Protocolo de Ligação de Dados

RELATÓRIO

Francisco Teixeira Lopes
ei11056@fe.up.pt
Nelson André Garrido da Costa
up201403128@fe.up.pt

ÍNDICE

Sumário

Este trabalho insere-se no âmbito da Unidade Curricular Redes de Computadores, do Mestrado Integrado em Engenharia Informática e da Computação da Faculdade de Engenharia da Universidade do Porto, abordando conteúdos programáticos, abordados nas aulas teóricas. Entre eles, destacam-se o Nível Físico – *Physical Layer* -, a Camada de Ligação de Dados - Data *Link Layer* – e a Camada de Aplicação – *Application Layer*. A proposta de trabalho teve como base a transferência, por cabo, de ficheiros entre dois computadores, através das suas portas de série.

Este relatório prende-se na consolidação do trabalho e conhecimento adquirido até à data, fazendo-se a união entre as componentes prática e teórica sobre as quais se debruça o projeto apresentado, que se pretende mostrar bem entendidas e implementadas.

INTRODUÇÃO

O projeto foi trabalhado no contexto da Unidade Curricular de Redes de Computadores, inserida no plano do 3º ano do Mestrado Integrado em Engenharia Informática e Computação.

Os objetivos principais deste trabalho prendem-se essencialmente na implementação do protocolo de ligação de dados, através de uma aplicação responsável pela transferência de ficheiros entre dois computadores - o emissor e o recetor -, recorrendo às suas portas de série e combinando características de protocolos já existentes, como a transparência e a transmissão organizada em tramas – de informação, controlo e não numeradas - e utilizando integralmente transmissão em série assíncrona.

De modo a que o processo fosse feito em conformidade com estes objetivos, foram incluídos, na aplicação, mecanismos de deteção de erros como, por exemplo, dados enviados em duplicado ou falhas de conexão entre as máquinas, e respetiva resposta, assim como suporte para dois tipos de pacotes enviados pelo emissor: os de dados (com porções do ficheiro) e os de controlo para iniciar – *start* - e finalizar a transmissão – *end*.

Este relatório, sendo o *pós-laboral* do trabalho prático, serve como suporte à sua fácil compreensão e está estruturado da seguinte forma:

- Introdução: indicação dos objetivos do trabalho e do relatório; descrição da lógica do relatório;
- Arquitetura: blocos funcionais e interfaces;
- Estrutura do código: APIs consultadas e utilizadas, principais estruturas de dados usadas, principais funções e sua relação com a arquitetura;
- Casos de uso principais;
- Protocolo de ligação lógica: identificação dos principais aspetos funcionais e descrição da estratégia de implementação destes aspetos;
- Protocolo de aplicação: tal como no ponto anterior, explana-se a maneira de pensar nesta fase;
- Validação: testes realizados e quantificação de resultados.
- Eficiência do protocolo de ligação de dados: caracterização estatística da eficiência do código desenvolvido comparativamente a uma caracterização teórica;
- Conclusões: considerações sintéticas finais sobre o trabalho realizado.

ARQUITETURA

A aplicação está organizada em duas camadas:

- Camada do protocolo de ligação de dados: funções do protocolo, especificando funcionalidades de sincronismo, numeração de tramas, controlo da transferência, tratamento de erros, abertura e definição das propriedades da porta de série – LinkLayer.c/LinkLayer.h
- Camada da aplicação: funções de escrita e leitura do ficheiro, através do processamento e envio de pacotes de controlo e dados, sendo fundamental para o funcionamento do programa. – AppLayer.c/AppLayer.h

Em relação à interface da aplicação, todos os valores globais às camadas e estruturas funcionais estão incluídos no ficheiro **Globals.h,** como, por exemplo, o *baudrate*, macros que definem valores predefinidos, como as *FLAGS*, tamanhos máximos para o nome e tamanho do ficheiro a transmitir, entre outros. É também apresentada, na linha de comandos, ao utilizador recetor, uma barra de progresso relativa à transferência com a respetiva percentagem. Mensagens *log* estão previstas, se o utilizador ativar a macro EN*ABLE_DEBUG* no ficheiro suprarreferido.

ESTRUTURA DO CÓDIGO

- **Camada de ligação de dados:**

Utiliza-se uma ***struct*** para a sua representação, onde é guardada, entre outra informação, a porta de série, o ***baudrate***, um número de **sequência** e a última resposta, que determina qual a trama a esperar receber ou enviar de seguida, o tamanho, variáveis de ***timeout***, para o contador de segundos de espera, e as configurações da ***termios***.

Das funções desta camada, destacam-se a ***llopen()*** - que tenta estabelecer a ligação -, ***llclose_transmit()***/***llclose_receive()*** - que tentam terminar a ligação -, ***llread()*** – que lê e valida pacotes de dados -, ***llwrite()*** – que processa e envia pacotes de dados.

```
typedef struct linkLayer {
  char port[PORT_NAME_SIZE];
  uint32_t baudrate;
  uint8_t sequence_number;
  uint32_t length;
  volatile uint8_t timeout_count;
  volatile uint8_t timeout_flag;
  uint16_t current_index;
  uint8_t last_response;
  serialState state;
  struct termios oldtio;
  struct termios newtio;
  uint8_t frame[FRAME_MAX_SIZE];
} linkLayer;
```

Fig1: código da *struct* da camada de ligação de dados.

```
/*
Attempts to establish connection by sending or receiving SET / UA and returns the file descriptor for the connection
*/
int llopen(int port, serialStatus role);

/*
Attempts to close connection by sending DISC, receiving DISC and sending UA, returns whether it succeeded
*/
int llclose_transmit(int fd);

/*
Attempts to close connection by receiving DISC, sending DISC and waiting on UA, returns whether it succeeded
*/
int llclose_receive(int fd);

/*
Attempts to send data_frame after byte stuffing it and appending a frame header
*/
int llwrite(int fd, uint8_t* data_frame, size_t length);

/*
Attempts to read a data_frame, byte destuffs it and validates it
*/
int llread(int fd, uint8_t* data_frame);
```

Fig2: código da declaração das principais funções da camada de ligação de dados.

- **Camada de aplicação:**

Utiliza-se uma ***struct*** para a sua representação, onde é guardada, entre outras informações, o ***file descriptor*** da porta de série, o **tamanho** e o **nome** do ficheiro a transferir, um **contador** de envio e o **estado** do envio.

Das principais funções afetas a esta camada, destacam-se a ***alsend()*** – que abre o ficheiro a enviar e o processa em pacotes de dados-, ***alreceive()*** – que tenta ler e guardar o ficheiro no computador recetor -, ***createSEPacket()***/***createDataPacket()*** – que geram os pacotes de controlo e dados resultantes do ficheiro a enviar/receber.

```
typedef struct applicationLayer {
  int serial_fd;
  serialStatus status;
  uint16_t tx_counter;
  int filesize;
  char filename[FILENAME_SIZE];
  uint8_t data_frame[DATA_FRAME_SIZE];
} applicationLayer;
```

Fig3: código da *struct* da camada de aplicação.

```
/*
Parses a string and converts it to unsigned long in specified base
*/
unsigned long parseULong(char* str, int base);

/*
Opens specified file and creates data packets for sending the file
*/
int alsend(int port, char* filename);

/*
Attempts to receive a file and writes it to disk
*/
int alreceive(int port);

/*
Creates a start/end data frame for initiating or ending a file transfer
*/
int createSEPacket(uint8_t control);

/*
Creates a data packet with data read from file, returns the size of the frame
*/
int createDataPacket(FILE* fd);
```

Fig4: código da declaração das principais funções da camada de aplicação.

CASOS DE USO PRINCIPAIS

Os casos de uso do programa incluem aqueles que são responsáveis pela transferência dos ficheiros:

- Configuração da ligação com escolha do papel de emissor ou recetor e identificação do ficheiro, através do seu nome, pelo emissor.
- Estabelecimento da ligação.
- Computador emissor envia o ficheiro particionado em pacotes.
- Computador recetor recebe os dados e escreve-os no ficheiro a guardar.
- Terminação da ligação e fecho dos ficheiros até ao momento abertos.

PROTOCOLO DA LIGAÇÃO LÓGICA

No que diz respeito ao protocolo da ligação lógica, implementa-se as principais funções relativas à camada de ligação lógica.

