Primeiro Trabalho Labortorial

Hugo Guimarães, Pedro Ponte

November 16, 2020

1 Sumário

Este trabalho foi realizado no contexto da cadeira Redes de Computadores, com o objetivo de implementar um protocolo de ligação de Dados através de uma porta série, permitindo a transmissão de um ficheiro entre 2 computadores.

Deste modo, o trabalho foi concluído com sucesso, dado que foi possível implementar uma aplicação que cumprisse os objetivos estabelecidos.

2 Introdução

Este trabalho pretende implementar um protocolo de ligação de dados baseado no guião fornecido, de modo a ser possível transferir ficheiros através de uma porta série.

O relatório pretende descrever detalhadamente a aplicação implementada, estando dividida nas seguintes secções:

2.1 Arquitetura

Descrição dos blocos funcionais e interfaces implementados.

2.2 Estrutura do Código

Descrição das APIs, principais estruturas de dados, principais funções e a sua relação com a arquitetura

2.3 Casos De Uuso Principais

Identificação dos principais casos de uso e da sequência de chamada de funções

2.4 Protocolo De Ligação Lógica

Descrição dos principais aspetos funcionais do protocolo de ligação lógica e da sua estratégia de implementação com apresentação de extratos de código.

2.5 Protocolo De Aplicação

Descrição dos principais aspetos funcionais do protocolo de aplicação e da sua estratégia de implementação com apresentação de extratos de código.

2.6 Validação

Descrição dos testes efetuados com apresentação quantificada dos resultados

2.7 Eficiência Do Protocolo De Ligação De Dados

Caracterização estatística da eficiência do protocolo de Stop&Wait implementado.

2.8 Conclusões

Síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

3 Arquitetura

O trabalho está divido em 2 secções fundamentais, o emissor e recetor. Ambos incorporam a sua própria camada de ligação de dados e aplicação.

4 Estrutura do Código

O código está dividido em vários ficheiros.

Os ficheiros llfunctions.c e stateMachines.c são responsáveis pelo tratamento do protocolo da ligação de dados, sendo stateMachines.c unicamente responsável pela implementação das máquinas de estado de aceitação de mensagens.

O ficheiro application.c é responsável pelo tratamento do protocolo de aplicação

Os ficheiros emissor.c e recetor.c são responsáveis pelo fluxo de execução do programa, dos lados do emissor e recetor, respetivamente. Ambos contêm apenas a função main e todas as funções chamadas estão implementadas nos restantes ficheiros.

emissor.c

- **main** - Controla os processos ao nível da camada da aplicação da parte do emissor e faz as chamadas às funções da camada de ligação.

recetor.c

- **main** - Controla os processos ao nível da camada da aplicação da parte do recetor e faz as chamadas às funções da camada de ligação.

llfunctions.c

- **llopen** - Do lado do emissor, envia uma trama de supervisão SET e recebe uma trama UA, enquanto no lado do recetor este espera pela trama de controlo SET enviada pelo emissor e responde com uma trama UA.
- **llclose** - Do lado do emissor, envia uma trama de supervisão DISC, espera que o emissor responda com uma trama DISC e envia uma trama UA. No lado do recetor, este aguarda pela trama DISC enviada pelo emissor, responde com uma trama DISC e depois recebe uma trama UA.

- **llwrite** - Faz o stuffing das tramas I e envia-as, recebendo REJ ou RR como resposta.
- **llread** - Lê as tramas I enviadas pelo llwrite e envia uma resposta do tipo RR, no caso das tramas I recebidas sem erros detetados no cabeçalho e no campo de dados, ou do tipo REJ, no caso das tramas I sem erro detetado no cabeçalho, mas com erros no campo de dados.
- **alarmHandler** - Substituição do handler do alarme para permitir que as tramas sejam enviadas MAXTRIES vezes em situação de erro.
- **getBcc2** - Gera o BCC2 no lado do emissor.
- **stuffBCC2** - Realiza o stuff do BCC2 no lado do emissor, após a geração do BCC.

