Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Software Básico para Acesso a um Dispositivo PCI em Sistema Operacional de Tempo Real e Sistema Mostrador de Tempo na Rede

Relatório de Projeto Orientado em Computação II

Autor: André Goddard Rosa <goddard@dcc.ufmg.br>
Orientadora: Mariza Andrade da Silva Bigonha <mariza@dcc.ufmg.br>

Belo Horizonte – MG

13 de Julho de 2004

Resumo. *Este trabalho tem como objetivo extrair informações de tempo de um dispositivo ligado ao barramento PCI de um computador, permitindo disponibilizar essas informações em uma rede de computadores. As informações obtidas através deste dispositivo (hora universal, contagem regressiva, hora de lançamento, estado da contagem e presença de sinal) são utilizadas para auxílio e monitoração de lançamentos de foguetes realizados no Centro de Lançamento da Barreira do Inferno (CLBI). Para isso foi desenvolvido um software básico capaz de extrair dados do dispositivo, um sistema capaz de propagar os dados obtidos na rede, e um aplicativo que permite a visualização dos dados que chegam pela rede em estações que não têm a placa PCI instalada.*

1 Introdução

A hora universal, obtida a partir de um dispositivo GPS com grande precisão, é um requisito importante em diversas aplicações de tempo real, incluindo o lançamento de foguetes. Uma codificação comum para a informação de tempo obtida a partir de um dispositivo GPS é o código IRIG-B [1].

O IRIG-B é um código de tempo capaz de transportar, sob uma interface analógica, sinais codificados de data e hora com resolução de 1 milésimo de segundo. O padrão IRIG é universalmente aceito como meio de sincronização de equipamentos.

De maneira particular, o Centro de Lançamento da Barreira do Inferno (CLBI) [2] usa uma variação de codificação do código IRIG-B denominado IRIG-B Composto, o qual se diferencia do original pelo transporte simultâneo das informações de Hora Corrente (TU – *Temps Universel*) e Contagem Regressiva (TD – *Temps Deroulant*), além da informação de parada de cronologia de lançamento.

2 Descrição do Problema

Os equipamentos atualmente empregados no Centro de Lançamento da Barreira do Inferno foram desenvolvidos no início dos anos 90 e tratam-se de equipamentos micro-processados, denominados MDT-100, os quais recebem o sinal IRIG-B Composto e exibem simultaneamente, em telas distintas, a Hora Corrente (TU), a Contagem Regressiva (TD) e o Estado da Contagem Regressiva. Todo o sincronismo de eventos durante uma campanha de lançamento de foguetes é garantido por estes dispositivos.

Nestes equipamentos existem saídas contendo a informação de tempo codificada em um sinal analógico que pode ser conectada a cartões de entrada ligados ao barramento ISA [3] dos computadores para uso posterior desta informação.

Com a modernização dos microcomputadores que compõe as posições operacionais do CLBI, não é mais possível utilizar a placa de barramento ISA, pois os computadores mais modernos não têm mais este tipo de barramento.

O padrão ISA está obsoleto, sendo difícil encontrar computadores modernos com suporte a este tipo de barramento. O padrão vigente no mercado é o PCI [4], que é mais largo (tem mais bits nas linhas de endereço e de dados) e funciona com uma freqüência mais alta. Este é um padrão universal de interface de *hardware* tanto em plataforma CISC quanto em plataformas RISC, dispondo de uma especificação aberta, publicada e mantida pela organização PCI Special Interest Group [5].

A empresa Audiolab/Concert [6] foi responsável pelo desenvolvimento do cartão padrão PCI que virá a substituir o antigo cartão padrão ISA no CLBI, sendo o escopo deste trabalho limitado à especificação e implementação de todo o conjunto de *software* necessário para garantir a funcionalidade do Sistema de Sincronização atualmente implantado no Centro de Lançamento, sem que a modernização venha a acarretar perda de desempenho em alguma função já existente. Para isso tem-se como requisito uma variação de 1 (um) segundo no pior caso entre o horário dos equipamentos de *hardware* e os horários exibidos nas estações de trabalho dos operadores. Além disso, objetiva-se com a transmissão das informações de tempo na rede, evitar um gasto maior com outros equipamentos de *hardware* que seriam necessários para a exibição dessas informações para um acompanhamento remoto.

Dessa forma, são assegurados que todos os componentes utilizados na modernização do Sistema de Sincronização sejam componentes atuais, existentes em suas respectivas linhas de produção, evitando que o projeto venha a utilizar partes ou peças obsoletas, comprometendo assim as futuras atividades de manutenção.

3 Análise do Problema

Para garantir uma precisão de não mais que um segundo de atraso, foi escolhida como plataforma para o desenvolvimento do *software* básico para acesso ao dispositivo PCI, um sistema operacional de tempo real, chamado QNX [7], que garante uma latência mínima e uma grande predictabilidade no tempo de resposta.

Para garantir o transporte de informações de tempo através do meio físico da rede, foram considerados diversos fatores que contribuem para uma imprecisão no transporte das informações, sendo um deles o próprio atraso do meio e a latência envolvida no envio e recepção de mensagens [8]. Para isso foi utilizado um protocolo padrão, chamado SNTP [9], utilizado em todos os sistemas operacionais modernos para sincronização de relógios entre computadores, que cumpre esses objetivos considerando os problemas de imprecisão que ocorreriam normalmente pela utilização da rede.

As outras informações, como o estado da contagem e a hora de lançamento, são transmitidas utilizando o protocolo UDP em uma mensagem de broadcast. Isso evita a geração de uma grande latência ao enviar essa mensagem seqüencialmente para cada um dos clientes interessados em exibir estas informações, com o efeito colateral de aumentar a demanda de largura de banda do meio. Isso não apresenta um grande problema, pois a maior mensagem trafegada carrega somente 11 bytes nesta implementação.

4 Aspectos de Implementação

Nesta seção será apresentada uma abordagem abstrata de alto nível e um detalhamento gradual dos aspectos relevantes para a implementação do sistema.

4.1 Esquemáticos da Arquitetura

Figura 1: Cartão PCI demodulador de código IRIG-B Composto

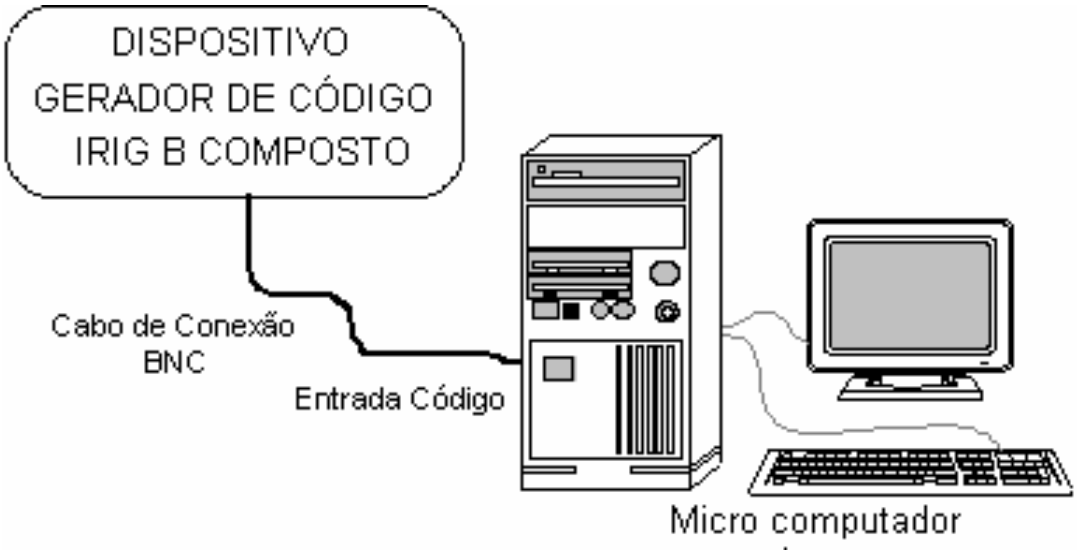

Figura 2: Diagrama de obtenção dos dados do cartão PCI

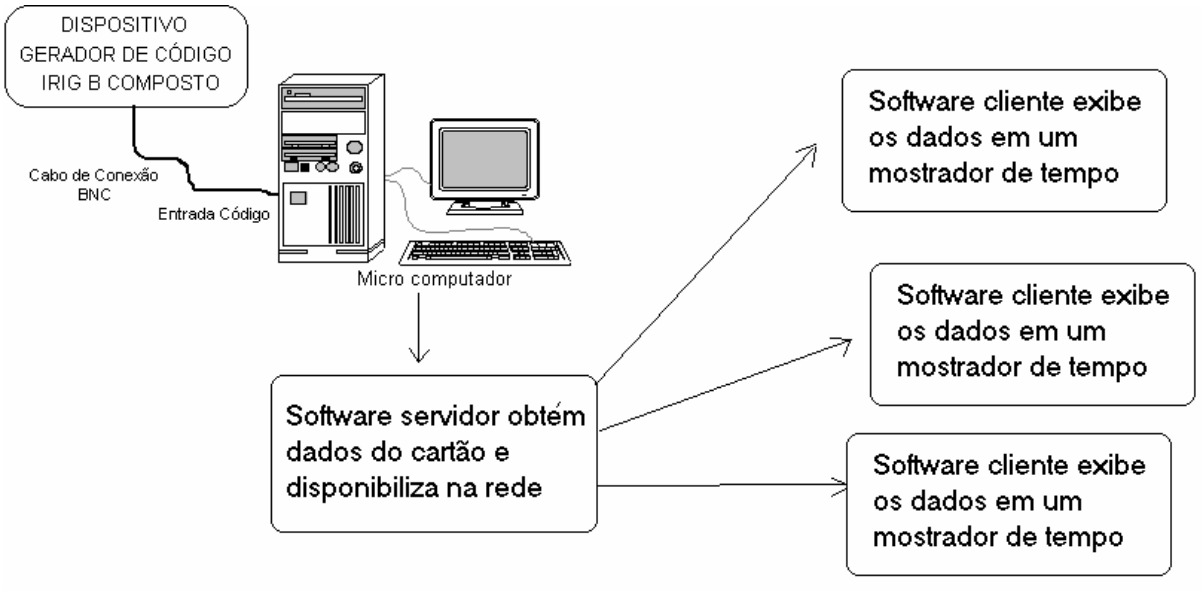

Figura 3: Diagrama geral de funcionamento

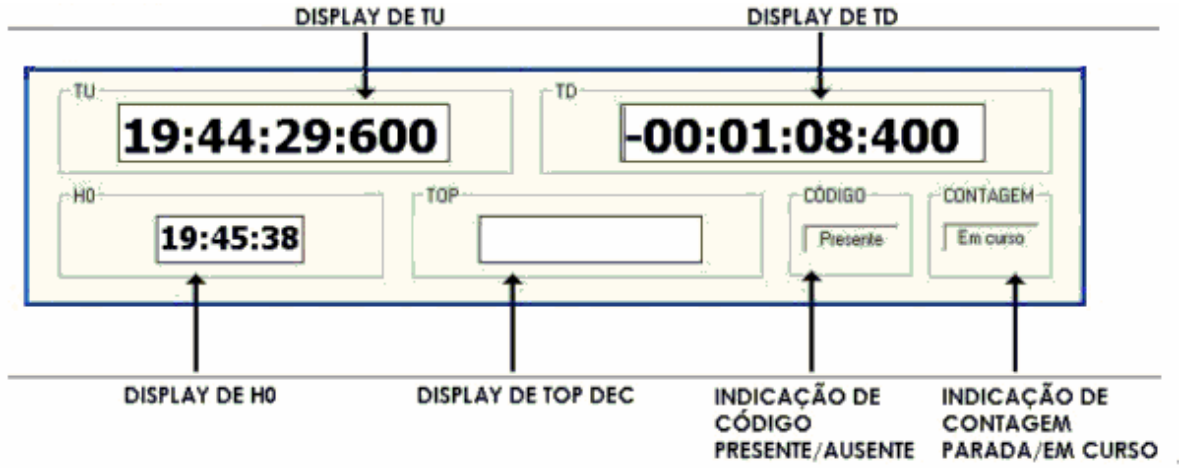

Figura 4: Sugestão para Interface Cliente do Mostrador de Tempo

4.2 Aspectos de Alto Nível

Um *software* básico, executado no sistema operacional de tempo real QNX, extrai as informações do cartão disponibilizando-as na rede para *software* clientes.

