação acerca da estruturação e organização do programa, para clarificar o funcionamento interno da aplicação. É também apresentada a forma correta de utilizar as funcionalidades fornecidas.

2. Arquitetura

A arquitetura presente na aplicação pode ser dividida em duas partes:

A **camada de ligação** utiliza um mecanismo *ARQ Stop-and-wait*, e é responsável pela conexão entre ambos os computadores, estabelecendo o início e o fim da ligação, e sincronizando e numerando as tramas a serem enviadas. Além disso, trata de eventuais erros que surjam na transferência dos dados.

A **camada da aplicação** é uma camada de mais alto nível e é responsável pelo tratamento da informação nos ficheiros, utilizando a camada de ligação de dados.

Do lado do emissor, esta camada deve ler a informação do ficheiro, e usar a camada de ligação para enviar essa mesma informação dividida em tramas. O recetor deve receber as várias tramas, fazendo uso, também, da camada de ligação e guardá-las no ficheiro designado.

2.1. Estrutura do código

Para a camada de ligação de dados existem quatro funções principais responsáveis pela correta ligação (fig.1 anexo II).

Ao nível da camada de aplicação, temos uma estrutura (fig.2 anexo II), que vai guardar o modo de ligação (transmissor ou recetor), o descritor da porta série, o nome do ficheiro, os dados a transferir, o tamanho do nome do ficheiro, e o tamanho do ficheiro.

Quanto às funções principais desta camada temos quatros funções: appopen, appwrite, appread, appclose (fig.3 anexo II).

No total o programa contém 11 ficheiros, devidamente separados e estruturados, para que a compreensão e leitura do código se torne mais percetível.

- **Alarm.h** e **Alarm.c** - responsáveis pela gestão de alarmes;

- **Application.h** e **Application.c** - contêm todo o código que diz respeito à camada de aplicação;
- **DataLink.h** e **DataLink.c** - contêm todo o código pertencente à camada de ligação de dados;
- **Settings.h** e **Settings.c** - responsáveis pela interface com o utilizador, e por guardar as definições pretendidas;
- **Statistics.h** e **Statistics.c** - guardam os contadores de eventos, que produzem as estatísticas finais;
- **transfer.c** - contém o corpo principal do programa.

3. Casos de uso principais

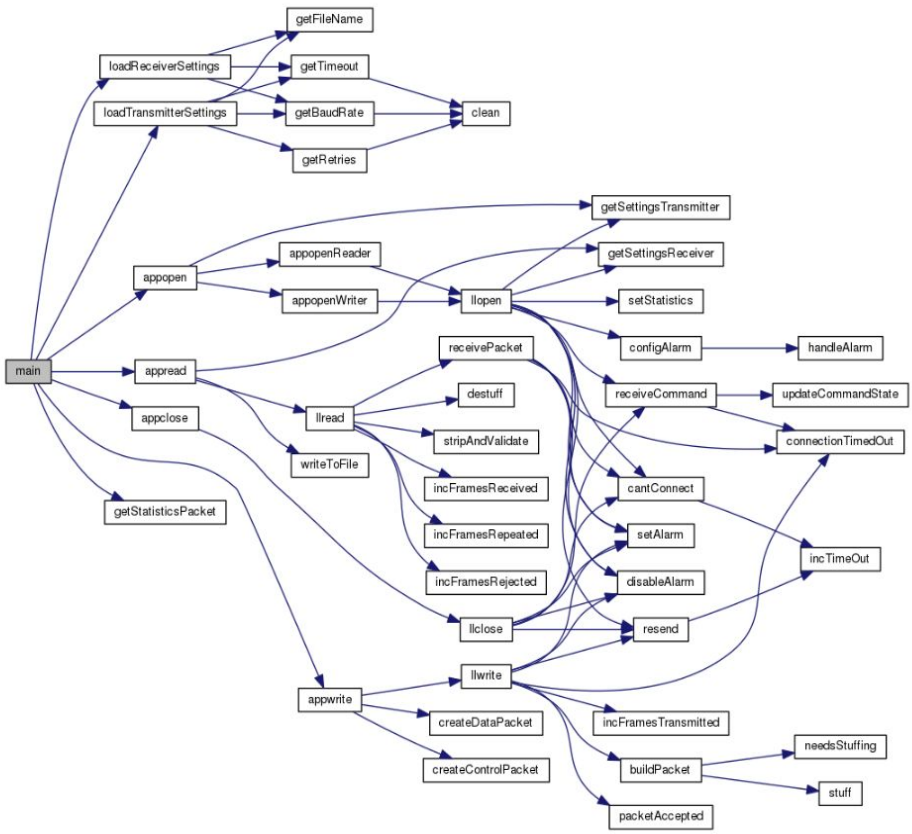
A estrutura do código foi pensada para permitir ao utilizador tirar máximo partido das funcionalidades do programa, de forma simples. Por isso, as principais funções com que estes interagem são:

