UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO MESQUITA FILHO"
FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA – DEE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

VITAL: SISTEMA MICROCONTROLADO PARA SUPERVISIONAR O CONTROLE DE ACESSO A LABORATÓRIOS INTERDISCIPLINARES

Dissertação de Mestrado submetida à Faculdade de Engenharia de Ilha Solteira Universidade Estadual Paulista - UNESP, como parte dos requisitos necessários para obtenção do título de Mestre em Engenharia Elétrica.

Tecnólogo André Luis Scagnolato
Autor

Orientador: Eng. Eletr. Alexandre César Rodrigues da Silva – FEIS / UNESP
Co-Orientador: Prof. Dr. Ricardo Tokio Higuti – FEIS / UNESP

ILHA SOLTEIRA SP
DEZEMBRO DE 2005

FICHA CATALOGRÁFICA

Elaborada pela Seção Técnica de Aquisição e Tratamento da Informação/Serviço Técnico
de Biblioteca e Documentação da UNESP-Ilha Solteira

Scagnolato, André Luis
S278v Vital : sistema microcontrolado para supervisionar o controle de acesso a laboratórios interdisci-
plinares / André Luis Scagnolato. -- Ilha Solteira : [s.n.], 2005
142 p. : il.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha
Solteira, 2005

Orientador: Alexandre César Rodrigues da Silva
Co-orientador: Ricardo Tokio Higuti
Bibliografia: p. 120-122

1. Ibutton. 2. Computadores - Controle de acesso. 3. Banco de dados distribuído. 4. Delphi
(Programa de computador). 5. Microcontrolador.

unesp

UNIVERSIDADE ESTADUAL PAULISTA
CAMPUS DE ILHA SOLTEIRA
FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA

CERTIFICADO DE APROVAÇÃO

TÍTULO: VITAL: Sistema Microcontrolado para Supervisionar o Controle de Acesso a Laboratórios Interdisciplinares

AUTOR: ANDRE LUIS SCAGNOLATO
ORIENTADOR: Prof. Dr. ALEXANDRE CESAR RODRIGUES DA SILVA

Aprovado como parte das exigências para obtenção do Título de MESTRE em ENGENHARIA ELÉTRICA pela Comissão Examinadora:

Prof. Dr. ALEXANDRE CESAR RODRIGUES DA SILVA
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Profa. Dra. MARIA DO CARMO G DA SILVEIRA
Analista de Informática - Polo Computacional da FE - UNESP - Campus de Ilha Solteira

Prof. Dr. DELCIO CARDIM
Departamento de Ciências da Computação - Faculdade Adamantinenses Integradas - FAI - Adamantina/SP

Data da realização: 22 de dezembro de 2005.

Presidente da Comissão Examinadora
Prof. Dr. ALEXANDRE CESAR RODRIGUES DA SILVA

Dedico esse trabalho a minha mãe Helena, a meu pai Antonio e a minha irmã Érica, pelo apoio e incentivo dados durante o decorrer desse projeto.

AGRADECIMENTOS

Ao meu orientador Professor Alexandre César Rodrigues da Silva, pela dedicação, atenção e orientação durante esse período, onde jamais deixou de transmitir seus conhecimentos e ensinamentos para a conquista dessa pesquisa, sempre acreditando em meu trabalho.

Ao meu cunhado José Aparecido de Aguiar Viana, pelo incentivo e ajuda nos momentos difíceis.

Ao amigo Robinson Ledesma, pela ajuda e pelo apoio constante no decorrer desse trabalho.

Aos amigos de laboratório Carlos Eduardo da Silva Santos e Tércio Alberto dos Santos Filho pela amizade, ajuda e pelos momentos de descontração.

Aos amigos João Marcos, Giorjety Dias, Silvano Renato Rossi, Ricardo Ferraz e Thiago Prado pela paciência e pelo apoio constante.

Aos professores e funcionários do Departamento de Engenharia Elétrica, sempre prestativos e atenciosos.

RESUMO

Apresenta-se nesse trabalho um sistema para o controle de acesso de pessoas as dependências de laboratórios interdisciplinares, utilizando o *iButton* como chave eletrônica. O crescente aumento do índice de furtos dentro de instituições de ensino tem motivado o desenvolvimento de novos métodos e ferramentas para inibir ou até mesmo coibir tal ato, servindo como forma de motivação para inúmeros trabalhos tanto no âmbito acadêmico como comercial. O *iButton* apresenta características únicas que o definem como um meio de armazenamento de informações barato, seguro e eficiente, dispensando o uso da chave tradicional. O software, denominado de VITAL (Vigilância Integrada Total aos Alunos do Laboratório), foi implementado em ambiente visual *Delphi* versão 7.0, sendo sua base de dados desenvolvida em *SQL* e todo o seu gerenciamento realizado pelo *Interbase*, um poderoso sistema gerenciador de banco de dados amplamente utilizado nos dias atuais. O *software* é composto por módulos independentes que controlam a entrada e saída dos usuários as dependências de determinado laboratório de pesquisa, comunicando-se com o sistema com apenas um toque do *iButton* no dispositivo de leitura. Utiliza-se para a obtenção do número de série do *iButton* um microcontrolador *Motorola MC68HC908*. Todos os usuários previamente autorizados a utilizar as dependências do laboratório, são cadastrados no banco de dados, bem como todas as transações realizadas por cada usuário. O sistema permite a emissão de vários tipos de relatórios para fins de auditoria e o envio de e-mail ao administrador em caso de tentativa de acesso indevido, com foto em anexo, obtida através de uma câmera localizada no lado externo do laboratório. O projeto disponibiliza para o administrador do sistema, a consulta de usuários que se encontram no laboratório via *WEB*, mediante o fornecimento de um nome de usuário e senha de acesso.

Palavras Chave: *iButton*. banco de dados. *SQL*. *Delphi*. *Interbase*. sistema de controle de acesso. microcontrolador. M68HC908.

ABSTRACT

This paper presents an access control system to people into the university laboratories, using the iButton as an electronic lock. The growing rate of theft at universities motivate the development of new methods and tools to inhibit or even avoid such intrusions, as a motivation for numbers of assignments both in academic and business environment. The IButton presents unique characteristics that define it as a non-expensive, safe and efficient database resource, in which has no need of traditional keys. This software denominated of VITAL (Integrated Vigilance to the Students of Laboratory) was implemented in Delphi language, version 7.0, its database was developed in SQL and its whole management was done by Interbase, a powerful database management largely used nowadays. The system is composed of independent modules which control the entry of users into a certain university laboratory, communicating to the system just with a touch of the IButton in the reading device. It is used for the obtaining of the number of series of the iButton a Motorola MC68HC908 microcontroller. All users, priory authorized, are recorded on the database system, as well as all transactions made by each one of them. The system allows varies reports issues to be used in audit, and the e-mail sending to the administrator in case of attempt of improper access, with picture enclosed, obtained through a located camera on the external side of the laboratory. The project available for the administrator of the system, the users' consultation that you they are at the laboratory through WEB, by the supply of an user name and access password.

Keywords: iButton. database. SQL. Delphi. Interbase. access control system.microcontroller.MC68HC908.

LISTA DE ABREVIATURAS E SIGLAS

Assíncrono: Forma de transmissão de dados na qual os dados são enviados caractere a caractere, com intervalos de tempo variáveis entre cada transmissão.

Bits: Forma reduzida de *binary digit* (dígito binário); o zero ou o um do sistema binário de numeração.

Bugs: Um erro na codificação ou na lógica que faz com que um programa não funcione corretamente ou que produza resultados incorretos.

CRC: Acrônimo de *cyclical (ou cyclic) redundancy check*. Um procedimento utilizado para verificar a ocorrência de erros na transmissão de dados. A verificação de erros com a técnica de *CRC* emprega cálculos complexos para gerar um número baseado nos dados transmitidos. O equipamento emissor faz cálculos antes da transmissão. O equipamento receptor repete o mesmo cálculo depois da transmissão. Se ambos chegarem ao mesmo resultado, o sistema considerará que a transmissão foi efetuada sem erros. O procedimento é conhecido como teste de redundância cíclica, porque cada transmissão, além dos dados, contém valores extras (redundantes) usados especificamente para a verificação de erros.

CMOS: Acrônimo de *complementary metal-oxide semiconductor*. O *CMOS* é um dispositivo semicondutor formado por dois transistores de efeito de campo (*MOSFETs*) baseados em óxido metálico: um do tipo N e outro do tipo P, integrados em um mesmo *chip* de silício. Geralmente usados na memória de acesso aleatório (*RAM*) e em aplicações de comutação, esses dispositivos operam em alta velocidade e consomem muito pouca energia elétrica. No entanto, eles são extremamente suscetíveis a danos provocados pela eletricidade estática.

Drivers: Um dispositivo de *hardware* ou programa que controla ou regula outro dispositivo. Os *drivers* de dispositivo são programas de controle específicos para cada dispositivo, cuja finalidade é permitir que o computador consiga utilizá-los; por exemplo, existem *drivers* para impressoras e unidades de disco.

E-Business: Modalidade de comércio nas quais as transações são feitas através da *Internet*.

EPROM: Acrônimo de ***e****rasable* ***p****rogrammable* ***r****ead-****o****nly* ***m****emory*. Um *chip* de memória não-volátil programado em uma etapa posterior à sua fabricação. Os *EPROMs* podem ser reprogramados removendo-se o revestimento protetor da superfície do *chip* e expondo-se o *chip* à luz ultravioleta. Apesar de os *EPROMs* serem mais caros do que os *chip*s *PROM*, eles podem ter uma relação custo/benefício mais interessante se forem necessárias muitas alterações.

EEPROM: Acrônimo de *electrically erasable programmable read-only memory*. Uma variação de *EPROM* que pode ser apagada através da aplicação de um sinal elétrico a um ou mais pinos. Esse tipo de *chip* é conveniente em aplicações que exigem uma memória estável durante longos períodos de tempo sem energia, mas que também possa ser reprogramada.

Half-Duplex: Comunicação eletrônica bidirecional que ocorre em apenas uma direção de cada vez.

Kbit: Abreviado como *Kb* ou *Kbit*. Unidade de dados igual a 1.024 *bits*.

Kbps: Abreviado como *Kbps*. Velocidade de transferência de dados – em uma rede, por exemplo – medida em múltiplos de 1.024 *bits* por segundo.

NVRAM: *(Nonvolatile Memory) Memória Não Volátil* - Sistema de armazenamento que não perde os dados quando a energia é desligada. Normalmente associado a núcleos magnéticos, *ROM*, *EPROM*, memória *flash*.

POST: Conjunto de rotinas armazenadas na memória de leitura (ROM) do computador, cuja finalidade é testar componentes do sistema, como a RAM, as unidades de disco e o teclado, para ver se estão conectados corretamente e aptos a funcionar.

PROM: Acrônimo de ***p****rogrammable* ***r****ead-****o****nly* ***m****emory*. Tipo de memória de leitura (*ROM*) que permite a gravação de dados com um dispositivo chamado *PROM programmer*

(programador de *PROM*). Depois que um *chip* de *PROM* é programado, seu conteúdo não pode ser mais modificado.

Polipropileno: Polímero utilizado para isolar condutores de correntes elétricas de alta freqüência.

RAM: Acrônimo de *random access memory*. Memória baseada em *chip*s semicondutores que podem ser lidos e gravados (escritos) pelo microprocessador e outros dispositivos. As posições de armazenamento de dados na *RAM* podem ser acessadas em qualquer ordem.

ROM: Acrônimo de *read-only memory*. Memória baseada em *chips* semicondutores que contêm instruções e dados, cujo conteúdo pode ser lido, mas não modificado.

SRAM: *(Static RAM)* Uma forma de *RAM* baseada em um circuito lógico conhecido como *flip-flop*. As *RAMs* estáticas costumam ser reservadas para *caches*.

Scratch Pad: Uma área de armazenamento temporário usada por um programa ou sistema operacional para cálculos, dados e outros trabalhos em andamento.

Threads: Na programação, consiste em um processo que faz parte de um processo maior ou programa. Em uma estrutura de dados em forma de árvore, um ponteiro que identifica o nó imediatamente superior (pai), sendo usado para facilitar o percurso da árvore.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

CAPÍTULO 1 - Introdução

1.1 Motivação

O acesso por alunos a determinados setores dentro de uma universidade, ou até mesmo a entrada e saída de funcionários em setores restritos dentro de uma empresa sempre foi uma questão problemática enfrentada por esses setores. O aumento da incidência de roubos e furtos tornou a questão de segurança um assunto abordado em diversos trabalhos, tanto no meio acadêmico como em aplicações comerciais [1], [2], [3], [4], [5], [6] e [7].

Pode-se citar como exemplo, eventos ocorridos na Faculdade de Engenharia de Ilha Solteira – UNESP durante os últimos anos, onde ocorrências de furtos têm sido registradas constantemente pelo setor de vigilância, como o furto de projetores multimídia, componentes eletrônicos dos laboratórios de ensino prático, *laptops*, etc.

A fragilidade do sistema interno de segurança e a ausência de um sistema de controle possibilitaram a crescente demanda do número de furtos, ocasionando enormes prejuízos para a instituição e para os usuários.

Os conceitos de controle e segurança são itens fundamentais para o bem estar de uma instituição e devem ser avaliados e estruturados de forma a não apresentarem falhas e operarem de maneira eficiente e interativa.

1.2 Descrição do Sistema

Neste trabalho, desenvolveu-se um sistema de controle de acesso com emprego de *iButton*, utilizando um microcomputador para realizar o controle de entrada e saída dos acadêmicos, um servidor de banco de dados e um microcontrolador, proporcionando um gerenciamento mais eficiente no acesso aos laboratórios de pesquisa e ensino.

O sistema aborda alguns princípios fundamentais de segurança necessários para o controle de acesso de pessoas a locais restritos, possibilitando o controle individual de cada usuário, sendo possível atribuir permissões individuais para cada um, como o dia da semana, horário, período e laboratório permitido. O *software* dispõe de módulos independentes que realizam tarefas distintas no aspecto de gerenciamento, auditoria e no controle de entrada e saída de usuários, proporcionando uma maior segurança e integridade dos dados.

Para o desenvolvimento do sistema, foi utilizado o ambiente de programação visual orientado a objetos *Borland Delphi* [8] versão 7.0. Um ambiente visual *Delphi* proporciona maior praticidade do programador durante o desenvolvimento do sistema, pois além de possuir a qualidade do ambiente de desenvolvimento visual, possui a flexibilidade e a capacidade de redimensionamento da arquitetura de seu banco de dados [9].

O *software* utiliza o gerenciador de banco de dados relacional *Interbase* [10], o qual se encontra disponível para *download* gratuitamente na *Internet*. Com o uso do *Interbase*, é possível o desenvolvimento de aplicações em *Delphi* utilizando *SQL (Structured Query Language)* [11], ou linguagem de consulta estruturada, sem a necessidade da aquisição de sistemas gerenciadores de banco de dados mais caros. O *SQL* baseia-se em uma linguagem de consulta constantemente utilizada em sistemas de clientes / servidores, para a realização de pesquisas em bancos de dados existentes. O uso do *SQL* torna o acesso à base de dados do sistema mais ágil e dinâmica, proporcionando assim, uma maior velocidade de acesso ao sistema, uma vez que o banco de dados tem seu tamanho constantemente alterado.

O *software* desenvolvido interage com o componente *iButton*, abordado no capítulo II, com mais de 27 milhões de unidades em circulação [12], e já vem mostrando superioridade em relação aos demais meios de identificação utilizados atualmente.

O *iButton* possui alta durabilidade e resistência, podendo sofrer quedas, arranhões e até mesmo o contato com a água, sendo amplamente utilizado em diversos segmentos do

mercado, como residências, prédios comerciais, veículos, etc, substituindo o uso da chave convencional.

Atualmente no mercado existem inúmeras tecnologias de armazenamento de informações, como o cartão magnético, código de barras, etc. O *iButton* apresenta características únicas que o definem como um meio de armazenamento de informação seguro, barato, durável e eficiente.

1.3 Trabalhos Relevantes na Área

1.3.1 Um Novo Sistema de Controle de Acesso para Laboratórios Interdisciplinares

Um modelo de sistema de controle de acesso similar foi utilizado no Campus da Faculdade de Engenharia Elétrica – UNESP [1], laboratório de Processamento de Sinais e Sistemas Digitais. Esse sistema realizava o controle de entrada e saída de alunos no laboratório, utilizando para isso o *iButton* no lugar da chave convencional, um leitor denominado *Blue Dot*, localizado no lado externo e interno da porta para realizar a leitura dos *iButton´s*, e um microcomputador *IBM* no qual se encontra o *software* desenvolvido em linguagem C, responsável pelo controle de entrada e saída dos alunos. Caso o usuário estivesse autorizado, o sistema enviava um sinal através da interface de comunicação paralela do microcomputador até um fecho eletromagnético, liberando o seu acesso e registrando sua entrada ou saída no sistema. Não sendo validado o *iButton* pelo sistema, o mesmo emitia um sinal sonoro e registra internamente em seu arquivo de auditoria a tentativa de acesso indevido. Nesta dissertação de mestrado o sistema foi aprimorado com a inserção de novos recursos e ferramentas, como o controle de permissões individuais, acessos a vários laboratórios, relatórios de auditoria e procedimentos para a detecção de tentativas de acessos indevidos, entre outros, apresentados no capítulo 4.

1.3.2 Sistema de Control de Aceso com iButton

O sistema de controle de acesso com *iButton* [2], desenvolvido na Universidade Javeriana, Colômbia, utiliza o *iButton* como chave de identificação. O sistema é dividido em três partes distintas: o módulo de acesso, responsável em realizar a validação dos usuários para permitir a sua entrada às dependências, o servidor de gestão, cuja função é registrar todos os eventos ocorridos no sistema e um protocolo de comunicação, responsável em realizar a comunicação do módulo de acesso com o servidor de gestão.

O sistema conta com um limite máximo de até 255 módulos autônomos, utilizados na validação dos usuários automatizando o processo de entrada e saída, onde quer que seja instalado. O módulo possui a capacidade de armazenar até 1440 números de *iButton´s*, utilizando para isso um microcontrolador MC68HC908. De um ponto central (servidor), é possível realizar a administração e a comunicação de até 255 módulos de acesso remoto.

O programa servidor do sistema foi desenvolvido em *Delphi* versão 4, o qual se comunica com os módulos através de um protocolo de comunicação específico, onde todas as informações relacionadas ao sistema, como as salas, os usuários e os eventos, são armazenadas em uma base de dados.

1.3.3 Sistema de Controle de Acesso por *iButton* com Verificação Biométrica por Geometria da Mão.

O Sistema de Controle de Acesso por *iButton* com Verificação Biométrica por Geometria da Mão[3] foi desenvolvido no Centro Universitário Positivo – UNICEMP, Curitiba PR, o qual consiste em um sistema de controle de acesso baseado em uma chave eletrônica (*iButton*) e uma chave biométrica, formada por primitivas geométrica extraídas a partir de imagens obtidas via um *scanner* tradicional. O sistema apresenta uma técnica de extração de primitivas a partir de uma análise do perfil de curvatura da imagem que é

invariante a rotação e a translação, não exigindo o uso de pinos ou dispositivos restritos a posição da mão no equipamento de aquisição de dados. Trata-se de um sistema muito seguro e eficiente, uma vez que a possibilidade de uma pessoa se passar por outra é bastante difícil, pois além de possuir o *iButton*, o usuário necessitaria ainda possuir uma geometria de mão compatível.