Ao ser invocada pela máquina emissora, *llopen()* envia o comando *SET* para tentar estabelecer a ligação, esperando pela resposta do recetor - a trama *UA* - que, se não for recebida dentro do período de tempo predefinido – 3 segundos -, o faz reenviar o *SET*. Este procedimento é replicado um número também predefinido de vezes – 3 vezes – que, se excedido, faz terminar o programa sem sucesso. Uma vez concluído este processo, a ligação é estabelecida e a execução continua.

De seguida, *llwrite()* recebe, como argumento, a informação a transmitir, que será usada para a partição em tramas, numeradas com número de sequência – 0 ou 1 – e que serão enviadas ao recetor. O programa fica, então, à espera duma resposta deste último, mais uma vez, durante um período de tempo, que findo, o faz reenviar a trama, e repetir o processo até se exceder as 3 tentativas – terminação sem sucesso. Quando se obtém, como resposta, a trama *RR*, o recetor recebeu a trama enviada sem erros e a função termina com sucesso. Se, pelo contrário, a resposta for uma trama *REJ*, o processo de envio repete-se.

A função *llread()* corre a par da *llwrite()*, na máquina recetora, e permanece em ciclo até receber uma trama, que é validada, recorrendo a uma máquina de estados que controla se os bytes recebidos são os esperados. Casos todos os campos estejam válidos, a função envia a trama *RR* e, caso contrário, envia a *REJ*, com os números de sequência da trama recebida. Por outro lado, se a função receber o comando *DISC*, a ligação deve ser terminada.

Por último, a função llclose() é invocada pelo emissor, que envia o comando *DISC*, indicando o fim da ligação, esperando receber, do recetor, um comando igual, que quando recebido, o faz enviar o comando *UA* – unnumbered acknowledgement.

PROTOCOLO DE APLICAÇÃO

Relativamente ao protocolo de aplicação, é feita a leitura e a conversão dos argumentos passados na linha de comandos, nomeadamente o papel de cada computador, a porta de série e o nome do ficheiro a transferir.

De seguida, o computador emissor invoca *alsend()*, onde lê o ficheiro a enviar e o particiona em pacotes de dados.

O recetor, por sua vez, invoca *alreceive()*, que recebe os pacotes de dados, escrevendo o ficheiro internamente.

VALIDAÇÃO

Para validar a eficácia do projeto desenvolvido, recorreu-se a vários métodos, que correram com sucesso:

- Interromper a ligação, enquanto a informação é transmitida.
- Geração de picos de tensão no cabo que liga as portas de série.
- Usar valores de *baudrate* e tamanho de pacote diferentes.
- Transmissão de diferentes ficheiros, com diferentes tamanhos.

EFICIÊNCIA DO PROTOCOLO DE LIGAÇÃO DE DADOS

Variável: BAUDRATE			
Ficheiro:	Penguin.gif	10968 bytes	
Packet size:	128 bytes		
Baudrate	Tempo de transferência (ms)	R (bits / s)	S = R / C
2400	54490,078	1610,275	0,671
4800	27246,977	3220,321	0,671
9600	13633,977	6435,686	0,670
19200	6828,169	12850,296	0,669
38400	3424,562	25621,964	0,667
57600	2290,677	38304,833	0,665
115200	1148,925	76370,533	0,663

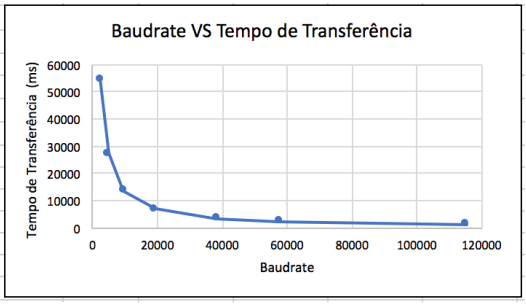

Fig5: Eficiência do programa, quando variada a taxa de transmissão, *baudrate*.

Variável: PACKET_SIZE			
Ficheiro:	Penguin.gif	10968 bytes	
Baudrate:	38400		
Packet Size	Tempo de transferência (ms)	R (bits / s)	S = R / C
128	3424,732	25620,691	0,667
256	3154,883	27812,127	0,724
512	3019,856	29055,690	0,757
1024	2954,543	29697,992	0,773
2048	2925,299	29994,879	0,781
4096	2906,626	30187,579	0,786
65535	2894,455	30314,517	0,789

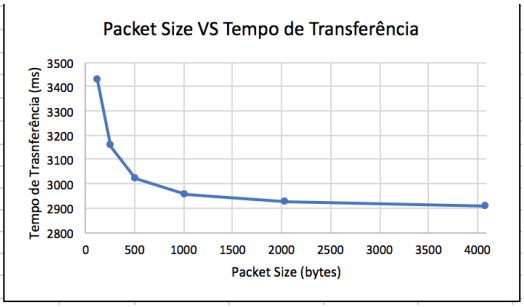

Fig6: Eficiência do programa, quando variado o tamanho dos pacotes de dados.

Variável: T_Prop			
Ficheiro:	Penguin.gif	10968 bytes	
Baudrate:	38400		
Packet size:	512		
Atraso (ms)	Tempo de transferência (ms)	R (bits / s)	S = R / C
0	3020,167	29052,696	0,757
500	12021,163	7299,127	0,190
1000	24020,64258	3652,858	0,095
1500	36020,96411	2435,915	0,063
2000	48021,09106	1827,197	0,048

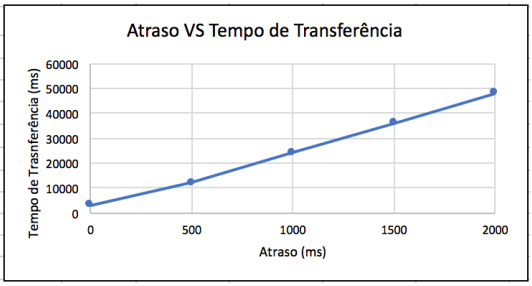

Fig7: Eficiência do programa, quando introduzido um atraso na receção das tramas, fazendo variar o tempo de propagação.

Variable FER			
Ficheiro:	Penguin.gif	10968 bytes	
Baudrate:	38400		
Packet size:	512		
Probabilidade de erro (%)	Tempo de transferência (ms)	R (bits / s)	S = R / C
0	3019,914	29055,129	0,757
5	12729,921	6892,737	0,179
10	12456,738	7043,899	0,183
15	25131,655	3491,374	0,091
20	34743,218	2525,500	0,066
25	72451,450	1211,073	0,032
30	41167,644	2131,383	0,056
35	47653,397	1841,296	0,048
40	47676,275	1840,412	0,048
45	152998,042	573,498	0,015
50	126386,168	694,253	0,018

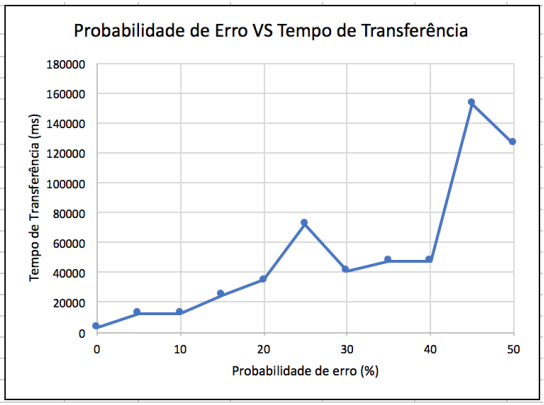

Fig8: Eficiência do programa, quando simulados erros, recorrendo a valores de probabilidade de ocorreram.

Relativamente ao teste de eficiência do programa com erros, convém referir que, para conseguirmos chegar a uma medição, alteramos o número de tentativas de *timeout*, de 3 para 10, de modo a que o programa não terminasse sem sucesso, principalmente com valores de probabilidade de erro mais elevados.

Comparando com os valores teóricos de referência relativos ao modem de linha telefónica, lecionado nas aulas teóricas, concluímos que o nosso projeto é eficiente, visto que o valor de *a* é sempre inferior a 1.

Neste contexto, não fizemos comparação teórica no teste de eficiência à T_Prop, visto que estamos a induzir atrasos reais que afetam diretamente o tempo de execução do programa, o que resulta em valores de *a* muito díspares.

