Universidade do Porto

Faculdade de Engenharia

FEUP

Trabalho Prático n.º 1

Protocolo de Ligação de Dados

Redes de Computadores (RCOM)

Ano lectivo 2010/2011 - 1º Semestre

Nuno Machado Matos – 080509140

Rolando Emanuel Lopes Pereira – 080509150

Tiago Carvalhido Morim Casanova – 080509124

Tiago Daniel Sá Cunha - 080509142

11 de Novembro de 2010

Sumário

O trabalho que está a ser analisado neste relatório incide sobre a transferência de informação entre dois processos, em computadores diferentes, através do uso da porta de série. Para tal, foram implementados dois programas diferentes com os propósitos de escrever e ler informações transferidas.

O trabalho foi concluído de forma total, sendo que cada programa respeitou a sequência de comandos enviados e recebidos, previne e corrige situações de erro e estabelece uma interface simples e eficaz com o utilizador. Em termos de valorizações, o nosso programa implementa todas as valorizações sugeridas, à excepção do reenvio do comando SET e posterior reinício do processo.

Introdução

Este trabalho, a ser desenvolvido no âmbito da unidade curricular de Redes de Computadores (RCOM) do 3º ano do Mestrado Integrado em Engenharia Informática e Computação (MIEIC) da Faculdade de Engenharia da Universidade do Porto (FEUP), tem como objectivos principais a implementação de um protocolo de ligação de dados e a execução e teste de uma aplicação simples de transferência de ficheiros.

O trabalho a realizar consiste no desenvolvimento de uma aplicação, composta por dois programas, usada para o envio de um ficheiro de um computador para outro através do porto de série. Pretende-se que a transferência seja fiável sendo eventuais erros que possam surgir durante a transmissão (ex. falha na ligação; envio dados/informação em duplicado) detectados e corrigidos.

O relatório encontra-se dividido em dez secções, nomeadamente a introdução presente, a descrição da Arquitectura do programa, a estruturação do código implementado, a apresentação dos diagramas de casos de uso, as elucidações dos protocolos de ligação lógica e do protocolo da aplicação, a validação da nossa solução para a resolução do problema pretendido com este trabalho, os elementos de valorização introduzidos, as conclusões finais do trabalho e nos anexos apresentamos o código por nós desenvolvido.

Arquitectura

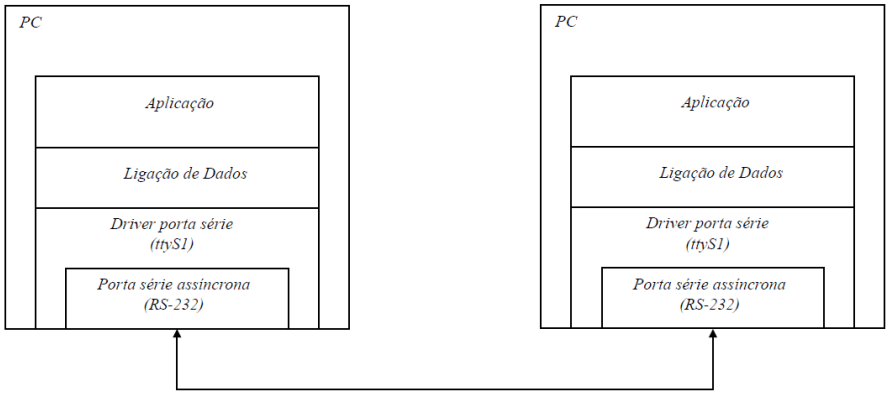
Os nossos programas apresentam uma interface de camadas, em que conforme o utilizador se identifique como emissor ou receptor, o utilizador será o primeiro a interagir ou não, respectivamente.

Cada programa tem uma camada que implementa uma ligação lógica e outra que serve a aplicação de transferência de ficheiros.

A camada lógica é responsável porimplementar o protocolo de ligação lógica que usa Stop & Wait ARQ (Automatic Repeat ReQuest) para prevenção de erros e retransmissão de tramas.

Por sua vez, a camada de aplicação é responsável, no emissor, pela separação do ficheiro em tramas e envio destas tramas através da ligação lógica, e no receptor é responsável por criar um novo ficheiro exactamente igual ao enviado e anexar, a cada iteração, a informação correspondente ao ficheiro, contida na trama recebida.

Entre ambas estas camadas existem ainda as camadas responsáveis pela interacção directa na porta de série assíncrona RS-232, nomeadamente a camada do driver que permite o uso das funções da API do Linux para leitura e escrita de tramas.



Ao nível das interfaces estabelecidas com o utilizador, de destacar que no receptor são expressas no ecrã variadas mensagens, quer sejam de sucesso na recepção da trama, de erro e pedido de reenvio por parte do emissor, bem como dados estatísticos tais como o nome e tamanho do ficheiro a ser transferido, o número da trama recebida, o número de tramas rejeitadas, entre outras.

Por sua vez, no emissor, para além de requisitar o tamanho das tramas de informação, o número máximo de retransmissões e o intervalo de tempo para o alarme, e de imprimir no ecrã o número da trama transmitida ou o número de alarmes já ativados, mostra mensagens com informação útil ao utilizador bem como estatísticas relativas à transmissão aquando o término da mesma.

Estrutura do código

Neste projecto, para além das funções da API do Linux, utilizou-se o driver da porta série (ttyS0 ou ttyS1), que permitia o acesso para leitura e escrita na porta de série através das funções read() e write() da APi do Linux, para depois ser entregue, por meio de um cabo de série na porta de série do outro computador.

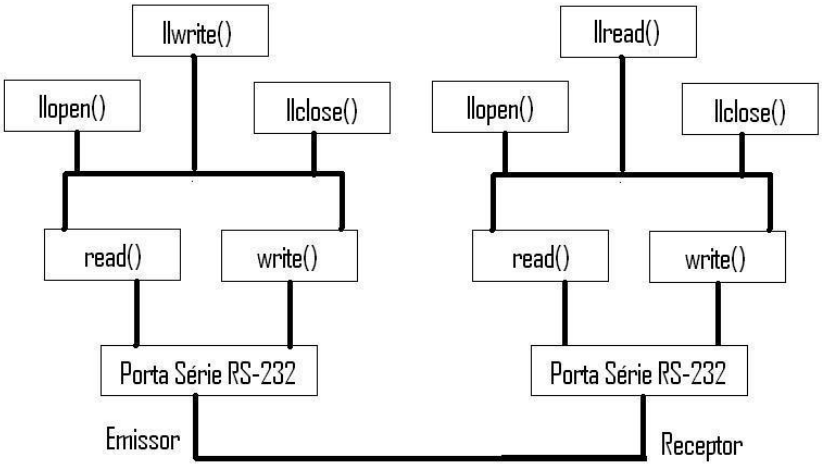
Neste projecto as estruturas de dados usadas foram apenas cadeias de caracteres (char*) para as tramas recebidas e enviadas, quer entre os processos, nas tramas, quer entre os níveis da aplicação e, para a aplicação foi usado, também, a estrutura FILE* tanto no acesso aos dados do ficheiro original como na criação de uma cópia do ficheiro no computador de destino.