1.3.4 Sistema de Controle de Acesso Acadêmico

O sistema de controle de acesso acadêmico [4], desenvolvido no Colégio de *Canyons*, sul da Califórnia, tem por finalidade controlar o acesso de alunos as salas de aula e laboratório, através de um computador administrador, que tem a função de gerenciar as fechaduras de acesso e auditoria.

A reprogramação das fechaduras é realizada com a utilização de um *PDA* ou *laptop*, onde através dele é possível implantar ou apagar dados das credenciais eletrônicas, inserirem novos pontos de acesso e novos usuários. O sistema possui a flexibilidade de se utilizar qualquer tipo de identificação eletrônica (*iButton*, cartão magnético, cartão de proximidade, etc).

1.3.5 Controle de Acesso Eletrônico

O Sistema de Controle de Acesso Eletrônico [5], desenvolvido pela empresa *LockLink,* consiste em uma fechadura *StandAlone,* microcontrolada e eletromecânica, que utiliza o *iButton* como meio de identificação. A fechadura é de fácil instalação e funciona com 4 baterias do tamanho AA, o que possibilita seu acionamento em torno de 80.000 vezes, sendo que o módulo possui uma capacidade de armazenamento de 1000 usuários e 1000 eventos.

Operacionalmente, a alavanca da fechadura é travada no lado externo e liberada no lado interno, sendo livre o egresso do usuário. As configurações e auditorias são realizadas através do *software Express LockLink*, sendo possível a sua instalação em qualquer sistema

operacional da família *Microsoft*. Através do *software* é possível realizar o cadastro de novos usuários, pontos de acesso e restrições individuais.

1.3.6 Orion Segurança – Sistema Ronda

O sistema Ronda [6], comercializado pela empresa Orion Segurança, foi desenvolvido para realizar a supervisão dos vigilantes em suas rondas, utilizando o *iButton* como meio de identificação. O sistema vem pré-programado de fábrica para um conjunto de pontos de ronda, com horários pré-estabelecidos. Com o auxílio de um microcomputador, é possível alterar a programação, definindo os horários e pontos de ronda, cadastrando funcionários e eventos, permitindo assim, um controle mais avançado das atividades dos vigilantes. O programa que acompanha o *hardware* permite a emissão de diversos tipos de relatórios, possibilitando assim, uma auditoria eficiente nas atividades dos vigilantes. Cada funcionário é identificado através de um toque do *iButton* que encontra-se fixo em seu crachá no dispositivo de leitura.

1.3.7 Sistema de Gerência de Segurança de Acesso Físico (SGS)

O sistema de gerência de segurança de acesso físico (SGS) [7] foi desenvolvido para uso comercial, fornecido pela empresa ATI Telecomunicações.

O SGS realiza o controle de acesso dos usuários, às instalações (prédios, salas, armários), podendo ser utilizado como meio de identificação o *iButton*, o cartão de proximidade, ou o *SmartCard*. O sistema é composto por três componentes principais: o servidor, as unidades remotas e o identificador do usuário. Cada usuário deve estar cadastrado no banco de dados do servidor. Quando um usuário cadastrado deseja entrar em uma instalação, ele introduz o seu identificador na unidade leitora, que após a validação positiva dos dados realiza a abertura da porta através do acionamento da fechadura eletrônica da porta.

1.4 Descrição do Sistema Integrado

O sistema desenvolvido nesse trabalho provê um controle de acesso empregando o *iButton* como meio de identificação individual para cada usuário, utilizando um microcontrolador *Motorola MC68HC908* para realizar a leitura do número de série do *iButton*.

O sistema dispõe de um servidor de banco de dados central, possibilitando o controle de vários laboratórios simultaneamente, onde cada laboratório possui um grupo individual de usuários com permissões de acessos definidas individualmente pelo administrador do laboratório.

Para cada laboratório, existe um administrador, este, definido pelo administrador geral do sistema, que pode manipular e gerenciar seu grupo de usuários da maneira que considerar mais conveniente, atribuindo permissões de acesso individuais para cada um.

A segurança proporcionada nesse sistema inclui desde a emissão de *e-mail´s* ao administrador do laboratório para cada tentativa de acesso indevida, como por exemplo, um usuário que tentou realizar o acesso fora de seu perfil de permissões, ou até mesmo o acesso a um laboratório o qual não possui permissão, como a captura da imagem do usuário por uma *webcam* instalada no lado externo do laboratório, anexando essa imagem no e-mail e enviado-a ao administrador de laboratório.

É possível ainda, verificar a lista de usuários que estão utilizando o laboratório, via *Internet*, além dos diversos relatórios de auditoria disponíveis no sistema. No decorrer dos próximos capítulos, serão descritos todos os materiais e métodos utilizados para o desenvolvimento do sistema.

CAPÍTULO 2 - *iButton*

2.1 Introdução

O *iButton* [12] mostrado na Figura 2.1, consiste em um pastilha *(chip)* de silício armazenado em uma cápsula de metal confeccionada em aço inoxidável com um diâmetro de 16 mm, chamado de *MicroCan*, mostrado na Figura 2.2.

Figura 2.1: *iButton*.
Fonte: <http://www.ibutton.com/ibuttons.index.html>

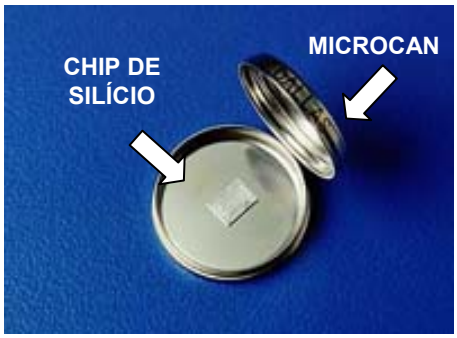


Figura 2.2: Cápsula aberta.
Fonte: <http://www.ibutton.com/ibuttons.index.html >

Com o emprego do *iButton*, as informações podem viajar com pessoas ou objetos, pois o seu encapsulamento é bastante resistente para suportar diversos ambientes, podendo ser usado como um acessório digital no dia a dia.

As funções lógicas dos *iButton´s* [13] vão desde um simples número de série de 64 *bits* de *RAM* não volátil ou *EEPROM* até relógio em tempo real e sensor de temperatura.

Existem *iButton´s* com diferentes características, sendo que todos os modelos possuem um número de série único *(ROM Registration Number)* e uma outra identificação chamada *Part Number* (número da família) que designa o tipo de *iButton*.

2.2 Características Físicas

O encapsulamento de metal *(MicroCan)* do *iButton* [13] possui 16 mm de diâmetro, possuindo dois tipos de espessura: 3,1 mm e 5,9 mm, designados por F3 e F5, respectivamente, mostrados na Figura 2.3. A versão F5 é padrão para todos os dispositivos que contém uma fonte de energia interna.

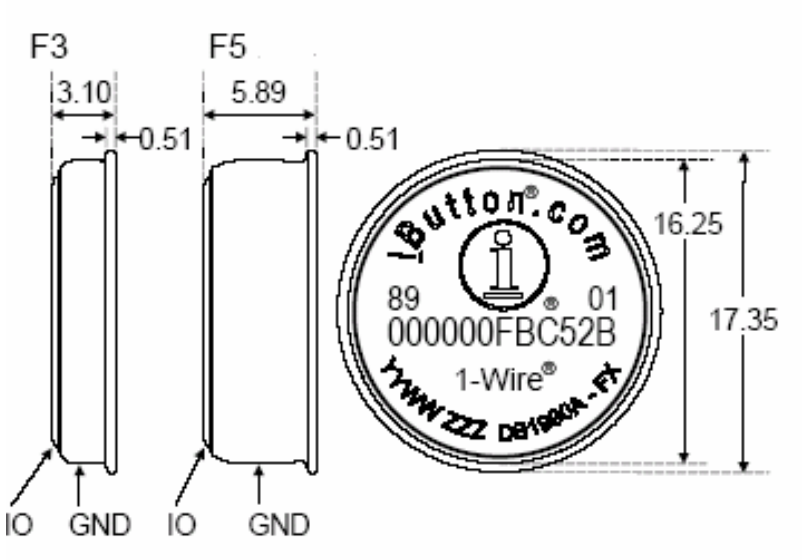


Figura 2.3: Medidas dos *iButton´s*.
Fonte: <http://www.maxim-ic.com/appnotes.cfm/appnote_number/937>

A espessura do aço da cápsula *(MicroCan)* é de 0,254 mm, sendo dividido em duas partes: o contato da parte superior, e o contato da parte inferior, ambos separados por um anel de polipropileno preto, que serve para evitar que as duas partes tenham contato entre si, servindo também para inibir os raios ultra violetas. Esse encapsulamento de metal é utilizado tanto para proteger o *chip* de silício interno, como para realizar a comunicação com o sistema, proporcionando também uma excelente estabilidade mecânica e ótima resistência à corrosão.

Todas as *MicroCan´s* resistem a choques mecânicos de 500g (1g = 9,81 m/s^2) em qualquer direção. Uma queda de 1,5 m de altura em uma superfície de concreto não danifica o seu encapsulamento nem o seu conteúdo [13].

O encapsulamento do *iButton* possui inúmeras informações, mostradas na Figura 2.4, sendo:

- YYWW: ano e semana de fabricação do *iButton*;
- CC: CRC *(Cyclic Redudancy Check)*;
- SSSSSSSSSSSS: número de série hexadecimal de 12 dígitos;
- RR: Revisão da embalagem;
- FF: código da família do *iButton*;
- ZZZZ: parte de número genérico;
- NNN: 001 até FFF para código padrão;
- FX: *MicroCan* padrão F3 ou F5.

Figura 2.4: Descrição das informações gravadas na parte superior do *iButton*.

2.3 Tecnologia

A energia necessária para o funcionamento do *iButton* é fornecida pelo sistema *Parasite Power* [14]. O *chip* dentro da cápsula utiliza a tecnologia *CMOS* e consome corrente de fuga apenas quando está em um estado inativo. Para manter o consumo de energia o mais baixo possível, durante o período ativo, e para ser compatível com as famílias existentes, uma linha

de dados do *iButton* é designada como uma saída em dreno aberta. A fonte de corrente da linha de dados para o terra, somente é retornada se o *iButton* for removido do meio de comunicação. A interface de dreno aberto faz com que os *iButton´s* sejam compatíveis com todos os microprocessadores e sistemas com lógica padrão. Em um meio *CMOS*, é necessário apenas um resistor de 5kΩ para 5 V_{DD} para entrar em condições normais de operação em uma porta bidirecional de dreno aberto. Na Figura 2.5 apresenta-se um diagrama de blocos do *iButton* internamente.

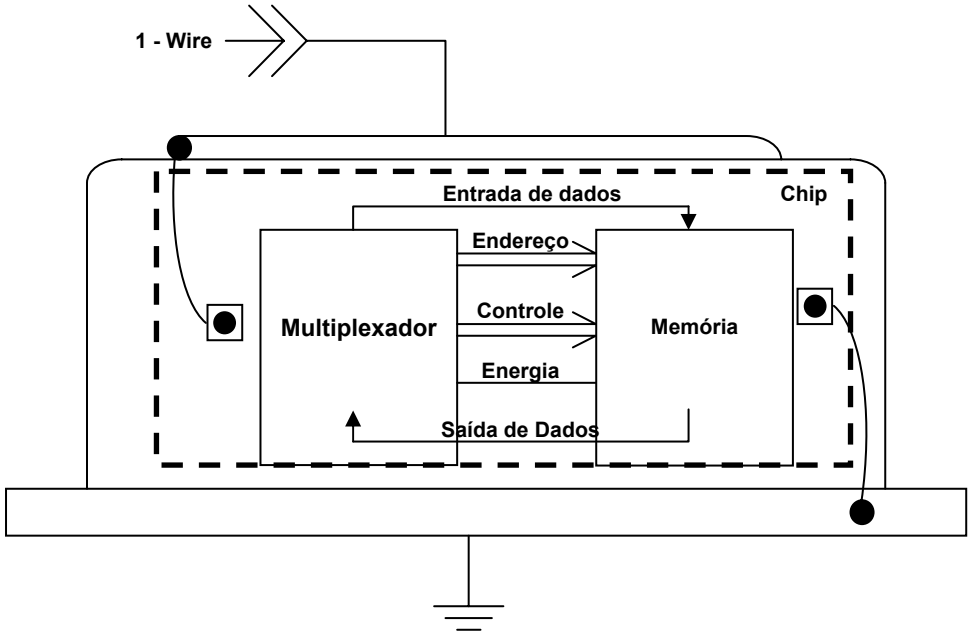


Figura 2.5: Diagrama de blocos do *iButton.*

2.4 Número de ROM

Todos os *iButton´s* [14] possuem uma memória *ROM* única, a qual é gravada e testada em fábrica. Essa memória possui 64 *bits*, sendo que os 8 primeiros *bits* identificam a família do *iButton*. Os próximos 48 *bits* identificam o número de série único e os 8 últimos *bits* constituem o chamado *CRC (Cylic Redundancy Check)* dos 56 *bits* anteriores. Na Tabela 2.1 apresenta-se como os *bits* estão organizados.

Tabela 2.1 : Divisão da *ROM* contida nos *iButton´s*.

64-BIT ROM - Número de Registro					
8-BIT CRC	Identificador Único			Número de Série do Dispositivo	Código da Família
	5	E	7	MSB (Formato Hexadecimal)	91
1 Byte	12 Bits			36 Bits	1 Byte

O *CRC* da memória *ROM* é gerado utilizando o polinômio $X^8 + X^5 + X^4 + 1$. O registrador operando como o acumulador *CRC* é inicializado em 0. Então, começando com o *bit* menos significativo do código da família, um *bit* de cada vez é armazenado. Depois que o 8º *bit* do código de família for registrado, o número de série é armazenado. Depois que o 48º número de série for armazenado, o registrador contém o valor *CRC*.

O número de série pode representar qualquer número decimal até $2{,}81 \times 10^{14}$. Se 1000 bilhões (10^{12}) de dispositivos da mesma família forem produzidos por ano, o número de série será suficientemente grande para suprir a demanda de 128 anos. Além disso, existem 128 famílias de *iButton´s* disponíveis [14].

2.5 Comunicação

A comunicação dos *iButton´s* com o microcomputador é realizada através do protocolo *1-Wire* (1 Fio) [15], sendo realizada com apenas um toque do *iButton* na interface de comunicação *1-Wire*. A interface de comunicação permite dois padrões de velocidade de comunicação: o modo *Standard*, na velocidade de 16 *kbps* e o modo *OverDrive*, com 142 *kbps* de velocidade de comunicação. Atualmente existem diversas interfaces [16] de comunicação, como o B*lue Dot* (serial ou paralelo), canetas para leitura, adaptadores seriais, paralelos, *USB*, etc. Na Figura 2.6 apresenta-se esses dispositivos.

Figura 2.6: Interfaces e adaptadores para os *iButton´s*.

Fonte: <http://www.ibutton.com/products/readers.html>

Pode-se operar vários *iButton´s* simultaneamente através da rede para comunicação digital *MicroLan*. Todos os *iButton´s* possuem uma interface para comunicação com essa rede.

2.6 Modelos de iButton´s

Atualmente existem 128 modelos de *iButton´s* [14] para os mais diversos fins. A Tabela 2.2 relaciona os modelos de *iButton´s* e suas principais características.

Tabela 2.2: Tipos de *iButton´s*.

iButton	**Características**
DS1990A – ROM de 64 *Bits*.	Possui apenas o número de registro único de 64 *Bits*. É o mais simples dos *ibutton´s*.
DS1971 – EEPROM de 256 *Bits*. *DS1973 – EEPROM* de 4 *Kbits*.	Memória não volátil, reprogramável para armazenar informações sobre produtos ou pessoas.
DS1963L – iButton monetário.	Memória não volátil de leitura ou escrita de 4096 *bits*; quatro páginas de 32 *Bits*; *chip* de 16 *Bits CRC*; 32
DS1982 – EPROM de 1 *Kbit*. *DS1985 – EPROM* de 16 *Kbits*. *DS1986 – EPROM* de 64 *Kbits.*	*EPROM* que pode ser programada várias vezes acrescentando-se informações.
DS1991 – 1 *Kbit* protegido por senha.	Memória não volátil de 1 *Kbit,* divididas em três áreas protegidas por senha que pode agir como três chaves eletrônicas separadas.
DS1992 – Memória de 1 *Kbit*. *DS1993* – Memória de 4 *Kbits*. *DS1994* – Memória de 4 *Kbits*, com calendário e relógio. *DS1995* – Memória de 16 *Bits*. *DS1996* – Memória de 64 *Kbits*.	*SRAM* (Não Volátil) organizada em páginas de 256 *Bits* com acesso em alta velocidade (modo *OverDrive*) a um registrador (*Scratchpad*) de 256 *Bits*; O DS1994 possui um relógio em tempo real e um calendário.
DS1920 – Termômetro	Termômetro digital e 2 *Bytes* de *EEPROM*.

2.7 Acessórios

O *iButton* possui diversos acessórios [14] de fixação e de uso pessoal. Os acessórios de fixação são usados para colar o *iButton* em uma superfície de um objeto qualquer, através de adesivos ou anéis de fixação. Entre os acessórios de uso pessoal, estão chaveiros, carteiras, anéis, pulseiras de relógio, entre outros, apresentados na Figura 2.7.

Figura 2.7: Chaveiros e adesivos para *iButton´s*.
Fonte: <http://www.ibutton.com/products/ibuttons.html>

2.8 A Linguagem TMEX

O ambiente *TMEX* [17] é um conjunto de *drivers* e programas utilitários que permitem a comunicação dos *iButton´s* e acessórios, utilizando um microcomputador. O *software TMEX* esta disponível gratuitamente na *Internet* no endereço *www.ibutton.com*, em versões para ambiente *MS-DOS* e *Windows*. Após a instalação do *TMEX* e realizadas as devidas configurações (como o número da porta serial) pode-se utilizar os aplicativos já prontos para interagir com os *iButton´s*. Todavia, é necessário utilizar um adaptador serial ou paralelo e uma mídia para a realização da leitura do *iButton*. Os *drivers* e protocolos utilizados da linguagem *TMEX* são apresentados no anexo A.

Para o desenvolvimento do sistema foi utilizado o ambiente de programação *Borland Delphi 7* em conjunto com as bibliotecas *TMEX*, inseridas no contexto do programa fonte. O *Delphi* utiliza uma variação da linguagem de programação *Pascal*, denominada *Object Pascal*, uma linguagem totalmente orientada a objetos.