Os diversos *software* clientes, em sistemas operacionais Windows [10], exibem as seguintes informações:

- Display de TU: apresenta o horário atual (TU – Temps Universel), contido no código IRIG-B Composto e transmitido na rede através de SNTP para completa precisão.
- Display de TD: apresenta a contagem regressiva (TD – Temps Deroulant) para disparo, contida no código IRIG-B Composto e calculada em cada cliente.
- Display de H0: apresenta a hora de disparo (H0 – Hora Zero), calculada como a diferença entre o horário atual e a contagem regressiva para disparo. Isto é:

  TU – TD = H0

  A informação de H0 é transmitida em broadscast por UDP e é utilizada pelos clientes para obter os dados da contagem regressiva, fazendo-se a conta inversa:

  TU – H0 = TD

O tempo para disparo sempre é programado durante a transição de um segundo para outro de TU, de forma que H0 sempre seja um horário exato, com zero milissegundos. Assim, é desnecessária a exibição dos milissegundos de H0, uma vez que estes são sempre iguais a zero.

- Display de TOP DEC: apresenta o horário exato em que a interrupção de TOP de
  Decolagem é recebida. Este horário representa o momento exato do lançamento do foguete e é exibido com precisão de milissegundos.
- Indicação de Código Presente/Ausente: este campo apresenta o atual estado da presença do código IRIG-B Composto. Se há existência do código, este campo apresenta a indicação "Presente". Senão, apresenta a indicação "Ausente".
- Indicação de Contagem Parada/Em curso: este campo apresenta o estado atual de
  um indicador contido no código IRIG-B Composto que informa se a contagem regressiva está parada ou não. Se o indicador estiver ativado, este campo apresenta a informação "Parada", e o campo TD exibe um tempo parado. Neste caso, o campo H0 passa a ser incrementado de maneira síncrona com TU. Quando o indicador não está ativado, TD está em andamento, e o campo apresenta a indicação "Em curso". É importante notar que a paralização de TD é programada para acontecer sempre na transição de um segundo para outro de TU, fazendo com que o TD parado sempre seja um tempo exato, com zero milissegundos. A Figura 4 apresenta uma sugestão para a interface do mostrador de tempo de acordo com as informações relatadas nesta seção.

4.3 Detalhes da Implementação

O cartão PCI utilizado, ilustrado na Figura 1, permite a geração de dois tipos de interrupção: uma interrupção periódica programável e uma interrupção assíncrona ativada quando o botão de lançamento do foguete for acionado (Interrupção de TOP DEC).

Uma peça chave desse trabalho foi implementar o *software* básico capaz de obter os dados do cartão PCI periodicamente. Esse *software* básico é responsável por:

a) configurar o intervalo de interrupção periódica programável do cartão;

b) extrair as informações do cartão a cada vez que ocorre uma interrupção periódica;

c) identificar a causa da interrupção sempre que houver uma interrupção.

Para isso, foi especificada uma API de acesso, programada em linguagem C, que abstrai toda a complexidade de lidar com o baixo nível do *hardware*: endereços de registradores, tratamento de interrupções, etc. Essa API é utilizada pelo sistema que propaga as informações de tempo na rede. As Figuras 2 e 3 exibem, respectivamente, o diagrama para a obtenção dos dados do cartão PCI e o diagrama geral de funcionamento do sistema desenvolvido.

O sistema cliente, denominado mostrador de tempo, foi programado em linguagem C++ e obtém os dados de tempo da rede e realiza a exibição para o usuário, cuidando de atualizar a interface a cada segundo.

5 Metodologia de Desenvolvimento

Esta seção descreve as principais etapas e os principais obstáculos confrontados para a implementação desse sistema de *software*.

5.1 Ambientação com o sistema operacional QNX

A primeira parte deste trabalho consistiu na ambientação com o sistema operacional QNX, uma plataforma não convencional para desenvolvimento de *software*. Nesta fase foi preciso aprender muito para operar e configurar o sistema operacional corretamente. Sutilezas nos detalhes que a diferenciavam de outras plataformas existentes foram identificadas e entendidas.

5.2 Extração de informações da placa PCI

Para capturar as informações dos registradores da placa PCI, foi necessário estudar o modelo de desenvolvimento de *device drivers* para o sistema operacional de tempo real QNX. A seguir, foi preciso aprender a utilizar algumas chamadas de sistema necessárias para obtenção de informações dos dispositivos anexados ao barramento PCI. Obtidas essas informações, foi possível endereçar os registradores específicos que contém a informação de tempo e fazer o tratamento específico, como conversão de BCD (*Binary Coded Decimal*) [3] para decimal.

5.3 Captura de interrupção (manipulador de interrupção)

Após a extração da informação de qual vetor de interrupção [3] era atribuído à placa pelo BIOS (*Basic Input Output System*) [3], efetuou-se a captura e tratamento da mesma. Nesta fase do trabalho, identificou-se um problema no projeto do *hardware*, que tinha como efeito a habilitação de interrupções espúrias aleatoriamente. Com o auxílio do *software*, que permitiu atentar para o problema, e de um osciloscópio, que identificou a causa do problema, o mesmo pôde ser corrigido.

5.4 Projeto da API básica de acesso simplificado às informações

Como as informações obtidas do cartão não estão em um formato inteligível para o usuário, pois estão codificadas em BCD e espalhadas em diversos registradores, foi necessário implementar funções de alto nível que obtivessem os dados tais como: TU, TD, etc. Esta API mostrou-se bastante versátil, pois simplifica a tarefa de qualquer aplicativo que queira fazer uso das informações providas pelo cartão PCI.

5.5 Configuração do SNTP

Para permitir a transmissão precisa de tempo na rede, estudou-se como configurar um servidor de tempo SNTP e também os clientes de tempo SNTP.

5.6 Tráfego de informações na rede

Foi necessário especificar um mini-protocolo, que considerasse quaisquer diferenças entre os tipos de dados trafegados, por exemplo, indianismo e tamanho dos tipos, e que permitisse atribuir confiabilidade na atualização das informações no cliente. Para isso desenvolveu-se uma pequena máquina de estados cuja função é verificar o correto recebimento dos pacotes a cada segundo, avisando o usuário caso ocorra algum problema no recebimento desses pacotes.

6 Resultados

Foi possível avaliar a solução implementada, mostrando que ela atende às necessidades levantadas nos requisitos expostos na descrição do problema.

Com o desenvolvimento do *software* foi possível encontrar um erro no projeto do *hardware* envolvido, permitindo uma interação construtiva com os projetistas do cartão PCI.

7 Conclusão

Neste trabalho, uma das maiores preocupações está relacionada com a extensibilidade. Procurou-se implementar uma API de fácil uso, para que futuros trabalhos possam ser desenvolvidos utilizando este mesmo arcabouço.

Este trabalho tem importante apelo prático, porque será utilizado futuramente, após toda a validação necessária, no auxílio e acompanhamento do lançamento de foguetes efetuado pelo Governo Brasileiro através do Ministério da Aeronáutica, DEPED [11] [12] [13] [14] [15] [16] [17].

Também foi possível obter, durante o desenvolvimento deste *software*, um sólido conhecimento relacionado ao barramento PCI, tratamento de interrupções de *hardware*, acesso aos registradores de um cartão interligado ao computador e confecção de um *software* básico para acesso de baixo nível em um sistema operacional.

É interessante notar que o conhecimento empregado em sua confecção é nitidamente interdisciplinar, destacando-se as partes de *Software* Básico, Redes de Computadores e Sistemas Operacionais.

Finalmente, uma das maiores contribuições deste desenvolvimento é a habilidade adquirida para desenvolver qualquer *software* básico para dispositivos, pois com esta abordagem prática foi possível desmistificar toda a complexidade envolvida neste tipo de desenvolvimento de baixo nível especializado.

8 Trabalhos Futuros

Como trabalho futuro, seria interessante avaliar a possibilidade de implementar uma variação do protocolo SNTP que permitisse o tráfego de informações de contagem regressiva no pacote já utilizado pelo mesmo para transmissão das informações de tempo, excluindo todo o mecanismo usado para o transporte paralelo destas informações no trabalho descrito neste texto.

9 Referências Bibliográficas

[1] IEEE Std 1344-1995 IEEE Standard for Synchrophasors for Power Systems.
http://standards.ieee.org/reading/ieee/std_public/description/relaying/1344-1995_desc.html

[2] Centro de Lançamento da Barreira do Inferno (CLBI).
http://www.aeb.gov.br/centrolanc.htm
http://pessoais.digi.com.br/~clbi/

[3] Tanenbaum, Andrew S. *Organização Estruturada de Computadores*. Editora LTC. Quarta Edição, 2001.

[4] Shanley e Andreson, 1995b; Solari e Willse, 1998.

[5] PCI Special Interest Group.
http://www.pcisig.com

[6] Audiolab Sistemas de Automação e *Software*.
http://www.audiolab.com.br

[7] QNX *Real Time Operating System*.
http://www.qnx.com
[8] Peterson, Larry L. and Davie, Bruce S. (2000) Computer Networks: A Systems Approach, Second Edition
[9] SNTP Simple NTP (SNTP) version 4, RFC 2030.
http://www.eecis.udel.edu/~mills/database/rfc/rfc2030.txt
[10] Windows, *Microsoft*.
http://www.microsoft.com
[11] Ministério da Aeronáutica. Ciclo de Vida de Sistemas e Materiais da Aeronáutica, 1992 (DMA 400-6).
[12] Ministério da Aeronáutica. Implantação e Gerenciamento de Sistemas no Ministério da Aeronáutica, 1998 (IMA 700-1).

[13] Ministério da Aeronáutica. Segurança de Dados do MAER, 1999 (NSMA 7-13).

[14] Centro Técnico Aeroespacial. Plano de Qualidade de Fornecedor, 1998 (DHM-006-R02).

[15] Centro Técnico Aeroespacial. Procedimentos de Homologação, de Convalidação e de Qualificação de Material Aeroespacial de Emprego Militar, 1998 (DHM-012).

[16] Centro Técnico Aeroespacial. Requisitos e Procedimentos Contratuais de Garantia da Qualidade do MAER, 1998 (DHM-014).

[17] DEPED, Departamento de Pesquisas e Desenvolvimento. Sistema de Integrado Tratamento de Dados de Localização (SITDL), 2002 (ET-DRE-01/00 – Rev. 31 OUT 2002).

10 Anexo – Código Fonte

10.1 Código do *Software* Básico de Acesso ao Dispositivo PCI

10.1.1 Arquivo de Cabeçalho MDT200Api.h

```
// MDT200API.h :
//

#if !defined(MDT200API_H)
#define MDT200API_H

/* Mascara de bits que representa possiveis causas de interrupcao */
#define IDENT_TEMPO 0x1
#define IDENT_TOP   0x2
#define IDENT_AMBOS 0x3

/* Constantes usadas para representar retornos de funcoes */
#define SUCESSO   1
#define FALHA     0
#define TIMEOUT  -1

////////////////////////////////////////////////////////////////////////////////////////////////////

/* Armazena dados relativos ao Tempo Universal. */

typedef struct _TU_TYPE {
      unsigned char Hour;
      unsigned char Min;
      unsigned char Sec;
      unsigned short ms;
} TU_TYPE;

////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
/* Armazena dados relativos aa Contagem Regressiva. */

typedef struct _TD_TYPE {
      unsigned char Hour;
      unsigned char Min;
      unsigned char Sec;
      unsigned char signal;
} TD_TYPE;

////////////////////////////////////////////////////////////////////////////////////////////////////////////

/* Armazena dados relativos aa Hora de Lancamento. */

typedef struct _H0_TYPE {
      unsigned char Hour;
      unsigned char Min;
      unsigned char Sec;
} H0_TYPE;

////////////////////////////////////////////////////////////////////////////////////////////////////////////

/* Armazena flags de status do sinal IRIG-B recebidos. */

typedef struct _STATUS_TYPE {
      unsigned char top_dec;
      unsigned char cont_par;
      unsigned char clock;
      unsigned char sinal;
      unsigned char reserved;
      unsigned char cod_pres;
      unsigned char hab_dados;
      unsigned char env_irig;
} STATUS_TYPE;

////////////////////////////////////////////////////////////////////////////////////////////////////////////

/* Armazena os parametros do driver do cartao MDT 200. */

typedef struct _CARD_PARAM {
      char DriverName[20];
      char DriverVersion[10];
      unsigned char IRQ;
      unsigned long BaseAddress;
} CARD_PARAM;

/* Prototipo das funcoes da biblioteca */
int OpenDevice();
void CloseDevice();
int EnableProgIntr();
int DisableProgIntr();
int EnableTOPIntr();
int DisableTOPIntr();
int EnableH0Intr();
int DisableH0Intr();
int SetProgIntrPeriod(unsigned short period);
int InstallHandler();
int WaitInterrupt(long segundos_timeout, long milissegundos_timeout);
int TU_Read(TU_TYPE * tuTime);
int TD_Read(TD_TYPE * tdTime);
```

```
int H0_Calc(H0_TYPE * h0Time, TU_TYPE * tuTime, TD_TYPE * tdTime);
int Status_Read(STATUS_TYPE * stReg);
int GetCardParams(CARD_PARAM * CardParameters);

#endif
```