Emissor: *loadTransmitterSettings; appopen; appwrite; appclose; getStatisticsPacket.*

Recetor: *loadReceiverSettings; appopen; appread; appclose; getStatisticsPacket;*

Percebe-se, portanto, que são necessárias somente 5 funções (para cada computador), para aproveitar, na sua totalidade, o que aplicação tem para oferecer.

Para maior detalhe, apresenta-se a sequência de chamada de funções do programa:



4. Protocolo

4.1. Protocolo de Ligação Lógica

A camada de ligação é a camada de mais baixo nível, que é responsável pela transferência e gestão de informação.

Ao ser iniciado o programa, o emissor vai chamar a função *llopen()*, onde é configurada a porta de série. No caso do emissor, este deve também enviar o comando SET, e esperar pela resposta do recetor (comando UA).

```
    if (status == TRANSMITTER) {
                configAlarm(settingsT.retries, settingsT.timeout);
                 write(fd, SET_PACKET, COMMAND_LENGTH); // Send SET to
receiver
                setAlarm(resend, (char *) SET_PACKET, COMMAND_LENGTH);


                type = UA_SENDER;
                if (receiveCommand(type) != 0) {
                                fprintf(stderr, "Can't connect to the
receiver. Please try again later.\n");
                        return -1;
                }
                disableAlarm();
            }
```

Em caso de erro na resposta recebida, ou em caso de ter passado o tempo estipulado, existe tentativa de retransmissão do comando SET. Processo este repetido tantas vezes quantas o número de retransmissões definido na interface, pelo utilizador. Caso o número retransmissões tenha sido ultrapassado, o programa deve terminar imediatamente.

Nesta fase, o recetor é complementar ao emissor, pelo que deve esperar pelo comando SET, e depois enviar o respetivo comando UA.

```
else if (status == RECEIVER) {
                configAlarm(0, settingsR.timeout);
                setAlarm(cantConnect, "", 0);


                type = SET;
                if (receiveCommand(type) != 0) {
                        fprintf(stderr, "Can't connect to the
sender. Please try again later.\n");
                       return -1;
                }
                disableAlarm();


                write(fd, UA_SENDER_PACKET, COMMAND_LENGTH);
        }
```

O envio de tramas I, tramas de dados, é feito usando a função *llwrite()*. As tramas são construídas a partir dos argumentos passados a *llwrite()*, sendo estes *buffer* e *length*, que representam a informação a ser transmitida e o seu tamanho, respetivamente.

A construção da trama a enviar passa por colocar os cabeçalhos corretos na cabeça e na cauda do *buffer*, assim como proceder ao stuffing.

No envios de tramas I, também existe retransmissão da trama - se não for recebida nenhuma resposta no tempo estipulado, e enquanto não se esgotarem o número máximo de reenvios possíveis.

Sempre que uma trama de informação é enviada, o emissor fica à espera da resposta do recetor: se a resposta recebida for ***RR***, então significa que a trama foi recebida com sucesso, mas se a resposta for ***REJ***, então a trama deve ser retransmitida.

Por outro lado, a receção de tramas I (por parte do recetor), é feita usando a função *llread()*. Esta funcionalidade tem a responsabilidade de receber a informação enviada pelo transmissor (usando uma máquina de estados), proceder ao *destuffing* da trama, verificar a integridade da informação e enviar uma resposta ao emissor, de acordo com a validade da trama. Esta validade é verificada e assegurada nos *BCCs*[1], passados nos cabeçalhos da trama.

Dependendo da integridade da informação, a resposta a enviar ao emissor será: ***RR*** se a trama é válida **ou** se a trama é repetida; ***REJ*** se nenhuma das anteriores ocorrer.