Com a utilização do *Delphi*, foi possível o desenvolvimento do sistema utilizando-se um banco de dados para armazenamento de todas as informações do sistema. Optou-se pelo sistema gerenciador de banco de dados relacional *Interbase* para administrar e manipular todas as informações contidas no banco de dados, por se tratar de um sistema gerenciador eficiente para aplicações desenvolvidas em *Delphi*, além de ser distribuído gratuitamente na *Internet* até a sua versão 6.5.

As informações armazenadas no banco de dados são manipuladas por comandos *SQL*, uma linguagem de gerenciamento de dados amplamente utilizada nos dias atuais pelos principais sistemas gerenciadores de banco de dados. Descreve-se no próximo capítulo as principais características das ferramentas utilizadas.

CAPÍTULO 3 – Ferramentas Utilizadas para Desenvolvimento do Sistema

3.1 Programação Orientada a Objetos

O paradigma da programação orientada a objetos *(OOP)* [8], é uma metodologia de desenvolvimento de *software* que utiliza o conceito de que tudo no mundo real é objeto.

Essa metodologia modela as características de objetos reais e abstratos através da utilização de classes e objetos. Pode-se dizer que a programação orientada a objetos é uma metodologia de implementação nas quais os programas são organizados como sendo uma coleção de objetos que cooperam entre si, sendo que cada um desses objetos representa uma instância de uma classe, e cujas classes são todas membros de uma hierarquia, unidas pelas relações de herança.

3.2 Elementos da Programação Orientada a Objetos

3.2.1 Objeto

Um objeto é uma entidade [18] que exibe algum comportamento bem definido. Como exemplo informal, imagina-se uma caneta qualquer, essa caneta é um objeto, um exemplo daquilo que se conhece por caneta. Imagina-se uma outra caneta qualquer, novamente, essa outra caneta também é um objeto, outro exemplo de caneta. Portanto, independente do modelo, cor, tamanho, a primeira ou a segunda caneta, ambas são objetos. Em outro exemplo tem-se o seguinte trecho de programa:

```
Program Exemplo;
VAR
        a,b             : Integer;
        resultado       : Integer;
Function Somar (N1, N2 : Integer): Integer;
Begin
        Somar := N1 + N2;
End;
Begin
        {obtém a e b}
        {soma a + b ...}
        Resultado := Somar (a,b);
        {...mostra o resultado}
End.
```

A primeira etapa nesse exemplo é determinar quem irá fazer cada tarefa para resolver o problema, ou seja, identificar os objetos. Neste exemplo, os objetos são os números inteiros a e b. Tanto a como b são objetos, pois são exemplos daquilo que se conhece por número inteiro.

3.2.2 Classe

Uma classe [18] é um conjunto de objetos que compartilham uma estrutura e um comportamento comum. Enquanto um objeto é uma entidade concreta que existe no tempo e espaço, uma classe representa uma abstração. Conforme visto no exemplo anterior, tem-se a caneta como uma classe, porém a caneta azul, ou a caneta amarela é um objeto, independente de suas características físicas. No trecho de programa anterior, tem-se a e b definidos como objetos, pois são exemplos de números inteiros, portanto pertence a uma classe, a de números inteiros. Quando se diz números inteiros, há vários (infinitos) objetos, que são dessa classe, entre eles, a e b.

3.3 Bases da Programação Orientada a Objetos

A programação orientada a objetos [18] se baseia nos seguintes conceitos:

- Abstração;
- Encapsulamento;
- Modularidade;
- Hierarquia;
- Polimorfismo.

3.3.1 Abstração

A abstração [18] é uma das formas fundamentais para enfrentar a complexidade. Por meio da abstração é possível definir as características essenciais de um objeto que pertence ao mundo real, isto é, seus atributos e seu comportamento a fim de modelá-lo, logo depois, em um objeto de *software.*

Uma abstração [18] focaliza-se na visão exterior de um determinado objeto e serve para separar o comportamento essencial de um objeto da sua implementação.

A principal ferramenta para suportar a abstração é a classe. Uma classe descreve um grupo de objetos que compartilham características semelhantes. Estas características especificam-se por meio dos atributos e comportamentos. Em termos da programação orientada a objetos, diz-se que um objeto constitui uma instância de uma classe determinada.

3.3.2 Encapsulamento

O encapsulamento [18] permite ocultar os detalhes da implementação de um objeto. Correntemente, em programação, se utiliza a terminologia "caixa preta", isto é, não se faz necessário saber como um programa funciona internamente, mas sim como deve ser utilizado. Geralmente uma classe define-se em duas partes, uma interface através da qual os objetos

instanciados da mesma classe interagem dentro da aplicação e a implementação dos membros dessas classes (atributos e comportamentos).

3.3.3 Modularidade

De uma forma muito geral, a modularidade [18] é a ação de particionar um programa em componentes individuais que possam reduzir a complexidade do sistema em algum grau.

A modularidade permite modificar as características de uma classe sem modificar as outras classes que compõem a aplicação. Em termos de *software*, a aplicação pode ser dividida em módulos diferentes (classes) e podem ser compilados e modificados sem afetar os outros.

3.3.4 Hierarquia

Um conjunto de abstrações [18] geralmente forma uma hierarquia. Identificando as hierarquias dentro de um projeto, simplifica-se o entendimento de um problema. Deste modo, pode-se dizer que uma hierarquia é uma organização de abstrações.

A programação orientada a objetos permite definir duas importantes hierarquias em um sistema complexo: hierarquias entre classes e hierarquias entre objetos. Assim, têm-se as hierarquias conhecidas como herança simples, expressa em forma semântica através de "é um" e hierarquia "faz parte de" que se refere à agregação de objetos. A herança simples permite definir uma nova classe a partir de uma já existente

3.3.5 Polimorfismo

O polimorfismo [18] é a propriedade pela qual uma entidade pode assumir diferentes formas. Isto é útil quando se necessita que um mesmo comportamento, operação ou ação se realize de diferentes maneiras. No contexto da programação orientada a objetos, modela-se essa situação por meio de objetos de *software,* permitindo a uma classe assumir diferentes

formas. Este fato é conseguido, denominando os métodos da classe com o mesmo nome, mas com implementações diferentes.

Descreve-se nos próximos itens as principais características do ambiente de programação *Borland Delphi*, utilizado para o desenvolvimento desse trabalho.

3.4 Ambiente de Programação Borland Delphi

3.4.1 Introdução

Lançado pela *Borland* no primeiro semestre de 1995, o *Delphi* entrava no mercado como um produto inovador, desde o lançamento do *Turbo Pascal 3*, na década de 80, com a característica principal de ser uma ferramenta extremamente poderosa e ao mesmo tempo fácil de aprender a usar, sem desprezar a flexibilidade de expandir seus recursos [8].

O *Delphi* é um completo ambiente de desenvolvimento de aplicações, baseado no conceito denominado *RAD (Rapid Application Development)*. Essa metodologia não automatiza por completo o desenvolvimento de uma aplicação, mas sim, auxiliam muito as diversas fases do desenvolvimento de um projeto, principalmente as fases relacionadas às criações de interfaces com o usuário.

Isso significa que podem ser criadas todas as interfaces dos aplicativos de forma visual, apenas adicionando objetos gráficos aos formulários, como botões, caixas de edição, legendas, caixas de combinação, etc. Após a definição dessa interface, podem-se incluir os códigos de programação responsáveis pela lógica e processamento das informações do usuário. Todavia, a fase de modelagem de toda a base de dados e do sistema como um todo, deve ser cuidadosamente estudando pelo programador.

3.4.2 Principais Características

O *Delphi* utiliza como linguagem de programação uma variação do *Turbo Pascal 7.0*, denominado de *Object* Pascal, ou seja, o *Delphi* é ambiente de desenvolvimento totalmente orientada a objeto [18].

O *Delphi* possui um compilador de código nativo que gera aplicações em 32 *bits*, podendo ser utilizado com os sistemas operacionais *Windows 95* ao *Windows 2000 / XP*.

Seu uso possibilita o desenvolvimento desde pequenos programas até grandes aplicações para gerenciamento de base de dados. Atualmente, existem três tipos de distribuição do *Delphi*:

- *Personal*: versão disponível no *site* da *Borland* (*www.borland.com.br*), todavia, não tem suporte à programação de banco de dados ou mesmo, características avançadas de outras versões;
- *Professional*: voltada para desenvolvedores profissionais, permitindo o desenvolvimento de aplicações de banco de dados local (*dBase*, *Paradox*, *Access*), ou remoto (*InterBase, MySQL,* etc) e servidores *WEB*.
- *Enterprise*: possui todas as características da versão Professional, mais o suporte ao desenvolvimento de aplicações *e-business* (*WebSnap*, *BizSnap* e *DataSnap*), além do *Interbase*, *drivers* para base de dados *SQL Oracle, DB2, MS SQL, Informix* e *Sybase*.

3.4.3 A Linguagem *Object Pascal*

Object Pascal é uma linguagem orientada a objetos híbrida, por possuir características de programação não só visual, mas também escrita, para os programadores que já conhecem técnicas de estruturas de programação, como a linguagem *Basic*, *Pascal* ou *xBASE* entre outras [18]. A linguagem *Object Pascal* torna mais fácil o desenvolvimento no ambiente

Windows de aplicações livres, ou que utilizam banco de dados do tipo Cliente/Servidor, trabalha com o uso de ponteiros para a alocação de memória e todo o poder de um código totalmente compilável. Além disso, possibilita a criação e reutilização (vantagem de re-uso tão sonhado com a orientação a objetos) de objetos e bibliotecas dinâmicas *(Dynamic Link Libraries - DLL).*

Object Pascal contém todo o conceito da orientação a objetos incluindo encapsulamento, herança e polimorfismo. Algumas extensões foram incluídas para facilitar o uso tais como conceitos de propriedades, particulares e públicas, e tipos de informações em modo *run-time*, manuseamento de exceções e referências de classes. O resultado de toda esta junção faz com que *Object Pascal* consiga suportar as facilidades de um baixo nível de programação.

Para o desenvolvimento do projeto utilizou-se um sistema de banco de dados concentrado em um servidor, proporcionando um controle centralizado de todos os dados do sistema. Nos próximos itens serão abordadas as principais características de um sistema gerenciador de banco de dados.

3.5 Banco de Dados – Sistema Gerenciador de Banco de Dados

3.5.1 Introdução

Um sistema gerenciador de banco de dados (SGBD) é composto por uma coleção de dados inter-relacionados e um conjunto de programas para acessá-los [9]. Um conjunto de dados, chamado normalmente de banco de dados, contém informações sobre um empreendimento em particular. O principal objetivo de um SGBD é proporcionar um ambiente que seja conveniente e eficiente para recuperar e armazenar informações do banco de dados [6].

Os sistemas de banco de dados [11] são projetados para gerenciar grandes grupos de informações. O gerenciamento de dados envolve a definição de estruturas para armazenamento de informações e o fornecimento de mecanismos para manipular essas informações. O SGBD também é responsável por fornecer segurança para as informações armazenadas, caso o sistema apresente alguma pane, ou mesmo a tentativa de acesso indevida ao seu conteúdo [19].

A grande importância das informações na maioria das empresas e organizações, e a grande importância do seu banco de dados têm incentivado o desenvolvimento de um grande grupo de conceito e técnicas para o gerenciamento eficiente de dados.

3.5.2 Abstração de Dados

Um sistema de gerenciamento de banco de dados é composto por uma coleção de arquivos inter-relacionados e de um conjunto de programas que permitem aos usuários acessar e modificar esses arquivos [19]. O objetivo principal de um sistema de banco de dados é o de proporcionar aos usuários uma visão abstrata dos dados, ou seja, ocultar certos detalhes de como os dados são armazenados e gerenciados. Partindo do pressuposto de que muitos dos usuários de sistemas de banco de dados não possuem um conhecimento amplo em computação, a complexidade de todo o sistema está oculta em de diversos níveis de abstração, que simplificam a interação do usuário com o sistema.

A seguir, são descritos os principais níveis de abstração de dados:

- Nível Físico: É o nível mais baixo de abstração de dados e descreve como os dados realmente estão armazenados, sendo todas as estruturas de dados descritas em detalhes nesse nível;
- Nível Conceitual: É o nível que representa quais dados estão armazenados no banco de dados e as relações que existem entre eles, sendo o banco de dados nessa fase,

descrito em um pequeno número de estruturas relativamente simples. O nível conceitual é utilizado por administradores de banco de dados;

- Nível Visual: É o nível mais alto de abstração que descreve apenas parte do banco de dados. O nível visual de abstração é definido para simplificar a interação do usuário com o sistema, o qual pode fornecer muitas visões para o mesmo banco de dados.

3.5.3 Banco de Dados Distribuídos

Um sistema de banco de dados distribuídos é composto por uma coleção de nós, podendo ser um microcomputador, por exemplo, onde cada máquina pode participar na execução das transações que acessam dados em um ou diversos nós [20]. A diferença principal entre sistemas de banco de dados centralizados e distribuídos, é que no primeiro os dados estão localizados em um único local, e no último os dados estão localizados em locais distintos.

3.5.4 Prós e Contras do Banco de Dados Distribuído

Em um sistema de banco de dados distribuído, podem-se obter inúmeros benefícios, como o compartilhamento de dados, confiabilidade e disponibilidade [20]. Todavia juntamente com essas vantagens, existem inúmeras desvantagens, que incluem desde o custo de desenvolvimento do *software*, maior potencial de erro, caso o planejamento não seja feito de forma correta.

A principal vantagem de sistemas de banco de dados distribuídos é a capacidade de partilhar e acessar dados de uma maneira confiável e eficiente. Entre as principais vantagens da utilização de banco de dados distribuídos, tem-se:

- Partilhamento de dados e controle de distribuição: se uma série de nós estão conectados, então um usuário em um nó qualquer pode acessar dados disponíveis em outro nó;
- Confiabilidade e disponibilidade: se um nó falhar em um sistema distribuído, os nós restantes são capazes de continuar operando;
- Aceleração no Processamento de Consultas: se uma consulta envolve dados em diversos nós, é possível dividi-las em sub-consultas que podem ser executadas em paralelo.

Uma das principais desvantagens de se utilizar sistemas de banco de dados distribuídos é a complexidade adicional requerida para assegurar a própria coordenação entre os nós. Entre os principais problemas de complexidade, pode-se citar:

- Custo de Desenvolvimento de *Software*: é mais difícil implementar um sistema de banco de dados distribuído, portando mais caro;
- Potencial Maior de *Bugs*: uma vez que os nós que formam o sistema distribuído operam em paralelo, é mais difícil realizar a correção dos algoritmos;
- Redução do Desempenho de Processamento: a troca de mensagens e a computação adicional requeridas para realizar a coordenação inter-nós é uma sobrecarga de processamento que pode contribuir para a queda (ou redução) no desempenho final do sistema.

3.6 Interbase

3.6.1 Introdução

O *Interbase* [21] é um gerenciador de banco de dados relacional (SGBDR) cliente / servidor, multiplataforma, poderoso e ao mesmo tempo simples, sendo desenvolvido para ser um banco de dados independente de plataformas e sistemas operacionais [22].

O sistema gerenciador foi desenvolvido por um grupo de engenheiros de sistemas, em meados de 1985, foi chamado inicialmente de *Groton*. Esse produto sofreu várias alterações até finalmente em 1986, receber o nome de *Interbase*, iniciando em sua versão 2.0 [10].

O *Interbase* acompanha o *Delphi*, com exceção das versões *Standard* e *Personal* (essa última disponível para *download* na *Internet* no endereço *www.borland.com/interbase*). Essa característica proporcionou o desenvolvimento de projetos utilizando aplicações com base de dados, sem a necessidade de se adquirir sistemas caros, dispensando também a interferência freqüente de profissionais, especializados em manutenção de banco de dados.

O *Interbase* dispensa o uso de super-servidores, usando pouco espaço em disco para a sua instalação e utilizando pouca memória em situações normais de uso. Dessa forma a plataforma requerida para a sua instalação e utilização pode ser reduzida, diminuindo consideravelmente os custos de projeto, todavia é evidente que um desempenho mais satisfatório pode ser obtido com a utilização de um equipamento melhor.

Atualmente o *Intebase* possui seu código distribuído gratuitamente pela *Internet*, chamado de *Open Source*, sendo que as licenças de utilização e distribuição são totalmente gratuitas.

Atualmente existem diversos sistemas gerenciadores de banco de dados no mercado, como *ORACLE*, *SYBASE*, entre outros, todavia, utilizou-se o *Interbase* nesse trabalho, pois até a sua versão 6.5, é distribuído gratuitamente na *Internet*.

3.6.2 Principais Características

O *Interbase* oferece suporte à linguagem *SQL (Structured Query Language)* de acordo com o padrão *SQL-ANSI-92*, além de mecanismos eficientes de segurança, integridade de dados, flexibilidade requerida para a *Internet*, aplicações de bases de dados integradas [10], [22], entre outras características citadas a seguir:

- *Multi-Threading*: o servidor de banco de dados é implementado com múltiplos *threads* de execução, ele pode tirar grande proveito de ambientes multiprocessados, aumentado muito o desempenho;
- Fácil Configuração: configuração de acordo com a necessidade para maior desempenho automaticamente;
- *Triggers:* permite que sejam estabelecidas regras;
- Sombra: cópia idêntica à cópia que está sendo mantida pelo servidor, excelente para recuperação;
- Tipos Binários: suporte para arquivos binários, som, vídeo, imagens e suporte a a*rray* multidimensionais.

3.7 SQL (Structured Query Language)

A linguagem de manipulação de dados *SQL,* padronizada pelo *American National Standards Institute (ANSI),* atualmente é a linguagem de manipulação de dados relacionais mais utilizada em aplicações de banco de dados [11]. O *SQL* é uma linguagem de acesso a dados amplamente utilizada por diversos Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDR) comerciais, como *ORACLE*, *SYBASE*, *SQL Server*, *Interbase*, *Paradox*, entre outros, tornando-se assim, uma linguagem padrão para troca de informações entre computadores.

Os comandos de *SQL* podem ser usados interativamente como uma linguagem de consulta, ou podem ser incorporados a programas de aplicações. O *SQL* não é uma linguagem de programação, e sim uma sub-linguagem de dados ou linguagem de acesso a dados, juntamente incorporada em outras linguagens.

3.7.1 Estrutura Básica da Linguagem *SQL*

Basicamente, a estrutura de uma expressão *SQL* consiste em três clausulas: *select*, *from* e *where* [23], sendo:

- A cláusula *select* é utilizada para listar os atributos desejados no resultado de uma consulta;
- A cláusula *from* lista as relações a serem examinadas na avaliação da expressão;
- A cláusula *where* consiste em um predicado envolvendo atributos de relações que aparecem na cláusula *from*.

Considere o seguinte exemplo de um comando básico em *SQL*, onde na Tabela TBUsu, existe o campo "nome_usu" e "tel_usu". Deseja-se localizar todos os telefones dos usuários com o nome "ANDRE"

select "tel_usu"

from "TBUsu"

where "nome_usu" = 'ANDRE'

Na execução do comando são listados todos os telefones dos usuários que tenham o nome iniciado com "ANDRE".