10.1.2 Arquivo Fonte – MDT200Api.c

```
/*************************************************************************
*                                                                        *
*   AUDIOLAB Sistemas Eletronicos Ltda  - Sistema MDT200                 *
*                                                                        *
*   MODULO: MDTAPI200    TIPO: Implementacao      ARQUIVO: MDTAPI200.c   *
*                                                                        *
*   Versao: 1.0              15/02/2004        Atualizacao:              *
*                                                                        *
*   Descricao: Implementacao de uma biblioteca compativel para acesso PCI *
*                                                                        *
*   Atualizacoes:                                    *
*                                                                        *
*   Sistema Operacional: QNX 6.2.1          Compilador: Watcom C         *
*                                                                        *
*************************************************************************/

#include <stdio.h>
#include <hw/pci.h>
#include <sys/neutrino.h>
#include <x86/inout.h>

#include "MDT200Api.h"

/* Informacoes sobre o driver */
#define DRIVER_NAME      "MDT200Api"
#define DRIVER_VERSION   "1.0"

/* Identificador possivel para os dispositivos PCI */
#define PCI_VENDOR_ID    0x10B5
#define PCI_DEVICE_ID_1  0x3001
#define PCI_DEVICE_ID_2  0x9030

/* Armazena descritor do dispositivo PCI */
static void *hdl = NULL;

/* Estrutura de notificacao usada para acordar InterruptWait() */
static volatile struct sigevent event;

/* Armazena a causa de uma interrupcao ocorrida */
static volatile int causa = 0x0;

/* Indica se o dispositivo foi obtido pela Api */
static unsigned int openedDevice = 0;

/* Armazena retorno da operacao de associar handler com interrupcao */
static int id_attach = 0;

/* Armazena o identificador da IRQ utilizada pela placa */
static unsigned int irq = 0;

/*
 * Armazena, respectivamente:
 *     o endereco dos registradores de controle - MEM
```

```
 *     o endereco dos registradores de controle - IO
 *     o endereco dos registradores que armazenam o tempo - IO
*/
static volatile unsigned long pci_bar_0, pci_bar_1, pci_bar_3;

/*-------------------------------------------------------------------------+
| isr_handler()                                                            |
|                                                                          |
| Responsavel por tratar as interrupcoes geradas no barramento PCI.        |
|                                                                          |
| Retorna um evento para acordar um InterruptWait() ou NULL (sem efeito).  |
+-------------------------------------------------------------------------*/
const struct sigevent *isr_handler(void *arg, int id)
{
      volatile unsigned char conteudo4C;
      volatile unsigned char conteudo4D;

      /* Disable PCI Interrupt */
      out8(pci_bar_1 + 0x4C, in8(pci_bar_1 + 0x4C) & ~(1 << 6));

      causa = 0;
      conteudo4C = in8(pci_bar_1 + 0x4C);
      conteudo4D = in8(pci_bar_1 + 0x4D);

      if (conteudo4C & (1 << 2)) {
            out8(pci_bar_1 + 0x4D, conteudo4D | (1 << 2));
            causa |= IDENT_TEMPO;
      }

      if (conteudo4C & (1 << 5)) {
            out8(pci_bar_1 + 0x4D, conteudo4D | (1 << 3));
            causa |= IDENT_TOP;
      }

      /* Realiza uma leitura espuria */
      in8(pci_bar_3);

      /* Enable PCI Interrupt */
      out8(pci_bar_1 + 0x4C, conteudo4C | (1 << 6));

      if (causa > 0) {
            /* Dispositivo PCI gerou interrupcao; acorda InterruptWait() */
            return (const struct sigevent *)&event;
      }

      return NULL;
}

/*-------------------------------------------------------------------------+
| OpenDevice()                                                             |
|    Inicializa a biblioteca e permite sua utilizacao subsequente; deve ser |
|    a primeira funcao do modulo a ser chamada.                            |
|    Ela eh responsavel por localizar o dispositivo PCI e obter informacoes |
|    necessarias para o correto funcionamento da Api, como a IRQ utilizada  |
|    pelo dispositivo para comunicar com o barramento e tambem o endereco   |
|    base dos registradores.                                               |
|                                                                          |
| Retorno:                                                                 |
|    SUCESSO: sucesso na execucao                                          |
|    FALHA  : falha na execucao                                            |
```

```
+------------------------------------------------------------------------*/
int OpenDevice()
{
	struct pci_dev_info info;
	int pidx;

	if (openedDevice) {
		return SUCESSO;
	}

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

	if (pci_attach(0) < 0) {
		fprintf(stderr, "pci_attach FAILURE!\n");
		return FALHA;
	}

	/* Permite privilegios de I/O */
	ThreadCtl(_NTO_TCTL_IO, 0);

	/*
	 * Fill in the Vendor and Device ID
	 */
	pidx = 0;
	info.VendorId = PCI_VENDOR_ID;
	info.DeviceId = PCI_DEVICE_ID_1;

	if ((hdl = pci_attach_device(NULL, PCI_INIT_ALL, pidx, &info)) == NULL) {
		/* Tenta encontrar com outro device id */
		info.DeviceId = PCI_DEVICE_ID_2;
		if ((hdl = pci_attach_device(0, PCI_INIT_ALL, 0, &info)) == NULL) {
			fprintf(stderr, "pci_attach_device FAILURE!\n");
			return FALHA;
		}
	}

	/* Encontrou dispositivo */

	/* Obtem endereco base MEM dos registradores de controle */
	pci_bar_0 = PCI_MEM_ADDR(info.CpuBaseAddress[0]);

	/* Obtem endereco base IO dos registradores de controle */
	pci_bar_1 = PCI_IO_ADDR(info.CpuBaseAddress[1]);

	/* Obtem endereco base IO dos registradores que contem o tempo */
	pci_bar_3 = PCI_IO_ADDR(info.CpuBaseAddress[3]);

	/* Obtem IRQ da placa */
	irq = info.Irq;

	/* Executa leitura inicial arbitraria da placa */
	in8(pci_bar_3);

	out8(pci_bar_1 + 0x4C, 0x0);
	out8(pci_bar_1 + 0x4D, 0x0);

	fprintf(stderr, "\nVendor %x, DeviceId %x, Base Addr %x, Int %d\n",
			info.VendorId, info.DeviceId, pci_bar_3, irq);

	openedDevice = 1;
```

```
	SIGEV_INTR_INIT(&event);

	return SUCESSO;
}

/*-------------------------------------------------------------------------+
| CloseDevice()                                                            |
|    Finaliza a biblioteca liberando todos os recursos por ela alocados.   |
|    Para voltar a utilizar qualquer funcao novamente, apos finalizada, ela |
|    deve ser inicializada novamente com a funcao OpenDevice().            |
|                                                                          |
| Retorno:                                                                 |
|    void                                                                  |
+-------------------------------------------------------------------------*/
void CloseDevice()
{
	if (openedDevice) {
		openedDevice = 0;

		/* Disable PCI Interrupt */
		out8(pci_bar_1 + 0x4C, 0x0);
		out8(pci_bar_1 + 0x4D, 0x0);

		if (id_attach != 0) {
			InterruptDetach(id_attach);
		}
		pci_detach_device(hdl);
	}
}

/*-------------------------------------------------------------------------+
| EnableProgIntr()                                                         |
|    Habilita a recepcao da interrupcao programavel.                       |
|                                                                          |
| Retorno:                                                                 |
|    SUCESSO: sucesso na execucao                                          |
|    FALHA  : falha na execucao                                            |
+-------------------------------------------------------------------------*/
int EnableProgIntr()
{
	if (!openedDevice) {
		return FALHA;
	}

	out8(pci_bar_1 + 0x4C, in8(pci_bar_1 + 0x4C) | 0x43);
	out8(pci_bar_1 + 0x4D, 0x0);

	return SUCESSO;
}

/*-------------------------------------------------------------------------+
| DisableProgIntr()                                                        |
|    Desabilita a recepcao da interrupcao programavel.                     |
|                                                                          |
| Retorno:                                                                 |
|    SUCESSO: sucesso na execucao                                          |
|    FALHA  : falha na execucao                                            |
+-------------------------------------------------------------------------*/
int DisableProgIntr()
```

```
{
	if (!openedDevice) {
		return FALHA;
	}

	out8(pci_bar_1 + 0x4C, in8(pci_bar_1 + 0x4C) & ~(1 << 0));

	return SUCESSO;
}

/*-------------------------------------------------------------------------+
| EnableTOPIntr()                                                          |
|    Habilita a recepcao da interrupcao de TOP DEC.                        |
|                                                                          |
| Retorno:                                                                 |
|    SUCESSO: sucesso na execucao                                          |
|    FALHA  : falha na execucao                                            |
+-------------------------------------------------------------------------*/
EnableTOPIntr()
{
	if (!openedDevice) {
		return FALHA;
	}

	out8(pci_bar_1 + 0x4C, in8(pci_bar_1 + 0x4C) | 0x58);
	out8(pci_bar_1 + 0x4D, 0x0);

	return SUCESSO;
}

/*-------------------------------------------------------------------------+
| DisableTOPIntr()                                                         |
|    Desabilita a recepcao da interrupcao de TOP DEC.                      |
|                                                                          |
| Retorno:                                                                 |
|    SUCESSO: sucesso na execucao                                          |
|    FALHA  : falha na execucao                                            |
+-------------------------------------------------------------------------*/
int DisableTOPIntr()
{
	if (!openedDevice) {
		return FALHA;
	}

	out8(pci_bar_1 + 0x4C, in8(pci_bar_1 + 0x4C) & ~(1 << 3));

	return SUCESSO;
}

/*-------------------------------------------------------------------------+
| EnableH0Intr()                                                           |
|    Habilita a emissao do comando TOP H0.                                 |
|                                                                          |
| Retorno:                                                                 |
|    SUCESSO: sucesso na execucao                                          |
|    FALHA  : falha na execucao                                            |
+-------------------------------------------------------------------------*/
EnableH0Intr()
{
	if (!openedDevice) {
```

```
		return FALHA;
	}

	/* Habilita possibilidade de envio de sinal */
	out8(pci_bar_3 + 0xF, 0x1);

	return SUCESSO;
}

/*-------------------------------------------------------------------------+
| DisableH0Intr()                                                          |
|    Desabilita a emissao do comando TOP H0.                               |
|                                                                          |
| Retorno:                                                                 |
|    SUCESSO: sucesso na execucao                                          |
|    FALHA  : falha na execucao                                            |
+-------------------------------------------------------------------------*/
int DisableH0Intr()
{
	if (!openedDevice) {
		return FALHA;
	}

	/* Desabilita possibilidade de envio de sinal */
	out8(pci_bar_3 + 0xF, 0x0);

	return SUCESSO;
}

/*-------------------------------------------------------------------------+
| SetProgIntrPeriod()                                                      |
|    Define periodo para ocorrencia da interrupcao programavel.            |
|    Os valores permitidos com seus respectivos significados sao:          |
|        0 -   Desabilitado                                                |
|        1 -   1 ms                                                        |
|        2 -   2 ms                                                        |
|        3 -   5 ms                                                        |
|        4 -  10 ms                                                        |
|        5 -  20 ms                                                        |
|        6 -  50 ms                                                        |
|        7 - 100 ms                                                        |
|        8 - 200 ms                                                        |
|        9 - 500 ms                                                        |
|       10 -   1 s                                                         |
|                                                                          |
| Retorno:                                                                 |
|    SUCESSO: sucesso na execucao                                          |
|    FALHA  : falha na execucao                                            |
+-------------------------------------------------------------------------*/
int SetProgIntrPeriod(unsigned short periodo)
{
	if (!openedDevice) {
		return FALHA;
	}

	switch (periodo) {
	case 0x0:
	case 0x1:
	case 0x2:
	case 0x3:
```

```
	case 0x4:
	case 0x5:
	case 0x6:
	case 0x7:
	case 0x8:
	case 0x9:
	case 0xA:
		/* Configura a interrupcao por tempo */
		out8(pci_bar_3 + 0xE, periodo);
		return SUCESSO;
	default:
		return FALHA;
	}
}

/*-------------------------------------------------------------------------+
| InstallHandler()                                                         |
|    Instala um manipulador para a interrupcao, fazendo com que um         |
|    WaitInterrupt() continue a processar apos a chegada da interrupcao.   |
|                                                                          |
| Retorno:                                                                 |
|    SUCESSO: sucesso na execucao                                          |
|    FALHA  : falha na execucao                                            |
+-------------------------------------------------------------------------*/
int InstallHandler()
{
	if (!openedDevice) {
		return FALHA;
	}

	if (id_attach == 0) {
		if ((id_attach = InterruptAttach(irq, isr_handler, NULL, 0, 0)) == -1) {
			fprintf(stderr, "Unable to attach interrupt.\n");
			return FALHA;
		}
	}

	return SUCESSO;
}

/*-------------------------------------------------------------------------+
| WaitInterrupt()                                                          |
|    Funcao blocante que fica esperando que uma interrupcao ocorra ou um   |
|    tempo de timeout venca (o que ocorrer primeiro). O retorno desta      |
|    funcao indica a causa da interrupcao ou um flag indicador de timeout, |
|    caso o tempo especificado para timeout venca antes da chegada de uma  |
|    interrupcao.                                                          |
|                                                                          |
| Retorno:                                                                 |
|    IDENT_TEMPO - recebeu uma interrupcao de tempo antes do timeout vencer|
|    IDENT_TOP   - recebeu uma interrupcao de TOP antes do timeout vencer  |
|    IDENT_AMBOS - recebeu ambas interrupcoes ao mesmo tempo (tempo e TOP),|
|                  antes do timeout vencer                                 |
|    FALHA       - ocorreu uma falha, retorna flag indicativo 0            |
|    TIMEOUT     - timeout ocorreu antes da chegada de qualquer interrupcao,|
|                  retorna flag indicativo -1                              |
+-------------------------------------------------------------------------*/
int WaitInterrupt(long segundos_timeout, long milissegundos_timeout)
{
	uint64_t timeout;
```

```
	if ((!openedDevice) || (id_attach == 0)) {
		return FALHA;
	}

	/* timeout representa o tempo em nanossegundos */
	timeout = segundos_timeout * 1000000000 + milissegundos_timeout * 1000000;

	TimerTimeout(CLOCK_REALTIME, _NTO_TIMEOUT_INTR, NULL, &timeout, NULL);

	if (InterruptWait(NULL, NULL) == -1) {
		return TIMEOUT;
	}

	/* causa pode ser IDENT_TEMPO, IDENT_TOP ou IDENT_AMBOS */
	return causa;
}

/*---------------------------------------------------------------------------+
| TU_Read()                                                                  |
|    Realiza a leitura dos registros de hora, minutos, segundos e           |
|    milissegundos de TU.                                                    |
|                                                                            |
| Retorno:                                                                   |
|    SUCESSO: sucesso na execucao                                            |
|    FALHA  : falha na execucao                                              |
+---------------------------------------------------------------------------*/
int TU_Read(TU_TYPE * tuTime)
{
	unsigned char tu_hour_ms;
	unsigned char tu_hour_td_tu;
	unsigned char tu_min;
	unsigned char tu_sec;
	unsigned char tu_ms;

	if (!openedDevice) {
		return FALHA;
	}

	tu_ms = in8(pci_bar_3);
	tu_hour_ms = in8(pci_bar_3 + 1);
	tu_sec = in8(pci_bar_3 + 2);
	tu_min = in8(pci_bar_3 + 3);
	tu_hour_td_tu = in8(pci_bar_3 + 9);

	tuTime->ms = ((tu_hour_ms & 0x0f) << 8) + tu_ms;
	tuTime->Sec = tu_sec;
	tuTime->Min = tu_min;
	tuTime->Hour = (tu_hour_td_tu << 4) + (tu_hour_ms >> 4);

	return SUCESSO;
}

/*---------------------------------------------------------------------------+
| TD_Read()                                                                  |
|    Realiza a leitura dos registros de hora, minutos, segundos e           |
|    sinal de TU.                                                            |
|                                                                            |
| Retorno:                                                                   |
|    SUCESSO: sucesso na execucao                                            |
```

```
|    FALHA  : falha na execucao                                          |
+-----------------------------------------------------------------------*/
int TD_Read(TD_TYPE * tdTime)
{
      unsigned char flags;
      unsigned char td_hour_ms;
      unsigned char td_hour_td_tu;
      unsigned char td_min;
      unsigned char td_sec;

      if (!openedDevice) {
                return FALHA;
      }

      td_hour_ms = in8(pci_bar_3 + 5);
      td_sec = in8(pci_bar_3 + 6);
      td_min = in8(pci_bar_3 + 7);
      flags = in8(pci_bar_3 + 8);
      td_hour_td_tu = in8(pci_bar_3 + 9);

      tdTime->Sec = td_sec;
      tdTime->Min = td_min;
      tdTime->Hour = (td_hour_td_tu & 0xf0) + (td_hour_ms >> 4);

      if (flags & (1 << 4)) {
                tdTime->signal = 1;
      } else {
                tdTime->signal = 0;
      }

      return SUCESSO;
}

/*-----------------------------------------------------------------------+
| H0_Calc()                                                              |
|    Calcula a hora de lancamento a partir de TU e TD.                   |
|                                                                        |
| Retorno:                                                               |
|    SUCESSO: sucesso na execucao                                        |
|    FALHA  : falha na execucao                                          |
+-----------------------------------------------------------------------*/
int H0_Calc(H0_TYPE * h0Time, TU_TYPE * tuTime, TD_TYPE * tdTime)
{
      char carrysec;
      char carrymin;

      char HourTU;
      char MinTU;
      char SecTU;

      char HourTD;
      char MinTD;
      char SecTD;

      char HourH0;
      char MinH0;
      char SecH0;

      if (!openedDevice) {
                return FALHA;
```

```
	}

	HourTU = ((tuTime->Hour & 0x0f) + ((tuTime->Hour >> 4) * 10));
	MinTU = ((tuTime->Min & 0x0f) + ((tuTime->Min >> 4) * 10));
	SecTU = ((tuTime->Sec & 0x0f) + ((tuTime->Sec >> 4) * 10));

	HourTD = ((tdTime->Hour & 0x0f) + ((tdTime->Hour >> 4) * 10));
	MinTD = ((tdTime->Min & 0x0f) + ((tdTime->Min >> 4) * 10));
	SecTD = ((tdTime->Sec & 0x0f) + ((tdTime->Sec >> 4) * 10));

	carrysec = 0;
	carrymin = 0;
	if (tdTime->signal) {
		SecH0 = SecTU - SecTD;
		if (SecH0 < 0) {
			SecH0 += 60;
			carrysec = 1;
		}
		MinH0 = MinTU - MinTD - carrysec;
		if (MinH0 < 0) {
			MinH0 += 60;
			carrymin = 1;
		}
		HourH0 = HourTU - HourTD - carrymin;
		if (HourH0 < 0) {
			HourH0 += 24;
		}
		if (HourH0 < 0) {
			HourH0 += 24;
		}
	} else {
		SecH0 = SecTU + SecTD;
		if (SecH0 >= 60) {
			SecH0 %= 60;
			carrysec = 1;
		}
		MinH0 = MinTU + MinTD + carrysec;
		if (MinH0 >= 60) {
			MinH0 %= 60;
			carrymin = 1;
		}
		HourH0 = (HourTU + HourTD + carrymin) % 24;
		if (HourH0 < 0) {
			HourH0 += 24;
		}
		if (HourH0 < 0) {
			HourH0 += 24;
		}
	}
	h0Time->Hour = HourH0;
	h0Time->Min = MinH0;
	h0Time->Sec = SecH0;

	return SUCESSO;
}

/*--------------------------------------------------------------------------+
| Status_Read()                                                             |
|    Realiza a leitura dos registros de status da placa PCI.                |
|                                                                           |
```

```
| Retorno:                                                    |
|    SUCESSO: sucesso na execucao                             |
|    FALHA  : falha na execucao                               |
+-------------------------------------------------------------------------*/
int Status_Read(STATUS_TYPE * stReg)
{
      unsigned char regFlags;

      if (!openedDevice) {
            return FALHA;
      }

      regFlags = in8(pci_bar_3 + 0x08);

      stReg->top_dec = regFlags & 0x80;
      stReg->cont_par = regFlags & 0x40;
      stReg->clock = regFlags & 0x20;
      stReg->sinal = regFlags & 0x10;
      stReg->reserved = regFlags & 0x08;
      stReg->cod_pres = regFlags & 0x04;
      stReg->hab_dados = regFlags & 0x02;
      stReg->env_irig = regFlags & 0x01;

      return SUCESSO;
}

/*-------------------------------------------------------------------------+
| GetCardParams()                                              |
|    Retorna parametros do driver do cartao MDT 200.           |
|                                                              |
| Retorno:                                                    |
|    SUCESSO: sucesso na execucao                             |
|    FALHA  : falha na execucao                               |
+-------------------------------------------------------------------------*/
int GetCardParams(CARD_PARAM * CardParameters)
{
      if (!openedDevice) {
            return FALHA;
      }

      strncpy(CardParameters->DriverName,          DRIVER_NAME,          sizeof(CardParameters-
>DriverName));
      strncpy(CardParameters->DriverVersion,       DRIVER_VERSION,       sizeof(CardParameters-
>DriverVersion));
      CardParameters->IRQ = irq;
      CardParameters->BaseAddress = pci_bar_0;

      return SUCESSO;
}
```