As funções mais importantes no protocolo de ligação lógica seguem uma sequência lógica, na medida em que é efectuado o llopen() nos dois processos e apenas quando estes se verificarem, se introduz um ciclo de llwrite() por parte do emissor e de llread() por parte do receptor, até ser chamada a função de llclose() que assinala o final de transmissão e faz terminar ambos os processos.

- int llopen(int fd);
- int llwrite(int fd,char* buf, int sizebuf);
- int llread(int fd, char* result);
- int llclose(int fd).

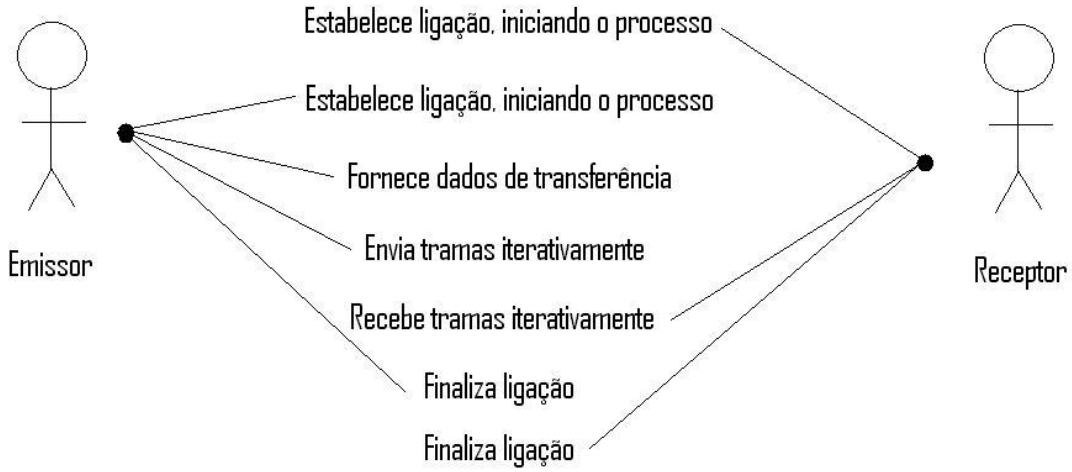
Quanto ao protocolo da aplicação, no receptor, caracteriza-se por um ciclo que recebe, a cada iteração, uma trama do nível do protocolo e conforme uma determinada codificação, define o inicio da transmissão, o envio de informação para o ficheiro e o término da transferência de dados para o ficheiro.

No emissor, o processo inicia-se com a abertura do ficheiro e a respetiva recolha de informação sobre o mesmo, criando um pacote de controlo indicando o início de ligação que é enviado após a introdução de alguns dados essenciais para a inicialização da transmissão, culminando com o envio de um outro pacote de controlo de fecho da ligação. Estes são separados por uma sequência de tramas de informação recolhida diretamente do ficheiro, sendo estas última intercaladas aleatoriamente por tramas com erros.



Casos de uso principais

Abaixo definem-se os casos de uso de ambos os processos, sendo que a ordem dos casos se encontra temporalmente ordenada e as atribuições são realizadas para quem comete as acções e quando.



Protocolo de ligação lógica

Relativamente ao protocolo de ligação lógica, existem três funções: *llopen(int fd)*, *llread(int fd, char* result)* ou *llwrite(int fd,char* buf, int sizebuf)* e *llclose(int fd)*. Apesar de o número de argumentos variar entre as funções, existe um argumento em comum: o '*file descriptor*' que abre a porta de série.

Emissor

Em *llopen()* após a inicialização do alarme e do envio da trama SET instanciada anteriormente conforme descrito no guião do trabalho, o programa fica à espera da recepção de

uma trama UA enviada pelo receptor. Se isto não acontece, o alarme é ativado e o processo reinicia-se tantas vezes quantas as definidas pelo utilizador no início do programa, ao fim das quais o programa termina com indicação do erro. Caso contrário, é verificada a trama recebida, e se esta corresponder a uma trama UA o processo continua normalmente.

Em *llwrite()* o processo inicia-se por definiros primeiros campos da trama a enviar, nomeadamente a flag 0x7E, o campo de endereço, o campo de controlo e os campos de protecção independentes BCC1 e BCC2, fazendo um ou-exclusivo de todos os dados na trama. Apesar de este último ser colocado depois dos campos de dados, tem de ser calculado antes da aplicação da transparência nos mesmos, o que é feito logo de seguida a todas os valores de flag ou ESC encontrados, e finalizado pela introdução da flag no final da trama.

Seguidamente é enviado o pacote ao receptor e ativado o alarme, ficando o programa à espera de uma resposta. Se isso não acontece, o alarme é ativado e como em *llopen()*, o processo reinicia tantas vezes quantas as definidas pelo utilizador no início do programa, ao fim das quais este termina com indicação do erro. Quando a resposta recebida não é nula, o alarme é desativado e a mesma é analisada, verificando a flag e o BCC1 da trama. Após a validação dos mesmo é verificado se a trama contém um valor RR ou REJ.

Se um valor REJ é recebido tal é indicado ao utilizador no ecrã, o contador de erros e os valores estatísticos são incrementados e a mesma trama é reenviada tantas vezes quantas as definidas pelo utilizador no início do programa, ao fim das quais este termina com indicação do erro. Por outro lado, quando um valor RR é recebido são verificados os valores de NS e NR, que sendo iguais causam o reenvio da mesma trama, mais uma vez, tantas vezes quantas as definidas pelo utilizador no início do programa. Se os valores forem diferentes o valor de NS é actualizado, os buffers usados são limpos e o seu espaço em memória é libertado e o método *llwrite()* retorna sucesso.

Em *llclose()*, após a definição da trama DISC e do seu envio, indicando o fim da ligação, o programa fica à espera da recepção de uma trama igual, que após validada é metamorfoseada numa trama UA que sinaliza o término do programa. Caso a trama recebida não seja uma trama DISC o programa termina com erro.

Receptor

Na função *llopen()*, o programa estabelece ligação com a porta de série. Por isso, inicialmente, esta função verifica se a trama enviada pelo emissor corresponde a uma trama SET. Para isso, existe um ciclo *while* que vai ler, carácter a carácter, aquilo que se encontra no *'file descriptor'* passado como argumento. A detecção de uma flag 0x7E marca o início da trama. Após a recepção dessa flag, o programa vai concatenar a um array temporário todos os caracteres lidos até encontrar a flag 0x03 e, depois de encontrar essa flag, prosseguirá com a concatenação de caracteres até voltar a encontrar a flag 0x7E. Por fim, faz a verificação das condições para o BCC e o C (byte de controlo), que caso sejam verdadeiras, levam o programa a responder com uma trama UA, à trama SET recebida anteriormente. Esta parte do código pode ser vista nos anexos deste relatório, com o nome "Código 1".