Note-se, no entanto, que em caso de trama repetida, a função *llread*, internamente, não a considera válida.

```
r = stripAndValidate(buffer, frame, frameSize, R);
if (r == 0) {  // valid packet
        validPacket = TRUE;
        suControl = (BUILD_R_CONTROL(R) ^ C_RR_SUFFIX);
        R = (R + 1) % 2;
}
else if (r == 1) {  // repeated packet
        validPacket = FALSE;
        suControl = (BUILD_R_CONTROL(R) ^ C_RR_SUFFIX);
}
else {  // invalid packet
        validPacket = FALSE;
        suControl = (BUILD_R_CONTROL(R) ^ C_REJ_SUFFIX);
}
```

A função *llclose()* é responsável por fechar as comunicações entre o emissor e o recetor, e repôr a configuração da porta de série.

Nesta fase, o emissor deve enviar o comando DISC, e seguidamente esperar pelo comando DISC do recetor. Caso não seja devolvido o comando DISC, o programa

[1] No guião do trabalho era somente pedido para verificar a integridade de *BCC2*. Neste projeto é também verificado o *BCC1*, para garantir, inequivocamente, que somente tramas repetidas são descartadas .

terminará com erro. Se DISC for recebido, o emissor envia a trama UA para o recetor e termina com sucesso. Tal como as outras funções, todas usam alarmes para gerir os *time-outs*.

```
if (status == TRANSMITTER) {
                write(fd, DISC_SENDER_PACKET, COMMAND_LENGTH);


                setAlarm(resend, (char *) DISC_SENDER_PACKET, COMMAND_LENGTH);
                type = DISC_RECEIVER;
                if (receiveCommand(type) != 0) {
                       fprintf(stderr, "Can't connect to the receiver. Please try again later.\n");
                        return -1;
                }
                disableAlarm();


                write(fd, UA_RECEIVER_PACKET, COMMAND_LENGTH);
}
```

Tal como acontecia na função *llopen()*, o recetor deve complementar as ações do emissor.

4.2. Protocolo da Aplicação

A camada da aplicação é a camada que é responsável pela interface e é onde o programa inicia e chama as funções do protocolo de ligação lógica.

É inicializada a camada da aplicação com a função *appopen()* onde é inicializada a respetiva estrutura, recorrendo à função *llopen()*.

```
app->filedes = llopen(path,oflag, TRANSMITTER);
app->filedes = llopen(path,oflag, RECEIVER);
```

No caso de ser emissor, *appopen()* vai chamar a função *appopenWriter()*, que abre e processa o ficheiro que vai ser transferido.

No caso do recetor, é chamada a função *appopenReader()* que vai apenas estabelecer a ligação com o emissor.

Quando o emissor chama a função *appwrite()* é criado um pacote de controlo de início de transferência que vai ser enviado ao recetor para este saber que se vai dar início à transferência dos dados.

De seguida, vão ser criado pacotes de dados, que serão enviados, um por um, ao recetor, com auxílio da função *llwrite()*.

```
for (i = 0; i < nrPackets; i++) {
        (...)
        createDataPacket(frame, &app.buffer[fileBytesSent], size);
        llwrite(app.filedes, frame, size + DATA_HEADER_SIZE);
        (...)
}
```

Quando todos os pacotes de dados tiverem sido enviados, é criado e depois enviado um pacote de controlo de fim de transferência (de forma semelhante ao pacote de início).

Quando a função *appread()* é invocada, são dadas as respostas aos pacotes/tramas recebidas e é lido aquilo que está a ser recebido com auxílio da função *llread()*.

Antes disso, já foi aberto o ficheiro onde se vai guardar a informação lida e faz-se a escrita nesse mesmo ficheiro.

A função appclose() é invocada no fim do protocolo de transferência, e apenas vai chamar a função *llclose()*, para dar como terminada a ligação entre os computadores, fechando a porta série. Antes de terminar são impressas todas as estatísticas do programa.

5. Validação

Para testar todo o código desenvolvido desde o início do ano, foram feitos alguns testes. Foi enviado de um computador para outro o *pinguim.gif*; foram enviadas imagens *JPEG* e *PNG* e também um ficheiro *gif* de maior tamanho.