10.2 Código do Programa de Teste da Biblioteca de *Software* Básico

```
/*************************************************************************
*                                                                        *
*   AUDIOLAB Sistemas Eletronicos Ltda  - Sistema MDT200                 *
*                                                                        *
*   MODULO: MDTAPI200    TIPO: Implementacao      ARQUIVO: mdt200.c      *
*                                                                        *
*   Versao: 1.0              20/04/2004          Atualizacao:            *
*                                                                        *
*   Descricao: Programa de teste da biblioteca MDTApi200.[ch]            *
```

```
*                                                                    *
*   Atualizacoes:                                                    *
*                                                                    *
*   Sistema Operacional: QNX 6.2.1          Compilador: Watcom C     *
*                                                                    *
**********************************************************************/

#include <stdlib.h>

#include "MDT200Api.h"

/* Define o tempo maximo para timeout da funcao de espera de interrupcao */
#define SEG_TIMEOUT      1
#define MS_TIMEOUT       500

/* Define alguns textos a serem exibidos */
#define COD_PRESENTE     "Codigo Presente"
#define COD_AUSENTE      "Codigo Ausente"
#define CONT_PARADA      "Contagem Parada"
#define CONT_CURSO       "Contagem em Curso"

/*---------------------------------------------------------------------------+
| finaliza()                                                                 |
|                                                                            |
| Rotina de finalizacao executada ao capturar sinais como CTRL + C.          |
| Responsavel por finalizar o programa, liberando os recursos amigavelmente.|
|                                                                            |
| Retorno:                                                                   |
|    void                                                                    |
|                                                                            |
+---------------------------------------------------------------------------*/
void *finaliza()
{
        CloseDevice();
        exit(0);
        return NULL;
}

/*---------------------------------------------------------------------------+
| XoSignalInstalaFinalizador()                                               |
|                                                                            |
| Rotina que instala uma funcao de tratamento para o encerramento do         |
| programa.                                                                  |
|                                                                            |
| recebe:                                                                    |
|    terminador - funcao a ser chamada para tratar o encerramento            |
| retorna:                                                                   |
|    SUCESSO - tratador instalado com sucesso                                |
|    FALHA - erro                                                            |
+---------------------------------------------------------------------------*/
void XoSignalInstalaFinalizador(void *terminador)
{
        struct sigaction actionSig;
        sigset_t set;

        sigemptyset(&set);
        actionSig.sa_flags = 0;
        actionSig.sa_mask = set;
        actionSig.sa_handler = terminador;
```

```
	/* trata todos os sinais de encerramento */
	sigaction(SIGPWR, &actionSig, NULL);
	sigaction(SIGINT, &actionSig, NULL);
	sigaction(SIGABRT, &actionSig, NULL);
	sigaction(SIGTTIN, &actionSig, NULL);
	sigaction(SIGTTOU, &actionSig, NULL);
	sigaction(SIGHUP, &actionSig, NULL);
	sigaction(SIGILL, &actionSig, NULL);
	sigaction(SIGTERM, &actionSig, NULL);
	sigaction(SIGPIPE, &actionSig, NULL);
	sigaction(SIGQUIT, &actionSig, NULL);
	sigaction(SIGKILL, &actionSig, NULL);
	sigaction(SIGSTOP, &actionSig, NULL);
	sigaction(SIGTSTP, &actionSig, NULL);
}

/*-------------------------------------------------------------------------+
| MenuEscolhaTempo()                                                       |
|    Apresenta uma interface que permite configurar o tempo de intervalo   |
|    periodico entre as interrupcoes de tempo.                             |
|                                                                          |
| Retorno:                                                                 |
|    unsigned short: representa um valor valido usado para configurar o    |
|                registrador da placa que armazena o intervalo de tempo    |
|                usado para gerar interrupcoes periodicas                  |
+-------------------------------------------------------------------------*/
unsigned short MenuEscolhaTempo()
{
	/* Apresenta menu para escolha do intervalo da interrupcao programavel */
	while (1) {
		printf("\n 0 -> Desabilitado\n");
		printf(" 1 ->    1 ms\n");
		printf(" 2 ->    2 ms\n");
		printf(" 3 ->    5 ms\n");
		printf(" 4 ->   10 ms\n");
		printf(" 5 ->   20 ms\n");
		printf(" 6 ->   50 ms\n");
		printf(" 7 ->  100 ms\n");
		printf(" 8 ->  200 ms\n");
		printf(" 9 ->  500 ms\n");
		printf(" A -> 1000 ms\n");
		printf
			("Digite o numero do intervalo da interrupcao programavel seguido de
[ENTER]: \n");
		switch (getchar()) {
		case '0':
			return 0x0;
		case '1':
			return 0x1;
		case '2':
			return 0x2;
		case '3':
			return 0x3;
		case '4':
			return 0x4;
		case '5':
			return 0x5;
		case '6':
			return 0x6;
		case '7':
```

```
                        return 0x7;
                case '8':
                        return 0x8;
                case '9':
                        return 0x9;
                case 'A':
                case 'a':
                        return 0xA;
                }
        }
}

/*-------------------------------------------------------------------------+
| CorrigeTD()                                                              |
|    Corrige TD em um segundo quando milissegundos de TU = 000             |
|                                                                          |
| Retorno:                                                                 |
|    void                                                                  |
+-------------------------------------------------------------------------*/
void CorrigeTD(TD_TYPE * TempoTD)
{
        if (TempoTD->Sec == 0)
                TempoTD->Sec = 0xFF;
        else if (TempoTD->Sec % 16 == 0)
                TempoTD->Sec = TempoTD->Sec - 7;
        else
                TempoTD->Sec--;
        if (TempoTD->Sec == 0xFF) {
                TempoTD->Sec = 0x59;
                if (TempoTD->Min == 0)
                        TempoTD->Min = 0xFF;
                else if (TempoTD->Min % 16 == 0)
                        TempoTD->Min = TempoTD->Min - 7;
                else
                        TempoTD->Min--;
                if (TempoTD->Min == 0xFF) {
                        TempoTD->Min = 0x59;
                        if (TempoTD->Hour % 16 == 0)
                                TempoTD->Hour = TempoTD->Hour - 7;
                        else
                                TempoTD->Hour--;
                }
        }
}

/*-------------------------------------------------------------------------+
| main()                                                                   |
|    Funcao principal do programa, por onde a execucao inicia.            |
|                                                                          |
| Retorno:                                                                 |
|    int: representa um valor a ser retornado para o sistema operacional  |
|         ao termino da execucao                                           |
+-------------------------------------------------------------------------*/
int main()
{
        int result = 0, tdMS = 0, tuMS = 0, topJaEmitido = 0;
        char sinal = '+', *codigo, *contagem;
        TU_TYPE topTime;
        TU_TYPE tuTime;
        TD_TYPE tdTime;
```

```
H0_TYPE h0Time;
STATUS_TYPE stReg;

XoSignalInstalaFinalizador(finaliza);

/* Inicializa estruturas que serao exibidas */
memset(&topTime, 0, sizeof(TU_TYPE));
memset(&tuTime, 0, sizeof(TU_TYPE));
memset(&tdTime, 0, sizeof(TD_TYPE));
memset(&h0Time, 0, sizeof(H0_TYPE));
memset(&stReg, 0, sizeof(STATUS_TYPE));

if (OpenDevice() != SUCESSO) {
        printf("Falha ao abrir dispositivo PCI!\n");
        return EXIT_FAILURE;
}

if (EnableTOPIntr() != SUCESSO) {
        printf("Falha ao habilitar interrupcao de TOP!\n");
        return EXIT_FAILURE;
}

if (EnableProgIntr() != SUCESSO) {
        printf("Falha ao habilitar interrupcao de tempo!\n");
        return EXIT_FAILURE;
}

if (SetProgIntrPeriod(MenuEscolhaTempo()) != SUCESSO) {
        printf("Falha ao configurar periodo de tempo!\n");
        return EXIT_FAILURE;
}

if (InstallHandler() != SUCESSO) {
        printf("Falha ao instalar manipulador de interrupcao!\n");
        return EXIT_FAILURE;
}

/* Loop infinito de espera por interrupcoes */
while (1) {
        /* Fica bloqueado ateh receber notificacao de interrupcao do
         * handler ou vencer o tempo especificado como timeout
         */
        switch (WaitInterrupt(1, 500)) {
        case IDENT_AMBOS:
                /* Neste caso efetuamos a leitura de todos os campos, inclusive TOP */
                topJaEmitido = 1;
                TU_Read(&topTime);
                /* Propagamos para a clausula seguinte; nao usamos break */

        case IDENT_TEMPO:
                /* Efetuamos a leitura de todos os campos, exclusive TOP */
                TU_Read(&tuTime);
                TD_Read(&tdTime);
                H0_Calc(&h0Time, &tuTime, &tdTime);
                Status_Read(&stReg);
                sinal = (tdTime.signal ? '+' : '-');
                codigo = (stReg.cod_pres ? COD_PRESENTE : COD_AUSENTE);
                if (stReg.cont_par) {
                        contagem = CONT_PARADA;
                        tdMS = 0;
```

```
                        } else {
                                contagem = CONT_CURSO;
                                tuMS = (tuTime.ms & 0x0F) + (((tuTime.ms & 0xF0) >> 4) * 10)
                                        + ((tuTime.ms >> 8) * 100);
                                if (tdTime.signal) {
                                        tdMS = tuMS;
                                } else {
                                        tdMS = (1000 - tuMS) % 1000;
                                        if (tuMS != 0) {
                                                CorrigeTD(&tdTime);
                                        }
                                }
                        }
                        break;

                case IDENT_TOP:
                        /* Neste caso mantemos os dados atuais, exceto para o TOP */
                        topJaEmitido = 1;
                        TU_Read(&topTime);
                        break;

                case TIMEOUT:
                        Status_Read(&stReg);
                        codigo = (stReg.cod_pres ? COD_PRESENTE : COD_AUSENTE);
                        contagem = (stReg.cont_par ? CONT_PARADA : CONT_CURSO);
                        break;

                default:
                        printf("Falha Fatal no WaitInterrupt!\n");
                        exit(EXIT_FAILURE);

                }                                               /* switch (WaitInterrupt()) */

                if (topJaEmitido) {
                        printf
                                ("\nTU:  %02x:%02x:%02x:%03x    TD:  %c%02x:%02x:%02x:%03d
H0: %02d:%02d:%02d  TOP: %02x:%02x:%02x:%03x\n%s  -  %s\n",
                                 tuTime.Hour, tuTime.Min, tuTime.Sec, tuTime.ms, sinal,
                                 tdTime.Hour, tdTime.Min, tdTime.Sec, tdMS, h0Time.Hour,
                                 h0Time.Min,        h0Time.Sec,        topTime.Hour,        topTime.Min,
topTime.Sec,
                                 topTime.ms, codigo, contagem);
                } else {
                        printf
                                ("\nTU:  %02x:%02x:%02x:%03x    TD:  %c%02x:%02x:%02x:%03d
H0: %02d:%02d:%02d  TOP:\n%s  -  %s\n",
                                 tuTime.Hour, tuTime.Min, tuTime.Sec, tuTime.ms, sinal,
                                 tdTime.Hour, tdTime.Min, tdTime.Sec, tdMS, h0Time.Hour,
                                 h0Time.Min, h0Time.Sec, codigo, contagem);
                }
        }                                                       // while (1)

        CloseDevice();
        return EXIT_SUCCESS;
}                                                               // main()
```