  Variaveis globais

  - **STP**
  - **counter** - Contador de chamadas ao alarmHandler, inicializada a 0.
  - **trama** - Representa o número sequencial da trama enviada pelo emissor (Ns), inicializada a 0.

stateMachines.c

- **sendMessage** - Faz o parse da trama e envia-a pela porta de série.
- **readSetMessage** - Máquina de estados que recebe a trama SET e verifica a sua correção.
- **readReceiverMessage** - Recebe as tramas REJ e RR enviadas pelo recetor e verifica a sua correção.
- **receiveUA** - Recebe as tramas UA e verifica a sua correção.
- **receiverRead_StateMachine** - Recebe as tramas I enviadas pelo emissor, verifica a sua correção, efetua o destuffing necessário, guarda os dados contidos nas tramas I num novo array e envia uma trama REJ ou RR como resposta, dependendo da ocorrência de erros nas tramas recebidas ou no respetivo destuffing.
- **receiveDISC** - Recebe as tramas DISC e verifica a sua correção.
- **checkBCC2** - Verifica a correção do BCC2 no lado do recetor.

  Variaveis globais

  - **STP**
  - **counter** - Contador de chamadas ao alarmHandler, inicializada a 0.

  - **trama** - Representa o número sequencial da trama enviada pelo emissor (Ns), inicializada a 0.

applications.c

- **openFile** - Abre o ficheiro enviado como argumento e obtém o seu conteúdo e tamanho.
- **parseControlPacket** - Gera o pacote de controlo de um ficheiro para depois ser enviado.
- **parseDataPacket** - Codifica a mensagem num pacote de acordo com o protocolo estabelecido.
- **splitPacket** - Obtém uma porção da mensagem, de modo a enviar os dados sob a forma de uma trama I.
- **checkStart _StateMachine** - Verifica se o primeiro pacote recebido pelo recetor é de facto o pacote de controlo de início.
- **checkEND** - Verifica se o pacote de controlo inicial é igual ao final.
- **assembleDataPacket** - Obtém os dados enviados pelo emissor através do pacote recebido pela porta série.
- **createFile** - Cria o ficheiro final após ter lido toda a informação através da porta série. **Variaveis globais**
  - **packetNumber** - Contagem do número de pacotes enviados.

5 Casos De Uso Principais

Este trabalho laboratorial tem 2 casos de uso distintos: a interface e a transmissão do ficheiro. A interface permite ao utilizador iniciar a aplicação. No lado do emissor, seleciona a porta de série que pretende utilizar (ex: /dev/ttyS0) e o ficheiro que pretende enviar. Do lado do recetor, basta apenas selecionar a porta de série a ser utilizada.

A transmissão do ficheiro, através da porta de série, entre os 2 dispositivos ocorre da seguinte forma:

- Configuração da ligação e escolha do ficheiro a ser enviado pelo emissor;
- Estabelecimento da ligação entre o emissor e o recetor;
- Envio, trama a trama, dos dados por parte do emissor; Receção dos dados enviados pelo recetor, que os guarda num ficheiro com o mesmo nome do original à medida que os vai recebendo;
- Terminação da ligação.

6 Protocolo De Ligação Lógica

O objetivo do protocolo de ligação lógica é estabelecer a ligação estável e fiável entre os 2 computadores, utilizando a porta de série. Para isso, implementamos, tal como é referido no enunciado, as funções llopen, llread, llwrite e llclose.

6.1 LLOPEN

Esta função é responsável por estabelecer a ligação entre o emissor e o recetor através da porta de série.

Do lado do transmissor, esta função instala o alarme que vai ser utilizado ao longo da ligação, envia uma trama SET ao recetor, ficando depois à espera que este envie na resposta uma trama do tipo UA. Caso o recetor não responda passados 3 segundos, o emissor volta a reenviar a trama SET, aguardando depois uma resposta do outro lado. Caso volte a ficar sem resposta ao fim dos 3 segundos, repete o envio mais uma vez e no caso de mais um insucesso o programa termina. Caso o recetor responda com a trama UA, então a ligação é estabelecida.

Do lado do recetor, este aguarda o envio da trama SET por parte do emissor e responde com o envio de uma trama do tipo UA.

6.2 LLWRITE

A função llwrite é responsável pelo stuffing e envio das tramas do tipo I.