Relativamente à função *llread()*, responsável pela implementação do protocolo da transferência de informação, tal como na função *llopen()*, também ocorre leitura de caracteres, um a um, e concatenação desses mesmos caracteres para um array (a concatenação é iniciada

quando é lida a flag 0x7E e terminada na mesma condição). Em seguida, com o array obtido, é verificada a condição para o BCC do header:

- Caso seja verdadeira, é executado o mecanismo de transparência, é passada para um novo array a trama já com transparência e é analisada a condição para o BCC2. Se for verdade, é enviada ao emissor uma trama RR e retornado pela função um inteiro correspondente ao número de caracteres presentes na trama inicial, entre as flags 0x7E, devidamente decrementado após o mecanismo de transparência. Por outro lado, se essa condição for falsa, é enviada ao emissor uma trama REJ. Esta parte do código pode ser vista nos anexos deste relatório, com o nome "Código 2".
- Caso seja falsa, o programa executa esta função desde o início.

A última função a ser chamada é a função *llclose()*. Nesta função são feitas duas leituras relativamente ao '*file descriptor'* recebido, ambas para verificação de comandos enviados pelo emissor: uma para o comando DISC e outra para o comando UA. Se for recebido o comando DISC, o receptor envia ao emissor igualmente um comando DISC, igual ao recebido. Esta parte do código pode ser vista nos anexos deste relatório, com o nome "Código 3".

Protocolo de aplicação

Receptor

No nível da aplicação é chamada a função *main(int argc, char** argv)*, responsável por chamar todas as funções relativas ao nível de aplicação. Inicialmente, chama a função *llopen()*, passando-lhe um *'file descriptor'* declarado anteriormente.

Em seguida, através de um ciclo, chama a função *llread()* de forma a poder obter as tramas enviadas pelo emissor. Para cada trama fará uma verificação relativa ao valor do campo de controlo recebido. Este varia entre três valores:

- 1, indica que a trama recebida é uma trama de START e obtém e imprime o nome do ficheiro que, posteriormente, irá criar;

- 0, significa que a trama recebida é uma trama de informação e, dentro dessa condição, imprime vários dados estatísticos, tais como a percentagem de conclusão da transferência e o número de tramas recebidas e escreve os dados para o ficheiro criado.

- 2, significa que a trama recebida é uma trama END. Quando recebe esta trama, o programa fecha o ficheiro para onde estava a copiar a informação e termina o ciclo.

Se não se verificar nenhum destes três valores no campo de controlo da trama recebida, o programa envia um REJ ao emissor, prosseguindo com o ciclo.

Emissor

Tal como no receptor, o emissor começa por chamar a função *main(int argc, char** argv)*, que trata de chamar, mais uma vez como o receptor, a função *llopen()*, estabelecendo assim a ligação entre estes.

Se a chamada do *llopen()* devolver sucesso, então o programa irá pedir ao utilizador para introduzir valores para o tamanho máximo das tramas a enviar e o número máximo de retransmissões e o número máximo de time-outs que podem ocorrer. O número de retransmissões indica o número de REJ que pode receber do receptor até desistir de enviar uma trama, e o time-out indica o número de vezes que o alarme pode ser ativado antes de o programa desistir.

Após receber estes valores, o programa manda, usando a função *llwrite()*, uma trama do tipo START para o receptor, indicando o nome do ficheiro a ser transmitido, bem como o seu tamanho em bytes.

Depois, o programa manda o ficheiro em bocados com o tamanho indicado pelo utilizador, usando para isso a função *llwrite()*. Este ciclo também verifica se o programa recebeu a trama de REJ três vezes, o que força o programa a terminar. No entanto, existe também uma probabilidade aleatória de se mandar um pacote com erros, servindo isto para testar o mecanismo de REJ do receptor e do emissor.

Finalmente, após enviar o ficheiro completamente, o programa envia ao receptor uma trama do tipo END indicando assim que o ficheiro terminou. Se esta trama for mandada com sucesso, o programa termina a ligação entre ele e o receptor chamando a função *llclose()*, que se encontra descrita na secção "Protocolo de Ligação Lógica".

Validação

De forma a serem testadas as funcionalidades do programa, optou-se pelo envio de ficheiros *.jpg, .gif, .zip* e *.c*.

Elementos de valorização

De todos os elementos de valorização para o projecto, seleccionaram-se os seguintes:

- Implementação do REJ, em duas situações: quando falha a verificação do BCC2 e quando o campo de controlo da trama não é 0, 1 ou 2;
- Registo das ocorrências, tanto no emissor como no receptor, são mostrados dados na consola, tais como, número de REJs enviados/recebidos, número de tramas enviadas/recebidas, entre outros;
- Geração aleatória de erros. Foi criado no emissor um número aleatório, que funcionará como número de tramas correctas enviadas entre cada trama de erro (criada pelo emissor, para testar a robustez do receptor), isto é, se o número gerado for 13, o emissor envia uma trama errada entre cada 13 tramas. Quando o emissor envia uma trama errada, o valor desse número é novamente mudado de forma aleatória;
- Selecção de parâmetros por parte do utilizador. No emissor, quando iniciado o programa, existem três parâmetros que podem ser inseridos pelo utilizador: número de retransmissões, intervalo de time-out e tamanho máximo da trama.

Conclusão

O objectivo fundamental do projecto era a transmissão de tramas de informação de um computador que funcionaria como emissor, para outro computador que, por sua vez, funcionaria como receptor, permitindo que o ficheiro enviado, chegasse sem erros e defeitos ao seu destino. Esse objectivo foi alcançado com distinção, através da criação de três funções relativas ao protocolo de ligação e uma função relativa ao protocolo de transmissão.

Inicialmente, o grupo prendeu-se com alguns problemas relativos à transmissão de dados, por algumas falhas de organização quanto à estruturação do código, e quanto à coordenação entre os diferentes ficheiros (emissor e receptor).

Em suma, pelo facto de o objectivo ter sido alcançado, os conhecimentos dos elementos do grupo, nesta área, foram ampliados.