Em fases mais iniciais de desenvolvimento, houve um uso extensivo de prints, para efeitos de debugging, assim como uma escolha cuidadosa da informação a processar e transferir - para garantir que as tramas eram transmitidas corretamente (cabeçalhos e stuffing), e recebidas e tratadas também corretamente (destuffing e validação).

É de salientar que em todos os testes efetuados foram introduzidos ruídos e foi desligado o cabo de porta série, verificando-se sempre sucesso.

6. Elementos de Valorização

No nosso projeto foram implementados os elementos de valorização seguintes:

- ✓ **Seleção de parâmetros pelo utilizador:** é iniciada uma interface aquando do começo do programa, que permite ao utilizador escolher certos parâmetros

essenciais ao correto funcionamento do programa. São entre eles o *baudrate*, número de retransmissões e intervalo de *timeout*.

É nas duas estruturas SettingsReceiver e SettingsTransmitter (fig.4 anexo II) onde são guardados os argumentos selecionados pelo utilizador: o *baud rate*, o intervalo de *timeout*, o número de tentativas de retransmissão, o nome do ficheiro a enviar e o nome do ficheiro a ser guardado, que são essenciais à ligação e transferência dos dados.

- ✓ **Implementação de REJ:** REJ será uma trama enviada pelo recetor ao emissor quando os dados recebidos no recetor contêm um erro no campo BCC1 e BCC2.

- ✓ **Registo de Ocorrências:** Durante o correr do programa são registadas certas informações, que irão ser mostradas ao utilizador aquando do término do programa. São mostrados, por exemplo, o número de *timeouts* ocorridos e o número de tramas recebidas/enviadas, rejeitadas e reenviadas.
  Existe, para que seja possível visualizar as ocorrências, a estrutura StatisticsPacket (fig. 5 anexo II). Serve para guardar informação sobre os principais eventos na troca de informação, durante o programa.

- ✓ **Verificação da integridade dos dados pela aplicação:** o programa verifica o tamanho do ficheiro, isto é, se o tamanho real coincide com o tamanho do pacote recebido no recetor. Verifica também se existe perda de pacotes ou se existem pacotes duplicados.

7. Conclusões

Durante a realização do trabalho a leitura e compreensão do guião tornou-se essencial para que fossem cumpridos todos os detalhes e objetivos pedidos. Tornou-se também uma necessidade estruturar e organizar o código, facilitando assim a correção de erros, a compreensão de todo o projeto e, uma vez que o guião continha uma quantidade substancial de pormenores, permitiu que nenhum deles fossem esquecidos.

Devido à organização do código e, após, várias e longas fases de testes serem feitas ao programa o grupo acredita que este protocolo foi cumprido com sucesso e robustez. Todos os requisitos foram cumpridos e ainda foram implementados elementos de valorização.

Em suma, este trabalho permitiu uma melhor compreensão do funcionamento da comunicação entre computadores, assim como todos os detalhes e fases envolvidas neste processo. Foi também possível assimilar melhor os conteúdos dados nesta unidade curricular e aperfeiçoar os conhecimentos na linguagem C.

8. Anexo I

O código fonte encontra-se numa pasta .zip

9. Anexo II

Estruturas e APIs da aplicação:

```
int llopen(const char *path, int oflag, int status);
int llwrite(int fd, char *buffer, int length);
int llread(int fd, char *buffer);
int llclose(int fd, int status);
```

Figura 1 - API da camada de ligação de dados

```
struct Application {
        int status;
        int filedes;
        char *fileName;
        char *buffer;
        int fileNameLength;
        unsigned int fileSize;
};
```

Figura 2 - Estrutura da camada da aplicação

```
int appopen(struct Application *app, const char *path, int oflag, int status);
int appwrite(struct Application app);
int appread(struct Application app);
int appclose(struct Application app);
```

Figura 3 - API da camada da aplicação

```
struct SettingsReceiver {
  char fileName[BUFFER_LENGTH];
  int timeout;
  int baudrate;
};
```

```
struct SettingsTransmitter {
  char fileName[BUFFER_LENGTH];
  int retries;
  int timeout;
  int baudrate;
};
```

Figura 4 - Estruturas para guardar as definições

```
struct StatisticsPacket {
  int framesTransmitted;
  int framesReceived;
  int framesRepeated;
  int framesRejected;
  int timeOutCounter;
};
```

Figura 5 - Estruturas para guardar as estatísticas