Inicialmente, é acrescentado um cabeçalho à mensagem, de acordo com o protocolo descrito no guião. De seguida, é feito o stuffing do BCC2 e da mensagem, pelo que a trama está pronta para ser enviada.

O processo de envio das tramas do tipo I está protegido por um alarme com 3 segundos de espera e 3 tentativas.

Após o envio, é esperada uma resposta pela parte do recetor, através do comando RR, que simboliza que a trama foi transmitida corretamente, ou do comando REJ, que indica problemas no envio da trama, originando um reenvio da trama original.

6.3 LLREAD

A função llread recebe as tramas do tipo I enviadas pelo emissor.

A trama recebida é lida e analisada através de uma máquina de estados, sendo feitas as verificações do cabeçalho e do campo de dados e realizado o respetivo destuffing caso seja necessário.

Caso a trama recebida não tenha erros no cabeçalho e caso seja uma nova trama, mas possua erros no campo de dados, é enviada uma resposta do tipo REJ para o emissor, pedindo uma retransmissão dessa trama. Caso contrário, é enviada uma resposta do tipo RR.

Se a trama recebida não possuir erros no cabeçalho e no campo de dados, ou caso a trama seja um duplicado, é confirmada ao emissor através de uma trama RR.

Tramas com o cabeçalho errado são ignoradas, sem qualquer ação.

6.4 LLCLOSE

A função llclose tem como objetivo concluir a ligação entre o emissor e o recetor.

O emissor envia uma trama DISC, esperando por uma resposta do emissor da mesma trama DISC. Caso a receba, envia uma trama UA para finalizar a ligação.

O emissor está protegido por um alarme de 3 tentativas de 3 segundos de espera, tal como as funções mencionadas anteriormente.

O recetor espera por uma trama DISC e, caso a receba, envia de volta uma trama DISC, esperando por uma trama UA para finalizar a ligação.

7 Protocolo De Aplicação

O protocolo de aplicação contém as seguintes funcionalidades:

- Leitura da informação sobre o ficheiro a enviar; Geração e leitura de pacotes de controlo do tipo START e END, contendo o tamanho e o nome do ficheiro;
- Divisão do ficheiro em fragmentos mais pequenos;
- Preenchimento de um fragmento do ficheiro com um cabeçalho de controlo;
- Leitura do ficheiro a criar, do lado do recetor, e criação do mesmo sem alterações;

7.1 OpenFile

Abre o ficheiro recebido e retorna os dados do ficheiro, assim como o seu tamanho.

7.2 ParseControlPacket

Gera um pacote de controlo do tipo START ou END, contendo o tamanho e o nome do ficheiro.

7.3 ParseDataPacket

Gera um pacote de dados, preenchendo-o com um cabeçalho contendo uma FLAG de controlo, o número de pacotes, o tamanho do ficheiro e o respetivo fragmento do ficheiro a ser enviado.

7.4 SplitPacket

Divide o ficheiro em fragmentos mais pequenos.

7.5 CheckStart

Verifica se o pacote de controlo foi recebido corretamente e obtém de lá o tamanho e o nome do ficheiro.

7.6 CheckEND

Compara o pacote de controlo do tipo START enviado antes da transmissão dos dados com o do tipo END recebido no final da transmissão, verificando se os campos com o tamanho e nome do ficheiro são iguais.

7.7 AssembleDataPacket

Retorna o campo de dados de um pacote.

7.8 CreateFile

Gera um ficheiro de acordo com os dados recebidos.

8 Validação

Foi testado o envio de vários ficheiros, incluindo ficheiros com uma elevada quantidade de dados, os quais foram enviados do emissor para o recetor corretamente, sem perda de informação.

Relativamente aos testes relativos à interrupção da ligação do cabo de série e geração de ruído, não fomos capazes de apresentar imagens relativas ao seu procedimento, porém, o seu sucesso foi comprovado na presença do docente no decurso da apresentação do projeto.

9 Eficiência Do Protocolo De Ligação De Dados

9.1 Variação da capacidade de Baudrate

Foi utilizada a imagem do pinguim, com um tamanho de 35.4KB, sobre a qual se fez variar os valores de Baudrate.