10.3 Código do Disseminador de Dados de Tempo na Rede

10.3.1 Arquivo de Cabeçalho mtrserver.h

```
#ifndef MDTSERV_H
```

```
#define MDTSERV_H

/* Define algumas posicoes no array de bytes da mensagem */
#define POS_HORA_H0              0
#define POS_HORA_TOP_INI         0
#define POS_MIN_H0               1
#define POS_MIN_TOP_INI          1
#define POS_SEG_H0               2
#define POS_SEG_TOP_INI          2
#define POS_SINAL_TD             3
#define POS_MS_MAIS_SIG_TOP_INI  3
#define POS_COD_PRES             4
#define POS_MS_MENOS_SIG_TOP_INI 4
#define POS_CONT_PAR             5
#define POS_IDX_BASE_DIA         6
#define POS_HORA_TOP_FIM         7
#define POS_MIN_TOP_FIM          8
#define POS_SEG_TOP_FIM          9
#define POS_MS_MAIS_SIG_TOP_FIM  10
#define POS_MS_MENOS_SIG_TOP_FIM 11

/* Define alguns codigos de retorno */
#define SUCESSO 1
#define FALHA   0
#define TIMEOUT -1

/* Define tempo de 1,5 segundos para timeout de recepcao de informacao de tempo */
#define TEMPO_TIMEOUT      1500

/* Define alguns codigos de frequencia */
#define FREQ_DESABILITADA 0
#define FREQ_SEGUNDO      10

/* Define o tamanho da mensagem em bytes a ser enviada via pacote UDP */
#define TAM_MSG (POS_MS_MENOS_SIG_TOP_FIM + 1)

/* Define o tamanho a ser enviado da mensagem: a mensagem com TOP tem 1 byte a mais
 * que a mensagem sem o TOP; este eh o flag usado para indicar presenca do TOP na
 * mensagem enviada
 */
#define TAM_MSG_TOP     5
#define TAM_MSG_COM_TOP (TAM_MSG)
#define TAM_MSG_SEM_TOP (TAM_MSG_COM_TOP - 5)

/* Define a porta usada para comunicacao */
#define PORT_NUMBER 9999

/* Define a mascara do IP de broadcast */
#define IP_BROADCAST "255.255.255.255"

/* Define alguns textos a serem exibidos */
#define MSG_COMO_USAR "\nUso: ./mdtserv [-p porta] [-v]\n[-p porta]\tParametro opcional que
define o numero da porta a ser usada\n[-v]\t        Parametro opcional que imprime informacoes sobre
o sistema\n\n"
#define MSG_NOME_DRIVER             "Nome do driver  : %s\n"
#define MSG_VERSAO_DRIVER           "Versao do driver: %s\n"
#define MSG_INTERVALO_TEMPO         "Programacao do intervalo de interrupcoes periodicas: 1
segundo\n"
#define MSG_EMISSAO_TOP             "Emissao de TOP H0 habilitada.\n"
#define MSG_INTERRUPCAO_PCI         "Interrupcao do barramento PCI: %d\n"
```

```
#define MSG_ENDERECO_BASE          "Endereco base do cartao MDT-200: %x\n\n"
#define MSG_OPCAO_DESCONHECIDA     "Opcao desconhecida `-%c'.\n"
#define MSG_CHAR_DESCONHECIDO      "Caracter de opcao desconhecido `\\x%x'.\n"
#define MSG_USO_PORTA              "Utilizando porta: %d\n\n"
#define MSG_CTRL_C                 "Para finalizar digite CTRL + C\n\n"
#define MSG_SUCESSO_REINICIALIZACAO_REDE "Sucesso ao recuperar de send\n"
#define ERRO_OPCAO_NAO_RECONHECIDA  "Opcao nao reconhecida: %s\n\n"
#define ERRO_INST_MDT200           "Erro ao inicializar Api do MDT200. Modulo Pci9030 esta carregado?\n"
#define ERRO_INI_SERVICO_REDE      "Erro ao inicializar servico de rede.\n"
#define ERRO_INST_HANDLER_INT_TEMPO "Erro ao instalar manipulador de interrupcao de tempo.\n"
#define ERRO_INST_HANDLER_INT_TOP  "Erro ao instalar manipulador de interrupcao de TOP\n"
#define ERRO_INST_TOP_HANDLER      "Erro ao instalar ouvinte de interrupcao de TOP.\n"
#define ERRO_TIMEOUT_INF_TEMPO     "ATENCAO: Timeout ao receber informacao de tempo do dispositivo MDT!\nNao foi possivel acertar relogio da maquina de acordo com GPS.\n"

#define COD_PRESENTE    "Codigo Presente"
#define COD_AUSENTE     "Codigo Ausente"
#define CONT_PARADA     "Contagem Parada"
#define CONT_CURSO      "Contagem em Curso"
#define SINAL_MAIS      "+"
#define SINAL_MENOS     "-"
#define SINAL_MAIS_CHAR  '+'
#define SINAL_MENOS_CHAR '-'

#endif
```