Anexo I

Código 1 – Verificação da recepção da trama SET e respectivo envio da trama UA (receiver.c)

```
char componente = final[1] ^ final[2];    //Verifica BCC1

if(final[3] == componente) {

//verificar C(Byte controlo) e envia UA
if(final[2] == 0x03) {     //se for SET

        printf("Recebi um SET\n");
        char UA[5];
        UA[0] = 0x7E;
        UA[1] = 0x03;
        UA[2] = 0x07;
        UA[3] = UA[2] ^ UA[1];
        UA[4] = 0x7E;
        write(fd,UA,strlen(UA)*sizeof(char)+1);
        printf("Enviei um UA\n"); }

        estado = 3; }      //termina ciclo
```

Código 2 – Cálculo do BCC2 e envio da resposta RR ou REJ para o emissor quando este envia uma trama (receiver.c)

```
int comp = buf2[1];
//Calculo de BCC2
for(m=2; m < bufsize-1;m++)
        comp = comp^buf2[m];

if(bcc2 == comp){

(…)
        char RR[5];
        RR[0] = 0x7E;
        RR[1] = 0x03;
        RR[2] = rr;
        RR[3] = RR[2] ^ RR[1];
        RR[4] = 0x7E;
        write(fd,RR,5);
(…)
}
else{
(…)
        char REJ[5];
        REJ[0] = 0x7E;
        REJ[1] = 0x03;
        REJ[2] = rej;
        REJ[3] = REJ[2] ^ REJ[1];
        REJ[4] = 0x7E;

        write(fd,REJ,5);
(…)
}
```

Código 3 – Processamento de um comando DISC por parte do receptor (receiver.c)

```
if((buf[0] == 0x7E) && (buf[1] == 0x03) && (buf[2] == 0x0B) &&     //verifica recepcao do comando DISC
(buf[3] == (buf[1]^buf[2])) && (buf[4] == 0x7E))
{
        printf("Recebi DISC!\n");
        char DISC[5];
        DISC[0] = 0x7E;
        DISC[1] = 0x03;
        DISC[2] = 0x0B;
        DISC[3] = DISC[2] ^ DISC[1];
        DISC[4] = 0x7E;
        limpar_buffer(fd);
        write(fd,DISC,5);
        printf("Enviei DISC!\n");
}
(…)
if((buf4[0] == 0x7E) && (buf4[1] == 0x03) && (buf4[2] == 0x07) &&//verifica recepcao do comando UA
(buf4[3] == (buf4[1]^buf4[2])) && (buf4[4] == 0x7E))
        printf("Recebi UA!\n");
```