Foi possível concluir que o aumento do Baudrate provoca uma diminuição da eficiência, embora o tempo de execução seja menor.

9.2 Variação do tamanho das tramas

Utilizando uma imagem de tamanho 35.4KB, e um Baudrate de 38400, fez-se variar o tamanho de envio das tramas em cada llwrite.

Foi possível concluir que o aumento do tamanho da trama de envio provocou o aumento da eficiência, sendo o tempo de execução menor.

9.3 Atraso no envio das tramas

Utilizando uma imagem de tamanho 35.4KB, e um Baudrate de 38400, e o envio de 128 bytes em cada trama, introduziu-se um atraso no envio de cada trama no llwrite, através da função usleep().

Tal como esperado, foi possível concluir que a introdução de um atraso no envio de tramas causa uma diminuição da eficiência do código, sendo o tempo de execução cada vez menor quanto maior o atraso introduzido.

9.4 Geração de erros no cabeçalho e no campo de dados

Foram criadas duas funções, generateRandomBCC e generateRandomBCC2, de modo a gerar erros no cabeçalho e campo de dados, respetivamente, a uma percentagem definida no ficheiro macros.h, através das macros BCC1ERRORRATE e BCC2ERRORRATE.

Utilizando uma imagem de tamanho 35.4KB, e um Baudrate de 38400, e o envio de 128 bytes em cada trama, fez-se variar os valores de BCC1ERRORRATE e BCC2ERRORRATE.

Tal como esperado, foi possível concluir que o aumento da taxa de erros gerados no cabeçalho e campo de dados provocou uma diminuição da eficiência, também como um aumento do tempo de execução.

10 Conclusões

Foi possível alcançar o objetivo proposto do trabalho, a implementação de um protocolo de ligação de Dados através de uma porta série, sendo possível enviar com sucesso ficheiros de diferentes tamanhos e extensões.

Foi possível compreender não só o processo de implementação de uma protocolo de ligação de dados, mas também as condições que afetam a eficiência do protocolo, através da alteração do tamanho da trama de envio, Baudrate, quantidade de erros e atraso no envio das tramas.

11 Anexos

11.1 Codigo

11.2 emissor.h

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>

#include <errno.h>
#include <signal.h>
#include <stdlib.h>
#include <time.h>

#include <string.h>

#include "llfunctions.h"
#include "application.h"


/**
 * \brief main function that starts the proggram flow
 * @param argc argument count
 * @param argv char pointer array with the arguments
 */
int main(int argc, char** argv);
```

11.3 emissor.c

```c
/*Non-Canonical Input Processing*/
#include "emissor.h"

extern unsigned int packetNumber;

int main(int argc, char** argv)
{
  int fd;

  if ((argc < 3) || ((strcmp("/dev/ttyS0", argv[1])!=0) && (strcmp(
    "/dev/ttyS1", argv[1])!=0))) {
    printf("Usage:\tnserial SerialPort File path\n\tex: nserial /
    dev/ttyS1 \t filename.jpg \n");
    return -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.
  */

  struct timespec initialTime, finalTime;
  clock_gettime(CLOCK_REALTIME, &initialTime);

  if ((fd = open(argv[1], O_RDWR | O_NOCTTY )) < 0) {
    perror(argv[1]);
    return -2;
  }

```