10.3.2 Arquivo Fonte - mtrserver.c

```
#include <stdlib.h>
#include <errno.h>
#include <resolv.h>
#include <time.h>
#include <signal.h>
#include <limits.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#include "MDT200Api.h"
#include "mtrserver.h"

/* Parametros da placa */
/* TODO - Separar funcoes de rede e de acesso aa placa em
 * modulos separados para evitar variaveis globais
 */
static unsigned int reg_address;
static int sd = 0;
static int porta = PORT_NUMBER;
static struct sockaddr_in addr;
/* topTime eh um recurso compartilhado entre as threads */
static TU_TYPE topTime;
/* jahOcorreuTop eh compartilhado, embora seja atualizado em somente uma thread */
static volatile sig_atomic_t jahOcorreuTop = 0;

void CorrigeTD(TD_TYPE * TempoTD)
{
```

```
		if (TempoTD->Sec == 0) {
			TempoTD->Sec = 0xFF;
		} else if (TempoTD->Sec % 16 == 0) {
			TempoTD->Sec = TempoTD->Sec - 7;
		} else {
			TempoTD->Sec--;
		}
		if (TempoTD->Sec == 0xFF) {
			TempoTD->Sec = 0x59;
			if (TempoTD->Min == 0) {
				TempoTD->Min = 0xFF;
			} else if (TempoTD->Min % 16 == 0) {
				TempoTD->Min = TempoTD->Min - 7;
			} else {
				TempoTD->Min--;
			}
			if (TempoTD->Min == 0xFF) {
				TempoTD->Min = 0x59;
				if (TempoTD->Hour % 16 == 0) {
					TempoTD->Hour = TempoTD->Hour - 7;
				} else {
					TempoTD->Hour--;
				}
			}
		}
}

/* As mensagens tem sua diferenciacao com base no tamanho da mensagem.
 * Para implementar pacotes diferentes com o mesmo tamanho sera necessario
 * incluir algum campo que identifique o tamanho da mensagem.
 * Estas mensagens foram otimizadas no tamanho, fazendo com que fiquem no
 * menor tamanho possivel.
 */
void PreparaMsgTOP(unsigned char * strMsg)
{
		// Evita modificacao de top durante seu uso
		strMsg[POS_HORA_TOP_INI] = topTime.Hour;
		strMsg[POS_MIN_TOP_INI] = topTime.Min;
		strMsg[POS_SEG_TOP_INI] = topTime.Sec;
		strMsg[POS_MS_MAIS_SIG_TOP_INI] = topTime.ms >> 8;
		strMsg[POS_MS_MENOS_SIG_TOP_INI] = topTime.ms & 0xFF;
}

void PreparaMsgComTOP(H0_TYPE * h0, TD_TYPE * td, STATUS_TYPE * status,
                                                unsigned char * strMsg)
{
		strMsg[POS_HORA_H0]      = h0->Hour;
		strMsg[POS_MIN_H0]       = h0->Min;
		strMsg[POS_SEG_H0]       = h0->Sec;
		strMsg[POS_SINAL_TD]     = td->signal;
		strMsg[POS_COD_PRES]     = status->cod_pres;
		strMsg[POS_CONT_PAR]     = status->cont_par;
		strMsg[POS_IDX_BASE_DIA] = (td->Hour > 23) ? 1 : 0;
		strMsg[POS_HORA_TOP_FIM] = topTime.Hour;
		strMsg[POS_MIN_TOP_FIM]  = topTime.Min;
		strMsg[POS_SEG_TOP_FIM]  = topTime.Sec;
		strMsg[POS_MS_MAIS_SIG_TOP_FIM]  = topTime.ms >> 8;
		strMsg[POS_MS_MENOS_SIG_TOP_FIM] = topTime.ms & 0xFF;
}
```

```
void PreparaMsgSemTOP(H0_TYPE * h0, TD_TYPE * td, STATUS_TYPE * status,
                                                unsigned char * strMsg)
{
        strMsg[POS_HORA_H0]      = h0->Hour;
        strMsg[POS_MIN_H0]       = h0->Min;
        strMsg[POS_SEG_H0]       = h0->Sec;
        strMsg[POS_SINAL_TD]     = td->signal;
        strMsg[POS_COD_PRES]     = status->cod_pres;
        strMsg[POS_CONT_PAR]     = status->cont_par;
        strMsg[POS_IDX_BASE_DIA] = (td->Hour > 23) ? 1 : 0;
}

void EnviaPacoteSincronizacao(unsigned char * msg, int tamanho_msg)
{
        if (send(sd, msg, tamanho_msg, 0) == -1) {
                perror("send:");
                // Tenta reinicializar servicos de rede
                if (InicializaServicoRedeUDP() != SUCESSO) {
                        fprintf(stderr, ERRO_INI_SERVICO_REDE);
                } else {
                        fprintf(stderr, MSG_SUCESSO_REINICIALIZACAO_REDE);
                }
        }
}

int ObtemDadosMDT(TU_TYPE * tuTime, TD_TYPE * tdTime, STATUS_TYPE * status)
{
        unsigned char strMsg[TAM_MSG];

        switch (WaitInterrupt(1, 500)) {
        case IDENT_TEMPO:
                // Ocorreu interrupcao de tempo; evita acesso multiplo aos recursos da placa
                TU_Read(tuTime);
                TD_Read(tdTime);
                Status_Read(status);

                return SUCESSO;

        case IDENT_AMBOS:
                jahOcorreuTop = 1;
                TU_Read(tuTime);
                TD_Read(tdTime);
                Status_Read(status);

                return SUCESSO;

        case IDENT_TOP:
                jahOcorreuTop = 1;
                TU_Read(&topTime);
                PreparaMsgTOP(strMsg);
                EnviaPacoteSincronizacao(strMsg, TAM_MSG_TOP);

                return SUCESSO;

        default:
                printf("Falha Fatal no WaitInterrupt!\n");
                exit(EXIT_FAILURE);

        case TIMEOUT:
                // Ocorreu timeout; evita acesso multiplo aos recursos da placa
```

```
                Status_Read(status);

                return TIMEOUT;
        }
}

void AlteraRelogioSistema(TU_TYPE * tu)
{
        time_t simpleTime;
        struct tm *p;
        struct timespec ts;
        struct timeval tv;
        char tempoStr[TAM_MSG];

        if (clock_gettime(CLOCK_REALTIME, &ts) == -1) {
                perror("clock_gettime:");
        }

        p = localtime(&ts.tv_sec);

        snprintf(tempoStr, TAM_MSG, "%02x%02x%02x", tu->Hour, tu->Min, tu->Sec);
        sscanf(tempoStr, "%02d%02d%02d", &p->tm_hour, &p->tm_min, &p->tm_sec);

        if ((p->tm_hour == 23) && (p->tm_min >= 55)) {
                // Nao atualizamos a hora local nos ultimos 5 minutos do dia
                // para permitir virada do dia pelo relogio do sistema
                return;
        }

        if ((simpleTime = mktime(p)) == (time_t) - 1) {
                perror("mktime:");
        }

        tv.tv_sec  = simpleTime;
        tv.tv_usec = 0;

        if (settimeofday(&tv, NULL) != 0) {
                perror("settimeofday:");
        }
}

int InicializaApiMDT(int frequencia, CARD_PARAM * card_params)
{
      /* Loop infinito de espera por interrupcoes */

        if (frequencia < FREQ_DESABILITADA || frequencia > FREQ_SEGUNDO) {
                return FALHA;
        }
        if (OpenDevice() != SUCESSO) {
                return FALHA;
        }

        if (GetCardParams(card_params) != SUCESSO) {
                return FALHA;
        }

        if (SetProgIntrPeriod(frequencia) != SUCESSO) {
                return FALHA;
        }
        if (EnableProgIntr() != SUCESSO) {
```

```
		return FALHA;
	}
	if (EnableTOPIntr() != SUCESSO) {
		return FALHA;
	}
	if (InstallHandler() != SUCESSO) {
		return FALHA;
	}

	return SUCESSO;
}

int InicializaServicoRedeUDP()
{
	const int on = 1;

	if ((sd = socket(PF_INET, SOCK_DGRAM, 0)) == -1) {
		perror("socket:");
		return FALHA;
	}

	/* Autoriza utilizacao de broadcast */
	if (setsockopt(sd, SOL_SOCKET, SO_BROADCAST, &on, sizeof(on)) != 0) {
		perror("setsockopt:");
		return FALHA;
	}

	/* Inicializa com endereco de broadcast */
	bzero(&addr, sizeof(struct sockaddr_in));
	addr.sin_family = AF_INET;
	addr.sin_port = htons(porta);
	inet_aton(IP_BROADCAST, &addr.sin_addr);

	if (connect(sd, (struct sockaddr *)&addr, sizeof(struct sockaddr_in)) != 0) {
		perror("connect:");
		return FALHA;
	}

	return SUCESSO;
}

void ImprimeInformacoesMDT(CARD_PARAM * card_params)
{
	printf(MSG_NOME_DRIVER, &card_params->DriverName[0]);
	printf(MSG_VERSAO_DRIVER, card_params->DriverVersion);
	printf(MSG_INTERVALO_TEMPO);
	printf(MSG_EMISSAO_TOP);
	printf(MSG_INTERRUPCAO_PCI, card_params->IRQ);
	printf(MSG_ENDERECO_BASE, card_params->BaseAddress);
}

int main(int argc, char **argv)
{
	int c;
	TU_TYPE tuTime;
	TD_TYPE tdTime;
	STATUS_TYPE status;
	CARD_PARAM card_params;

	printf(MSG_COMO_USAR);
```

```
// Tenta inicializar placa PCI do dispositivo MDT
if (InicializaApiMDT(FREQ_SEGUNDO, &card_params) != SUCESSO) {
	fprintf(stderr, ERRO_INST_MDT200);
	return EXIT_FAILURE;
}
// Efetua parse da linha de comando
while ((c = getopt(argc, argv, "vp:")) != -1) {
	switch (c) {
	case 'v':
		ImprimeInformacoesMDT(&card_params);
		return EXIT_SUCCESS;

	case 'p':
		porta = atoi(optarg);
		break;

	case '?':
		if (isprint(optopt)) {
			fprintf(stderr, MSG_OPCAO_DESCONHECIDA, optopt);
		} else {
			fprintf(stderr, MSG_CHAR_DESCONHECIDO, optopt);
		}
		return EXIT_FAILURE;

	default:
		abort();
	}
}

if (optind < argc) {
	printf(ERRO_OPCAO_NAO_RECONHECIDA, argv[optind]);
	return EXIT_FAILURE;
}

ImprimeInformacoesMDT(&card_params);

printf(MSG_USO_PORTA, porta);

printf(MSG_CTRL_C);

memset(&tuTime, 0, sizeof(tuTime));
memset(&topTime, 0, sizeof(topTime));
memset(&tdTime, 0, sizeof(tdTime));
memset(&status, 0, sizeof(status));
tdTime.signal = 1;

// Tenta inicializar servicos de rede
if (InicializaServicoRedeUDP() != SUCESSO) {
	fprintf(stderr, ERRO_INI_SERVICO_REDE);
	return EXIT_FAILURE;
}

// A thread principal eh a encarregada de obter informacoes da interrupcao
// de tempo, alterar o relogio do sistema e tambem repassar informacoes de
// tempo para a rede
while (1) {
	unsigned char strMsg[TAM_MSG];
	int tdms = 0;
	char *contagem, *codigo, *sinal;
```

```
		H0_TYPE h0Time;

		if (ObtemDadosMDT(&tuTime, &tdTime, &status) == SUCESSO) {
			AlteraRelogioSistema(&tuTime);
		} else {
			// Entao ocorreu timeout
			fprintf(stderr, ERRO_TIMEOUT_INF_TEMPO);
		}

		// Calcula H0 que sera impresso
		H0_Calc(&h0Time, &tuTime, &tdTime);

		// Obtem sinal de TD e o estado do codigo e da contagem
		sinal = (tdTime.signal ? SINAL_MAIS : SINAL_MENOS);
		codigo = (status.cod_pres ? COD_PRESENTE : COD_AUSENTE);
		contagem = (status.cont_par ? CONT_PARADA : CONT_CURSO);

		// Calcula milissegundos de TD (nao usado atualmente) e corrige TD se necessario
		if (status.cont_par) {
			tdms = 0;
		} else {
			int tums =
				(tuTime.ms & 0x0F) + (((tuTime.ms & 0xF0) >> 4) * 10) +
				((tuTime.ms >> 8) * 100);
			if (tdTime.signal == 0) {
				tdms = (1000 - tums) % 1000;
				if (tums != 0) {
					CorrigeTD(&tdTime);
				}
			} else {
				tdms = tums;
			}
		}

		if (jahOcorreuTop) {
			// Imprime tempo com TOP na tela; evita modificacao de TOP durante seu uso
			printf
				("TU: %02x:%02x:%02x TD: %s%02x:%02x:%02x TOP:
%02x:%02x:%02x:%03x H0: %02d:%02d:%02d %s %s\n",
				 tuTime.Hour, tuTime.Min, tuTime.Sec, sinal, tdTime.Hour,
				 tdTime.Min, tdTime.Sec, topTime.Hour, topTime.Min, topTime.Sec,
				 topTime.ms, h0Time.Hour, h0Time.Min, h0Time.Sec, codigo,
				 contagem);

			PreparaMsgComTOP(&h0Time, &tdTime, &status, strMsg);
			EnviaPacoteSincronizacao(strMsg, TAM_MSG_COM_TOP);
		} else {
			// Imprime tempo sem TOP na tela
			printf
				("TU: %02x:%02x:%02x TD: %s%02x:%02x:%02x H0:
%02d:%02d:%02d %s %s\n",
				 tuTime.Hour, tuTime.Min, tuTime.Sec, sinal, tdTime.Hour,
				 tdTime.Min, tdTime.Sec, h0Time.Hour, h0Time.Min, h0Time.Sec,
				 codigo, contagem);

			PreparaMsgSemTOP(&h0Time, &tdTime, &status, strMsg);
			EnviaPacoteSincronizacao(strMsg, TAM_MSG_SEM_TOP);
		}
	}
```

```
	return EXIT_SUCCESS;
}
```