Variável: BAUDRATE

Ficheiro:	Penguin.gif	10968 bytes			
Packet size:		128 bytes			
Baudrate	de transferênc	R (bits / s)	S = R / C	a	S Teórico
2400	54490,078	1610,275	0,671	0,00009375	0,99981254
4800	27246,977	3220,321	0,671	0,0001875	0,99962514
9600	13633,977	6435,686	0,670	0,000375	0,99925056
19200	6828,169	12850,296	0,669	0,00075	0,99850225
38400	3424,562	25621,964	0,667	0,0015	0,99700897
57600	2290,677	38304,833	0,665	0,00225	0,99552016
115200	1148,925	76370,533	0,663	0,0045	0,99108028

Variável: PACKET_SIZE

Ficheiro:	Penguin.gif	10968 bytes			
Baudrate:		38400			
Packet Size	de transferênc	R (bits / s)	S = R / C	a	S Teórico
128	3424,732	25620,691	0,667	0,0015	0,99700897
256	3154,883	27812,127	0,724	0,00075	0,99850225
512	3019,856	29055,690	0,757	0,000375	0,99925056
1024	2954,543	29697,992	0,773	0,0001875	0,99962514
2048	2925,299	29994,879	0,781	0,00009375	0,99981254
4096	2906,626	30187,579	0,786	4,6875E-05	0,99990626
65535	2894,455	30314,517	0,789	2,9297E-06	0,99999414

Variable FER

Ficheiro:	Penguin.gif	10968 bytes			
Baudrate:		*38400*			
Packet size:		512			
Probabilidade de erro (%)	de transferênc	R (bits / s)	S = R / C	a	S Teórico
0	3019,914	29055,129	0,757	0,000375	0,99925056
5	12729,921	6892,737	0,179	0,000375	0,94928803
10	12456,738	7043,899	0,183	0,000375	0,89932551
15	25131,655	3491,374	0,091	0,000375	0,84936298
20	34743,218	2525,500	0,066	0,000375	0,79940045
25	72451,450	1211,073	0,032	0,000375	0,74943792
30	41167,644	2131,383	0,056	0,000375	0,69947539
35	47653,397	1841,296	0,048	0,000375	0,64951287
40	47676,275	1840,412	0,048	0,000375	0,59955034
45	152998,042	573,498	0,015	0,000375	0,54958781
50	126386,168	694,253	0,018	0,000375	0,49962528

Fig9: Eficiência do programa, comparativamente aos valores teóricos.

CONCLUSÃO

O protocolo de ligação de dados foi correta e eficazmente implementado, tendo sido possível a transmissão resistente a erros de transferência entre dois computadores, utilizando as suas portas de série.

O programa possui duas camadas, que são inter-independentes, como estava previsto, sendo que na camada de ligação de dados não é feito qualquer processamento que incida sobre o cabeçalho do pacote de dados a enviar e na camada de aplicação não é feito qualquer processamento ou endereçamento ao cabeçalho da trama. As componentes características de outros protocolos, como a transparência, foram implementadas.

A eficiência do programa, depois de testada e analisada pode ser justificada pelo facto de se gastar uma parte considerável do tempo a processar a informação, em detrimento da sua simples transmissão.

Para concluir este relatório, considera-se por bem evidenciar que os objetivos deste projeto foram alcançados, tendo sido feita com sucesso, a união e a consolidação da componente teórica e da prática da Unidade Curricular respetiva.

Anexo 1 – Source Code

AppLayer.h

```
#ifndef APPLAYER_H
#define APPLAYER_H

/*
Prints program usage
*/
void printUsage(char* program_name);

/*
Parses a string and converts it to unsigned long in specified base
*/
unsigned long parseULong(char* str, int base);

/*
Opens specified file and creates data packets for sending the file
*/
int alsend(int port, char* filename);

/*
Attempts to receive a file and writes it to disk
*/
int alreceive(int port);

/*
Get the filesize of a file with up to INT_MAX bytes
*/
int getFilesize(FILE* fd);

/*
Creates a start/end data frame for initiating or ending a file transfer
*/
int createSEPacket(uint8_t control);

/*
Creates a data packet with data read from file, returns the size of the frame
*/
int createDataPacket(FILE* fd);

/*
Writes received data frame to specified fd
*/
int writeFile(FILE* fd, uint8_t* data_frame);
```

```
/*
UPDATE AND SHOW PROGRESS BAR
*/
VOID UPDATEPROGRESSBAR(SIZE_T CURRENT_BYTES);

/*
HELPER FUNCTION FOR PRINTING A DATA FRAME
*/
VOID PRINTDATAFRAME(SIZE_T SIZE);

/*
EXTRACT FILESIZE AND FILENAME FROM DATA_FRAME
*/
INT EXTRACTFILEINFO(UINT8_T* DATA_FRAME, SIZE_T LENGTH);

/*
RESETS THE DATA FRAME IN THE APPLAYER STRUCT
*/
VOID RESETDATAFRAME();

#ENDIF /* APPLAYER_H */
```

AppLayer.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <stdint.h>
#include <unistd.h>
#include <signal.h>
#include <limits.h>
#include <errno.h>
#include <math.h>

#include "Globals.h"
#include "AppLayer.h"
#include "LinkLayer.h"
#include "Helper.h"

static applicationLayer app_layer;

/*
Prints program usage
*/
void printUsage(char* program_name) {

 printf("\tUsage  : %s tx <port number [0,1]> <filename>\n",
program_name);
 printf("\tUsage  : %s rx <port number [0,1]>\n", program_name);
 printf("\tExample: %s tx 0 penguin.gif\n", program_name);
 printf("\tExample: %s rx 1\n", program_name);
}

/*
Parses a string and converts it to unsigned long in specified base
*/
unsigned long parseULong(char* str, int base) {

 char *endptr;
 unsigned long val;
 int errsv; //Preserve errno

 //Convert string to unsigned long
```

```
  errno = 0; //Clear errno
  val = strtoul(str, &endptr, base);
  errsv = errno;

  //Check for conversion errors
  if (errsv != 0) {
    return ULONG_MAX;
  }

  if (endptr == str) {
    return ULONG_MAX;
  }

  //Successful conversion
  return val;
}

/*
Helper function for printing a data frame
*/
void printDataFrame(size_t size) {

  size_t i;
  for(i = 0; i < size; i++) {
    LOG_MSG("|0x%02X", app_layer.data_frame[i]);
  }

  LOG_MSG("|\n");
}

/*
Creates a start/end data frame for initiating or ending a file transfer,
returns the size of the frame
*/
int createSEPacket(uint8_t control) {

  size_t index = 0;
  app_layer.data_frame[0] = control;

  /* 1st parameter, filesize */
  app_layer.data_frame[1] = T_SIZE;

  int filesize_length = (int) ((ceil(log10(app_layer.filesize)) + 1) *
sizeof(char));
  app_layer.data_frame[2] = filesize_length;
```

```
 char filesize[FILESIZE_MAX_DIGITS];
 snprintf(filesize, FILESIZE_LENGTH, "%d", app_layer.filesize);

 size_t i;
 for(i = 0; i < FILESIZE_LENGTH; i++) {

  app_layer.data_frame[3 + i] = filesize[i];
 }

 index = i + 3;

 /* 2nd parameter, filename */
 app_layer.data_frame[index] = T_NAME;
 app_layer.data_frame[index + 1] = strlen(app_layer.filename) + 1;

 for(i = 0; i < app_layer.data_frame[index + 1]; i++) {

  app_layer.data_frame[index + 2 + i] = app_layer.filename[i];
 }

 return index + 2 + i;
}

/*
Creates a data packet with data read from file, returns the size of the frame
*/
int createDataPacket(FILE* fd) {

 app_layer.data_frame[0] = C_DATA;
 app_layer.data_frame[1] = app_layer.tx_counter % 255;

 int nread = fread(app_layer.data_frame + 4, 1, PACKET_SIZE, fd);

 app_layer.data_frame[2] = (nread >> 8) & 0xFF;
 app_layer.data_frame[3] = nread & 0xFF;

 return (nread + 4);
}

/*
Resets the data frame in the appLayer struct
*/
void resetDataFrame() {
```

```
 BZERO(&APP_LAYER.DATA_FRAME, SIZEOF(APP_LAYER.DATA_FRAME));
}

/*
EXTRACT FILESIZE AND FILENAME FROM DATA_FRAME
*/
INT EXTRACTFILEINFO(UINT8_T* DATA_FRAME, SIZE_T LENGTH) {

 SIZE_T CURRENT_INDEX = DCONTROL_INDEX + 1;

 IF(DATA_FRAME[CURRENT_INDEX] == T_SIZE) {

  SIZE_T PAR_SIZE = DATA_FRAME[CURRENT_INDEX + 1];