```
28
29  int fileNameSize = strlen(argv[2]);
30  char* filename = (char*)malloc(fileNameSize);
31  filename = (char*)argv[2];
32  off_t fileSize = 0;
33  int sizeControlPacket = 0;
34
35  unsigned char *data = openFile(filename, &fileSize);
36
37  // Dealing with the SET and UA
38  if(llopen(fd, TRANSMITTER) == ERROR){
39    puts("TRANSMITTER: Error on llopen");
40    return -3;
41  }
42
43  // Start Control packet
44  unsigned char *start = parseControlPacket(CT_START, fileSize,
      filename, fileNameSize, &sizeControlPacket);
45
46  if(llwrite(fd, start, sizeControlPacket) != TRUE ){
47    puts("TRANSMITTER: Error writing START control packet");
48    return -4;
49  }
50  free(start);
51
52  // Ciclo de envio dos packets
53  int packetSize = PACKETSIZE;
54  off_t index = 0;
55
56  while(index < fileSize && packetSize == PACKETSIZE){
57    unsigned char* packet = splitPacket(data, &index, &packetSize,
      fileSize);
58
59    int length = packetSize;
60
61    unsigned char* message =  parseDataPacket(packet, fileSize, &
      length);
62
63    if(llwrite(fd, message, length) != TRUE){
64      puts("TRANSMITTER: Error sending data packet");
65      return -5;
66    }
67
68    printf("Sent packet number: %d\n", packetNumber);
69
70    free(message);
71  }
72
73
74  // End Control packet
75  unsigned char *end = parseControlPacket(CT_END, fileSize,
      filename, fileNameSize, &sizeControlPacket);
76
77  if(llwrite(fd, end, sizeControlPacket) != TRUE ){
78    puts("TRANSMITTER: Error writing END control packet");
79    return -6;
80  }
```

```
  free(end);


  if(llclose(fd, TRANSMITTER) == ERROR){
    puts("TRANSMITTER: Error on llclose");
    return -7;
  }

  clock_gettime(CLOCK_REALTIME, &finalTime);

  double accum = (finalTime.tv_sec - initialTime.tv_sec) + (
    finalTime.tv_nsec - initialTime.tv_nsec) / 1E9;

  printf("Seconds passed: %f\n", accum);

  sleep(1);
  close(fd);
  free(data);

  return 0;
}
```

11.4 recetor.h

```
/*Non-Canonical Input Processing*/

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

#include <time.h>


// Created files

#include "llfunctions.h"
#include "application.h"
```

11.5 recetor.c

```
#include "recetor.h"

extern unsigned int packetNumber;

int main(int argc, char** argv)
{
  int fd;
  off_t index = 0;

  if ( (argc < 2) ||
       ((strcmp("/dev/ttyS0", argv[1])!=0) &&
```

```
12          (strcmp("/dev/ttyS1", argv[1])!=0) )) {
13      printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS1\n"
        );
14      return -1;
15    }
16
17  /*
18    Open serial port device for reading and writing and not as
        controlling tty
19    because we don't want to get killed if linenoise sends CTRL-C.
20  */
21
22    struct timespec initialTime, finalTime;
23    clock_gettime(CLOCK_REALTIME, &initialTime);
24
25    fd = open(argv[1], O_RDWR | O_NOCTTY );
26    if (fd <0) {
27      perror(argv[1]);
28      return -2;
29    }
30
31    if(llopen(fd, RECEIVER) == ERROR){
32      puts("Error on llopen");
33      return -3;
34    }
35
36    unsigned char* start = malloc(0);
37    unsigned int size, sizeStart;
38
39
40    size = llread(fd,start);
41    sizeStart = size;
42
43
44    unsigned int fileSize = 0;
45    unsigned int nameSize = 0;
46    char *fileName  = (char *)malloc(0);
47
48    if(checkStart(start,&fileSize,fileName,&nameSize) == ERROR){
49      puts("Error on checkStart");
50      return -4;
51    }
52
53    // Loop for reading all llwrites from the emissor
54    unsigned char* dataPacket;
55    unsigned int packetsRead = 0;
56    unsigned int messageSize;
57
58    unsigned char* final;
59
60    unsigned char* result = (unsigned char*)malloc(fileSize); //
        Creates null pointer to allow realloc
61
62    while(TRUE){
63      unsigned int packetSize = 0;
64      unsigned char* message = malloc(0);
65      messageSize = 0;
```