10.4 Código do Mostrador de Tempo na Rede

10.4.1 Arquivo Fonte – MTR100Dlg.cpp

```
// MTR100Dlg.cpp : implementation file
//

#include "stdafx.h"
#include "MTR100.h"
#include "MTR100Dlg.h"
#include "Config.h"

#include <MDT200API.h>
#include <process.h>
#include <errno.h>
#include <winsock2.h>


#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/timeb.h>


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////////

typedef struct _TD_CALCULADO
{
	int	Hour;
	int	Min;
	int	Sec;
	int  signal;
} TD_CALCULADO;

#define METADE_SEG_EM_MILISEG 500

#define TAM_TEMPO_FORMATADO 9

#define POS_HORA_H0		0
#define POS_MIN_H0		1
#define POS_SEG_H0		2
#define POS_SINAL_TD		3
#define POS_COD_PRES		4
#define POS_CONT_PAR		5
#define POS_HORA_TOP		6
#define POS_MIN_TOP		7
#define POS_SEG_TOP		8
#define POS_MS_MAIS_SIG_TOP	9
#define POS_MS_MENOS_SIG_TOP	10
```

```
#define TAM_MSG_TOP      5
#define TAM_MSG_COM_TOP  (POS_MS_MENOS_SIG_TOP + 1)

#define SINAL_MAIS_CHAR   '+'
#define SINAL_MENOS_CHAR  '-'

#define MSG_CODIGO_PRESENTE  "Presente"
#define MSG_CODIGO_AUSENTE   "Ausente"
#define MSG_CONTAGEM_CURSO   "Em curso"
#define MSG_CONTAGEM_PARADA  "Parada"

#define GREEN   RGB(  0,255,  0)
#define RED     RGB(255,  0,  0)

#define ERRO_INI_WIN_SOCKET   "Erro ao inicializar biblioteca Windows Socket"
#define ERRO_CRIAR_SOCKET     "Erro ao criar socket!!!"
#define ERRO_BIND_SOCKET      "Erro ao tentar atar ao socket criado!!!"

//VARIAVEIS GLOBAIS

typedef unsigned (WINAPI *CAST_FUNCTION)(LPVOID);  //Cast necessario para as threads
typedef unsigned *CAST_LPDWORD;                    //Cast necessario para as
threads

static HANDLE   hInterruptEvent;                   //Handle para a interrupcao programavel
static HANDLE   hTOPEvent;                         //Handle para a
interrupcao de TOP DEC
static HANDLE   hMutexRecebeuMsg;        //Mutex para sincronizacao de acessos a recursos
static HANDLE   hMutexMudouPorta;    //Mutex para sincronizacao de acessos a recursos

static SOCKET sd = 0;
struct sockaddr_in addr;

static HANDLE         hDevice;                     //Handle para acessar o
dispositivo
static DEVICE_LOCATION Device;
//Informacoes sobre a placa
static U32            RegAddress;                  //Recebe endereco para
leitura e escrita de registros

static HANDLE   hIntThread;                        //Handle para a thread de
interrupcoes periodicas
static DWORD    pIntThreadId;                      //ID da thread de interrupcoes
periodicas

static HANDLE   hWaitTimeThread;                   //Handle para a thread de TOP
DEC
static DWORD    pWaitTimeThreadId;                 //ID da thread de TOP DEC

DWORD WINAPI InterruptThread(CMTR100Dlg *Window);   //Prototipo da thread de interrupcoes
periodicas
DWORD WINAPI WaitTimeThread(CMTR100Dlg *Window);    //Prototipo da thread

void CalculaTD(TD_CALCULADO *tdTime, CString &strTempo1, CString &strH0, bool sinal);

static volatile int recebeuMsg = 1;
static volatile int mudouPorta = 0;

////////////////////////////////////////////////////////////////////////////////////////////////
```

```
/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////


CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////


void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////


BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////


CMTR100Dlg::CMTR100Dlg(CWnd* pParent /*=NULL*/)
	: CDialog(CMTR100Dlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CMTR100Dlg)
	//}}AFX_DATA_INIT
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);

	WORD wVersionRequested;     /* socket dll version info */
	WSADATA wsaData;            /* data for socket lib initialisation */

	wVersionRequested = MAKEWORD(1, 1);

	/*
	* We need to call the WSAStartup routine BEFORE we try to use any of
	* the Winsock dll calls.
	*/

	if (WSAStartup(wVersionRequested, &wsaData ) != 0) {
	   AfxMessageBox(ERRO_INI_WIN_SOCKET);
	}
}

////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////


void CMTR100Dlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CMTR100Dlg)
	DDX_Control(pDX, IDC_RICHEDIT_TOP, m_EditTOP);
	DDX_Control(pDX, IDC_RICHEDIT_H0, m_EditH0);
	DDX_Control(pDX, IDC_RICHEDIT_TD, m_EditTD);
	DDX_Control(pDX, IDC_RICHEDIT_TU, m_EditTU);
	DDX_Control(pDX, IDC_RICHEDIT_CODIGO, m_EditCodigo);
	DDX_Control(pDX, IDC_RICHEDIT_CONTAGEM, m_EditContagem);
	//}}AFX_DATA_MAP
}

////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////


BEGIN_MESSAGE_MAP(CMTR100Dlg, CDialog)
	//{{AFX_MSG_MAP(CMTR100Dlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_COMMAND(ID_CONFIGURACOES, OnMenuConfig)
	ON_COMMAND(ID_AJUDA_SOBRE, OnMenuSobre)
	ON_COMMAND(ID_SEMPRE_VISIVEL, OnMenuSempreVisivel)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//INICIALIZACOES

BOOL CMTR100Dlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	// Faz a janela ficar sempre no topo
	SetWindowPos(&CWnd.wndTopMost, 0, 0, 0, 0,
SWP_NOMOVE|SWP_NOSIZE|SWP_NOACTIVATE);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}

	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	//Insere barra de menu
	m_Menu.LoadMenu(IDR_MENUBAR);
	SetMenu(&m_Menu);

	CHARFORMAT cfG;
	CHARFORMAT cfM;

	//Seta estilo das fontes grandes
	cfG.dwMask = CFM_SIZE | CFM_BOLD | CFM_CHARSET | CFM_COLOR;
   cfG.dwEffects = CFE_BOLD;
   cfG.yHeight = 520;
	cfG.crTextColor = RGB(0,0,0);
   cfG.bCharSet = GREEK_CHARSET;
   cfG.bPitchAndFamily = FF_MODERN;
   strcpy(cfG.szFaceName,"Arial");
   cfG.cbSize = sizeof(cfG);

	//Seta estilo das fontes medias
	cfM.dwMask = CFM_SIZE | CFM_BOLD | CFM_CHARSET;
   cfM.dwEffects = CFE_BOLD;
   cfM.yHeight = 375;
   cfM.crTextColor = RGB(0,0,0);
   cfM.bCharSet = GREEK_CHARSET;
   cfM.bPitchAndFamily = FF_MODERN;
```

```
   cfM.cbSize = sizeof(cfM);

	m_EditTU.SetDefaultCharFormat(cfG);
	m_EditTD.SetDefaultCharFormat(cfG);
	m_EditH0.SetDefaultCharFormat(cfM);
	m_EditTOP.SetDefaultCharFormat(cfM);

	//Cria objeto para sincronizacao de acesso a recursos
	hMutexRecebeuMsg = CreateMutex(NULL, FALSE, "Exclusao Mutua Recebeu Msg");
	//Cria objeto para sincronizacao de acesso a recursos
	hMutexMudouPorta = CreateMutex(NULL, FALSE, "Exclusao Mutua Mudou Porta");

	hIntThread = (HANDLE) _beginthreadex(NULL,
                                                0,

(CAST_FUNCTION)InterruptThread,
                                                this,
                                                0,

(CAST_LPDWORD)&pIntThreadId
                                                );
	if ((int)hIntThread == -1) AfxMessageBox("Erro ao inicializar thread de timer");

	//Inicia thread de recepcao de TOP DEC
	hWaitTimeThread = (HANDLE) _beginthreadex(NULL,
                                                0,

(CAST_FUNCTION)WaitTimeThread,
                                                this,
                                                0,

(CAST_LPDWORD)&pWaitTimeThreadId
                                                );
	if ((int)hWaitTimeThread == -1) AfxMessageBox("Erro ao inicializar thread de recepcao de
TOP DEC");

   return TRUE;
}

////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////


void CMTR100Dlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
```

```
void CMTR100Dlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}


////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////


HCURSOR CMTR100Dlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}


////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//Funcao de finalizacao do aplicativo

void CMTR100Dlg::OnCancel()
{
	//Termina thread
	DWORD ExCode = -1;
	TerminateThread(hIntThread, ExCode);

	//Fecha os handles
	CloseHandle(hIntThread);
	CloseHandle(hWaitTimeThread);

	//Fecha janela do aplicativo
	CDialog::OnCancel();
}


int CMTR100Dlg::getPorta()
{
```

```
	return porta;
}

void CMTR100Dlg::IniciaSocketRede(int configPorta)
{
	// Finaliza
	shutdown(sd, SD_BOTH);
	closesocket(sd);

	// Inicializa
	porta = configPorta;
	if ((sd = socket(PF_INET, SOCK_DGRAM, 0)) == INVALID_SOCKET) {
		AfxMessageBox(ERRO_CRIAR_SOCKET);
		return;
	};

	memset(&addr, 0, sizeof(addr));
	addr.sin_family      = AF_INET;
	addr.sin_port        = htons(configPorta);
	addr.sin_addr.s_addr = INADDR_ANY;

	if (bind(sd, (struct sockaddr*)&addr, sizeof(addr)) != 0) {
		AfxMessageBox(ERRO_BIND_SOCKET);
		return;
	}
}



////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//Esta funcao abre a janela de configuracao do aplicativo. Se o usuario clicar em OK, as novas configuracoes
//sao setadas para a placa.
void CMTR100Dlg::OnMenuConfig()
{
	CConfig ConfigDlg(this);

	int nResponse = ConfigDlg.DoModal();
	if (nResponse == IDOK)
	{
		IniciaSocketRede(ConfigDlg.nPorta);

		WaitForSingleObject(hMutexMudouPorta,INFINITE);
		mudouPorta = 1;
		ReleaseMutex(hMutexMudouPorta);
	}
	else if (nResponse == IDCANCEL)
	{
	}

}


////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//Esta funcao abre a janela de ABOUT
```

```
void CMTR100Dlg::OnMenuSobre()
{
	CAboutDlg AboutDlg;
	int nResponse = AboutDlg.DoModal();
	if (nResponse == IDOK)
	{
	}
}


void CMTR100Dlg::OnMenuSempreVisivel()
{
	if (m_Menu.CheckMenuItem(ID_SEMPRE_VISIVEL, MF_BYCOMMAND) ==
MF_CHECKED) {
		m_Menu.CheckMenuItem(ID_SEMPRE_VISIVEL, MF_UNCHECKED);
		// Nao faz a janela ficar sempre no topo
		SetWindowPos(&CWnd.wndNoTopMost, 0, 0, 0, 0,
SWP_NOMOVE|SWP_NOSIZE|SWP_NOACTIVATE);
	}
	else {
		m_Menu.CheckMenuItem(ID_SEMPRE_VISIVEL, MF_CHECKED);
		// Faz a janela ficar sempre no topo
		SetWindowPos(&CWnd.wndTopMost, 0, 0, 0, 0,
SWP_NOMOVE|SWP_NOSIZE|SWP_NOACTIVATE);
	}
}