sender.c

```
/*Non-Canonical Input Processing*/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>


#define BAUDRATE B38400
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE  1

#define SET     0x7e03007e
#define UA      0x7e03077e
#define FLAG  0x7e
#define FLAG2 0x7d

int alarme_activo = 0,   //varia entre o alarme de llopen e llwrite
        flag = 1,                   //activa ou descativa alarme do llopen
        conta = 1,                  //contador do numero de alarmes do llopen
        conta2 = 1,                 //contador do numero de alarmes do llwrite
        terminar = 0,    //determina se llwrite terminou com erro ou nao
```

```
        terminar_tudo = 0,      //determina se llopen terminou com erro ou nao
        ns_local = 0,   //NS
        tam_i,          //tamanho das tramas de informacao
        num_max_retrans,        //numero maximo de retransmissoes
        time_out_length,        //intervalo de tempo do alarme
        num_t_retrans = 0,      //numero de tramas de informacao retransmitidas
        num_time_out = 0,       //numero de alarmes activados
        num_REJ_rec = 0,        //numero de REJ recebidos
        n_trama = 0;    //numero da trama I a enviar(N)

// atende alarme
void atende()
{
        if (alarme_activo == 0) {
                if (conta < 4) {
                        printf("Alarme do llopen # %d\n", conta);       //imprime no maximo 3
alarmes
                        flag = 1;                                       //reactiva a flag para
permitir novo alarme
                        conta++;
                }
                else
                        terminar_tudo = 1;                              //activa flag para
terminar com erro
        }
        else {
                if (conta2 < 4) {
                        printf("Alarme do llwrite # %d\n", conta2+1);   //imprime no maximo 3
alarmes
                        conta2++;
                }
        }
}

//tira o lixo (caso haja) do buffer
void limpar_buffer(intfd) {
        char temp;
        int lidos = -1;

        do {
                lidos = read(fd, &temp, 1);                             //le o que esta no buffer
carater a carater...
        } while (lidos != 0);                                           //...ate ter apenas zeros
}

//estabelece ligação com o recetor
int llopen(int fd) {
        char buf[5];                                                    //buffer que vai conter o
SET
        bzero(buf, sizeof(buf));                                //limpa o buffer do SET

        buf[0] = 0x7e;                                                  //define...
        buf[1] = 0x03;
        buf[2] = 0x03;                                                  //...o...
        buf[3] = buf[1] ^ buf[2];
```

```
	buf[4] = 0x7e;                                          //...SET

	printf("A estabelecer ligacao...\n");

	while(conta < 4)                                        //enquanto o numero de
alarmes nao for 3
	{
		if(flag==1){
			alarm(3);                                   // activa alarme de 3s
			flag=0;
		}

		write(fd,buf,5);                            //envia o SET para o recetor

		char final[5];                                  //buffer que vai conter o
UA
		bzero(final, sizeof(final));                    //limpa o buffer do UA

		read(fd, final, 6);                             //le um UA
		limpar_buffer(fd);                              //limpa o resto do
buffer

		if(final[0] == 0)
			atende();                               //se nao tiver lido nada,
atende o alarme
		else
		{
			if (final[0] == 0x7e &&final[1] == 0x03 &&
				final[2] == 0x07 &&     final[3] == (final[1] ^ final[2])
				&& final[4] == 0x7e) {

				printf("Ligacao Estabelecida (UA recebido)\n");          //se
verificar que recebe um UA...
				conta = 4;
	//...termina o ciclo de leitura com alarme
			}
			else {    return -1; }                                  //retorna  -1  se
terminar com erro desconhecido
		}
	}

	return 1;                                                       //retorna  1  se
terminar sem problemas
}

//termina a ligacao com o recetor
int llclose(int fd) {
	char buf[5];                                            //buffer que vai conter o
DISC
	bzero(buf, sizeof(buf));                        //limpa o buffer do DISC

	buf[0] = 0x7e;                                          //define...
	buf[1] = 0x03;
	buf[2] = 0x0B;                                          //...o...
	buf[3] = buf[1] ^ buf[2];
```

```
	buf[4] = 0x7e;                                          //..DISC

	limpar_buffer(fd);                                      //limpa o buffer para
assegurar que nao enviara lixo
	write(fd,buf,5);                                        //envia o DISC

	char final[5];                                          //buffer que vai conter o
DISC(recetor)
	bzero(final, sizeof(final));                            //limpa o buffer que vai
conter o DISC(recetor)
	sleep(1);
	read(fd, final, 5);                                     //le um DISC(recetor)

	printf("A desligar...\n");

	if (final[0] == 0x7e &&final[1] == 0x03 &&
		final[2] == 0x0B &&    final[3] == (final[1] ^ final[2])
		&& final[4] == 0x7e) {

		buf[2] = 0x07;                                  //se recebe um
DISC(recetor)...
		buf[3] = buf[1] ^ buf[2];                       //...muda o DISC para
um UA

		sleep(1);
		printf("A confirmar encerramento...(a enviar UA)\n");
		write(fd,buf,5);                                //manda um UA
	}
	else
		return -1;                                      //retorna -1 se terminar
com erro desconhecido

	return 1;                                               //retorna 1 se terminar
sem problemas
}

//calcula bcc2 para as tramas
char calcular_bcc2(char dados[], int tamanho) {
	char bcc2 = dados[0];

	int k;
	for (k = 1; k != tamanho; k++) {
		bcc2 ^= dados[k];                               //faz um "ou-exclusivo"
de todos os bits da trama
	}

	return bcc2;
}

//estrutura para calcular transparencia
typedefstruct {
	char* buf;                                              //buffer com
transparencia
	int contador;                                           //novo tamanho de buf
} teste;
```

```
teste* calcular_transparencia(char dados[], int tamanho) {
	char* destino = malloc(2*tamanho);

	bzero(destino, 2*tamanho);

	char* limite_destino = destino;
	int k = 0;
	int contador = 0;

	for (k = 0; k != tamanho; k++) {
		if (dados[k] == FLAG) {
uma FLAG...
			*limite_destino = 0x7d;
transparencia...
			limite_destino++;
apontador
			contador++;
			*limite_destino = 0x5e;
		}
		else if (dados[k] == 0x7d) {
			*limite_destino = 0x7d;
transparencia...
			limite_destino++;
apontador
			contador++;
			*limite_destino = 0x5d;
		}
		else
			*limite_destino = dados[k];
		limite_destino++;
		contador++;
	}

	teste* return_type = malloc(sizeof(teste));

	return_type->buf = destino;
	return_type->contador = contador;

	returnreturn_type;
}

//determina se recebe um RR ou um REJ
int* processar_resposta(char *buf) {

	int* devolver = malloc(2*sizeof(int));
	bzero(devolver, 2*sizeof(int));


	if (buf[0] != FLAG)
termina
		return devolver;

	char bbc1 = buf[1] ^ buf[2];
recebida...
```

```
	if (bbc1 == buf[3]) {                                   //...e verifica
		int mascara = 0x7;
		if ((buf[2] & mascara) == 1) // RR              //se os 3 primeiros bits
tiverem o valor 1
		{
			devolver[0] = 1;                        //devolve esse valor
			devolver[1] = (buf[2] >> 5);
		}
		else if ((buf[2] & mascara) == 5) // REJ     //se os 3 primeiros bits tiverem
o valor 5
		{
			devolver[0] = 2;
			devolver[1] = (buf[2] >> 5);
		}
	}
	return devolver;
}

//calcula campo de controlo das tramas
char calcular_C(void) {
	if (ns_local == 0)
		return 0x00;
	else if (ns_local == 1)
		return 0x02;
	else {
		printf("Erro ao calcular o C\n");
		return -1;
	}
}

//envia pacotes de dados ao recetor
int llwrite(intfd,char* buf, intsizebuf){

	char buf_sem_trans[256];
	bzero(buf_sem_trans, sizeof(buf_sem_trans));

	buf_sem_trans[0] = FLAG;
	//FLAG de inicio de pacote
	buf_sem_trans[1] = 0x03;
	//A
	buf_sem_trans[2] = calcular_C();
	//Campo de controlo
	buf_sem_trans[3] = buf_sem_trans[1] ^ buf_sem_trans[2];	//BCC1

	char bcc2 = calcular_bcc2(buf, sizebuf);                        //BCC2
(necessario calcular sem a transparencia aplicada)

	char buf2[sizebuf+4];
	//buffer auxiliar...
	bzero(buf2, sizebuf+4);

	int k;
	for (k = 0; k != sizebuf; k++) {
```

```
		buf2[k] = buf[k];
	}

	*(buf2+sizebuf) = bcc2;

	teste* buf_trans = calcular_transparencia(buf2, sizebuf+1);	//...para calcular e aplicar a transparencia

	char *temp = buf_trans->buf;


	for (k = 4; k != buf_trans->contador+4; temp++) {
		buf_sem_trans[k] = *temp;
	//passa a informação com transparencia para o buffer a enviar(inclui BCC2)
		k++;
	}

	buf_sem_trans[k] = FLAG;
	//finaliza o buffer a enviar com a FLAG


	free(buf_trans);
	//desfaz-se do buffer auxiliar com a transparencia

	// Processar confirmacao
	char final[256];
	int mm;

	conta2 = 0;
			//inicializa o contador de alarme do llwrite
	alarme_activo=1;
		//muda o alarme de llopen para llwrite
	int* valores = 0;
	int contador_erro = 0,
	//contador de erros(REJ & NS==NR)
		volta = 1;
			//flag para saber se deve renviar a trma

	do {
		write(fd, buf_sem_trans, sizeof(buf_sem_trans));		//envia pacote ao recetor

		if(volta == 1)
		{
			alarm(time_out_length);
		//activa o alarme
			mm = read(fd, final, 5);
	//le a resposta