3.8 Protocolo 1-Wire

Para a realização da leitura do número de *ROM* do *iButton*, o microcontrolador *Motorola MC68HC908* foi programado utilizando o protocolo *1-Wire* e rotinas para o envio de informações pela interface paralela do microcomputador [26].

O protocolo *1-Wire* foi desenvolvido pela *Dallas Semiconductor* (atualmente pertencente à *Maxim*) para permitir a construção de dispositivos periféricos para microcontroladores e microprocessadores com um uso mínimo de recursos tanto de *hardware* como de *software* [26].

Existe uma grande diversidade de dispositivos compatíveis com o protocolo, tais como: *chips* de identificação, memórias *EPROM* e *EEPROM* seriais, relógios de tempo real, sensores de temperatura, conversores A/D, etc.

E estrutura do protocolo é baseada em uma arquitetura mestre-escravo, utilizando o barramento de apenas um fio em modo *half-duplex* [26].

Eletricamente, o protocolo estabelece que os elementos de conexão ao barramento devem ser todos do tipo dreno-aberto, ou seja, um dispositivo somente pode forçar o barramento em nível zero. Sendo assim, deve haver um resistor de *pull-up* externo de forma a garantir nível lógico 1 quando não existirem elementos forçando o barramento ao zero lógico.

O protocolo em si é do tipo assíncrono e utiliza uma arquitetura de *time-slots*, ou fatias de tempo, para permitira a comunicação bidirecional *half-duplex*. A velocidade máxima de comunicação é de 16 *kbps* para o modo normal e 143 *kbps*, para o modo *OverDrive*, o que é mais que suficiente para as pequenas tarefas de aquisição de dados e controle geral. Como características principais do protocolo, podem-se citar [26]:

- Alimentação entre 3 e 5,5 volts;
- Possibilidade de alimentação do dispositivo a partir da própria linha do barramento (modo de alimentação parasita). Nesse caso, a quantidade de fios para a comunicação resume-se a apenas dois: comunicação e terra;
- Número único de identificação de 64 *bits* para cada dispositivo. Este código de 64 *bits* é composto de 8 *bits* para *CRC*, 48 *bits* com o número serial do dispositivo e mais 8 *bits* para a identificação da família do dispositivo.

3.8.1 Arquitetura Geral

Como o protocolo *1-Wire* é do tipo mestre-escravo, cabe ao dispositivo mestre do barramento a tarefa de iniciar e controlar a comunicação.

A inicialização do barramento é feita pelo dispositivo mestre que deve colocar o barramento em nível lógico zero por um período mínimo de 480µs. Este é chamado pulso de *reset* do barramento [26].

Entre 15 e 60µs após o *reset* do barramento, os dispositivos escravos presentes aterram a linha do barramento, indicando que existem dispositivos presentes no sistema. Esse pulso de presença possui entre 60 e 240µs de duração. Na Figura 3.1, apresenta-se os tempos do pulso de *reset* e de presença [26].

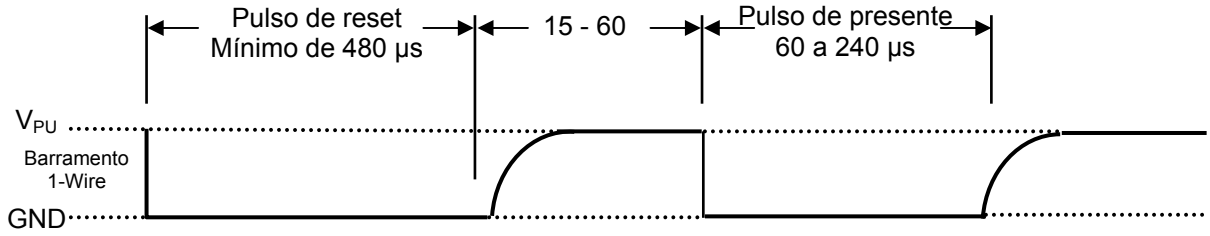


Figura 3.1: Tempos do pulso de *reset* e pulso de presença.

Se após o pulso de *reset* do barramento ele continuar em nível lógico 1 por mais de 240µs, é sinal de que não há dispositivos conectados ao barramento. Na Figura 3.2 apresenta-se o tempo do pulso de *reset* sem a detecção de nenhum dispositivo.

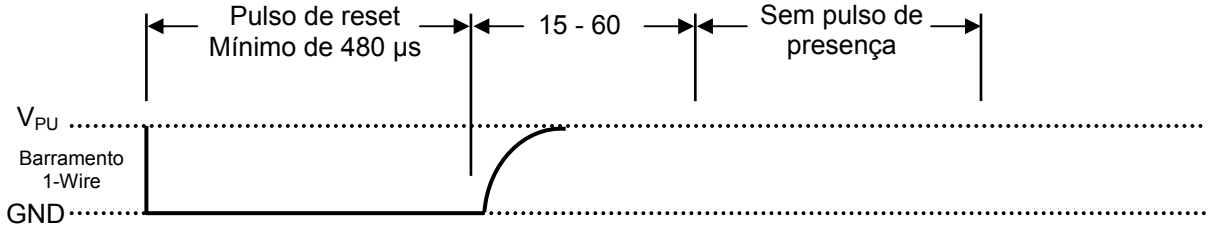


Figura 3.2: Tempo do pulso de reset sem a detecção de dispositivo.

Após o *reset* e verificação de presença de dispositivos no barramento, o dispositivo mestre pode iniciar a comunicação com o(s) dispositivo(s) escravos. Nessa situação existem dois casos possíveis [26]:

1. Há apenas um dispositivo escravo no barramento (esta condição deve ser prevista no projeto do sistema);

2. Existe mais de um dispositivo escravo no barramento.

No primeiro caso, o dispositivo mestre pode iniciar a comunicação com o dispositivo escravo imediatamente após o ciclo de *reset* / detecção de dispositivos. No segundo caso, antes de iniciar a comunicação com os dispositivos escravos, o mestre deve providenciar a identificação individual dos dispositivos no barramento [26].

Doravante, consideremos a existência de apenas um elemento escravo conectado ao barramento. Assim, após o pulso de *reset* e a resposta de presença do dispositivo, o elemento mestre pode enviar comandos para o dispositivo escravo [26].

Os comandos são divididos em escrita e leitura. Para realizar uma escrita no barramento, o elemento mestre deve manter a linha em nível lógico "0" por no mínimo 1µs. Após esse período, a linha é mantida em "0"ou desligada "nível 1", de acordo com o valor do *bit* a ser escrito. Na Figura 3.3 apresenta-se os comandos de leitura e escrita.

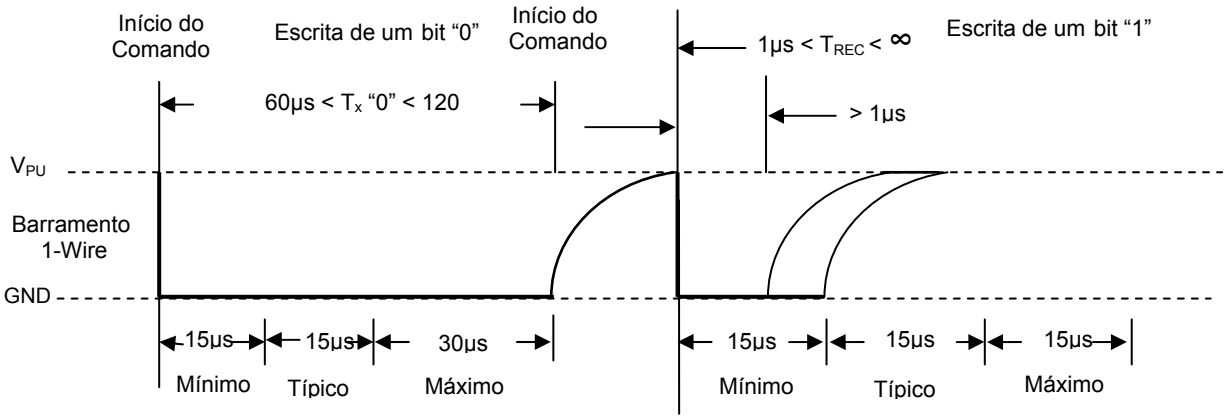


Figura 3.3: Gráfico de leitura e escrita.

A sincronização dos elementos escravos com o dispositivo mestre é feita pela detecção da borda de descida do sinal emitido pelo mestre do barramento.

A operação de leitura de um *bit* é realizada de forma muito parecida com o da escrita, mas com a diferença de que o elemento mestre inicia a operação colocando a linha do

barramento em nível lógico “0” por um mínimo de 1µs, em seguida desliga a saída (fazendo com que o barramento vá a nível “1” por força do resistor *pull-up* externo). Aproximadamente 15µs após a borda de descida do sinal, o elemento escravo reconhece a operação e coloca o dado na linha por um período mínimo de 45µs. Na Figura 3.4 apresenta-se a operação de leitura iniciada pelo elemento mestre.

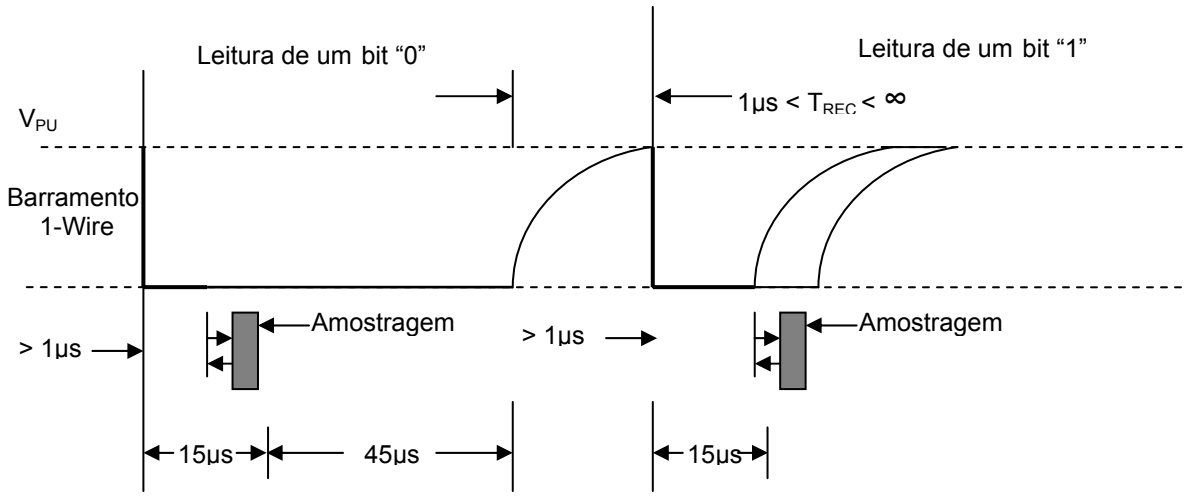


Figura 3.4: Operação de leitura e escrita iniciada pelo dispositivo mestre.

A duração dos comandos (leitura e escrita) deve estar entre 60µs (mínimo) e 120µs (máximo).

Os comandos e dados que circulam pelo barramento *1-Wire* são sempre transmitidos iniciando pelo *bit* menos significativo (*LSB*).

CAPÍTULO 4 – Sistema Microcontrolado para Supervisionar o Acesso a Laboratórios Interdisciplinares

4.1 Introdução

A proposta para o desenvolvimento do projeto surgiu da necessidade de atualização do sistema de controle de acesso, na época em uso no Laboratório de Processamento de Sinais e Sistemas Digitais – FEIS - UNESP, criando um sistema integrado que proporcionasse ao administrador toda a operacionalidade da linguagem visual, fornecendo uma interface amigável, possibilitando dessa forma, um gerenciamento mais eficiente no acesso às dependências do laboratório de pesquisa, através de diversos relatórios de auditoria, permissões de dia e hora e segurança quanto a possíveis tentativas de acessos indevidos.

Pensando nessas características, reformulou-se o sistema antigo, criando-se um sistema de controle de acesso, cujas informações, estão centralizadas em um servidor de banco de dados. Através do servidor é possível realizar todo o gerenciamento do sistema, emitir relatórios de auditoria e demais informações pertinentes. Um ambiente controlado dessa maneira permite ao acadêmico um acesso aberto e seguro às dependências do laboratório de ensino, abolindo do uso das chaves tradicionais e, até mesmo, a necessidade de funcionários para supervisionar a entrada e saída dos acadêmicos.

O sistema foi implementado no ambiente de desenvolvimento *Borland Delphi* [8] versão 7, sendo sua base de dados desenvolvida em *SQL* e utilizando o *Interbase 6.0* como sistema gerenciador de banco de dados, proporcionando assim, todas as vantagens oferecidas pela linguagem de programação visual e um controle mais eficiente e ágil.

Durante a implementação do sistema, optou-se pela sua divisão em módulos, cada um responsável em realizar funções distintas. A modularização proporciona uma maior eficiência

e operacionalidade do sistema, uma vez que cada módulo é independente, e sua aplicação dar-se-á de acordo com as necessidades da implantação, permitindo que apenas o módulo necessário seja instalado.

Os módulos são denominados de módulo leitura e módulo administrador, os quais serão descritos detalhadamente nos próximos itens. A modularidade permite que um administrador realize um cadastro de usuário, alterações de suas permissões estando em qualquer microcomputador localizado na rede local, não havendo a necessidade de instalação de todo o sistema.

O sistema é composto em duas partes distintas em nível de *hardware* e *software*, onde ambos se interagem para realizar o controle de entrada e saída de usuários.

4.2 Descrição da Estrutura de Software – Módulo Administrador

O módulo administrador tem a função de realizar o gerenciamento dos usuários que acessam as dependências do laboratório de pesquisa. Para isso, disponibiliza, diversos recursos, como o cadastro de usuários, cadastro de permissões de acesso, onde é possível definir as permissões para cada usuário e os relatórios de auditoria. O acesso ao módulo administrador é permitido mediante o fornecimento de um nome de usuário e senha.

No sistema, foram definidos dois tipos distintos de administradores, o administrador geral do sistema, sendo ele o responsável em atribuir as demais permissões aos outros administradores, possuindo acesso global a todos os dados do sistema e o administrador do laboratório, esse, responsável em administrar os usuários de seu respectivo laboratório de pesquisa, possuindo acesso somente aos dados pertinentes aos seus usuários, não possuindo nenhum outro acesso aos demais dados dos outros laboratórios. Todavia, é facultado ao administrador geral do sistema conceder os seus privilégios de acesso aos administradores de laboratórios. Cada laboratório pode ter apenas um administrador. Descreve-se a seguir, as principais características do módulo administrador.

4.2.1 Acesso ao módulo administrador

O acesso ao módulo administrador é feito com a utilização de um nome de usuário e uma senha de acesso. O nome de usuário e a senha de acesso do administrador geral do sistema são únicos, ou seja, existe apenas um administrador geral em todo o sistema, todavia, podem existir outros administradores de laboratório com níveis de acesso de administrador, os quais são definidos durante o seu cadastro. O formulário de "*logon*" é apresentado na Figura 4.1.

No ato de instalação do *software*, o administrador geral do sistema deve cadastrar seu nome de usuário e senha de acesso. Os acessos às opções do menu variam de acordo com o tipo de administrador. A descrição do cadastro de nome de usuário e senha dos administradores é abordada no item 4.2.1.1.

Figura 4.1 : Interface principal de acesso ao módulo administrador.

Para uma maior segurança do sistema, foi desenvolvido um algoritmo de criptografia e descriptografia de dados. Todas as senhas armazenadas no banco de dados do sistema são criptografadas pelo algoritmo de criptografia, sendo que nem mesmo o administrador de banco de dados consegue identificar as senhas cadastradas. Mesmo que um usuário mal

intencionado consiga acessar o banco de dados do sistema, este, não conseguirá identificar as senhas armazenadas nas tabelas, tornando o sistema mais seguro e menos propenso a invasões e falhas de segurança.

No processo de verificação de senha, quando o administrador efetua o acesso ao sistema, o algoritmo descriptografa a senha armazenada no banco de dados, realizando a comparação com a senha fornecida, permitindo ou não o seu acesso. A seguir, serão apresentados os principais recursos do módulo administrador.

4.2.1.1 Cadastro de Usuários

Na opção de cadastro de usuários, é permitido ao administrador geral e ao administrador de laboratórios, realizar a inclusão, alteração, consulta e exclusão dos dados de usuários no banco de dados do sistema. Para o administrador do laboratório, somente será permitido a inclusão de usuários, pertinente ao seu laboratório de pesquisa, onde a opção de preenchimento "laboratório de pesquisa" é bloqueada para alterações. Já para o administrador geral do sistema, este poderá realizar o cadastro de usuários para qualquer laboratório de pesquisa, pois a opção é desbloqueada.

Durante o cadastro de usuários, cuja interface é apresentada na Figura 4.2, o administrador deve informar ao sistema todos os dados pertinentes ao usuário. Um dos campos requeridos para o cadastro do usuário é o número do seu *iButton*. Este pode ser obtido de maneira automática, desde que o microcomputador possua a interface de comunicação *1-Wire (Blue Dot)*, bastando apenas pressionar o botão "Leitura". Durante essa operação, o sistema adquire o número do *iButton*, atribuindo o valor automaticamente no campo "*iButton*", dispensando a digitação.

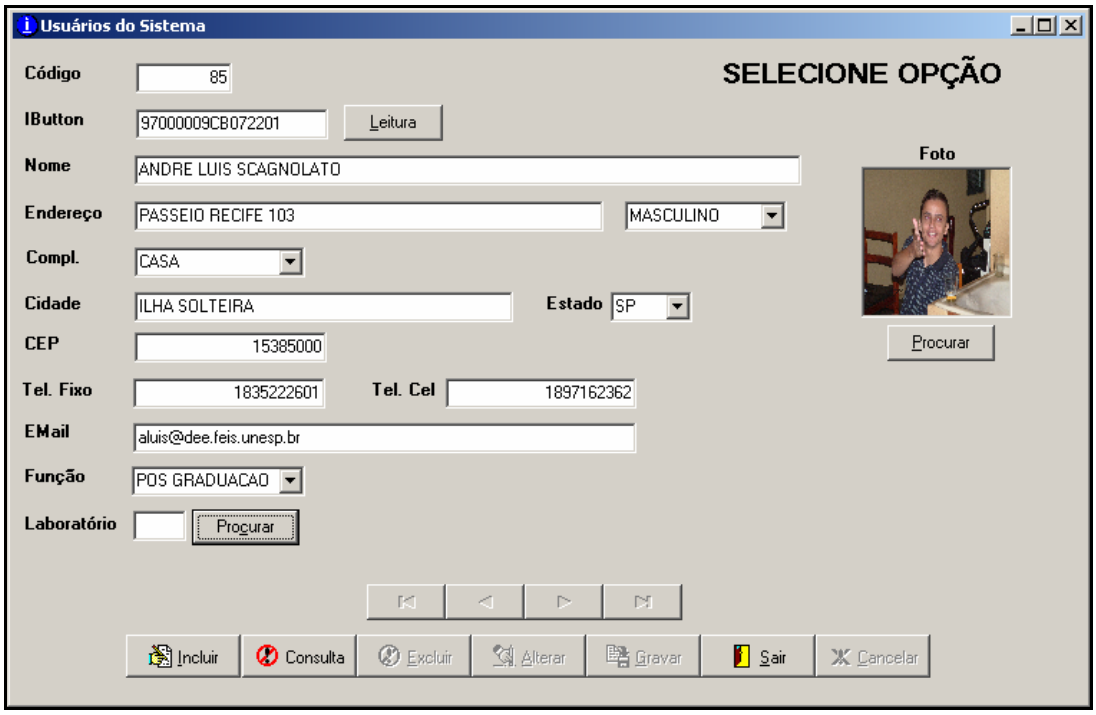


Figura 4.2 : Interface de cadastro de usuários do sistema.