  CHAR FILESIZE[PAR_SIZE];

  SNPRINTF(FILESIZE, PAR_SIZE, "%S", DATA_FRAME + CURRENT_INDEX + 2);

  APP_LAYER.FILESIZE = PARSEULONG(FILESIZE, 10);

  CURRENT_INDEX += PAR_SIZE + 2;

 } ELSE RETURN -1;

 IF(DATA_FRAME[CURRENT_INDEX] == T_NAME) {

  SIZE_T PAR_SIZE = DATA_FRAME[CURRENT_INDEX + 1];

  SNPRINTF(APP_LAYER.FILENAME, PAR_SIZE, "%S", DATA_FRAME + CURRENT_INDEX
+ 2);

 } ELSE RETURN -1;

 RETURN 0;
}

/*
WRITES RECEIVED DATA FRAME TO SPECIFIED FD
*/
INT WRITEFILE(FILE* FD, UINT8_T* DATA_FRAME) {

 UINT8_T MSB = DATA_FRAME[2];
 UINT8_T LSB = DATA_FRAME[3];

 SIZE_T NWRITE = (MSB << 8) | (LSB & 0XFF);
```

```
 FWRITE(DATA_FRAME + 4, 1, NWRITE, FD);

 RETURN 0;
}

/*
UPDATE AND SHOW PROGRESS BAR
*/
VOID UPDATEPROGRESSBAR(SIZE_T CURRENT_BYTES) {

 PRINTF("\033C");

 DOUBLE RATIO = (DOUBLE) CURRENT_BYTES / (DOUBLE) APP_LAYER.FILESIZE;

 RATIO *= 100.0;

 RATIO = CEIL(RATIO);

 UINT8_T FILLED_SIZE = RATIO / 100.0 * PROGRESS_BAR_SIZE;

 PRINTF("\U250C");

 SIZE_T I;

 FOR(I = 0; I < PROGRESS_BAR_SIZE; I++) {
  PRINTF("\U2500");
 }

 PRINTF("\U2510\N");
 PRINTF("\U2502");

 FOR(I = 0; I < PROGRESS_BAR_SIZE; I++) {
  IF(FILLED_SIZE > I) PRINTF("\U2588");
  ELSE PRINTF("\U2591");
 }

 PRINTF("\U2502");
 PRINTF(" %D%%\N", (INT) RATIO);

 PRINTF("\U2514");

 FOR(I = 0; I < PROGRESS_BAR_SIZE; I++) {
  PRINTF("\U2500");
 }
```

```
  printf("\u2518\n");
}

/*
Opens specified file and creates data packets for sending the file
*/
int alsend(int port, char* filename) {

  /* Try to open file for sending */
  strcpy(app_layer.filename, filename);

  FILE* fd = fopen(app_layer.filename, "rb");

  if(fd == NULL) {
    printf("%s not found!\n", app_layer.filename);
    exit(1);
  }

  /* Try to establish a connection */
  app_layer.serial_fd = llopen(port, TRANSMIT);

  if(app_layer.serial_fd < 0) {
    printf("alsend: couldn't open serial port for communication.\n");
    exit(1);
  }

  printf("Connection established!\n");

  /* Validate the filesize */
  app_layer.filesize = getFilesize(fd);

  LOG_MSG("Filesize    : %d\n", app_layer.filesize);

  if(app_layer.filesize <= 0) {
    printf("alsend: file has size 0 or exceeds 2 147 483 647 bytes
(~2.14GiB).\n");
    llreset(app_layer.serial_fd);
    exit(1);
  }

  /* Send the start I frame */
  resetDataFrame();
  size_t packet_size = createSEPacket(C_START);
```

```
 LOG_MSG("S/E I FRAME :");
 PRINTDATAFRAME(PACKET_SIZE);

 UINT8_T END_FLAG = FALSE;
 SIZE_T BYTES_RECEIVED = 0;

 WHILE(TRUE) {

  UINT8_T* DATA_FRAME = (UINT8_T*) MALLOC(PACKET_SIZE * 2); //ALLOCATE
DOUBLE AMOUNT FOR BYTE STUFFING
  MEMCPY(DATA_FRAME, APP_LAYER.DATA_FRAME, PACKET_SIZE);

  INT NWRITE = LLWRITE(APP_LAYER.SERIAL_FD, DATA_FRAME, PACKET_SIZE);

  IF(NWRITE < 0) {
   LLRESET(APP_LAYER.SERIAL_FD);
   EXIT(1);
  }

  APP_LAYER.TX_COUNTER++;

  IF(END_FLAG) {
   FREE(DATA_FRAME);
   BREAK;
  }

  RESETDATAFRAME();
  PACKET_SIZE = CREATEDATAPACKET(FD);

  FREE(DATA_FRAME);

  IF(PACKET_SIZE <= HEADER_SIZE - 1) {
   PACKET_SIZE = CREATESEPACKET(C_END);
   END_FLAG = TRUE;
  }

  IF(!END_FLAG) {
   BYTES_RECEIVED += PACKET_SIZE - DPACKET_HEADER_SIZE;
#IF ENABLE_DEBUG == 0
   UPDATEPROGRESSBAR(BYTES_RECEIVED);
#ENDIF
  }
 }

 LLCLOSE_TRANSMIT(APP_LAYER.SERIAL_FD);
```

```
  printf("Connection closed, %s transferred successfully!\n",
app_layer.filename);

  close(app_layer.serial_fd);
  fclose(fd);
  return 0;
}

/*
Attempts to receive a file and writes it to disk
*/
int alreceive(int port) {

  app_layer.serial_fd = llopen(port, RECEIVE);

  if(app_layer.serial_fd < 0) {
    printf("alreceive: couldn't open serial port for communication.\n");
    exit(1);
  }

  printf("Connection established!\n");

  uint8_t* data_frame = (uint8_t*) malloc(FRAME_MAX_SIZE);
  FILE* fd;
  size_t bytes_received = 0;

  while(TRUE) {

    int nread = llread(app_layer.serial_fd, data_frame);

    if(nread >= 0 && data_frame[DCONTROL_INDEX] == C_DATA) {
      bytes_received += nread - HEADER_SIZE;
#if ENABLE_DEBUG == 0
      updateProgressBar(bytes_received);
#endif
    }

    if(nread < 0) continue;

    if(data_frame[DCONTROL_INDEX] == C_START) {
      extractFileInfo(data_frame, nread);
      fd = fopen(app_layer.filename, "wb");
      LOG_MSG("Filename: %s Filesize: %d\n", app_layer.filename,
app_layer.filesize);
```

```
    } else if(data_frame[dcontrol_index] == c_data) {
      writeFile(fd, data_frame);
    } else if(data_frame[dcontrol_index] == c_end) break; //todo
validate filesize / filename
  }

  free(data_frame);

  llclose_receive(app_layer.serial_fd);

  printf("Connection closed, %s transferred successfully!\n",
app_layer.filename);

  close(app_layer.serial_fd);
  return 0;
}

/*
Get the filesize of a file with up to INT_MAX bytes
*/
int getFilesize(FILE* fd) {

  fseek(fd, 0, SEEK_END);
  int filesize = ftell(fd);
  fseek(fd, 0, SEEK_SET);

  if(filesize == INT_MAX) return -1;
  return filesize;
}

/*
Program entry point, parses user input and continues if validation
succeeded
*/
int main(int argc, char** argv) {

  if(argc < 3) {
    printf("%s: wrong number of arguments!\n", argv[0]);
    printUsage(argv[0]);
    exit(1);
  } else if((strcmp("tx", argv[1]) != 0) && (strcmp("rx", argv[1]) != 0)) {
    printf("%s: mode selected does not exist, tx or rx are the available
modes.\n", argv[0]);
    printUsage(argv[0]);
    exit(1);
```