```
66
67      if((messageSize = llread(fd,message)) == ERROR){
68        puts("Error on llread data packet ");
69        return -5;
70      }
71      printf("message size = %d\n",messageSize);
72
73      if(messageSize == 0){
74        continue;
75      }
76      else if (message[0] == CT_END){
77        puts("Reached Control End Packet");
78        final = (unsigned char*)malloc(messageSize);
79        memcpy(final,message,messageSize);
80        break;
81      }
82
83      packetsRead++;
84
85      printf("Received packet number: %d\n", packetsRead);
86
87      dataPacket = assembleDataPacket(message,messageSize,&packetSize
        );
88
89      for(int i= 0; i < packetSize; i++){
90        result[index + i] = dataPacket[i];
91      }
92
93      index += packetSize;
94
95      free(dataPacket);
96    }
97
98    if(checkEND(start, sizeStart, final, messageSize) == 1) {
99      puts("Start and End packets are different!");
100     return -6;
101   }
102
103   printf("Received a file of size: %u\n", fileSize);
104
105   // Creating the file to be rewritten after protocol
106   createFile(result,fileSize,fileName);
107
108
109   if(llclose(fd, RECEIVER) == ERROR){
110     puts("Error on llclose");
111     return -7;
112   }
113
114
115   clock_gettime(CLOCK_REALTIME, &finalTime);
116   double accum = (finalTime.tv_sec - initialTime.tv_sec) + (
        finalTime.tv_nsec - initialTime.tv_nsec) / 1E9;
117   printf("Seconds passed: %f\n", accum);
118   sleep(1);
119
120   free(fileName);
```

```
121   free(result);
122
123   close(fd);
124
125   return 0;
126 }
```

11.6 llfunctions.h

```
1
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <unistd.h>
5  #include <sys/types.h>
6  #include <sys/stat.h>
7  #include <fcntl.h>
8  #include <termios.h>
9  #include <errno.h>
10 #include <signal.h>
11 #include <string.h>
12
13 #include "stateMachines.h"
14 #include "macros.h"
15
16 /**
17  * \brief Deals with the protocol initiation establishment
18  * @param fd file descriptor for the serial port to be used for the
       connection
19  * @param status If 0, sends SET message and waits for UA, if 1,
      waits for set and sends UA
20  * @return returns 0 upon sucess, -1 otherwise
21  */
22 int llopen(int fd, int status);
23
24
25 /**
26  * \brief gets BCC2
27  * @param message gets BCC2 from this message
28  * @param size message size
29  * @return returns BCC2
30  */
31 unsigned char getBCC2(unsigned char *mensagem, int size);
32
33 /**
34  * \brief stuffs BCC2
35  * @param bcc2 bcc2 char to be stuffed
36  * @param size size of BCC2 after stuffing
37  * @return returns the stuffed BCC2
38  */
39 unsigned char* stuffBCC2(unsigned char bcc2,unsigned int *size);
40
41 /**
42  * \brief Sends an I packet from a message from buffer to the
      serial port
43  * @param fd fiel desriptor of the serial port
44  * @param buffer containing the messsage to be sent
```

```
 * @param length length of the message to be sent
 * @return TRUE(1) upon sucess, FALSE(0) upon failure
 */
int llwrite(int fd, unsigned char *buffer, int length);

/**
 * \brief Reads an I packets sent trough the serial port
 * @param fd file descriptor for the serial port
 * @param buffer buffer read from the serial port
 * @return size of the read buffer
 */
unsigned int llread(int fd, unsigned char *buffer);

/**
 * \brief Termination of the protocol by serial port
 * @param fd file descriptor of the serial port
 * @param status if 0, acts as sender. if 1, acts as receiver for
    the termination protocl
 * @return returns 0 upon sucess, -1 otherwise
 */
int llclose(int fd, int status);

/**
 * \brief handles the alarm
 * @param signo signal number to be handled
 */
void alarmHandler(int signo);


unsigned char* generateRandomBCC(unsigned char* packet, int
    packetSize);