Os campos requeridos para preenchimento e suas respectivas características são apresentados na Tabela 4.1:

Tabela 4.1 : Descrição dos campos requeridos no cadastro de usuários do sistema.

Número Campo	Nome Campo	Descrição
1	Código do Usuário	Código do usuário, gerado automaticamente a cada cadastro.
2	Número *iButton*	Número *iButton*
3	Nome	Nome completo do usuário.
4	Endereço	Endereço do usuário.
5	Sexo	Sexo do usuário.
6	Complemento	Complemento (casa, prédio, etc).
7	Cidade	Cidade do usuário.
8	Estado	Estado do usuário.
9	CEP	CEP do usuário.
10	Telefone Fixo	Telefone Fixo do usuário.
11	Telefone Celular	Telefone Celular do usuário.
12	*E-Mail*	*E-Mail* do usuário.
13	Função	Função do usuário (graduação, pós graduação, funcionário, etc).
14	Laboratório	Laboratório de pesquisa ao qual o usuário pertence.

Na opção de escolha de laboratório, onde é definido qual laboratório de pesquisa o usuário pertence, esta, é liberada somente para o administrador geral do sistema, sendo vetada essa opção de escolha para os administradores de laboratórios.

O código do laboratório é atribuído automaticamente, de acordo com código do laboratório de pesquisa ao qual o administrador é responsável. Por exemplo, o administrador de laboratório "Alexandre", pertencente ao laboratório de código "43", com o nome "Laboratório de Processamento de Sinais e Sistemas Digitais", dessa forma, somente realiza cadastros de usuários pertinentes ao seu laboratório de pesquisa. Nesse caso, é atribuído automaticamente o código "43" no campo laboratório, não sendo possível a sua alteração.

Contudo, se o administrador de laboratório "Alexandre" possuísse os privilégios do administrador geral do sistema, o mesmo poderia realizar a inclusão do usuário em qualquer laboratório de pesquisa.

4.2.1.2 Cadastro de Permissões de Acesso

Na opção de cadastro de permissões de acesso dos usuários, é permitido ao administrador realizar a inclusão, alteração, consulta e exclusão dos dados referentes às permissões de usuários no banco de dados. As permissões de acesso são definidas individualmente por usuário, e pode ser manipulada de acordo com as características de cada um. Na Figura 4.3 apresenta-se a interface de cadastro das permissões de acesso, podendo ser do tipo:

- Semanal: é possível definir os dias da semana, hora inicial e hora final em que o usuário pode utilizar o laboratório de pesquisa;
- Período: pode-se definir o período em que o usuário pode utilizar o laboratório de pesquisa. Por exemplo, pode-se combinar a permissão semanal em conjunto com a

permissão por período, na necessidade de um usuário utilizar o laboratório de pesquisa por um tempo determinado;

- Irrestrito: um usuário pode ser definido como irrestrito, ou seja, ele pode freqüentar as dependências do laboratório sem restrições, todavia, somente no laboratório de pesquisa ao qual foi cadastrado pelo administrador do laboratório, não podendo acessar os demais laboratórios.

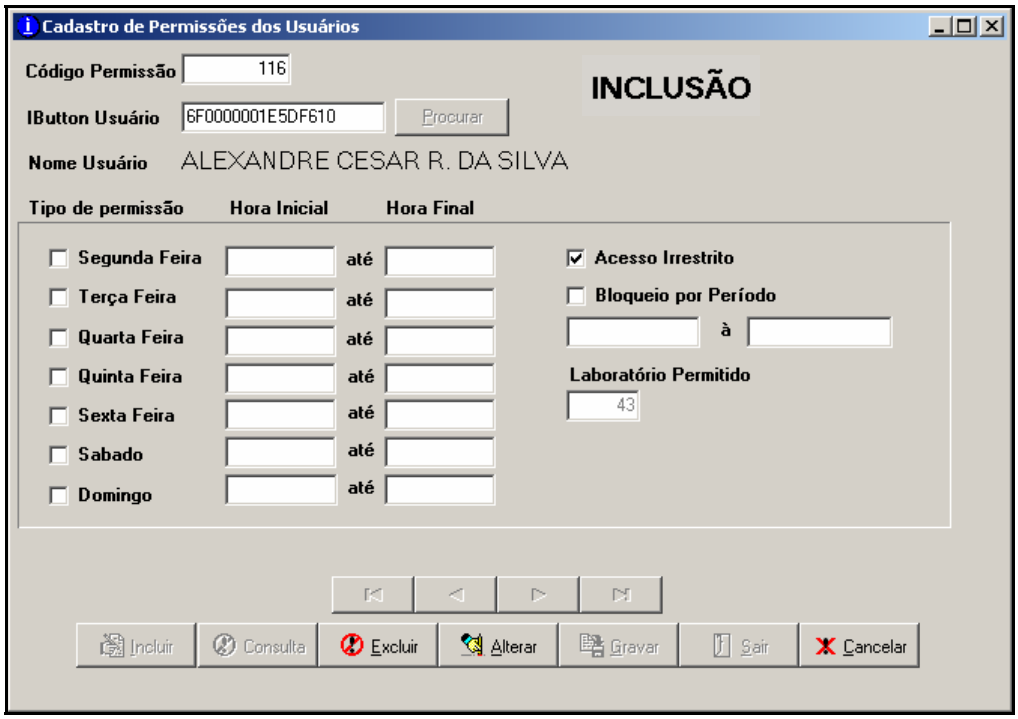


Figura 4.3 :Interface de cadastro de permissões do usuário.

O campo "Laboratório Permitido" é inalterável nessa opção, sendo exibido somente a título de comentário, uma vez que somente pode ser alterado na opção de "alteração" no menu "Cadastro de Usuários". Na Tabela 4.2 apresenta-se os campos requeridos no preenchimento do cadastro de permissões.

Tabela 4.2 : Descrição dos campos requeridos no cadastro de permissões de usuários.

Número Campo	Nome Campo	Descrição
1	Código da Permissão	Código do usuário, gerado automaticamente a cada cadastro.
2	Número *iButton*	Número *iButton.*
3	Nome Usuário	Nome do usuário.
4	Segunda Feira	Habilitado para permissão da segunda feira.
5	Hora Inicial	Hora inicial permitida na segunda feira.
6	Hora Final	Hora final permitida na segunda feira.
7	Terça Feira	Habilitado para permissão da terça feira.
8	Hora Inicial	Hora inicial permitida na terça feira.
9	Hora Final	Hora final permitida na terça feira.
10	Quarta Feira	Habilitado para permissão da quarta feira.
11	Hora Inicial	Hora inicial permitida na quarta feira.
12	Hora Final	Hora final permitida na quarta feira.
13	Quinta Feira	Habilitado para permissão da quinta feira.
14	Hora Inicial	Hora inicial permitida na quinta feira.
15	Hora Final	Hora final permitida na quinta feira.
16	Sexta Feira	Habilitado para permissão da sexta feira.
17	Hora Inicial	Hora inicial permitida na sexta feira.
18	Hora Final	Hora final permitida na sexta feira.
19	Sábado	Habilitado para permissão no sábado.
20	Hora Inicial	Hora inicial permitida no sábado.
21	Hora Final	Hora final permitida no sábado.
22	Domingo	Habilitado para permissão no domingo.
23	Hora Inicial	Hora inicial permitida no domingo.
24	Hora Final	Hora final permitida no domingo.
25	Acesso Restrito	Preenchimento para acesso irrestrito.
26	Bloqueio por Período	Habilita bloqueio por período.
27	Período Inicial	Período inicial.
28	Período Final	Período final.
29	Laboratório Permitido	Código do laboratório permitido.

4.2.1.3 Cadastro de Múltiplas Permissões de Acesso

Nessa opção, é possível incluir permissões de acesso a outros laboratórios de pesquisa, ou seja, um usuário pode acessar outro laboratório, com as mesmas permissões de acesso ou não. Por exemplo, um usuário com permissão de acesso irrestrito ao laboratório de código "43", nome "Laboratório de Processamento de Sinais e Sistemas Digitais", pode ser incluído para ter acesso ao laboratório "41", nome "Laboratório de Computação", todavia com permissão de acesso somente na segunda feira, das "14:00 as 18:00 horas".

O cadastro é semelhante ao cadastro de permissões de acesso, contudo é incluído o botão "Procurar" para escolha do laboratório de pesquisa adicional. Na Figura 4.4 apresenta-se a interface de cadastro de múltiplas permissões de acesso.

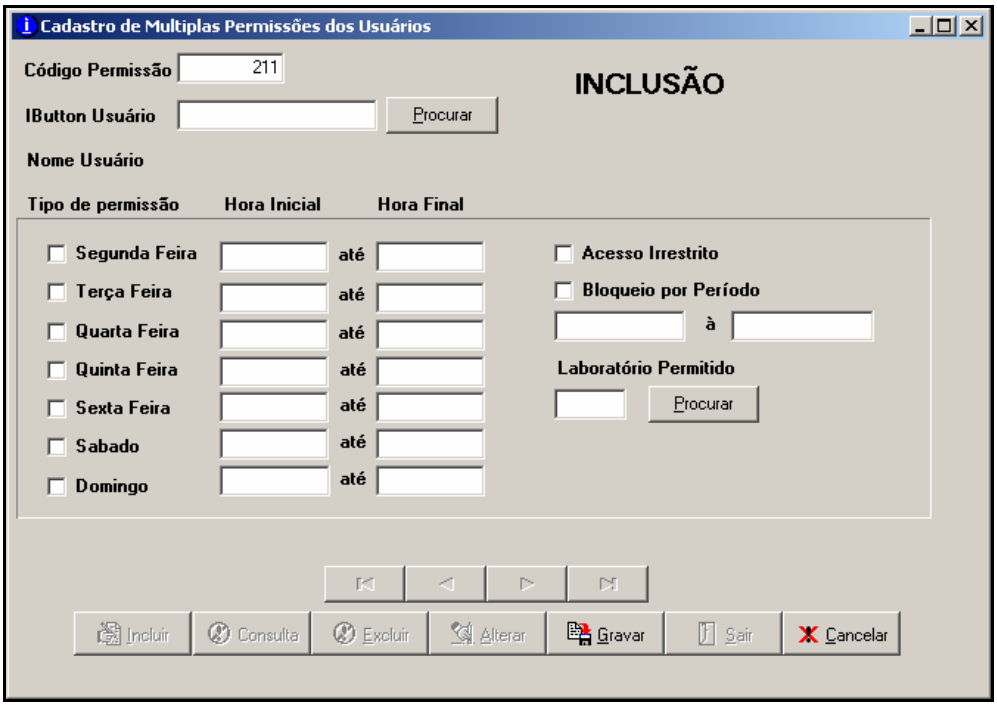


Figura 4.4 : Interface de cadastro de múltiplas permissões de acesso.

4.2.1.4 Cadastro de Laboratórios

A opção de cadastro de laboratórios somente é disponível ao administrador geral do sistema. Nessa opção é possível cadastrar os laboratórios de pesquisa, onde é informado o nome do laboratório de pesquisa, o nome do docente responsável, *e-mail*, nome de usuário e senha, sendo os dois últimos itens, os que serão utilizados pelo administrador de laboratórios para gerenciamento de seus usuários. Na opção "Acesso Administrador", quando preenchida, torna o administrador de laboratório um administrador geral do sistema, com todos os seus privilégios. Essa opção deve ser preenchida, caso se deseje mais de um administrador geral do sistema.

Na Figura 4.5 apresenta-se a interface de cadastro de laboratórios de pesquisa.

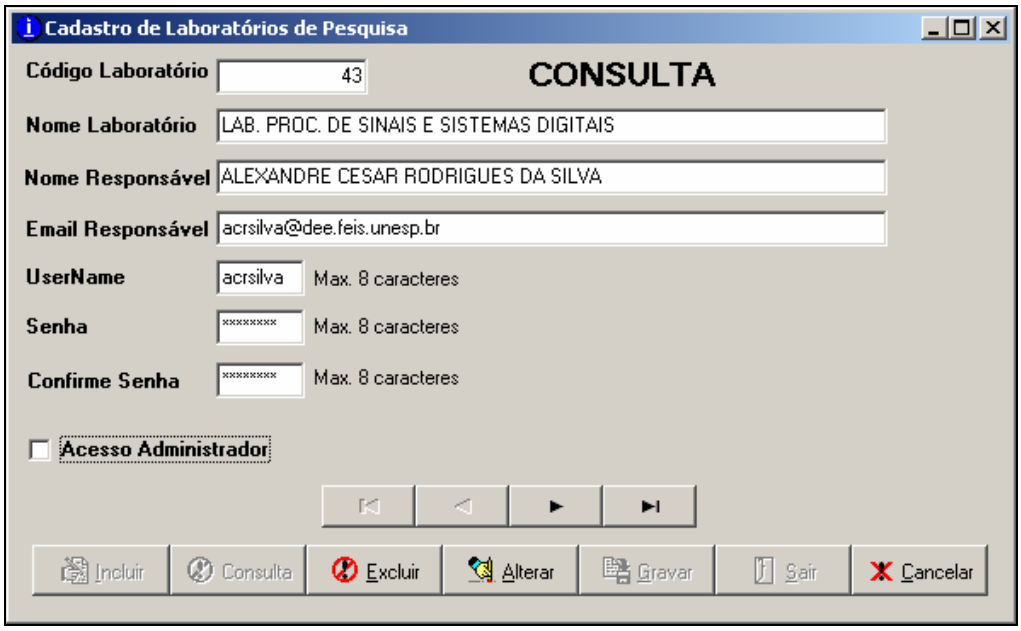


Figura 4.5 : Interface de cadastro de laboratórios de pesquisa.

Na Tabela 4.3 apresenta-se os campos requeridos para o cadastro de laboratórios de pesquisa.

Tabela 4.3 : Descrição dos campos requeridos no cadastro de laboratórios de pesquisa.

Número Campo	Nome Campo	Descrição
1	Código do Laboratório	Código do laboratório, gerado automaticamente a cada cadastro.
2	Nome do Laboratório	Nome do laboratório de pesquisa.
3	Nome Responsável	Nome completo do responsável.
4	*E-mail* Responsável	Endereço de e-mail do responsável.
5	*UserName*	Nome de usuário (máximo de 8 caracteres)
6	Senha	Senha (máximo 8 caracteres)
7	Confirmação Senha	Confirmação da senha digitada no campo anterior.

4.2.1.5 Alteração de Dados do Administrador

Nessa opção, exclusiva do administrador geral do sistema, é permitida a alteração de seus dados. A interface de alteração de dados do administrador é apresentada a partir da Figura 4.6.

Figura 4.6 : Interface de alteração dos dados do administrador.

Na Tabela 4.4 apresenta-se os campos requeridos na alteração de dados do administrador.

Tabela 4.4 : Descrição dos campos requeridos nas configurações do administrador.

Número Campo	Nome Campo	Descrição
1	Nome	Nome do administrador do sistema.
2	*E-mail*	*E-mail* do administrador.
3	*Username*	Nome de usuário (máximo 8 caracteres).
4	Senha	Senha (máximo 8 caracteres).
5	Confirmação de senha	Confirmação de senha digitada anteriormente.

4.2.1.6 Relatórios de Auditoria (Administrador geral do sistema)

Com o intuito de avaliar o grau de segurança no acesso às dependências do laboratório de pesquisa, o sistema proporciona diversos relatórios para fins de auditoria.

Os relatórios podem ser gerados de acordo com o tipo de administrador, como por exemplo, um administrador do laboratório com código “43”, somente poderá ter acesso aos dados dos usuários pertinentes ao seu laboratório de pesquisa, não podendo ter acesso aos demais dados.

Os relatórios fornecidos para o administrador geral do sistema são apresentados na Figura 4.7.

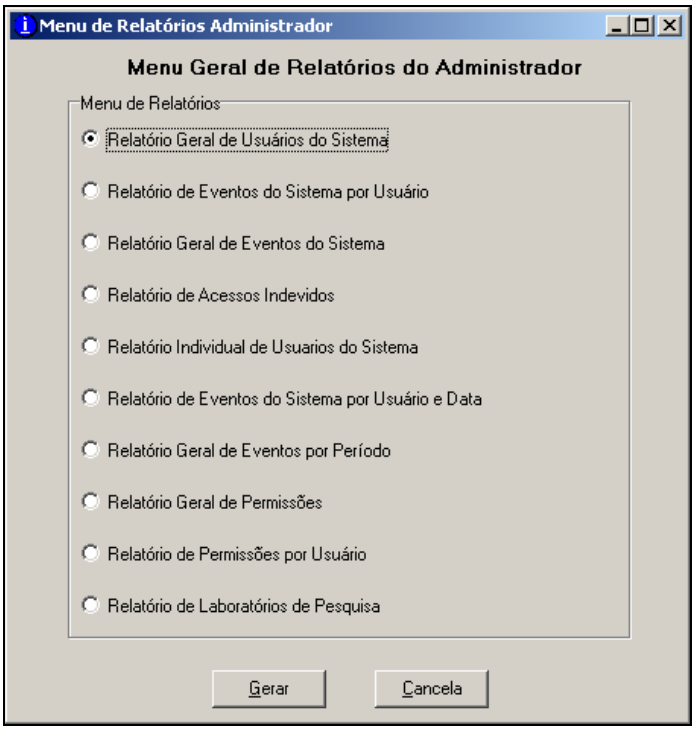


Figura 4.7 : Opções de relatórios de sistema para o administrador geral do sistema.

Os tipos de relatórios fornecidos pelo sistema são descritos a seguir:

Relatório geral de usuários do sistema: esse relatório exibe todos os usuários cadastrados no sistema, onde são apresentados o seu código, nome do usuário, número do *iButton* e função. Como é um relatório permitido somente ao administrador geral do sistema, é necessário informar o laboratório de pesquisa que se deseja listar o relatório de usuários, ou escolher a opção para listagem geral de todos os usuários de todos os laboratórios. Na Figura 4.8 apresenta-se a interface de opções de laboratórios de pesquisa.

Relatório de eventos do sistema por usuário: o relatório de eventos do sistema por usuário permite que sejam visualizadas todas as transações de um determinado usuário a partir de seu nome. Para isso, é necessário informar o nome de usuário que se deseja emitir o relatório. Os dados apresentados no relatório são respectivamente, o código do usuário, nome do usuário, número do *iButton*, data do evento e a hora do evento. A interface de procura de usuários é apresentada na Figura 4.9.