			if(mm > 0){
			//se a resposta nao for nula..
				alarm(0);
			//..desativa o alarme..
				limpar_buffer(fd);
		//..limpa o resto do buffer..
```

```
                        volta = 0;
                //..e indica que a trama nao deve ser reenviada(temp)

                        valores = processar_resposta(final);
//verifica o que recebeu(RR ou REJ?)

                        if (valores[0] == 2) {
//se recebeu REJ..
                                printf("Recebi REJ com valor: %d\n", valores[1]);
                                printf("A reenviar a trama: ");
                                num_t_retrans++;
                                num_REJ_rec++;
                                contador_erro++;
                                volta = 1;
                //..indica que deve reenviar a trama
                        }

                        if (valores[0] == 1) {
//se recebeu RR
                                if (valores[1] == ns_local) {
//se este for igual ao NS(erro)..
                                        printf("Recebi um RR com NR == NS\n");
                                        bzero(final, sizeof(final));
//..limpa o buffer com a resposta..
                                        contador_erro++;
                                        volta = 1;
                //..e indica que deve reenviar a trama
                                }
                                else {
                                        //ns_local = (ns_local + 1) % 2;
//senao actualiza NS
                                        free(valores);
        //liberta a variavel com a resposta processada
                                        bzero(final, sizeof(final));
//limpa o buffer com a resposta

                                        return 1;
                //retorna sucesso
                                }
                        }


                        free(valores);
                }
                else
                {
                        sleep(time_out_length-1);
//assegura que espera tempo suficiente para dar o alarme
                        atende();
                //assegura que atende o alarme
                        num_time_out++;
                        bzero(final, sizeof(final));
//limpa o buffer com a resposta
```

```
                        volta = 1;
                    //indica que deve reenviar a trama
                        contador_erro = 0;
                    }
                }
        } while (contador_erro < num_max_retrans && conta2 < num_max_retrans);    //tenta
no maximo 3 vezes, por alarme ou por erros

        printf("Rejeitado 3 vezes, a desistir\n");                              //se    sair    do
ciclo, llwrite termina com erro
        terminar = 1;
            //indica ao programa para terminar com erro
        return 0;
}

//envia trama inicial para definir a informação a enviar
int mandar_start(intfd, char* filename, intsize_file) {

        char* tamanho_string = malloc(10);
        sprintf(tamanho_string, "%d", size_file);

        intsizestr_tamanho = strlen(tamanho_string);
        intsizestr_nome = strlen(filename);


        char* trama_start = malloc(40);

        trama_start[0] = 0x1;
        //tipo de trama
        trama_start[1] = 0x0;
        //tipo de dados seguinte(tamanho do ficheiro)
        trama_start[2] = sizestr_tamanho;
        //tamanho em octetos do tamanho do ficheiro
        strcpy(&trama_start[3], tamanho_string);
        //tamanho do ficheiro
        trama_start[3+sizestr_tamanho] = 0x1;                                   //tipo
de dados seguinte(nome do ficheiro)
        trama_start[3+sizestr_tamanho+1] = sizestr_nome;                        //tamanho    em
octetos do nome do ficheiro
        strcpy(&trama_start[3+sizestr_tamanho+2], filename);            //nome do ficheiro

        llwrite(fd, trama_start, 3+sizestr_tamanho+2+sizestr_nome);     //envia trama ao recetor

        return 0;
}

//envia tramas de informação
void mandar_tramaI(int fd, char* buf, int sizebuf){

        char *tramaI = malloc(4+sizebuf);

        tramaI[0] = 0x0;
            //Campo controlo = 0 significa dados
        tramaI[1] = n_trama;
        //numero da trama a enviar
```

```
	n_trama = (n_trama + 1) % 256;
	//atualiza o numero da trama

	tramaI[2] = sizebuf >> 8;
	//numero de octetos do campo de dados(L2)
	tramaI[3] = sizebuf & 0xFF;
	//numero de octetos do campo de dados(L1)

	int k = 0;
	for (k = 0; k != sizebuf; k++)
		tramaI[k+4] = buf[k];
	//coloca os dados na trama

	llwrite(fd, tramaI, sizebuf+4);
	//envia a trama

	free(tramaI);
		//liberta o espaco ocupado
}

//envia trama final
void mandar_end(int fd) {
	char buffer3[7] = {2, 'c', FLAG, 'e', 'f', FLAG2, 0};		//apenas e necessario
definir que C = 2

	llwrite(fd, buffer3, sizeof(buffer3));

	printf("End enviado\n");
}

int main(intargc, char** argv)
{
	intfd;
	structtermiosoldtio,newtio;

	if ( (argc< 3) ||
		((strcmp("/dev/ttyS0", argv[1])!=0) &&
		 (strcmp("/dev/ttyS1", argv[1])!=0) )) {
		printf("Usage:\tnserialSerialPort	filename\n\tex:	nserial	/dev/ttyS1
pinguim.gif\n");
		exit(1);
	}


	/*
	  Open serial port device for reading and writing and not as controlling tty
	because we don't want to get killed if linenoise sends CTRL-C.
	*/


	fd = open(argv[1], O_RDWR | O_NOCTTY );
	if (fd<0) {perror(argv[1]); exit(-1); }

	if ( tcgetattr(fd,&oldtio) == -1) { /* save current port settings */
		perror("tcgetattr");
```

```
            exit(-1);
        }

        bzero(&newtio, sizeof(newtio));
        newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
        newtio.c_iflag = IGNPAR;
        newtio.c_oflag = 0;

        /* set input mode (non-canonical, no echo,...) */
        newtio.c_lflag = 0;

        newtio.c_cc[VTIME]    = 1;   /* inter-character timer unused */
        newtio.c_cc[VMIN]     = 0;   /* blocking read until 5 chars received */



        /*
          VTIME e VMIN devem ser alterados de forma a proteger com um temporizador a
          leitura do(s) próximo(s) caracter(es)
        */



        tcflush(fd, TCIOFLUSH);

        if ( tcsetattr(fd,TCSANOW,&newtio) == -1) {
            perror("tcsetattr");
            exit(-1);
        }

        (void) signal(SIGALRM, atende);
        //handler do alarme

        llopen(fd);
                        //abre a ligacao

        if(terminar_tudo == 1)
        //verifica se llopen terminou com erro
            return 0;

        /*
          O ciclo FOR e as instruções seguintes devem ser alterados de modo a respeitar
          o indicado no guião.
        */


        char erro[7] = {6, 'f', FLAG, 'r', 7, FLAG2, 5};                    //buffer  para  simular
erros

        // Logica de aplicacao

        char *nome = argv[2];

        FILE* ficheiro_enviar = fopen(nome, "rb");                                          //abre  o
ficheiro passado pelo utilizador
```

```
	if (ficheiro_enviar == NULL) {
	//verifica se o nome do ficheiro foi passado
		printf("Ficheiro \"%s\" nao existe\n", nome);
		printf("Numero tramas retransmitidas = %d\nNumero de time outs = %d\nNumero de REJs recebidos = %d\n", num_t_retrans, num_time_out, num_REJ_rec);
		return -1;
	}

	struct stat st;
	stat(nome, &st);
	int tamanho_ficheiro = st.st_size;

	//dados a ser introduzidos pelo utilizador
	do{
		printf("Escolha o tamanho máximo das tramas de informação (2 - 127): ");
		scanf ("%d",&tam_i);
	}while(tam_i < 2 || tam_i > 127);

	do{
		printf("Escolha o numero maximo de retransmissões (se não tiver a certeza, escreva 3): ");
		scanf ("%d",&num_max_retrans);
	}while(num_max_retrans < 0);

	do{
		printf("Escolha o intervalo de time-out (se não tiver a certeza, escreva 3): ");
		scanf ("%d",&time_out_length);
	}while(time_out_length< 0);

	mandar_start(fd, nome, tamanho_ficheiro);
	//enviatramainicial
	limpar_buffer(fd);
		//limpa o buffer

	char* bufferI = malloc(tam_i);
	char* aux = malloc(tam_i);

	srand(time(NULL));
	int numerot = 1;
		//variavel para identificar o numero da trama
	int numerot2  = (int)(rand()%20);
				//variavel para simulacao de erros
	while (tam_i*numerot < tamanho_ficheiro+tam_i) {
		if(numerot2 == 20)
		//simulacao de erros
		{
			printf("Enviando trama com erro!\n");
			llwrite(fd, erro, 7);
			num_t_retrans++;
			num_REJ_rec++;
			numerot2=(int)(rand()%20);
			aux = bufferI;
		}
		else
```

```
		{
			fread(bufferI, 1, tam_i, ficheiro_enviar);			//le do ficheiro
			mandar_tramaI(fd, bufferI, tam_i);
	//envia trama lida

			printf("Enviando trama numero: %i\n", numerot);
			limpar_buffer(fd);
			numerot++;
			numerot2++;

			if(terminar == 1){
		//verifica se termina com erro
				printf("A terminar(erro)...\n");
				printf("Numero tramas retransmitidas = %d\nNumero de time outs = %d\nNumero de REJs recebidos = %d\n", num_t_retrans, num_time_out, num_REJ_rec);
				return 0;
			}
		}
		if(tam_i*numerot == tamanho_ficheiro)
	//verifica se as tramas enviadas equivalem ao tamanho do ficheiro
			break;
	}

	printf("A escrever END\n");
	mandar_end(fd);
			//envia trama final

	llclose(fd);
		//fecha a aplicacao
	// Termina a logica da aplicacao

	if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
		perror("tcsetattr");
		exit(-1);
	}

	printf("Confirmado!\nA fechar a aplicacao...\n");

	printf("\nEstatísticas:\nNumero tramas retransmitidas = %d\nNumero de time outs = %d\nNumero de REJs recebidos = %d\n", num_t_retrans, num_time_out, num_REJ_rec);

	close(fd);
	return 0;
}
```