unsigned char* generateRandomBCC2(unsigned char* packet, int
    packetSize);
```

11.7 recetor.c

11.8 recetor.c

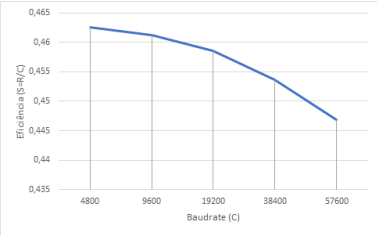
11.9 recetor.c

11.10 Variação do Baudrate

11.11 Variação do tamanho das tramas

11.12 Introdução de atraso de propragação

11.13 Introdução de erros nas tramas



(a) label 1

C	Time	R	S
4800	130,604498	2220,419698	0,462587437
9600	65,486258	4428,361138	0,461287619
19200	32,9410888	8803,497716	0,458515506
38400	16,646928	17420,43937	0,453657275
57600	11,264985	25743,20339	0,446930614

(b) label 2

Figure 1: 2 Figures side by side

C	Time	R	S
4800	130,604498	2220,419698	0,462587437
9600	65,486258	4428,361138	0,461287619
19200	32,9410888	8803,497716	0,458515506
38400	16,646928	17420,43937	0,453657275
57600	11,264985	25743,20339	0,446930614

Figure 2: Caption

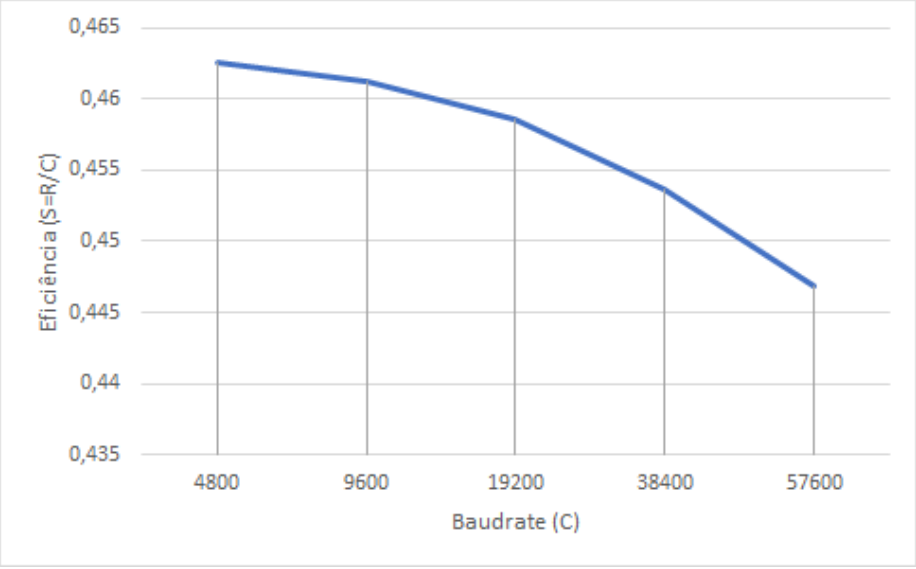


Figure 3: Caption