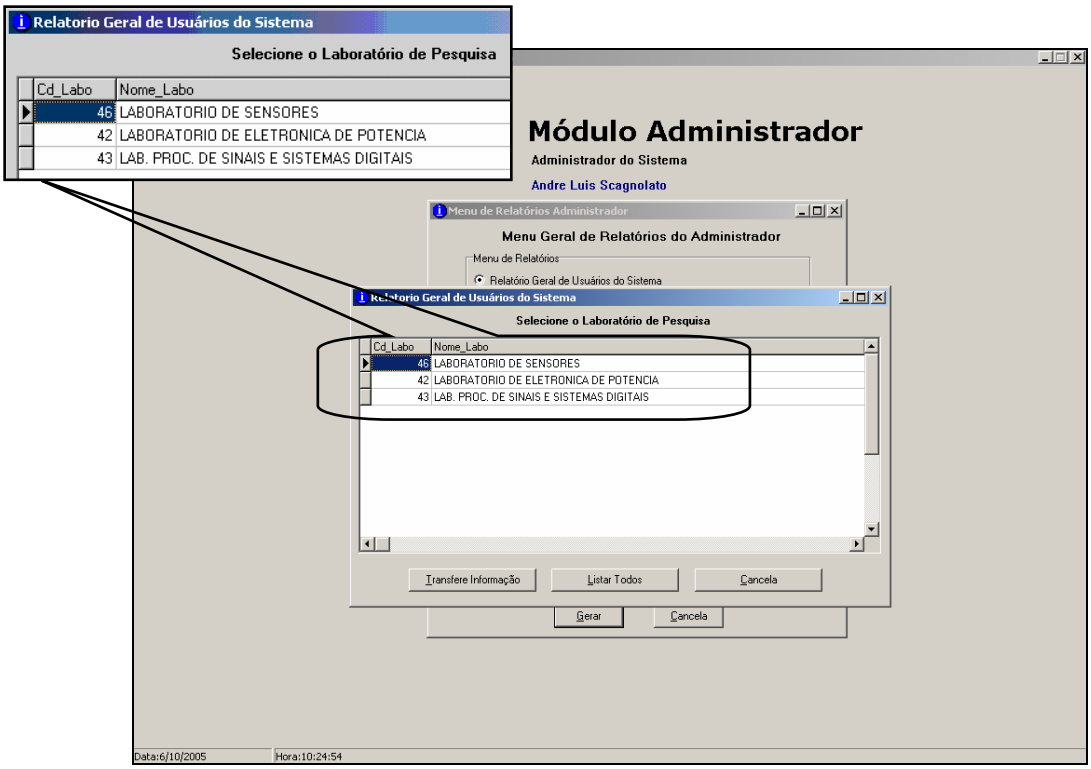


Figura 4.8 : Interface para escolha de laboratório no relatório geral de usuários do sistema.

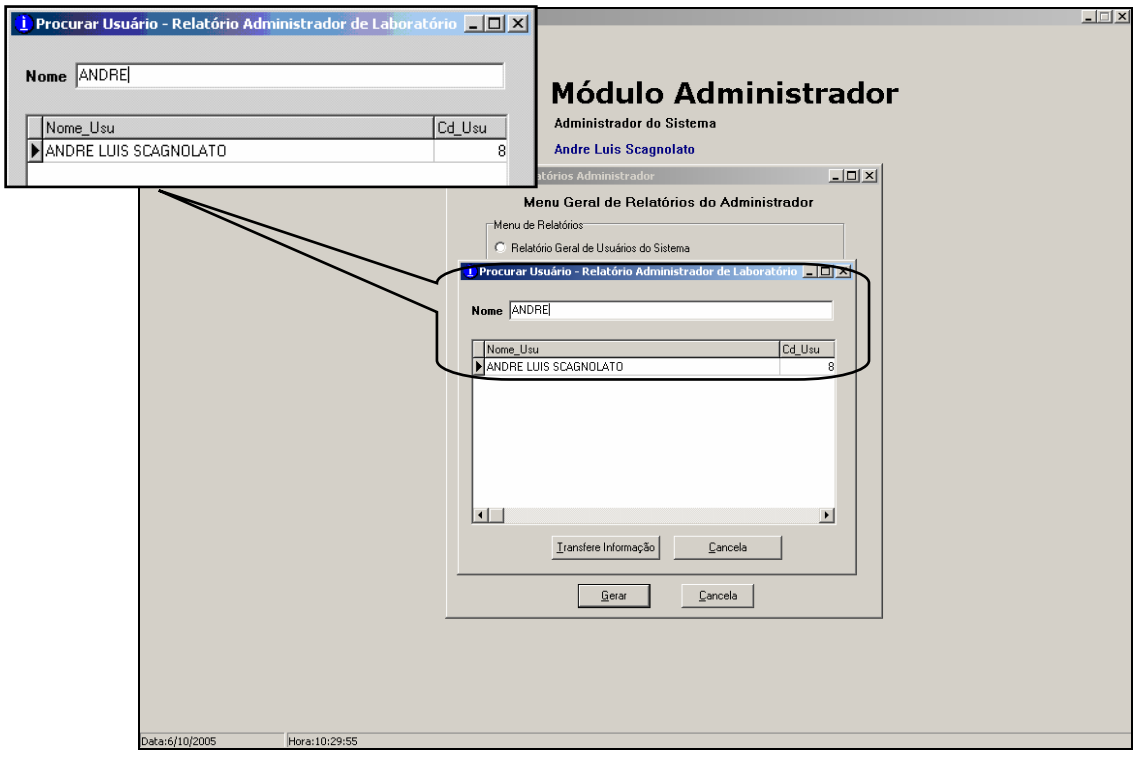


Figura 4.9 : Interface para procura de usuário - relatório de eventos do sistema por usuário.

Relatório geral de eventos do sistema: esse relatório exibe todos os acessos ocorridos no sistema, onde são apresentadas a data do evento, o horário, o nome do usuário, número do

iButton e a sua função. É necessário informar o laboratório de pesquisa que se deseja listar o relatório, ou escolher a opção para listagem geral. Apresenta-se na Figura 4.10 a interface de procura por laboratórios de pesquisa.

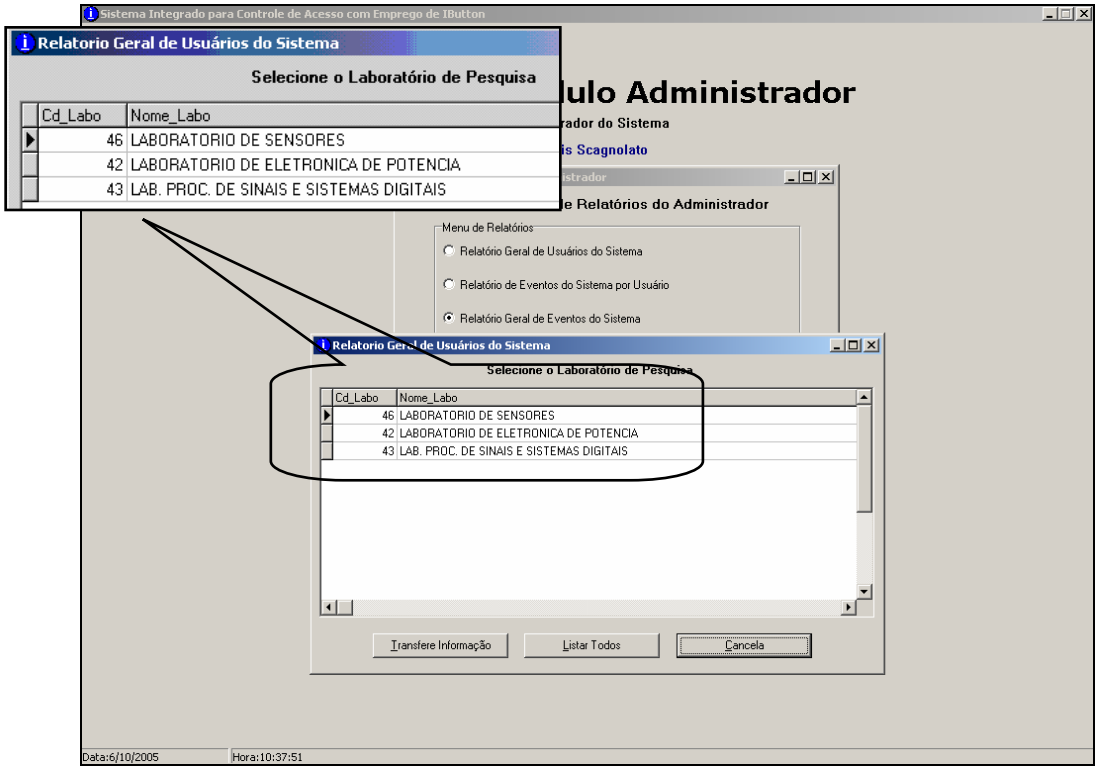


Figura 4.10 : Relatório geral de eventos do sistema - interface de procura por laboratório de pesquisa.

Relatório de Acessos Indevidos: Esse relatório fornece os acessos indevidos realizados pelos usuários, de acordo com o tipo de evento ocorrido. No ato de sua geração, é solicitado ao administrador escolher qual laboratório deseja verificar os eventos, ou, é facultada a escolha de todos os laboratórios. Após essa escolha, é solicitado que se opte por um determinado tipo de evento ou ainda, é possível listar todos os tipos de eventos ocorridos. Apresenta-se na Figura 4.11 a interface de escolha de tipos de eventos disponíveis.

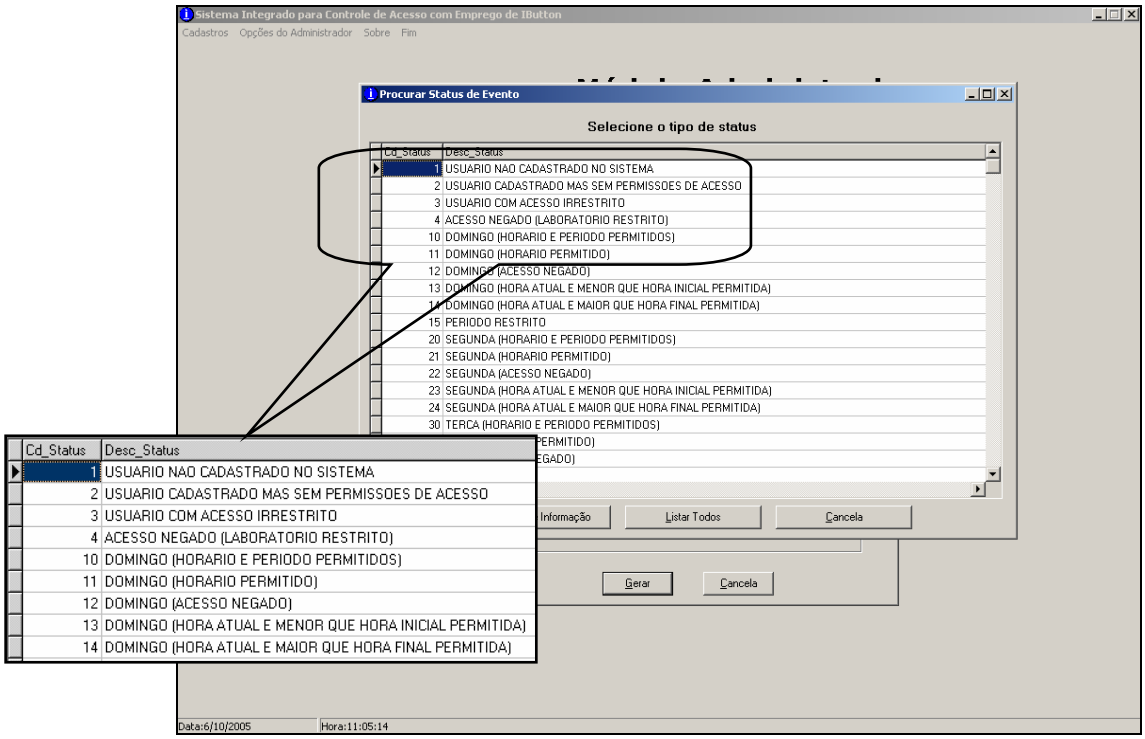


Figura 4.11 : Relatório de acessos indevidos - Interface de escolha de tipo de evento.

Relatório Individual de Usuários do Sistema: Esse relatório tem por finalidade exibir todos os dados relacionados a um determinado usuário. Nesse relatório são exibidos todos os seus dados pessoais. É necessário informar qual laboratório o usuário pertence e qual o seu nome. Para isso, é disponibilizada uma ferramenta de procura, apresentada na Figura 4.12.

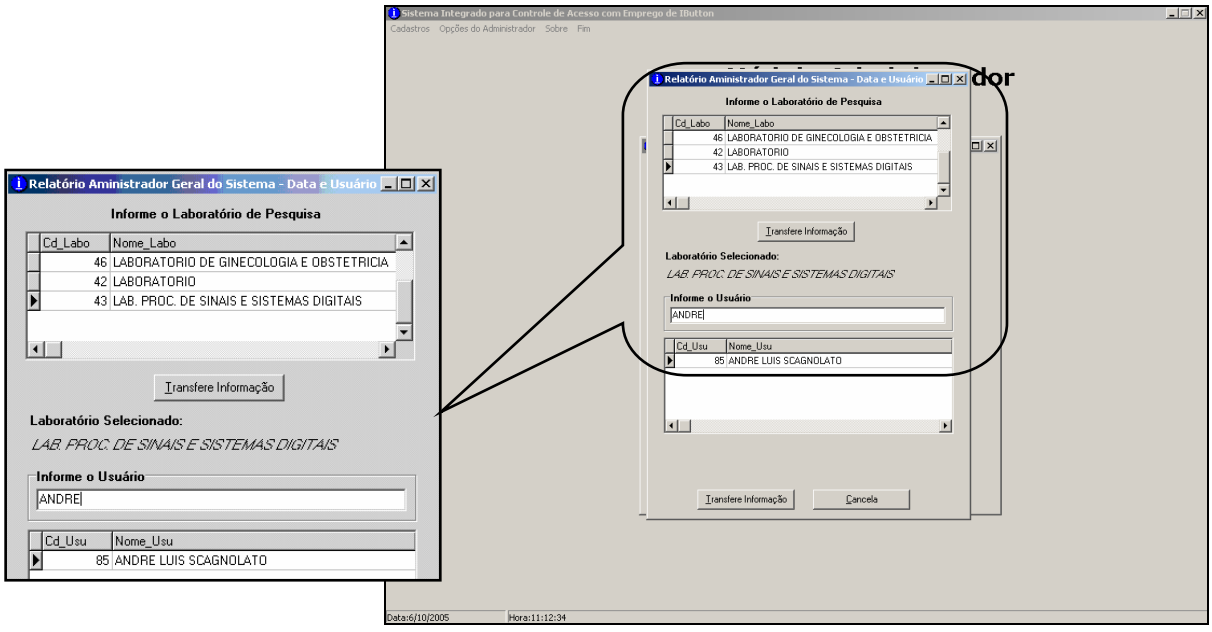


Figura 4.12 : Relatório individual de usuários do sistema - Interface de escolha.

Relatório de Eventos do Sistema por Usuário e Data: Esse relatório permite listar todos os eventos de um determinado laboratório e um determinado usuário através de uma data informada. A interface de escolha de opções é apresentada na Figura 4.13.

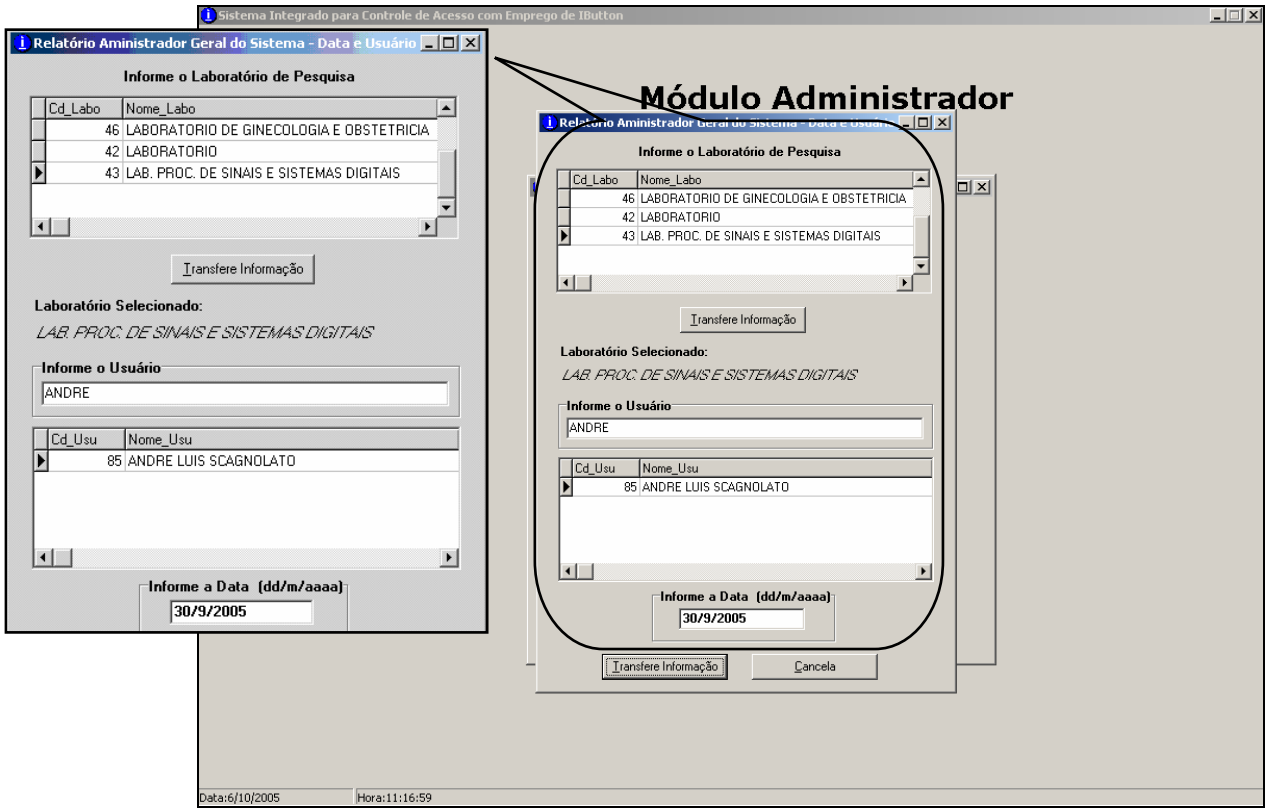


Figura 4.13 : Relatório de eventos do sistema por usuário e data – Interface de escolha.

Relatório Geral de Eventos por Período: Esse relatório fornece todos os eventos de um determinado laboratório, a partir de uma data informada. A interface de escolha é apresentada a partir da Figura 4.14.

