1º Trabalho Laboratorial - Ligação de Dados

FEUP - RC - Turma 7, Grupo 7

2021/22

Duarte Guedes Sardão
up201905497

José Pedro Ferreira
up201904515

Lucas Calvet Santos
up201904517

Introdução

O objetivo do trabalho laboratorial era criar uma aplicação e protocolo que permitissem o envio de ficheiros através de uma porta série com o uso de tramas. O objetivo do relatório será explicitar o seu desenvolvimento. Este relatório será divido nas seguintes secções:

1. Arquitetura
2. Estrutura do código
3. Caso de uso
4. Protocolo de ligação lógica
5. Protocolo de aplicação
6. Validação
7. Eficiência do protocolo de ligação de dados
8. Conclusões

Caso de Uso

Para fazer a transferência de um ficheiro com writenoncanonical.c e readnoncanonical.c é necessário compilar e executar os referidos programas, em cada computador, respetivamente, e com a porta de série como argumento. No caso do write, será necessário também o argumento com o path do ficheiro a enviar e no caso do read, o utilizador poderá, facultativamente, definir o path do ficheiro de output como argumento. Ambos aceitam a flag `-v` que ativará os prints *verbose*, para debug.

Compilar **writenoncanonical.c** : `gcc -Wall writenoncanonical.c link_layer.c app.c [-o <Nome do executável>]`

Compilar **readnoncanonical.c** : `gcc -Wall readnoncanonical.c link_layer.c app.c [-o <Nome do executável>]`

Considerando, a título de exemplo, `wnc` como o executável resultante da compilação da writenoncanonical.c e `rnc` o executável resultante da compilação da readnoncanonical.c

Executar o binário referente ao **writenoncanonical.c** :

wnc SerialPort InputFile [Verbose Mode]
wnc /dev/ttyS [-v]

- X is the serial port number
- file_path is the path of the file to read and then send
- v (optional) enables verbose mode

Exemplos: `./wnc /dev/ttyS1 pinguim.gif\n` ou `./wnc /dev/ttyS1 pinguim.gif\n -v`

Executar o binário referente ao **readnoncanonical.c** :

rnc SerialPort OutputFile [Verbose Mode]
rnc /dev/ttyS [-v]

- X is the serial port number
- file_path is the path of the file to create
- v (optional) enables verbose mode

Exemplos: `./rnc /dev/ttyS1 pinguim.gif\n` ou `./rnc /dev/ttyS1 pinguim.gif\n -v`

O writenoncanonical.c irá primeiro ler e tratar dos argumentos. Depois de abrir o ficheiro dado para envio, irá abrir a porta lógica.

```
int port_fd = llopen(argv[1], TRANSMITTER);
if (port_fd == -1)
  error(1, errno, "cannot open the serial port");
if (port_fd == -2)
  error(1, 0, "no response after %d tries, cannot establish a connection", TRIES);
if (port_fd == -3)
  error(1, 0, "got unexpected response, cannot establish a connection");
if (port_fd < 0)
  error(1, 0, "error opening the serial port");
if (verbose)
  printf("Connection established as the transmitter.\n");
```

De seguida, procedirar-se ao envio em si, preparando o pacote de controlo e fazendo write.

```
unsigned char *app_packet;
int size, res;
size = make_control_package(TRUE, file_info.st_size, basename(filepath), &app_packet);
res = llwrite(port_fd, app_packet, size);
free(app_packet);
```

Se este não for recebido, irá tentar repetidas vezes, saindo sem sucesso passado número de TRIES, definido em macros. No outro caso, poderá proceder-se à escrita do ficheiro em si: num while loop, writenoncanonical lê MAX_DATA_SIZE – 4 bytes do ficheiro para um buffer, preparando então o pacote de dados e enviando. Além disso, manterá conta do tamanho dos dados já escritos.

```
size = make_data_package(seq_n, read_buffer, read_size, &app_packet);
seq_n++;
int llw_res = llwrite(port_fd, app_packet, size);

if (llw_res < 0)
{
  free(app_packet);
  error(1, 0, "error in llwrite");
```

```
    }

    write_file_size += size - 4;
```

Quando não ler mais dados do ficheiro, o loop de write irá terminar. Aí, escreverá um pacote de controlo final e chamará llclose, para fechar a porta de série e close, para fechar o ficheiro agora enviado.