receiver.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#define BAUDRATE B38400
#define _POSIX_SOURCE 1
#define FALSE 0
#define TRUE 1

//Variaveis Globais
int nr = 0, tamanho_fich = 0, tamanho=0,fim=0, check=0, nrejs=0;
FILE *fp;

//Prototipos de funcoes
intllopen(intfd);
intllread(intfd, char* result);
intllclose(intfd);
voidlimpar_buffer(intfd);


int main(intargc, char** argv)
{
intfd;
structtermiosoldtio,newtio;

if ( (argc< 2) ||
             ((strcmp("/dev/ttyS0", argv[1])!=0) &&
              (strcmp("/dev/ttyS1", argv[1])!=0) )) {
printf("Usage:\tnserialSerialPort\n\tex: nserial /dev/ttyS1\n");
exit(1);
   }

  /*
    Open serial port device for reading and writing and not as controlling tty
because we don't want to get killed if linenoise sends CTRL-C.
  */

fd = open(argv[1], O_RDWR | O_NOCTTY );
if (fd<0) {perror(argv[1]); exit(-1); }

if ( tcgetattr(fd,&oldtio) == -1) { /* save current port settings */
perror("tcgetattr");
exit(-1);
   }

bzero(&newtio, sizeof(newtio));
newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
newtio.c_iflag = IGNPAR;
```

```
newtio.c_oflag = 0;

   /* set input mode (non-canonical, no echo,...) */
newtio.c_lflag = 0;

newtio.c_cc[VTIME]    = 1;   /* inter-character timer unused */
newtio.c_cc[VMIN]     = 0;

  /*
    VTIME e VMIN devem ser alterados de forma a proteger com um temporizador a
    leitura do(s) proximo(s) caracter(es)
  */

    tcflush(fd, TCIOFLUSH);

    if ( tcsetattr(fd,TCSANOW,&newtio) == -1) {
perror("tcsetattr");
exit(-1);
    }

printf("New termios structure set\n");


/*
    O ciclo WHILE deve ser alterado de modo a respeitar o indicado no guiao
  */

  /*
   *       As alterações ao código inicial fornecido começam aqui
   */

          printf("A iniciar transferencia..\n");

          int numero = 0;             //numero das tramas recebidas
          char buffer[256];           //array com informação do ficheiro

          llopen(fd);

          while(1)
          {

                    int tam=0;        //tamanho de caracteres uteis lido da trama recebida

                    tam = llread(fd,buffer);

                    int i;

                    if(buffer[0] == 0)          //Campo de controlo indica que é uma trama de
informaçao (Trama I)
                    {

                              int k = buffer[2]*256+buffer[3]; // calculo do tamanho recebido na
trama atraves do seu conteudo (256*L2 + L1 = K)

                              numero++;       //incrementa numero de trama
```

```
                        printf("Recebida trama numero %d\n", numero);

                        char buffer1[k], final[k];
                        bzero(final,sizeof(final));
                        bzero(buffer1,sizeof(buffer1));

                        for(i=4; i < tam;i++)     //retira os 3 primeiros caracteres da trama (nao
contem informacao)
                                buffer1[i-4] = buffer[i];

                        int temp = tam-4;
                        tamanho_fich += tam-4;          //actualizacao do tamanho recebido

                        for(i=0; i < tam-4; i++)
                                final[i] = buffer1[i];

                        if(tamanho_fich + temp > tamanho)
                        {
                                char temp1[tamanho - tamanho_fich+temp];      //restringe
tamanho da trama ao numero de elementos estritamente necessario
                                bzero(temp1, sizeof(temp1));
                                int h;
                                for(h=0; h < tamanho - tamanho_fich+temp; h++)
                                        temp1[h] = buffer1[h];


                                if(check==1)    //verifica se e a ultima trama porque o tamanho
a escrever no ficheiro sera menor
                                {
                                        fim=1;
                                        fwrite (temp1 , 1 , (tamanho - tamanho_fich+temp) , fp
);
                                }
                                check++;
                        }


                        if(tamanho_fich > tamanho)
                                tamanho_fich = tamanho;
                        printf("Tamanho Actual / Tamanho Total : %d/%d\n", tamanho_fich,
tamanho);

                        //Calculo da percentagem
                        float t1 = (float) (tamanho_fich);
                        float t2 = (float) (tamanho);
                        float div = t1/t2;
                        printf("%.2f%s\n", 100*div, "%");

                        if(div==1)
                                printf("Transferencia concluida\nForam enviados %d REJ\n",
nrejs);

                        if(fim==0)
                                fwrite (final , 1 , temp , fp );      //escreve qualquer trama,
excepto a ultima
```

```
		}
		else if(buffer[0] == 1)   //Campo de controlo indica que é uma trama de START
		{
			char tam [10];
			bzero(tam,10);
			for(i=0; i < buffer[2]; i++)	//obtem o valor do tamanho, concatenando todos os caracteres no array
				tam[i] = buffer[i+3];

			tamanho = atoi(tam);	//obtem o valor inteiro a partir dos caracteres concatenados

			int k=i+3;
			charnome[buffer[k+1]];
			int m=k;
			k++;

			for(i=0; i < buffer[k]+1; m++,i++)	//Obtem o nome do ficheiro a ser criado
				nome[i] = buffer[m+2];

			unsignedint z;
			printf("Nome do ficheiro: ");
			for(z=0; z < sizeof(nome); z++)
				printf("%c", nome[z]);
			printf("\n");

			//criar ficheiro
			fp = fopen(nome, "a");
		}

		else if(buffer[0] == 2)   //Campo de controlo indica que é uma trama de END
		{
			fclose(fp);
			break;
		}
		else
		{
			printf("Erro ao receber a trama numero %d!\n", numero);
			//Envia trama REJ se o campo de controlo nao for nenhum dos especificados anteriormente
			intrej = 0;
			if(nr == 0)
				rej = 0x05;
			else
				rej = 0x25;

			char REJ[5];

			REJ[0] = 0x7E;
			REJ[1] = 0x03;
			REJ[2] = rej;
			REJ[3] = REJ[2] ^ REJ[1];
			REJ[4] = 0x7E;
```

```
                        write(fd,REJ,5);
                        nrejs++;
                        printf("Enviei REJ numero %d com nr = %d\n", nrejs, nr);
                }
                printf("\n\n");
                bzero(buffer, sizeof(buffer));
        }

   llclose(fd);

  /*
   *      As alterações ao código inicial fornecido acabam aqui
*/

tcsetattr(fd,TCSANOW,&oldtio);
close(fd);
return 0;
}

//Funcao para apagar os valores indesejados lidos da porta de serie
void limpar_buffer(intfd)
{
        char temp;
        int lidos = -1;
        do{
                lidos = read(fd,&temp,1);
        }while(lidos != 0);
}