DWORD WINAPI WaitTimeThread(CMTR100Dlg *Window)
{
	Window->IniciaSocketRede(PORTA_DEFAULT);

	while(1)
	{
		TU_TYPE	tmTOP;
		TD_CALCULADO tmTD;
		H0_TYPE	tmH0;
		CTime	theTime;
		struct _timeb tstruct;
		CString	strTU, strTD, strH0, strTOP;
		U8	buffer[TAM_MSG_COM_TOP];
		int	bytes, addr_len = sizeof(addr);
		int	contagemParada, codigoPresente;

		// Funcao blocante; aguardamos mensagem da rede
		bytes = recvfrom(sd, (char *)buffer, sizeof(buffer), 0, (struct sockaddr*)&addr,
&addr_len);
		// Obtemos hora da maquina imediatamente ao desblocar
		theTime = CTime::GetCurrentTime();
		// Obtemos milissegundos da hora da maquina imeditamente ao desblocar
		_ftime(&tstruct);

		if (bytes == 0 || bytes == SOCKET_ERROR) {
			// Verifica erro ao receber mensagem da rede
			continue;
		}

		//Evitando acesso multiplo a recursos...
		WaitForSingleObject(hMutexRecebeuMsg,INFINITE);
		recebeuMsg = 1;
```

```
		ReleaseMutex(hMutexRecebeuMsg);

		// Obtem dados da mensagem
		tmH0.Hour      = buffer[POS_HORA_H0];
		tmH0.Min       = buffer[POS_MIN_H0];
		tmH0.Sec       = buffer[POS_SEG_H0];
		tmTD.signal    = buffer[POS_SINAL_TD];
		codigoPresente = buffer[POS_COD_PRES];
		contagemParada = buffer[POS_CONT_PAR];
		tmTOP.Hour     = buffer[POS_HORA_TOP];
		tmTOP.Min      = buffer[POS_MIN_TOP];
		tmTOP.Sec      = buffer[POS_SEG_TOP];
		tmTOP.ms       = buffer[POS_MS_MAIS_SIG_TOP];
		tmTOP.ms       = tmTOP.ms << 8;
		tmTOP.ms      |= buffer[POS_MS_MENOS_SIG_TOP];

		if (!codigoPresente) {
			// Entao ignora quaiquer outros dados da mensagem
			Window->m_EditCodigo.SetWindowText(MSG_CODIGO_AUSENTE);
			Window->m_EditCodigo.SetBackgroundColor(false, RED);
			continue;
		}
		Window->m_EditCodigo.SetBackgroundColor(false, GREEN);
		Window->m_EditCodigo.SetWindowText(MSG_CODIGO_PRESENTE);

		if (contagemParada) {
			Window-
>m_EditContagem.SetWindowText(MSG_CONTAGEM_PARADA);
			Window->m_EditContagem.SetBackgroundColor(false, RED);
		}
		else {
			Window->m_EditContagem.SetWindowText(MSG_CONTAGEM_CURSO);
			Window->m_EditContagem.SetBackgroundColor(false, GREEN);
		}

		// Obtemos TU como string formatada; acrescentamos 1 segundo caso seja necessario
		strTU = theTime.Format("%H:%M:%S");
		if (tstruct.millitm > METADE_SEG_EM_MILISEG) {
			// Entao arredonda o tempo somando um segundo e ignorando milissegundos
			int hora, minuto, segundo;
			char mili[TAM_TEMPO_FORMATADO];
			LPTSTR p = strTU.GetBuffer(TAM_TEMPO_FORMATADO);
			sscanf(p, "%02d:%02d:%02d", &hora, &minuto, &segundo);
			segundo++;
			if (segundo == 60) {
				segundo = 0;
				minuto++;
				if (minuto == 60) {
					minuto = 0;
					hora++;
					hora %= 24;
				}
			}
			sprintf(mili, "%02d:%02d:%02d", hora, minuto, segundo);
			strTU = CString(mili);
		}

		// Obtemos H0 como string formatada
		strH0.Format("%02d:%02d:%02d", tmH0.Hour, tmH0.Min, tmH0.Sec);
```

```
                    // Obtemos TD a partir de TU, H0 e do sinal de TD
                    if (tmTD.signal) {
                            CalculaTD(&tmTD, strTU, strH0, true);
                            strTD.Format("%c%02d:%02d:%02d",  SINAL_MAIS_CHAR,  tmTD.Hour,
tmTD.Min, tmTD.Sec);
                    }
                    else {
                            CalculaTD(&tmTD, strTU, strH0, false);
                            strTD.Format("%c%02d:%02d:%02d", SINAL_MENOS_CHAR, tmTD.Hour,
tmTD.Min, tmTD.Sec);
                    }

                    // Atualizamos TU a partir do relogio do sistema (arredondado se necessario)
                    Window->m_EditTU.SetWindowText(strTU);

                    // Determina tipo da mensagem pelo seu tamanho
                    if (bytes == TAM_MSG_COM_TOP) {
                            // Entao mensagem contem informacao de TOP na posicao final
                            strTOP.Format("%02x:%02x:%02x:%03x",      tmTOP.Hour,      tmTOP.Min,
tmTOP.Sec, tmTOP.ms);
                            Window->m_EditTOP.SetWindowText(strTOP);
                            Window->m_EditTD.SetWindowText(strTD);
                            Window->m_EditH0.SetWindowText(strH0);
                    }
                    else if (bytes != TAM_MSG_TOP) {
                            // Entao mensagem nao contem informacao de TOP
                            Window->m_EditTD.SetWindowText(strTD);
                            Window->m_EditH0.SetWindowText(strH0);
                    }
                    else {
                            // Entao mensagem soh contem TOP na posicao inicial (posicao de H0)
                            strTOP.Format("%02x:%02x:%02x:%03x",              buffer[POS_HORA_H0],
buffer[POS_MIN_H0],      buffer[POS_SEG_H0],      ((buffer[POS_SINAL_TD]      <<      8)      |
buffer[POS_COD_PRES]));
                            Window->m_EditTOP.SetWindowText(strTOP);
                    }
            }

            return 0;
}


DWORD WINAPI InterruptThread(CMTR100Dlg *Window)
{
            HANDLE hTimer = NULL;
            LARGE_INTEGER liDueTime;

            // Especifica um intervalo de 2 segundos
    liDueTime.QuadPart = -20000000;
            // Configura um objeto representando um timer de 2 segundos
    hTimer = CreateWaitableTimer(NULL, TRUE, "WaitableTimer");

            // Programa o timer e o espera vencer pois aguardamos inicializacao do programa
    SetWaitableTimer(hTimer, &liDueTime, 0, NULL, NULL, 0);
    WaitForSingleObject(hTimer, INFINITE);

            // Este algoritmo tem como objetivo detectar ausencia de recebimento de
            // mensagem em um intervalo de 2 segundos e tambem detectar o restabelecimento
            // de recebimento de mensagens apos ausencia de recebimento
            while (1) {
```

```
                    WaitForSingleObject(hMutexMudouPorta,INFINITE);
                    mudouPorta = 0;
                    ReleaseMutex(hMutexMudouPorta);

                    while (recebeuMsg) {
                            //Evitando acesso multiplo a recursos...
                            WaitForSingleObject(hMutexRecebeuMsg,INFINITE);
                            recebeuMsg = 0;
                            ReleaseMutex(hMutexRecebeuMsg);

                            // Pausa por 2 segundos
                            SetWaitableTimer(hTimer, &liDueTime, 0, NULL, NULL, 0);
                            WaitForSingleObject(hTimer, INFINITE);
                    }

                    if (!mudouPorta) {
                            AfxMessageBox("ATENCAO: A conexão com o servidor está rompida!",
MB_OK | MB_APPLMODAL | MB_ICONSTOP);
                    }

                    while (!recebeuMsg) {
                            if (mudouPorta) {
                                    WaitForSingleObject(hMutexMudouPorta,INFINITE);
                                    mudouPorta = 0;
                                    ReleaseMutex(hMutexMudouPorta);
                                    // Pausa por 1 segundo
                                    liDueTime.QuadPart = -10000000;
                            SetWaitableTimer(hTimer, &liDueTime, 0, NULL, NULL, 0);
                                    WaitForSingleObject(hTimer, INFINITE);
                               liDueTime.QuadPart = -20000000;
                                    if (recebeuMsg) {
                                            break;
                                    }
                                    AfxMessageBox("ATENCAO: A conexão com o servidor está
rompida! Está utilizando a porta correta?", MB_OK | MB_APPLMODAL | MB_ICONSTOP);
                            }
                            // Pausa por 2 segundos
                    SetWaitableTimer(hTimer, &liDueTime, 0, NULL, NULL, 0);
                            WaitForSingleObject(hTimer, INFINITE);
                    }

                    AfxMessageBox("ATENCAO: A conexão com o servidor foi restabelecida!", MB_OK |
MB_APPLMODAL | MB_ICONEXCLAMATION);
            }

            return 0;
}

void CalculaTD(TD_CALCULADO *tdTime, CString &strTempo1, CString &strTempo2, bool sinal)
{
            int horaTU, minTU, segTU, horaH0, minH0, segH0;
            LPTSTR pTU, pH0;

            char str[256];

            if (sinal) {
                    pTU = strTempo1.GetBuffer(TAM_TEMPO_FORMATADO);
                    pH0 = strTempo2.GetBuffer(TAM_TEMPO_FORMATADO);
            } else {
                    pH0 = strTempo1.GetBuffer(TAM_TEMPO_FORMATADO);
```

```
                pTU = strTempo2.GetBuffer(TAM_TEMPO_FORMATADO);
        }

        sscanf(pTU, "%02d:%02d:%02d", &horaTU, &minTU, &segTU);
        sscanf(pH0, "%02d:%02d:%02d", &horaH0, &minH0, &segH0);

        // Caso TD seja positivo: TD = TU - H0, pois strTempo1 = strTU e strTempo2 = strH0
        // Caso TD seja negativc: TD = H0 - TU, pois strTempo1 = strH0 e strTempo2 = strTU
        tdTime->Hour = horaTU - horaH0;
        tdTime->Min  = minTU  - minH0;
        tdTime->Sec  = segTU  - segH0;

        if (tdTime->Sec < 0) {
                tdTime->Sec += 60;
                tdTime->Min--;
        }
        if (tdTime->Min < 0) {
                tdTime->Min += 60;
                tdTime->Hour--;
        }

        if (!sinal) {
                if (horaH0 < horaTU) {
                        tdTime->Hour += 24;
                }
        }


   sprintf(str, "%d", horaTU);
   AfxMessageBox(str, MB_OK | MB_APPLMODAL | MB_ICONEXCLAMATION);

   sprintf(str, "%d", horaH0);
   AfxMessageBox(str, MB_OK | MB_APPLMODAL | MB_ICONEXCLAMATION);

}
```

Belo Horizonte, 13 de Julho de 2004.

_____

André Goddard Rosa

_____

Mariza Andrade da Silva Bigonha