Por sua vez, readnoncanonical.c passará um processo semelhante de interpretação de argumentos e abertura da porta série, esperando pelo pacote inicial de controlo de writenoncanonical.

```
  while (!received_start_cp)
  {
    package_len = llread(port_fd, &package);
    if (package[0] == CP_START)
    {
      if(verbose) printf("Start Control Package Received\n");
      read_control_package(package, package_len, &file_size, &file_name);
      received_start_cp = TRUE;
    }
    free(package);
  }
```

Depois de abrir o ficheiro para onde fará output, readnoncanonical.c entrará em um loop para proceder à leitura dos pacotes de dados enviados.

```
while (!end_package_stream)
  {
    if ((package_len = llread(port_fd, &package)) < 0)
    {
      free(package);
      error(1, errno, "llread failed");
    }

    switch (package[0])
    {
    case DP:
      package_len = read_data_package(package, &seq_n, &data);
      int written_size = write(file_fd, data, package_len);
      free(data);
      if (package_len <= 0 || written_size != package_len)
      {
        free(package);
        error(1, errno, "write to file failed");
      }
      read_file_size += written_size;
      break;
```

Quando receber o pacote de controlo final, verifica a integridade da escrita, verificando que o nome e tamanho do ficheiro escrito são congruentes com o recebido. Tal como write, fará llclose e close.

Protocolo de Ligação Lógica

O protocolo de ligação foi implementado no ficheiro link_layer.h e link_layer.c. O protocolo apresenta funções para abrir portas, llopen, escrever nestas, llwrite, ler de estas, llread e fechar, llclose. No caso de escritas, tanto em llwrite como em acknowledgements de open, read e close, será chamado timeout_write, que irá tentar repetir escritas falhadas, com uso de alarme.

```
if (alarm_set)
{
  alarm_set = FALSE;
  tries--;
  if (tries > 0)
  {
    write(fd, to_write, write_size);
    printf("Alarm triggered, trying again. %d tries left\n", tries);
    alarm(TIMEOUT);
  }
}
```

O protocolo também lida com a criação dos headers de tramas, dispondo de: make_bbc, make_info, funções de byte stuffing e destuffing e funções para definir SET, UA, RR, REJ, e DISC. Protocolo de Aplicação Nos ficheiros app.c e app.h estão definidas as funções que fazem e leem pacotes de dados e controlo.

Protocolo da Aplicação

O Protocolo de Aplicação foi definido nos ficheiros `app.h` e `app.c`. A aplicação suporta dois tipos de pacotes:

- **Pacotes de controlo** para sinalizar o início e o fim da transferência do ficheiro
- **Pacotes de dados** contendo fragmentos do ficheiro a transmitir

Assim, o Protocolo de Aplicação tem apenas a responsabilidade de criar e interpretar estes dois tipos de pacotes, "ignorando" a estrutura das tramas referentes ao Protocolo de Ligação de Dados. Para tal contém apenas quatro funções:

- `make_control_package`
- `read_control_package`
- `make_data_package`
- `read_data_package`

A função `make_control_package` cria os pacotes de controlo. Recebe um booleano que indica se se trata de um pacote de sinalização de início (com '2' no campo de controlo) ou de final (com '3' no campo de controlo). Além disso, recebe o tamanho do ficheiro a ser transmitido e o nome do mesmo, parâmetros que vão ser codificados com o formato TLV no pacote de controlo. O campo Type será 0 para o tamanho do ficheiro e 1 para o nome, por convenção. O pacote de controlo resultante será passado por referência para o argumento `control_package`.

```
int make_control_package(int start, int file_size, char *file_name, unsigned char **control_
{
    unsigned char *result_package = (unsigned char *)malloc(MAX_PACKET_SIZE * sizeof(unsigne
    if (start)
    {
        result_package[0] = CP_START;
    }
    else
    {
        result_package[0] = CP_END;
    }

    result_package[1] = CP_T_FSIZE;
    result_package[2] = sizeof(int);

    memcpy(&result_package[3], &file_size, sizeof(int));

    int index = 3 + sizeof(int);

    result_package[index++] = CP_T_FNAME;
    result_package[index++] = strlen(file_name) + 1;

    for (int i = 0; i < strlen(file_name) + 1; i++)
    {
        result_package[index + i] = file_name[i];
    }

    *control_package = result_package;
    return index + strlen(file_name) + 1;
}
```

Por sua vez, a função `read_control_package` cria os pacotes de controlo. Recebe um booleano que indica se se trata de um pacote de sinalização de início (com '2' no campo de controlo) ou de final (com '3' no campo de controlo). Além disso, recebe o tamanho do ficheiro a ser transmitido e o nome do mesmo, parâmetros que vão ser codificados com o formato TLV no pacote de controlo. O campo Type será 0 para o tamanho do ficheiro e 1 para o nome, por convenção.