//Funcao que estabelece a ligacao atraves da porta de serie
int llopen(intfd)
{
        charbuf[1],final[3];
        int estado = 0, res;
        bzero(final,3);
        buf[0]='\0';

        while (estado!=3)
        {
                res = read(fd, buf, 1);  //le um caracter
                if(res > 0)
                {
                        //verifica se o caracter = 7E
                        if(estado == 0)
                        {
                                if(buf[0] ==  0x7E)
                                        estado = 1;
                        }
                        //verifica o campo de endereço
                        if(estado == 1)
                        {
                                if(buf[0] ==  0x03)
                                        estado = 2;
                                else
                                        sprintf(final,"%s%s",final,buf);
```

```
                        }
                        if(estado == 2)
                        {

                                if(buf[0] !=  0x7E)
                                        sprintf(final,"%s%s",final,buf);
                                else
                                {
                                        sprintf(final,"%s%s",final,buf);
                                        char componente = final[1] ^ final[2];    //Verifica
BCC1
                                        if(final[3] == componente)
                                        {
                                                //verificar C(Byte controlo) e envia UA
                                                if(final[2] == 0x03)       //se for SET
                                                {
                                                        printf("Recebi um SET\n");
                                                        char UA[5];
                                                        UA[0] = 0x7E;
                                                        UA[1] = 0x03;
                                                        UA[2] = 0x07;
                                                        UA[3] = UA[2] ^ UA[1];
                                                        UA[4] = 0x7E;

        write(fd,UA,strlen(UA)*sizeof(char)+1);
                                                        printf("Enviei um UA\n");

                                                }
                                                estado = 3;        //termina ciclo
                                        }
                                        else estado=0;
                                }
                        }
                }

        }
        limpar_buffer(fd);
        return 1;
}

//Funcao que permite a terminacao controlada da transferencia
int llclose(intfd)
{
        int j=0;
        charbuf[5];
        limpar_buffer(fd);

        while(j<5)        //le um caracter de cada vez
        {
                read(fd, buf, 5);
                j++;
        }

        limpar_buffer(fd);
```

```
	if((buf[0] == 0x7E) && (buf[1] == 0x03) && (buf[2] == 0x0B) &&	//verifica
recepcao do comando DISC
	(buf[3] == (buf[1]^buf[2])) && (buf[4] == 0x7E))
	{
		printf("Recebi DISC!\n");
		char DISC[5];
		DISC[0] = 0x7E;
		DISC[1] = 0x03;
		DISC[2] = 0x0B;
		DISC[3] = DISC[2] ^ DISC[1];
		DISC[4] = 0x7E;
		limpar_buffer(fd);
		write(fd,DISC,5);
		printf("Enviei DISC!\n");
	}
	limpar_buffer(fd);
	j=0;
	sleep(1);
	char buf4[5];
	while(j<5)
	{
		read(fd, buf4, 5);
		j++;
	}
	limpar_buffer(fd);

	if((buf4[0] == 0x7E) && (buf4[1] == 0x03) && (buf4[2] == 0x07) &&//verifica
recepcao do comando UA
	(buf4[3] == (buf4[1]^buf4[2])) && (buf4[4] == 0x7E))
		printf("Recebi UA!\n");

	return 1;
}


//Funcao que implementa o protocolo de transferencia de informacao
int llread(intfd, char* result)
{
	while(1)
	{
		int j=0;
		charbuf[1],buffer[256];
		bzero(buffer, sizeof(buffer));
		while(j <256)
		{
			read(fd, buf, 1);//le um caracter de cada vez
			if(buf[0] == 0x7e)	//encontra flag inicial
			{
				buffer[j]=buf[0];	//grava no array
				j++;
				while(j < 256)		//enquanto o tamanho for menor do que
o estabelecido
				{
					read(fd, buf, 1);//le novo elemento
```

```
                                        if(buf[0] == 0x7e)      //enquanto nao for flag de terminacao
                                        {
                                                buffer[j]=buf[0];       //grava no array
                                                break;
                                        }
                                        buffer[j]=buf[0];       //guarda ultima flag
                                        j++;
                                }
                                break;
                        }

                }

                //Verifica bcc do header
                if(buffer[3] == (buffer[2]^buffer[1]))
                {
                        unsignedint l=0, m=0, n=0,bufsize = 0;
                        char buf2[sizeof(buffer)];
                        bzero(buf2, sizeof(buf2));

                        for(l=1; l < sizeof(buffer); l++) //retira flags do inicio e fim da trama
                        {
                                if(buffer[l] == 0x7e)
                                        break;
                                else bufsize++;
                        }

                        bufsize-=2;

                        for(m=3; m < sizeof(buffer); m++,n++) //desfaz transparencia a partir do terceiro caracter
                        {
                                if(buffer[m] == 0x7D && buffer[m+1] == 0x5E) //Transparencia para flag
                                {
                                        buf2[n] = 0x7E;
                                        m++;
                                        bufsize--;
                                }
                                else if(buffer[m] == 0x7D && buffer[m+1] == 0x5D) //Transparenciapara escape
                                {
                                        buf2[n] = 0x7D;
                                        m++;
                                        bufsize--;
                                }
                                else
                                        buf2[n] = buffer[m];
                        }


                        int comp = buf2[1];
                        //Calculo de BCC2
                        for(m=2; m < bufsize-1;m++)
```

```
		comp = comp^buf2[m];

	char bcc2 = buf2[bufsize-1];
	buf2[bufsize-1]='\0';	//Apaga ultimo elemento
	buf2[bufsize-1]='\0';	//Apaga ultimo elemento
	bufsize-=2;
	buf2[0] = buffer[2];

	if(bcc2 == comp)
	{
		if((buf2[0]&0x02)>>1 == nr) //Ns=Nr
		{
			if(nr == 0)
				nr = 1;
			else
				nr = 0;
		}

		intrr = 0;
		if(nr == 0)
			rr = 0x01;
		else
			rr = 0x21;

		char RR[5];
		RR[0] = 0x7E;
		RR[1] = 0x03;
		RR[2] = rr;
		RR[3] = RR[2] ^ RR[1];
		RR[4] = 0x7E;

		write(fd,RR,5);
		int x;
		//Actualiza a variavel a devolver a camada da aplicacao
		for(x=0; x < 256; x++)
			result[x] = buf2[x+1];

		return bufsize;	//sucesso
	}
	else
	{
		int rej = 0;
		if(nr == 0)
			rej = 0x05;
		else
			rej = 0x25;

		char REJ[5];

		REJ[0] = 0x7E;
		REJ[1] = 0x03;
		REJ[2] = rej;
		REJ[3] = REJ[2] ^ REJ[1];
		REJ[4] = 0x7E;
```

```
                                write(fd,REJ,5);
                                nrejs++;
                                printf("Enviei REJ numero %d com nr = %d\n", nrejs, nr);
                                sleep(2);
                        }
                }
        }
}
```