Tamanho	Time	R	S
32	16,646928	17420,43937	0,453657275
64	13,153735	22046,72665	0,574133507
96	12,014862	24136,50694	0,628554868
128	11,401286	25435,44649	0,662381419
160	11,082233	26167,7227	0,681451112
192	10,823469	26793,33216	0,697743025
224	10,660243	27203,58251	0,708426628

Figure 4: Caption

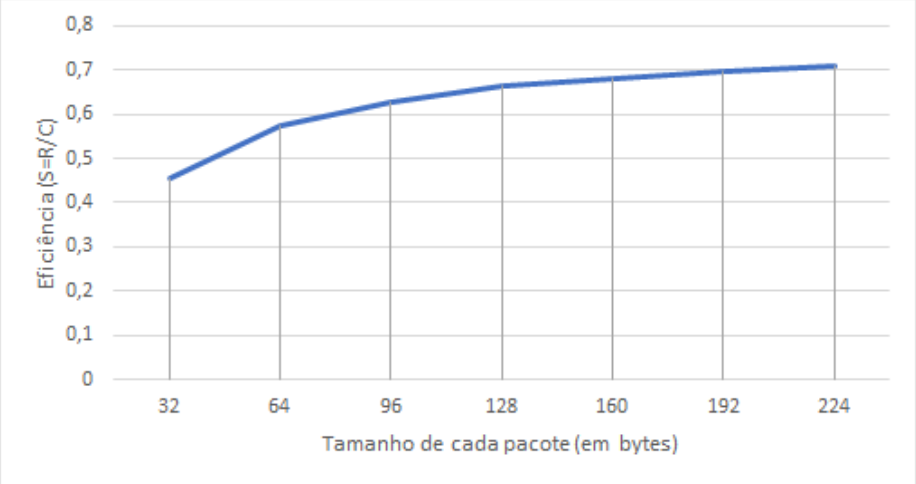


Figure 5: Caption

Atraso	Time	R	S
0,001	10,799957	26851,66246	0,699262043
0,05	14,439983	20082,90453	0,522992305
0,1	28,690873	10107,63249	0,263219596
0,15	42,946874	6752,454207	0,175845162

Figure 6: Caption

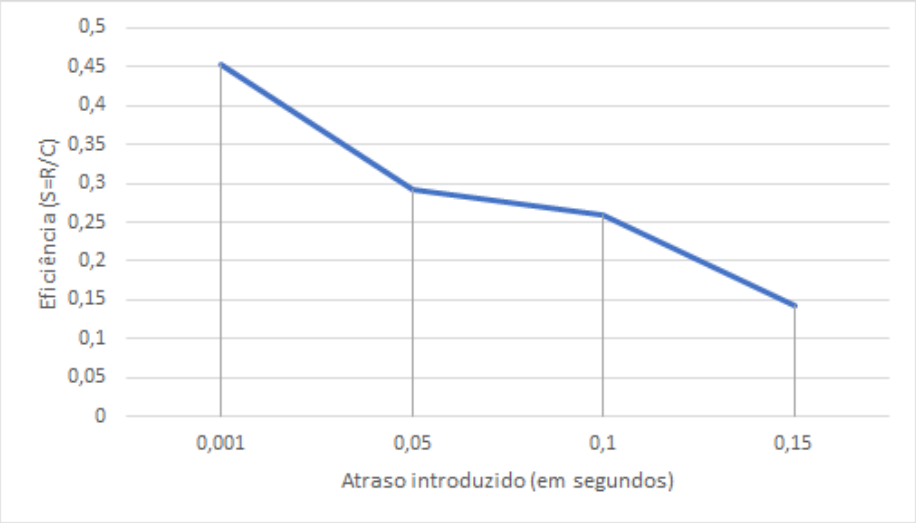


Figure 7: Caption

Erro(%)	Time	R	S
0+0	16,646928	17420,43937	0,453657275
1+1	25,969783	11166,70093	0,290799503
3+3	29,259915	9911,060917	0,258100545
5+5	52,640816	5508,972353	0,143462822
7+7	87,051704	3331,316754	0,08675304

Figure 8: Caption

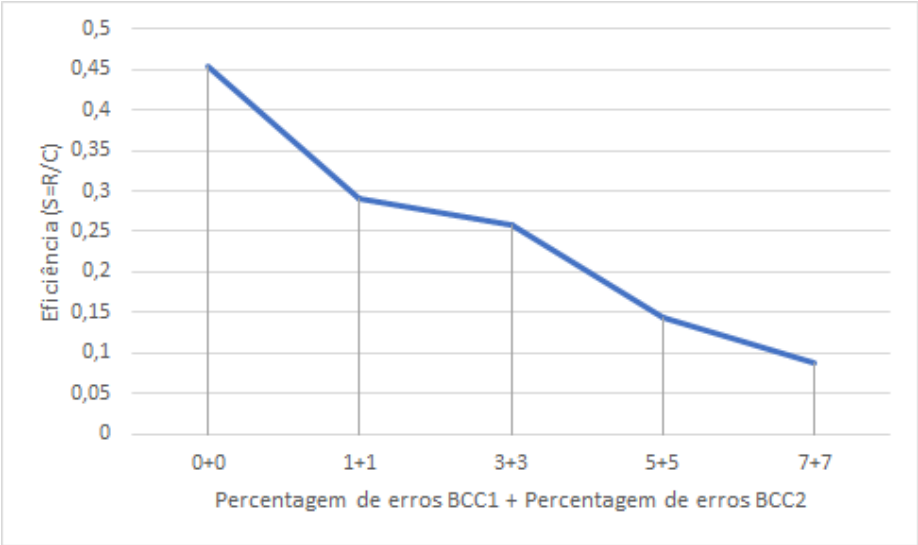


Figure 9: Caption