```
 } else if((strcmp("0", argv[2]) != 0) && (strcmp("1", argv[2]) != 0)) {
  printf("%s: port number must be 0 or 1!\n", argv[0]);
  printUsage(argv[0]);
  exit(1);
 } else if((strcmp("tx", argv[1]) == 0) && argc < 4) {
  printf("%s: missing filename after port number!\n", argv[0]);
  printUsage(argv[0]);
  exit(1);
 }

 if((strcmp("tx", argv[1]) == 0) && strlen(argv[3]) + 1 > FILENAME_SIZE)
{
  printf("Filename too long, can only be %d chars long.\n",
FILENAME_SIZE - 1);
  exit(1);
 }

 memset(&app_layer, 0, sizeof(app_layer));

 if(strcmp("tx", argv[1]) == 0) alsend(parseUlong(argv[2], 10), argv[3]);
 if(strcmp("rx", argv[1]) == 0) alreceive(parseUlong(argv[2], 10));

 return 0;
}
```

LINKLAYER.H

```
#IFNDEF LINKLAYER_H
#DEFINE LINKLAYER_H

/*
ATTEMPTS TO ESTABLISH CONNECTION BY SENDING OR RECEIVING SET / UA AND RETURNS THE FILE DESCRIPTOR FOR THE CONNECTION
*/
INT LLOPEN(INT PORT, SERIALSTATUS ROLE);

/*
ATTEMPTS TO CLOSE CONNECTION BY SENDING DISC, RECEIVING DISC AND SENDING UA, RETURNS WHETHER IT SUCCEEDED
*/
INT LLCLOSE_TRANSMIT(INT FD);

/*
ATTEMPTS TO CLOSE CONNECTION BY RECEIVING DISC, SENDING DISC AND WAITING ON UA, RETURNS WHETHER IT SUCCEEDED
*/
INT LLCLOSE_RECEIVE(INT FD);

/*
ATTEMPTS TO SEND DATA_FRAME AFTER BYTE STUFFING IT AND APPENDING A FRAME HEADER
*/
INT LLWRITE(INT FD, UINT8_T* DATA_FRAME, SIZE_T LENGTH);

/*
ATTEMPTS TO READ A DATA_FRAME, BYTE DESTUFFS IT AND VALIDATES IT
*/
INT LLREAD(INT FD, UINT8_T* DATA_FRAME);

/*
RESTORES PREVIOUS SERIAL PORT SETTINGS
*/
VOID LLRESET(INT FD);

#ENDIF /* LINKLAYER_H */
```

LINKLAYER.C

```
#INCLUDE <SYS/TYPES.H>
#INCLUDE <SYS/STAT.H>
#INCLUDE <FCNTL.H>
#INCLUDE <TERMIOS.H>
#INCLUDE <STDIO.H>
#INCLUDE <STDLIB.H>
#INCLUDE <STRING.H>
#INCLUDE <STRINGS.H>
#INCLUDE <STDINT.H>
#INCLUDE <UNISTD.H>
#INCLUDE <SIGNAL.H>
#INCLUDE <LIMITS.H>
#INCLUDE <ERRNO.H>

#INCLUDE "GLOBALS.H"
#INCLUDE "LINKLAYER.H"
#INCLUDE "HELPER.H"

STATIC LINKLAYER LINK_LAYER;

/*
ACTIVATES TIMEOUT FLAG AND INCREMENTS THE TIMEOUT COUNT
*/
VOID TIMEOUTALARM() {

 LINK_LAYER.TIMEOUT_FLAG = TRUE;
 LINK_LAYER.TIMEOUT_COUNT++;
 SIGNAL(SIGALRM, TIMEOUTALARM);
}

/*
CREATES A SUPERVISION OR UNNUMBERED FRAME FROM ADDRESS AND CONTROL FIELD
GIVEN (F | A | C | BCC | F)
*/
VOID CREATESUFRAME(UINT8_T ADDRESS, UINT8_T CONTROL) {

 LINK_LAYER.FRAME[0] = FLAG;
 LINK_LAYER.FRAME[1] = ADDRESS;
 LINK_LAYER.FRAME[2] = CONTROL;
 LINK_LAYER.FRAME[3] = BCC1(ADDRESS, CONTROL);
 LINK_LAYER.FRAME[4] = FLAG;
}

/*
```

```
INITIALIZES SERIAL PORT AND SETS UP THE NEEDED FIELS ON LINKLAYER STRUCT
*/
INT INITSERIAL(UINT32_T BAUDRATE, SERIALSTATUS ROLE) {

 INT FD;

 /* OPEN SERIAL PORT FOR READING AND WRITING AND NOT AS CONTROLLING
TERMINAL */
 FD = OPEN(LINK_LAYER.PORT, O_RDWR | O_NOCTTY);
 IF (FD < 0) {
  PRINTF("INITSERIAL: FAILED TO OPEN PORT %S.\N", LINK_LAYER.PORT);
  RETURN -1;
 }

 /* SAVE CURRENT PORT SETTINGS */
 IF(TCGETATTR(FD, &LINK_LAYER.OLDTIO) == -1) {
  PRINTF("INITSERIAL: COULDN'T GET PORT SETTINGS ON PORT %S.\N",
LINK_LAYER.PORT);
  RETURN -1;
 }

 BZERO(&LINK_LAYER.NEWTIO, SIZEOF(LINK_LAYER.NEWTIO));

 LINK_LAYER.NEWTIO.C_CFLAG = BAUDRATE | CS8 | CLOCAL | CREAD;
 LINK_LAYER.NEWTIO.C_IFLAG = IGNPAR;
 LINK_LAYER.NEWTIO.C_OFLAG = 0;
 LINK_LAYER.NEWTIO.C_LFLAG = 0; /* SET INPUT MODE: NON-CANONICAL / NO ECHO
*/

 LINK_LAYER.NEWTIO.C_CC[VTIME] = 1; /* INTER-CHARACTER TIMER */
 LINK_LAYER.NEWTIO.C_CC[VMIN] = 0; /* MINIMUM NUMBER OF CHARACTERS TO
READ */

 TCFLUSH(FD, TCIOFLUSH);

 IF(TCSETATTR(FD, TCSANOW, &LINK_LAYER.NEWTIO) == -1) {
  PRINTF("INITSERIAL: COULDN'T SET PORT SETTINGS ON PORT %S.\N",
LINK_LAYER.PORT);
  RETURN -1;
 }

 LINK_LAYER.BAUDRATE = BAUDRATE;
 IF(ROLE == TRANSMIT) LINK_LAYER.TIMEOUT_FLAG = TRUE;
 LINK_LAYER.STATE = INIT;
 LOG_MSG("USING %S.\N", LINK_LAYER.PORT);
```

```
 RETURN FD;
}

/*
LINK LAYER STATE MACHINE
*/
VOID LLSTATE(SERIALSTATE* STATE, SERIALEVENT EVENT) {

 SWITCH(*STATE) {
  CASE INIT:
  IF(EVENT == FLAG_E) *STATE = S_FLAG;
  ELSE IF(EVENT == TIMEOUT_E) *STATE = INIT;
  BREAK;
  CASE S_FLAG:
  IF(EVENT == FLAG_E) *STATE = E_FLAG;
  ELSE IF(EVENT == TIMEOUT_E) *STATE = INIT;
  BREAK;
  CASE E_FLAG:
  IF(EVENT == TIMEOUT_E) *STATE = INIT;
  BREAK;
 }
}

/*
HELPER FUNCTION FOR PRINTING A FULL FRAME
*/
VOID PRINTFRAME(SIZE_T SIZE) {

 SIZE_T I;
 FOR(I = 0; I < SIZE; I++) {
  LOG_MSG("|0X%02X", LINK_LAYER.FRAME[I]);
 }

 LOG_MSG("|\N");
}

/*
CHECKS IF ALARM HAS HAPPENED AND RESETS IT IF IT HAS
*/
VOID CHECKALARM() {

 IF(LINK_LAYER.TIMEOUT_FLAG) {
  LOG_MSG("ALARM #%D\N", LINK_LAYER.TIMEOUT_COUNT);
  ALARM(TIMEOUT_S);
```

```
   LINK_LAYER.TIMEOUT_FLAG = FALSE;
  }
}

/*
TURNS OFF ALARM AND RESETS FLAG AND TIMEOUT COUNT
*/
VOID DOWNALARM() {

  ALARM(0);
  LINK_LAYER.TIMEOUT_FLAG = FALSE;
  LINK_LAYER.TIMEOUT_COUNT = 0;
}

/*
CHECKS THE HEADER BCC AND RETURNS TRUE IF IT CHECKS OUT
*/
INT CHECKHEADERBCC() {

  IF(LINK_LAYER.FRAME[BCC1_INDEX] != (LINK_LAYER.FRAME[ADDRESS_INDEX]
^ LINK_LAYER.FRAME[CONTROL_INDEX])) {
   RETURN FALSE;
  } ELSE RETURN TRUE;
}