Relatório Geral de Permissões: Esse relatório fornece as permissões de todos os usuários de um determinado laboratório. Para a sua escolha, o sistema fornece um formulário com as opções de laboratórios disponíveis. A interface de escolha é apresentada a partir da Figura 4.15.

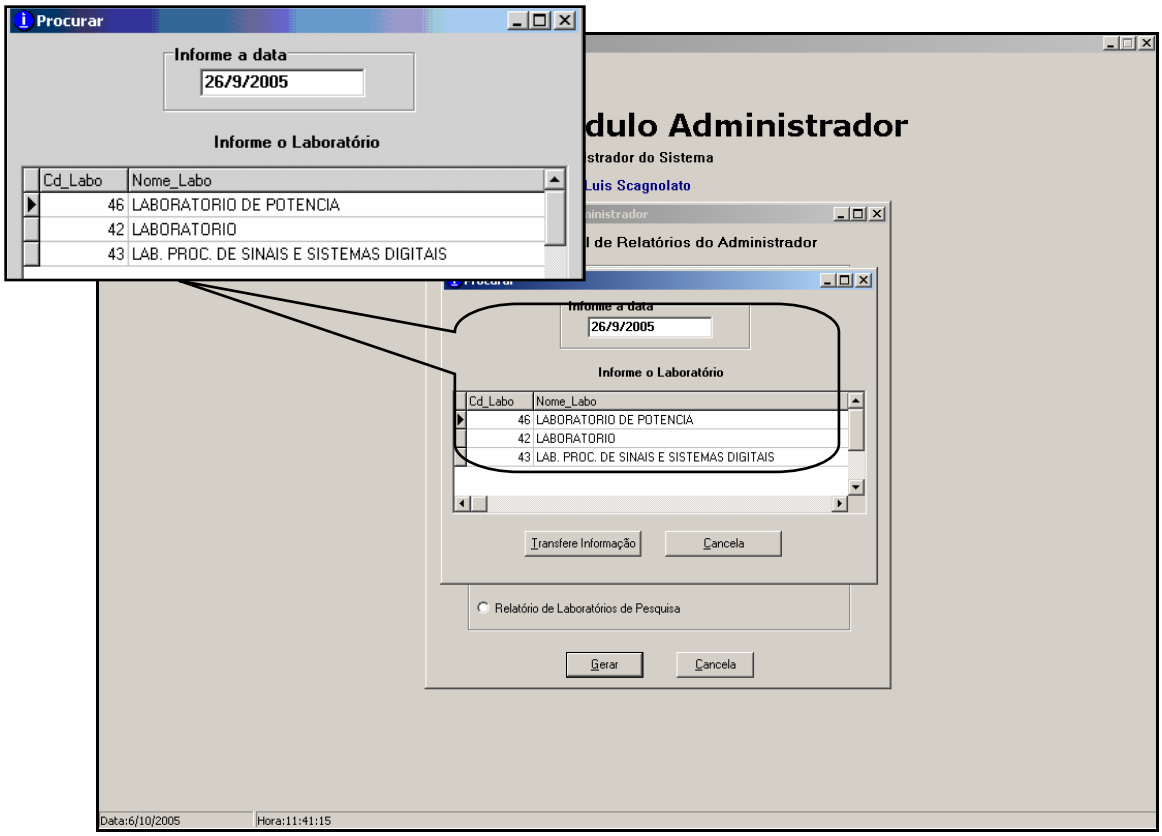


Figura 4.14 : Relatório geral de eventos do sistema por período – Interface de escolha.

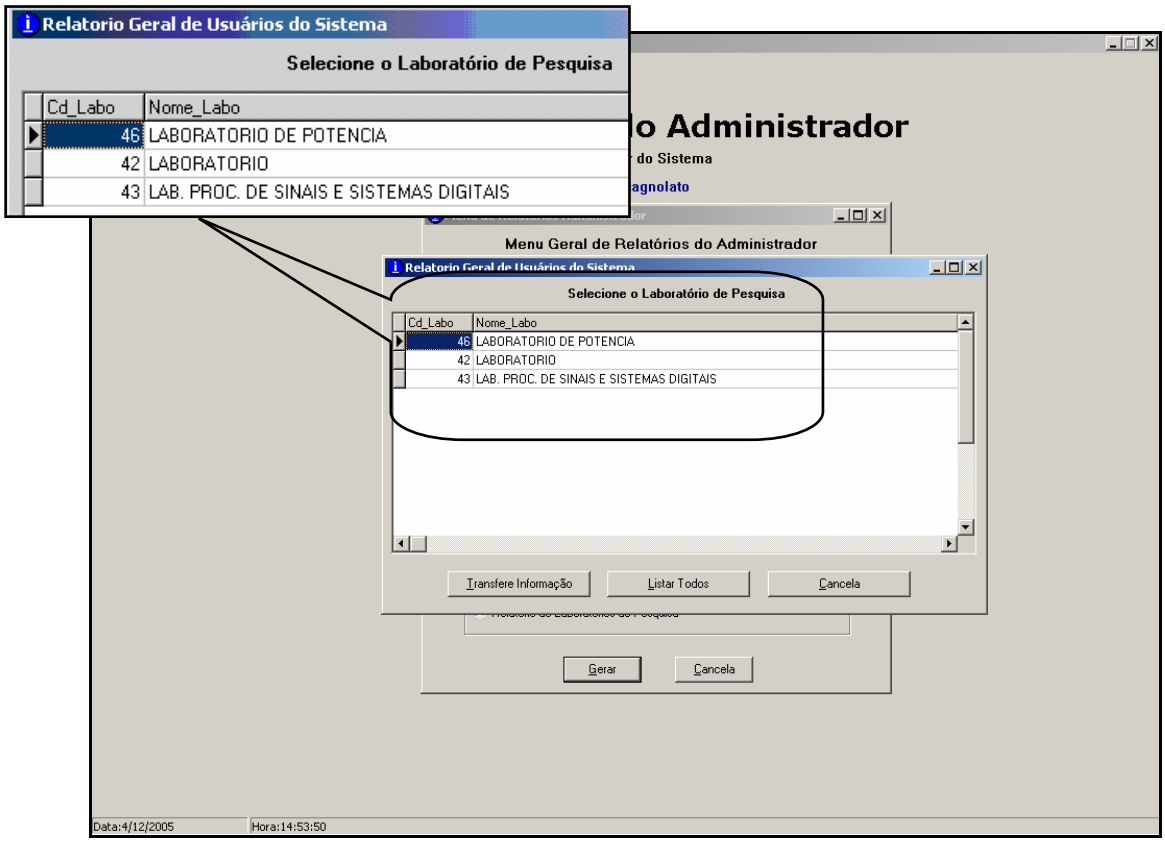


Figura 4.15 : Relatório geral de permissões – Interface de escolha.

Relatório de Permissões por usuário: Essa opção de relatório disponibiliza para o administrador as permissões de acesso individuais de um usuário. Para isso, é necessário informar no formulário de pesquisa, o laboratório ao qual o usuário pertence e o seu nome.

Relatório de Laboratórios de Pesquisa: Esse relatório fornece todos os laboratórios cadastrados no sistema, informando o nome, seu código, nome e *e-mail* do responsável.

4.2.1.7 Relatórios de Auditoria (Administrador de laboratórios)

Os relatórios de auditoria dos administradores de laboratórios são idênticos aos do administrador geral do sistema, todavia, somente serão exibidos os dados pertinentes ao seu laboratório de pesquisa. Por exemplo, um administrador de laboratório, com código "43" deseja gerar um relatório geral de usuários do sistema. O sistema somente ira lhe fornecer as informações pertinentes aos usuários cadastrados com o mesmo código, não possuindo acesso aos dados dos demais usuários.

4.2.2 Módulo Leitura

O módulo leitura foi desenvolvido para realizar a autenticação dos *iButton´s* no momento em que o usuário o toca na interface de comunicação *1-Wire* (*BlueDot*). O módulo leitura deve ser instalado no microcomputador que realiza o controle de entrada e saída de usuários, tendo a função de realizar a validação dos usuários, permitindo ou não o acesso dos mesmos às dependências do laboratório de pesquisa. O módulo leitura dispõe de rotinas de segurança, no caso de tentativa de acesso indevido, permitindo assim, um controle mais efetivo e restrito às dependências do laboratório de pesquisa.

No início da execução do módulo leitura, apresentado na Figura 4.16, é solicitado o código do laboratório de pesquisa o qual ele está realizando o controle. Caso o administrador não se lembre do código, o mesmo pode ser consultado através do botão "Procurar", localizado no lado direito do formulário de "*logon"*. O sistema apresenta um formulário com

todos os laboratórios cadastrados no sistema, bastando apenas “um clique” do *mouse* para selecionar o seu respectivo laboratório. Após a inserção do código, basta clicar em “*OK*” para inicializar o sistema.

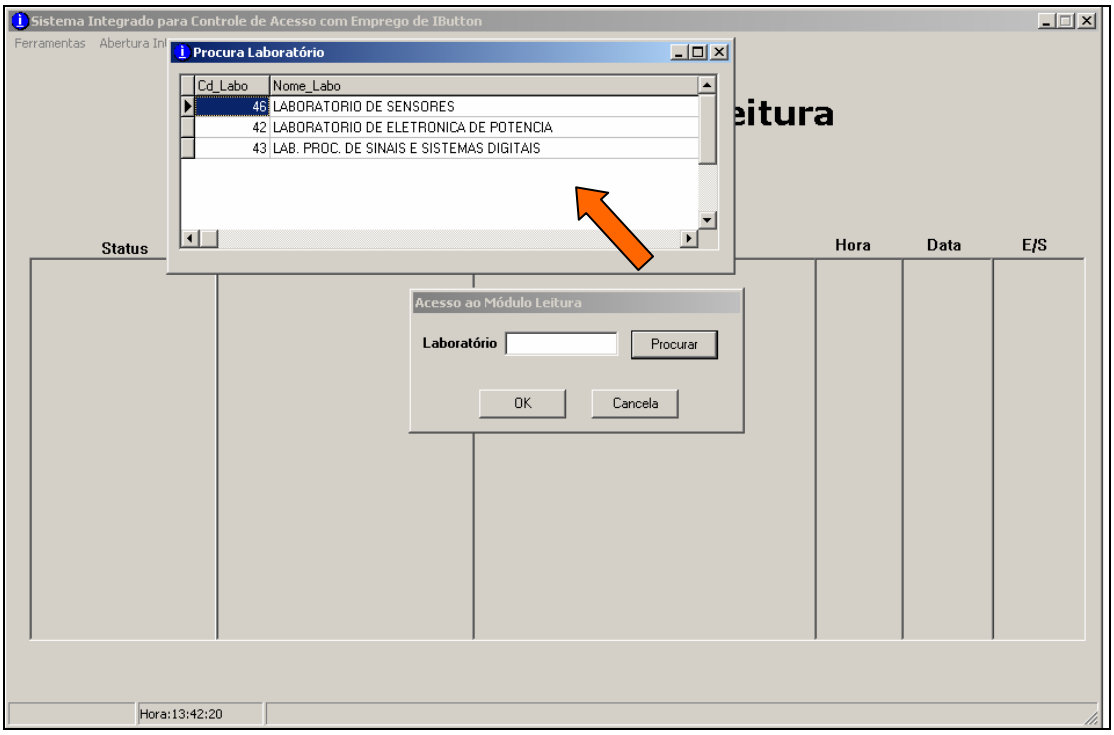


Figura 4.16 : Inicio da execução do módulo leitura.

Não é necessário deixar a janela minimizada ou maximizada para o módulo leitura continuar em operação. Mesmo fechando-se o formulário, o sistema permanece em atividade, até que seja explicitamente finalizado na opção “finalizar programa”. A indicação de sua atividade é apresentada através de um ícone localizado ao lado direito, próximo ao relógio padrão do sistema operacional *Windows*.

Clicando com o botão direito do mouse sobre o ícone, o sistema apresenta três opções. Na primeira opção “Restaurar”, o administrador tem a opção de restaurar a janela do módulo leitura, na segunda opção “Abertura Interna”, o sistema realiza a abertura automática da porta, sem a necessidade de se inserir um *iButton* na interface de comunicação “*Blue Dot*” e na terceira e última opção, “Finalizar Programa”, o módulo leitura é desativado. Na Figura 4.17 apresenta-se as opções do sistema, quando o mesmo encontra-se minimizado.

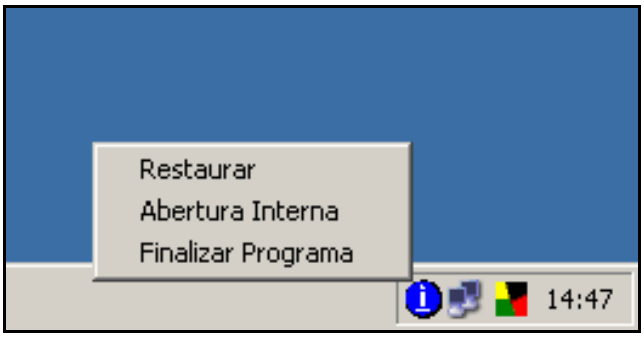


Figura 4.17 : Ícone do módulo leitura e suas respectivas funções.

Na interface principal, é possível visualizar todas as transações de entrada e saída dos usuários, onde é informado o status do evento, motivo, nome do usuário, data e hora. Todavia, essas informações contidas na tela servem somente para visualização, não podendo ser alteradas ou excluídas. No caso de uma possível manutenção, como a eliminação de registros antigos no banco de dados, essa tarefa só é possível ao administrador de banco de dados. Na Figura 4.18 apresenta-se a interface principal do módulo leitura. O menu principal disponibiliza quatro opções descritas nos itens a seguir.

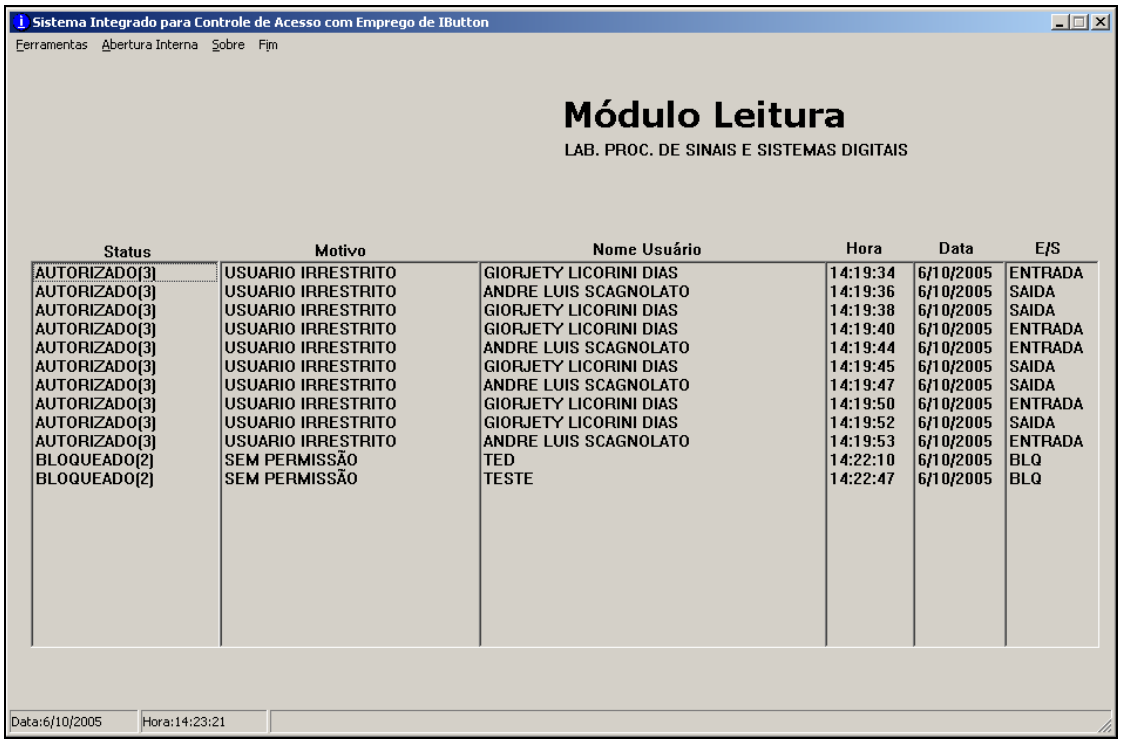


Figura 4.18: Interface principal do módulo leitura.

4.2.2.1 Menu Ferramentas – Limpar Eventos do Sistema

Essa opção permite que todos os eventos do sistema sejam apagados do formulário principal. Todavia, essas informações não são excluídas do banco de dados do sistema.

4.2.2.2 Menu Ferramentas – Configuração de *E-mail*

Durante uma tentativa de acesso indevido às dependências do laboratório de pesquisa, o módulo leitura pode ser configurado para enviar automaticamente um *e-mail* ao responsável pelo laboratório, informando a tentativa de acesso indevido. No conteúdo do *e-mail* são informados o número do *iButton*, o nome do usuário (caso ele esteja previamente cadastrado no banco de dados do sistema), o tipo de evento ocorrido, a data e a hora do evento. O cadastro do docente ao qual o sistema envia o *e-mail* é individual, ou seja, o sistema envia o *e-mail* somente para o docente responsável pelo laboratório de pesquisa. O acesso às configurações de *e-mail* somente é permitida ao administrador de laboratório, mediante a entrada de sua senha de acesso.

Nas configurações de *e-mail*, mostrado na Figura 4.19, devem ser informados, individualmente para cada docente, os seus dados de configuração. Os dados informados nesse formulário, servem para fornecer ao sistema as configurações básicas de *e-mail*, para onde deve ser enviado, no momento em que um usuário acessar o sistema indevidamente. Os dados a serem informados são:

- *E-mail* do destinatário (destinatário que ira receber o *e-mail* de tentativa de acesso indevido);
- Assunto (assunto do *e-mail*);
- Texto (texto de caráter livre);
- Remetente (endereço de *e-mail* do remetente);

- Nome do remetente;
- Nome do usuário (nome do usuário da conta de *e-mail*);
- Senha;
- Servidor de mensagens enviadas (*SMTP*);
- Porta (porta de acesso).