/*
EMPTIES SERIAL PORT OF DATA
*/
VOID DISCARDFRAME(INT FD) {

  SIZE_T NREAD;

  DO {
   NREAD = READ(FD, LINK_LAYER.FRAME, 1);
  } WHILE(NREAD != 0);

  LOG_MSG("LAST: 0X%02X\N", LINK_LAYER.FRAME[0]);
}

/*
READS A FULL FRAME FROM THE PORT AND CHECKS BCC1 AND THE CONTROL FIELD
EXPECTED, RETURNS ERROR TYPE OR NO ERROR ON SUCCESS
*/
SERIALERROR READFRAME(INT FD, UINT8_T CONTROL_EXPECTED) {
```

```
 SIZE_T NREAD = 0;
 SIZE_T TOTAL_READ = 0;
 LINK_LAYER.LENGTH = 0;

 WHILE(TRUE) {

  NREAD = READ(FD, LINK_LAYER.FRAME + LINK_LAYER.CURRENT_INDEX, 1);

  IF(NREAD > 0) {

   LOG_MSG("RECEIVED[%03I]: 0X%02X | BEFORE STATE: %I |",
LINK_LAYER.CURRENT_INDEX, LINK_LAYER.FRAME[LINK_LAYER.CURRENT_INDEX],
LINK_LAYER.STATE);

   IF(LINK_LAYER.CURRENT_INDEX == 0) {
    IF(LINK_LAYER.FRAME[LINK_LAYER.CURRENT_INDEX] != FLAG) {
     RETURN FLAG_ERR;
    }
   }

   IF(LINK_LAYER.FRAME[LINK_LAYER.CURRENT_INDEX] == FLAG) {
    LLSTATE(&LINK_LAYER.STATE, FLAG_E);
   }

   LOG_MSG(" AFTER STATE: %I\N", LINK_LAYER.STATE);

   LINK_LAYER.CURRENT_INDEX++;
   TOTAL_READ += NREAD;
  }

  IF(LINK_LAYER.STATE == E_FLAG) {
   DOWNALARM();
   LINK_LAYER.LENGTH = TOTAL_READ;
   IF(!CHECKHEADERBCC()) RETURN BCC1_ERR;
   IF(LINK_LAYER.FRAME[CONTROL_INDEX] != CONTROL_EXPECTED) {
    LINK_LAYER.LAST_RESPONSE = LINK_LAYER.FRAME[CONTROL_INDEX];
    RETURN CONTROL_ERR;
   }
   RETURN NO_ERR;
  } ELSE IF(LINK_LAYER.TIMEOUT_FLAG == TRUE) {
   RETURN TIMEOUT_ERR;
  }
 }
}
```

```
/*
RESETS THE LINKLAYER STRUCT FIELDS NEEDED TO RUN ANOTHER WRITE/READ LOOP
*/
VOID RESETFRAME() {

 LINK_LAYER.STATE = INIT;
 LINK_LAYER.CURRENT_INDEX = 0;
 BZERO(&LINK_LAYER.FRAME, SIZEOF(LINK_LAYER.FRAME));
}

/*
CALCULATES THE BCC CORRESPONDING TO A DATA FRAME
*/
UINT8_T CALCBCC2(UINT8_T* DATA_FRAME, SIZE_T LENGTH) {

 UINT8_T BCC = DATA_FRAME[0];

 SIZE_T I;
 FOR(I = 1; I < LENGTH; I++) {

    BCC ^= DATA_FRAME[I];
 }

 RETURN BCC;
}

/*
SEARCHES FOR OCCURRENCES OF FLAG OR THE ESCAPE CHARACTER IN THE DATA FRAME
AND REPLACES THEM
*/
SIZE_T BYTE_STUFF(UINT8_T* DATA_FRAME, SIZE_T LENGTH) {

 CHAR RESULT[FRAME_MAX_SIZE];
 SIZE_T ADD_LENGTH = 0;

 SIZE_T I;
 FOR(I = 0; I < LENGTH; I++) {
  IF(DATA_FRAME[I] == FLAG) {
    RESULT[I + ADD_LENGTH] = ESCAPE_CHAR;
    RESULT[I + 1 + ADD_LENGTH] = FLAG ^ XOCTET;
    ADD_LENGTH++;
  } ELSE IF(DATA_FRAME[I] == ESCAPE_CHAR) {
    RESULT[I + ADD_LENGTH] = ESCAPE_CHAR;
    RESULT[I + 1 + ADD_LENGTH] = ESCAPE_CHAR ^ XOCTET;
    ADD_LENGTH++;
```

```
    } else {
      result[i + add_length] = data_frame[i];
    }
  }

  bzero(data_frame, length);
  memcpy(data_frame, result, length + add_length);

  return (length + add_length);
}

/*
Searches for occurrences of the escape char and replaces it with the
escaped char
*/
size_t byte_destuff(uint8_t* data_frame, size_t length) {

  char result[FRAME_MAX_SIZE];
  size_t new_length = 0;

  size_t i;
  for(i = 0; i < length; i++, new_length++) {
    if(data_frame[i] == ESCAPE_CHAR) {
      result[new_length] = data_frame[i + 1] ^ XOCTET;
      i++;
    } else result[new_length] = data_frame[i];
  }

  bzero(data_frame, length);
  memcpy(data_frame, result, new_length);

  return new_length;
}

/*
Extract data frame from a full frame with BCC2 included
*/
void extractDataFrame(uint8_t* data_frame, size_t length) {

  size_t i;
  for(i = 0; i < length - HEADER_SIZE; i++) {

    data_frame[i] = link_layer.frame[i + 4];
  }
}
```

```
/*
CREATES AN INFORMATION FRAME FROM GIVEN PARAMETERS (F | A | C | BCC1 |
D1..DN | BCC2 | F)
*/
VOID CREATEIFRAME(UINT8_T ADDRESS, UINT8_T CONTROL, UINT8_T* DATA_FRAME,
SIZE_T LENGTH) {

 LINK_LAYER.FRAME[0] = FLAG;
 LINK_LAYER.FRAME[1] = ADDRESS;
 LINK_LAYER.FRAME[2] = CONTROL;
 LINK_LAYER.FRAME[3] = BCC1(ADDRESS, CONTROL);

 MEMCPY(LINK_LAYER.FRAME + 4, DATA_FRAME, LENGTH);

 LINK_LAYER.FRAME[LENGTH + 4] = FLAG;
}

/*
ATTEMPTS TO SEND DATA_FRAME AFTER BYTE STUFFING IT AND APPENDING A FRAME
HEADER
*/
INT LLWRITE(INT FD, UINT8_T* DATA_FRAME, SIZE_T LENGTH) {

 /* CALCULATE BCC2 AND APPEND IT TO DATA_FRAME */
 UINT8_T BCC_2 = CALCBCC2(DATA_FRAME, LENGTH);
 DATA_FRAME[LENGTH] = BCC_2;

 /* BYTE STUFF THE DATA_FRAME AND UPDATE LENGTH */
 SIZE_T STUFFED_LENGTH = BYTE_STUFF(DATA_FRAME, LENGTH + 1);
 LENGTH = STUFFED_LENGTH;

 LOG_MSG("BCC2       : 0X%02X\N", BCC_2);

 UINT8_T SUCCESS = FALSE;
 INT NWRITE;

 SIGNAL(SIGALRM, TIMEOUTALARM);
 LINK_LAYER.TIMEOUT_FLAG = TRUE;

 WHILE(LINK_LAYER.TIMEOUT_COUNT <= TIMEOUT) {

  RESETFRAME();
  CREATEIFRAME(A_TX, C_NS(LINK_LAYER.SEQUENCE_NUMBER), DATA_FRAME,
LENGTH);
```

```
   nwrite = write(fd, link_layer.frame, length + 5);

   LOG_MSG("Sent (b%03i)  :", nwrite);
   printFrame(length + 5); LOG_MSG("\n");

   checkAlarm();

   resetFrame();
   serialError status = readFrame(fd,
C_RR(!link_layer.sequence_number));

   if(status == NO_ERR) {
    success = TRUE;
    break;
   }

   if(status == BCC1_ERR) {
    link_layer.timeout_flag = TRUE;
   } else if(status == CONTROL_ERR) {
    LOG_MSG("Control field was not as expected.\n");
    if(link_layer.last_response == C_REJ(0)) link_layer.sequence_number
= 0;
    if(link_layer.last_response == C_REJ(1)) link_layer.sequence_number
= 1;
    LOG_MSG("Last response: 0x%02X\n", link_layer.last_response);
    link_layer.timeout_flag = TRUE;
   }
  }

  LOG_MSG("Received    :");
  printFrame(link_layer.current_index);

  if(success) {
   link_layer.sequence_number = !link_layer.sequence_number;
   return nwrite;
  }
  else {
   LOG_MSG("Maximum number of retries reached.\n");
   return -1;
  }
 }
```

```
/*
ATTEMPTS TO READ A DATA_FRAME, BYTE DESTUFFS IT AND VALIDATES IT
*/
INT LLREAD(INT FD, UINT8_T* DATA_FRAME) {

 UINT8_T SUCCESS = FALSE;
 UINT8_T BCC_ERROR = FALSE;

 WHILE(SUCCESS == FALSE) {

  RESETFRAME();
  SERIALERROR STATUS = READFRAME(FD, C_NS(LINK_LAYER.SEQUENCE_NUMBER));

  LOG_MSG("RECEIVED(%03D):", LINK_LAYER.LENGTH);
  PRINTFRAME(LINK_LAYER.CURRENT_INDEX);

  IF(STATUS == FLAG_ERR) {
   DISCARDFRAME(FD);
  } ELSE IF(STATUS == BCC1_ERR) {
   LOG_MSG("BCC1 FIELD WAS NOT AS EXPECTED.\N");
  } ELSE IF(STATUS == CONTROL_ERR) {
   LOG_MSG("CONTROL FIELD WAS NOT AS EXPECTED.\N");
   BREAK;
  } ELSE SUCCESS = TRUE;
 }

 IF(SUCCESS) {
  /* EXTRACT DATA FRAME WITH BCC2 INCLUDED */
  EXTRACTDATAFRAME(DATA_FRAME, LINK_LAYER.LENGTH);

  /* DESTUFF DATA FRAME EXTRACTED */
  SIZE_T DESTUFFED_LENGTH = BYTE_DESTUFF(DATA_FRAME, LINK_LAYER.LENGTH -
HEADER_SIZE);
  LINK_LAYER.LENGTH = DESTUFFED_LENGTH;

  UINT8_T BCC_2 = CALCBCC2(DATA_FRAME, LINK_LAYER.LENGTH - 1);

  RESETFRAME();

  IF(BCC_2 != DATA_FRAME[LINK_LAYER.LENGTH - 1]) {
   LOG_MSG("BCC2 ERROR\N");
   CREATESUFRAME(A_TX, C_REJ(LINK_LAYER.SEQUENCE_NUMBER));
   BCC_ERROR = TRUE;
  } ELSE CREATESUFRAME(A_TX, C_RR(!LINK_LAYER.SEQUENCE_NUMBER));
 } ELSE {
```

```
  resetFrame();
  createSUFrame(A_TX, C_RR(link_layer.sequence_number));
 }

 /* Send response */
#if ENABLE_DEBUG == 1
 int nwrite = write(fd, link_layer.frame, SU_FRAME_SIZE);
#else
 write(fd, link_layer.frame, SU_FRAME_SIZE);
#endif

 LOG_MSG("Sent (b%03i)  :", nwrite);
 printFrame(SU_FRAME_SIZE);

 if(!bcc_error && success) link_layer.sequence_number =
!link_layer.sequence_number; //Update sequence number on success

 if(bcc_error || !success) return -1;

 size_t length = link_layer.length - 1;
 link_layer.length = 0;
 return length;
}

/*
Connection establishment specific to the transmitter, returns 0 on success,
negative number otherwise
*/
int llopen_transmit(int fd) {

 uint8_t success = FALSE;

 while(link_layer.timeout_count <= TIMEOUT) {

  resetFrame();
  createSUFrame(A_TX, C_SET);

#if ENABLE_DEBUG == 1
  int nwrite = write(fd, link_layer.frame, SU_FRAME_SIZE);
#else
  write(fd, link_layer.frame, SU_FRAME_SIZE);
#endif

  LOG_MSG("Sent (%03d)   :", nwrite);
  printFrame(SU_FRAME_SIZE);
```

```
    checkAlarm();

    resetFrame();
    serialError status = readFrame(fd, C_UA);

    if(status == NO_ERR) {
      success = TRUE;
      break;
    }

    if(status == BCC1_ERR || status == CONTROL_ERR) {
      LOG_MSG("BCC1 or Control field was not as expected.\n");
      return -1;
    }
  }

  LOG_MSG("Received(%03d):", link_layer.length);
  printFrame(link_layer.current_index);

  if(success) return 0;
  else {
    LOG_MSG("Maximum number of retries reached.\n");
    return -1;
  }
}

/*
Connection establishment specific to the receiver, returns 0 on success,
negative number otherwise
*/
int llopen_receive(int fd) {

  resetFrame();
  serialError status = readFrame(fd, C_SET);

  LOG_MSG("Received(%03d):", link_layer.length);
  printFrame(link_layer.current_index);

  if(status == BCC1_ERR || status == CONTROL_ERR) {
    LOG_MSG("BCC1 or Control field was not as expected.\n");
    return -1;
  }

  resetFrame();
```

```
 createSUFrame(A_TX, C_UA);

#if ENABLE_DEBUG == 1
 int nwrite = write(fd, link_layer.frame, SU_FRAME_SIZE);
#else
 write(fd, link_layer.frame, SU_FRAME_SIZE);
#endif

 LOG_MSG("Sent (%03i)   :", nwrite);
 printFrame(SU_FRAME_SIZE);

 return 0;
}

/*
Attempts to establish connection by sending or receiving SET / UA and
returns the file descriptor for the connection
*/
int llopen(int port, serialStatus role) {

 memset(&link_layer, 0, sizeof(link_layer));
 signal(SIGALRM, timeoutAlarm);

 int fd;

 snprintf(link_layer.port, PORT_NAME_SIZE + 1, "%s%d", "/dev/ttyS",
port);

 fd = initSerial(BAUDRATE, role);

 if(fd < 0) return -1;

 if(role == TRANSMIT) {
  if(llopen_transmit(fd) != 0) {
   llreset(fd);
   return -1;
  }
 } else if(role == RECEIVE) {
  if(llopen_receive(fd) != 0) {
   llreset(fd);
   return -1;
  }
 }

 return fd;
```

```
}

/*
ATTEMPTS TO CLOSE CONNECTION BY SENDING DISC, RECEIVING DISC AND SENDING
UA, RETURNS WHETHER IT SUCCEEDED
*/
INT LLCLOSE_TRANSMIT(INT FD) {

 UINT8_T SUCCESS = FALSE;

 WHILE(LINK_LAYER.TIMEOUT_COUNT <= TIMEOUT) {

  RESETFRAME();
  CREATESUFRAME(A_TX, C_DISC);

#IF ENABLE_DEBUG == 1
  INT NWRITE = WRITE(FD, LINK_LAYER.FRAME, SU_FRAME_SIZE);
#ELSE
  WRITE(FD, LINK_LAYER.FRAME, SU_FRAME_SIZE);
#ENDIF

  LOG_MSG("SENT (%03D)  :", NWRITE);
  PRINTFRAME(SU_FRAME_SIZE);

  CHECKALARM();

  RESETFRAME();
  SERIALERROR STATUS = READFRAME(FD, C_DISC);

  IF(STATUS == NO_ERR) {
   SUCCESS = TRUE;
   BREAK;
  }

  IF(STATUS == BCC1_ERR || STATUS == CONTROL_ERR) {
   LOG_MSG("BCC1 OR CONTROL FIELD WAS NOT AS EXPECTED.\N");
   LLRESET(FD);
   RETURN -1;
  }
 }

 LOG_MSG("RECEIVED(%03D):", LINK_LAYER.LENGTH);
 PRINTFRAME(LINK_LAYER.CURRENT_INDEX);

 RESETFRAME();
```

```
 createSUFrame(A_RX, C_UA);

#if ENABLE_DEBUG == 1
 int nwrite = write(fd, link_layer.frame, SU_FRAME_SIZE);
#else
 write(fd, link_layer.frame, SU_FRAME_SIZE);
#endif

 LOG_MSG("Sent (%03d)   :", nwrite);
 printFrame(SU_FRAME_SIZE);

 sleep(CLOSE_WAIT);

 if(success) {
  llreset(fd);
  return 0;
 } else {
  LOG_MSG("Maximum number of retries reached.\n");
  llreset(fd);
  return -1;
 }
}

/*
Attempts to close connection by receiving DISC, sending DISC and waiting
on UA, returns whether it succeeded
*/
int llclose_receive(int fd) {

 resetFrame();
 serialError status = readFrame(fd, C_DISC);

 LOG_MSG("Received(%03d):", link_layer.length);
 printFrame(link_layer.current_index);

 if(status == BCC1_ERR || status == CONTROL_ERR) {
  LOG_MSG("BCC1 or Control field was not as expected.\n");
  llreset(fd);
  return -1;
 }

 resetFrame();
 createSUFrame(A_RX, C_DISC);

#if ENABLE_DEBUG == 1
```

```
 int nwrite = write(fd, link_layer.frame, SU_FRAME_SIZE);
#else
 write(fd, link_layer.frame, SU_FRAME_SIZE);
#endif

 LOG_MSG("Sent (%03i)   :", nwrite);
 printFrame(SU_FRAME_SIZE);

 resetFrame();
 status = readFrame(fd, C_UA);

 if(status == BCC1_ERR || status == CONTROL_ERR) {
   LOG_MSG("BCC1 or Control field was not as expected.\n");
   llreset(fd);
   return -1;
 }

 llreset(fd);
 return 0;
}

/*
Restores previous serial port settings
*/
void llreset(int fd) {

 if (tcsetattr(fd, TCSANOW, &link_layer.oldtio) == -1) {
   printf("llreset: couldn't set port settings on port %s.\n",
link_layer.port);
   exit(1);
 }
}
```

HELPER.H

```
#IFNDEF HELPER_H
#DEFINE HELPER_H

/*
HELPER FUNCTION FOR PRINTING A GENERIC ARRAY IN HEX FORMAT
*/
VOID PRINTARRAYASHEX(UINT8_T* BUFFER, SIZE_T SIZE);

/*
HELPER FUNCTION FOR PRINTING A DATA FRAME AS ASCII
*/
VOID PRINTARRAYASASCII(UINT8_T* BUFFER, SIZE_T SIZE);

#ENDIF /* HELPER_H */
```

Helper.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <stdint.h>
#include <unistd.h>
#include <signal.h>
#include <limits.h>
#include <errno.h>
#include <math.h>

#include "Globals.h"

/*
Helper function for printing a generic array in hex format
*/
void printArrayAsHex(uint8_t* buffer, size_t size) {

 size_t i;
 for(i = 0; i < size; i++) {
  LOG_MSG("|0x%02X", buffer[i]);
 }

 LOG_MSG("|\n");
}

/*
Helper function for printing a data frame as ASCII
*/
void printArrayAsASCII(uint8_t* buffer, size_t size) {

 size_t i;
 for(i = 0; i < size; i++) {
  LOG_MSG("|%c", buffer[i]);
 }

 LOG_MSG("|\n");
}
```

GLOBALS.H

```
#IFNDEF GLOBALS_H
#DEFINE GLOBALS_H

/*
 USEFUL MACROS
*/

#DEFINE ENABLE_DEBUG 0

#IF ENABLE_DEBUG
#DEFINE LOG_MSG PRINTF
#ELSE
#DEFINE LOG_MSG(...)
#ENDIF

#DEFINE BIT(N)    (0X01 << N)
#DEFINE BCC1(A, B) (A ^ B)
#DEFINE FALSE      0
#DEFINE TRUE       1
#DEFINE ABS(X)    ((X) < 0 ? -(X) : (X))

/*
 APPLICATION LAYER CONSTANTS
*/

#DEFINE PROGRESS_BAR_SIZE   50

#DEFINE PORT_NAME_SIZE      10
#DEFINE FILESIZE_MAX_DIGITS  11
#DEFINE DPACKET_HEADER_SIZE  4
#DEFINE FILENAME_SIZE       256
#DEFINE MAX_START_FRAME     272

/* MAXIMUM NUMBER OF BYTES FOR A START/END DATA FRAME, ALLOWS FOR 256
BYTES FILENAMES */
#DEFINE PACKET_SIZE        256
#DEFINE DATA_FRAME_SIZE     ((PACKET_SIZE + 4 < MAX_START_FRAME) ?
MAX_START_FRAME : PACKET_SIZE + 4)  /* PICKS THE HIGHEST VALUE A DATA
FRAME WILL EVER POSSIBLY NEED */

#DEFINE C_NS(N)           BIT(6) * N
#DEFINE C_DATA            0X01
#DEFINE C_START           0X02
#DEFINE C_END             0X03
```

```
#define T_SIZE          0
#define T_NAME          1

#define DCONTROL_INDEX      0

/*
  Link layer constants
*/

#define FLAG        0x7E            /* Start / End frame flag */
#define ESCAPE_CHAR   0x7D              /* Escape character for byte
stuffing */
#define XOCTET      0x20             /* Octet to xor with for byte stuffing
*/

#define A_TX        0x03            /* Address field when sending
commands as emitter or responses as receiver */
#define A_RX        0x01            /* Address field when sending
commands as receiver or responses as emitter */

#define C_SET       0x03             /* Set up command */
#define C_DISC      0x0B             /* Disconnect command */
#define C_UA        0x07            /* Unnumbered acknowledgment
response */
#define C_RR(n)     (0x05 | BIT(7 * n))     /* Receiver ready response */
#define C_REJ(n)    (0x01 | BIT(7 * n))     /* Reject response */

#define SU_FRAME_SIZE  5                /* Expected supervision frame size
*/
#define ADDRESS_INDEX  1                                        /* Frame
header address index */
#define CONTROL_INDEX  2                /* Frame header control index */
#define BCC1_INDEX    3                 /* Frame header BCC index */
#define HEADER_SIZE   5                 /* Header size without BCC2 */

#define BAUDRATE     B38400
#define TIMEOUT      3                  /* Number of retries on transmission
*/
#define FRAME_MAX_SIZE DATA_FRAME_SIZE * 2 + 6  /* Maximum size for
a frame, usually the maximum I frame possible */
#define TIMEOUT_S    3                  /* Number of seconds to wait per try
*/
#define CLOSE_WAIT    1                 /* Number of seconds to wait before
closing serial port */
```

```
/*
  ENUMERATORS
*/

TYPEDEF ENUM SERIALSTATUS {TRANSMIT, RECEIVE} SERIALSTATUS;
/* CURRENT ROLE OF THE PROGRAM */
TYPEDEF ENUM SERIALSTATE {INIT, S_FLAG, E_FLAG} SERIALSTATE;
/* LINK LAYER STATE MACHINE STATES */
TYPEDEF ENUM SERIALEVENT {FLAG_E, TIMEOUT_E} SERIALEVENT;
/* LINK LAYER STATE MACHINE EVENTS */
TYPEDEF ENUM SERIALERROR {NO_ERR, BCC1_ERR, CONTROL_ERR,
TIMEOUT_ERR, FLAG_ERR} SERIALERROR;  /* LINK LAYER POSSIBLE ERRORS */

/*
  STRUCT CONTAINING APPLICATION LAYER DATA
*/
TYPEDEF STRUCT APPLICATIONLAYER {
  INT SERIAL_FD;
  SERIALSTATUS STATUS;
  UINT16_T TX_COUNTER;
  INT FILESIZE;
  CHAR FILENAME[FILENAME_SIZE];
  UINT8_T DATA_FRAME[DATA_FRAME_SIZE];
} APPLICATIONLAYER;

/*
  STRUCT CONTAINING LINK LAYER DATA
*/
TYPEDEF STRUCT LINKLAYER {
  CHAR PORT[PORT_NAME_SIZE + 1];
  UINT32_T BAUDRATE;
  UINT8_T SEQUENCE_NUMBER;
  UINT32_T LENGTH;
  VOLATILE UINT8_T TIMEOUT_COUNT;
  VOLATILE UINT8_T TIMEOUT_FLAG;
  UINT16_T CURRENT_INDEX;
  UINT8_T LAST_RESPONSE;
  SERIALSTATE STATE;
  STRUCT TERMIOS OLDTIO;
  STRUCT TERMIOS NEWTIO;
  UINT8_T FRAME[FRAME_MAX_SIZE];
} LINKLAYER;

#ENDIF /* GLOBALS_H */
```

MAKEFILE

```
CC = GCC
CFLAGS = -WALL -STD=GNU99
SRCS = HELPER.C APPLAYER.C LINKLAYER.C

ALL: $(SRCS)
	$(CC) $(CFLAGS) $(SRCS) -LM -O RCOM
```