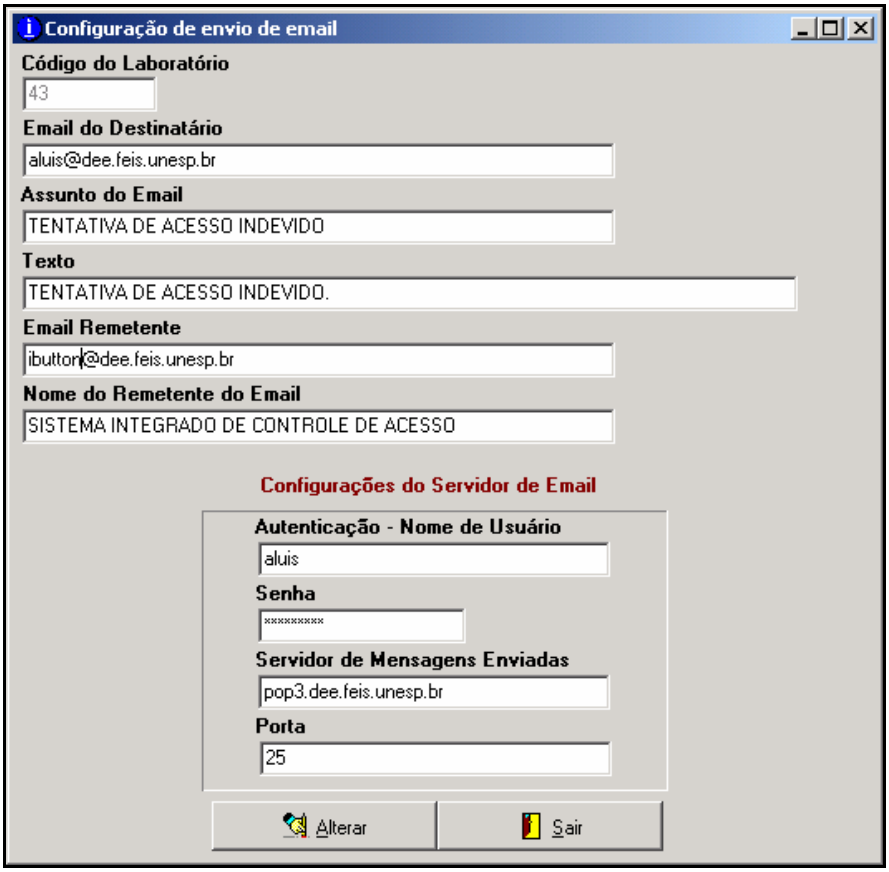


Figura 4.19 : Interface de configuração de *e-mail*.

4.2.2.3 Menu Ferramentas - Interface de Monitoramento por Vídeo

Como complemento dos recursos de segurança proporcionado pelo sistema, foi implementado a rotina de captura de imagem por uma câmera de vídeo. Tal implementação surgiu da necessidade de se identificar uma possível tentativa de acesso indevido quando um usuário mal intencionado inserisse um *iButton* que não estivesse cadastrado no banco de

dados. O sistema envia um e-mail ao administrador, contendo o número do *iButton*, data, horário, contudo, não era possível realizar a identificação visual do indivíduo que tentou realizar o acesso indevido.

Com a implementação dessa rotina de segurança, através de uma câmera localizada no lado externo do laboratório de pesquisa, no momento em que um indivíduo inserir um *iButton* na interface de leitura e o mesmo não encontrar-se cadastrado no banco de dados do sistema, a câmera é automaticamente acionada, sendo sua imagem capturada, gerando um arquivo e anexado-o no *e-mail* enviado ao administrador. Na Figura 4.20 apresenta-se a interface de captura de vídeo e na Figura 4.21 apresenta-se o e-mail enviado ao administrador com o arquivo em anexo.

Figura 4.20 : Interface de captura de vídeo.

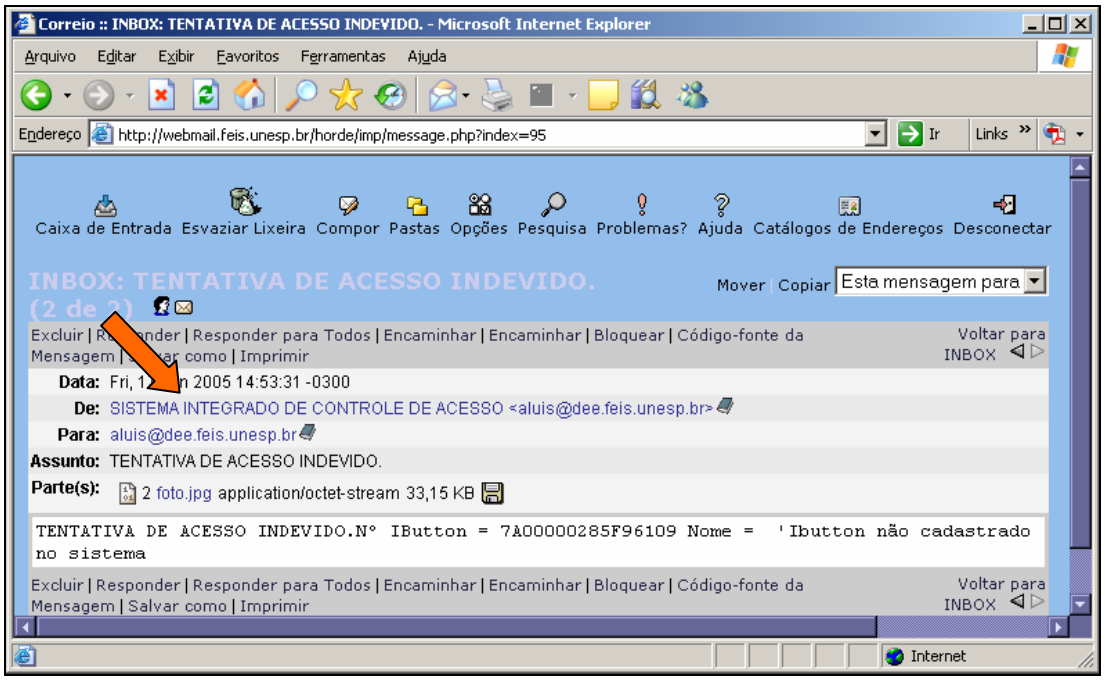


Figura 4.21 : E-mail enviado ao administrador com foto em anexo.

4.2.2.4 Menu Ferramentas – Localizar Usuário

Essa opção do sistema possibilita que um usuário seja localizado, estando ele em qualquer laboratório de pesquisa. Para isso basta entrar com o seu nome ou com suas iniciais, onde o formulário apresenta a relação dos usuários. Para selecionar o usuário, basta dar um "*click*" com o *mouse* sobre o nome escolhido. O sistema automaticamente mostra os locais e os acessos realizados pelo mesmo naquele dia. Na Figura 4.22 apresenta-se a interface de localização de usuários.

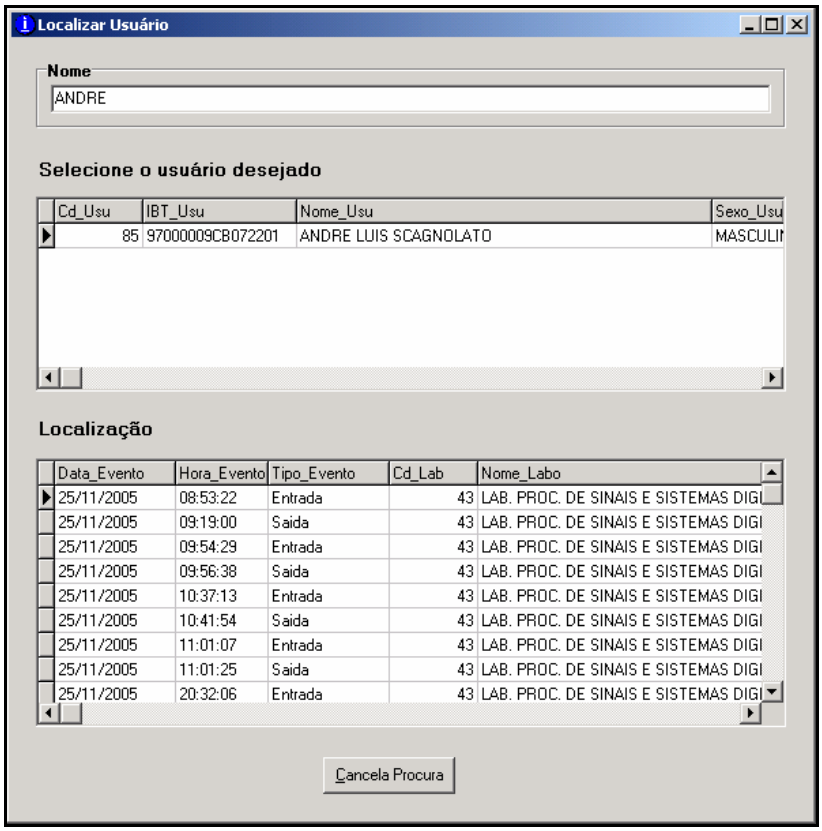


Figura 4.22 : Formulário de localização de usuários.

4.2.2.5 Menu Abertura Interna

A abertura interna possibilita que o fecho eletromagnético seja acionado sem a utilização do *iButton*. Essa opção é bastante útil, pois possibilita a abertura da porta sem a necessidade de se dirigir a mesma para inserir o *iButton* no leitor "*Blue Dot*". A opção pode ser acionada pela combinação das teclas quentes "ALT + A".

4.2.2.6 Menu Sobre e Menu Fim

O menu "Sobre" fornece informações, a título de comentário, sobre o sistema (autor, versão, contato, etc.) e o menu "Fim" realiza o encerramento do sistema.

4.2.3 Estrutura de Funcionamento do Módulo Leitura

A partir do instante em que o módulo leitura realiza a leitura do número de série do *iButton*, o sistema verifica no banco de dados se o mesmo encontra-se previamente cadastrado

para algum usuário. Sendo validado o número de série do *iButton*, o sistema verifica se o usuário possui permissões de acesso ao laboratório de pesquisa (dia da semana, horário, período e laboratório). Após a rotina de verificações, se o *iButton* estiver cadastrado e com permissões aceitas, o sistema envia um sinal pela interface de comunicação paralela que aciona o fecho eletromagnético, permitindo o ingresso do usuário às dependências do laboratório de pesquisa, registrando o acesso do usuário no banco de dados para possíveis fins de auditoria (hora, data, número do *iButton* e status do evento). Todas as verificações e acessos à base de dados do sistema são realizados por comandos em *SQL*.

Na Tabela 4.5 é apresentado um trecho do programa em que é verificada a permissão de acesso do usuário no domingo, onde são comparados os horários iniciais e finais permitidos. Na hipótese da condição ser satisfeita, o programa envia um sinal pela interface de comunicação paralela do microcomputador, consequentemente, o acionamento do fecho eletromagnético.

Tabela 4.5 :Trecho de programa para verificação das permissões do usuário.

```
IBQuery2.SQL.Clear;
IBQuery2.Close;
IBQuery2.SQL.Clear;
//Seleciona na "TBRes" se usuário tem permissão
IBQuery2.SQL.Add('select * from "TBRest" where "IBT_Usu" = '+""+VerificaIBT+"");
IBQuery2.Open;
If (IBQuery2.FieldByName('Domingo').AsBoolean = True) Then
   If ((time >= IBQuery2.FieldByName('HrIni_Dom').AsDateTime) and
     (time <= IBQuery2.FieldByName('HrFin_Dom').AsDateTime)) Then
        If (IBQuery2.FieldByName('Periodo').AsBoolean = True) Then
           If ((date >= IBQuery2.FieldByName('Per_Ini').AsDateTime) and
              (date <= IBQuery2.FieldByName('Per_Fin').AsDateTime)) Then
                 Begin
                   //Rotina para abertura da porta utilizando a interface paralela do PC
                   VarPino := 3;
```

```
                IOPort1.PortAddress := $378;
                IOPort1.PortData := VarPino;
                IOPort1.Write;
                MessageBeep(1);
                ListBox1.Items.Add('AUTORIZADO(10)');
                ListBox2.Items.Add('DOMINGO/HORARIO PERMITIDO');
                ListBox3.Items.Add(IBQuery1Nome_Usu.AsString);
                ListBox4.Items.Add (TimeToStr(time));
                ListBox5.Items.Add (DateToStr(date));
                IBQuery1.Close;
                IBQuery1.SQL.Clear;
                 IBQuery1.SQL.Add ('insert into "TBEvento"
                ("IBT_Usu","Hora_Evento","Data_Evento","StatusEvento") values
                ('+""+VerificaIBT+""+','+""+TimeToStr(time)
                 +""+','+""+DateToStr(date)+""+',10)');
                IBQuery1.Open;
              End;
```

No caso de uma tentativa de acesso indevido, como a utilização de um *iButton* que não se encontra cadastrado, ou o acesso fora de seu perfil de permissões, o sistema registra o evento no banco de dados do sistema (hora, data e número do *iButton*). O sistema nesse instante envia um e-mail para o responsável pelo laboratório de pesquisa, informando a tentativa de acesso indevido. O fluxograma apresentado na Figura 4.23 ilustra o funcionamento do módulo leitura na verificação da entrada do usuário.

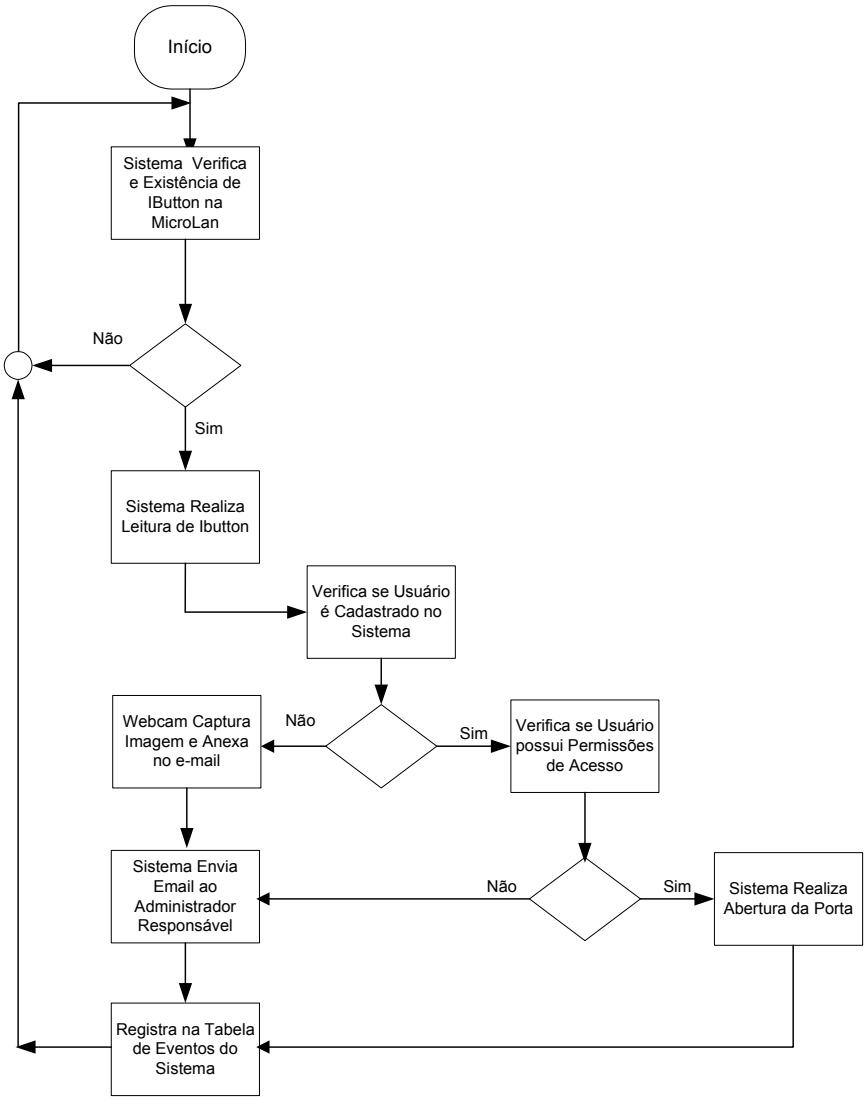


Figura 4.23: Fluxograma de funcionamento do módulo leitura na entrada de usuários.

4.2.4 Eventos do Sistema

São considerados eventos do sistema todas as transações de entrada, saída e tentativa de acessos indevidos as dependências do laboratório de pesquisa. Os registros de eventos do sistema são armazenados em uma tabela específica, no banco de dados, onde cada evento possui um nível de status, o qual especifica o tipo de evento, ocorrido, onde são registrados, além do código de status do evento, a hora, a data e o número do *iButton*. A partir da tabela de eventos é possível a emissão de diversos tipos de relatórios de auditoria.

Na Tabela 4.6 apresenta-se os níveis de status para cada evento do sistema, verificados durante os acessos às dependências do laboratório de pesquisa.

Tabela 4.6: Descrição dos níveis de status.

Status	Descrição	Status	Descrição
1	Usuário não cadastrado no sistema	40	Quarta (Horário e período permitidos)
2	Usuário cadastrado no sistema, porém sem permissões de acesso as dependências do laboratório de pesquisa	41	Quarta (Horário permitido)
3	Usuário com acesso irrestrito	42	Acesso negado na quarta
4	Acesso negado, laboratório de pesquisa restrito	43	Quarta (hora atual é menor que hora inicial permitida)
10	Domingo (Horário e período permitidos)	44	Quarta (hora atual é maior que hora final permitida)
11	Domingo (Horário permitido)	50	Quinta (Horário e período permitidos)
12	Acesso negado no domingo	51	Quinta (Horário permitido)
13	Domingo (hora atual é menor que hora inicial permitida)	52	Acesso negado na quinta
14	Domingo (hora atual é maior que hora final permitida)	53	Quinta (hora atual é menor que hora inicial permitida)
15	Período Restrito (data)	54	Quinta (hora atual é maior que hora final permitida)
20	Segunda (Horário e período permitidos)	60	Sexta (Horário e período permitidos)
21	Segunda (Horário permitido)	61	Sexta (Horário permitido)
22	Acesso negado na segunda	62	Acesso negado na sexta
23	Segunda (hora atual é menor que hora inicial permitida)	63	Sexta (hora atual é menor que hora inicial permitida)
24	Segunda (hora atual é maior que hora final permitida)	64	Sexta (hora atual é maior que hora final permitida)
30	Terça (Horário e período permitidos)	70	Sábado (Horário e período permitidos)
31	Terça (Horário permitido)	71	Sábado (Horário permitido)
32	Acesso negado na terça	72	Acesso negado no sábado
33	Terça (hora atual é menor que hora inicial permitida)	73	Sábado (hora atual é menor que hora inicial permitida)
34	Terça (hora atual é maior que hora final permitida)	74	Sábado (hora atual é maior que hora final permitida)

4.3 Acesso Via WWW (World Wide WEB)

É disponibilizado para o administrador, o acesso ao sistema via *WEB*. Com esse recurso é possível realizar a verificação dos usuários que estão utilizando o laboratório de pesquisa.

Para o desenvolvimento desse recurso, foi utilizada a linguagem de criação de *script PHP* (*Powered Hyper Processor*) [24], criada para o desenvolvimento de aplicações *WEB*. O desenvolvimento da página de acesso e página de usuários ativos foi desenvolvido em *HTML*

(*Hyper Text Transfer Protocol*), onde é possível inserir o código *PHP*, que é interpretado toda vez que a página for visitada.

Em todos os acessos realizados via *WEB*, o bando de dados do sistema é consultado, exibindo as informações para o administrador. O acesso é garantido mediante o fornecimento de um nome de usuário e senha de acesso.

O endereço para acesso ao sistema via *WEB* é fornecido colocando-se o endereço de acesso no navegador padrão. A interface de acesso é apresentada na Figura 4.24.

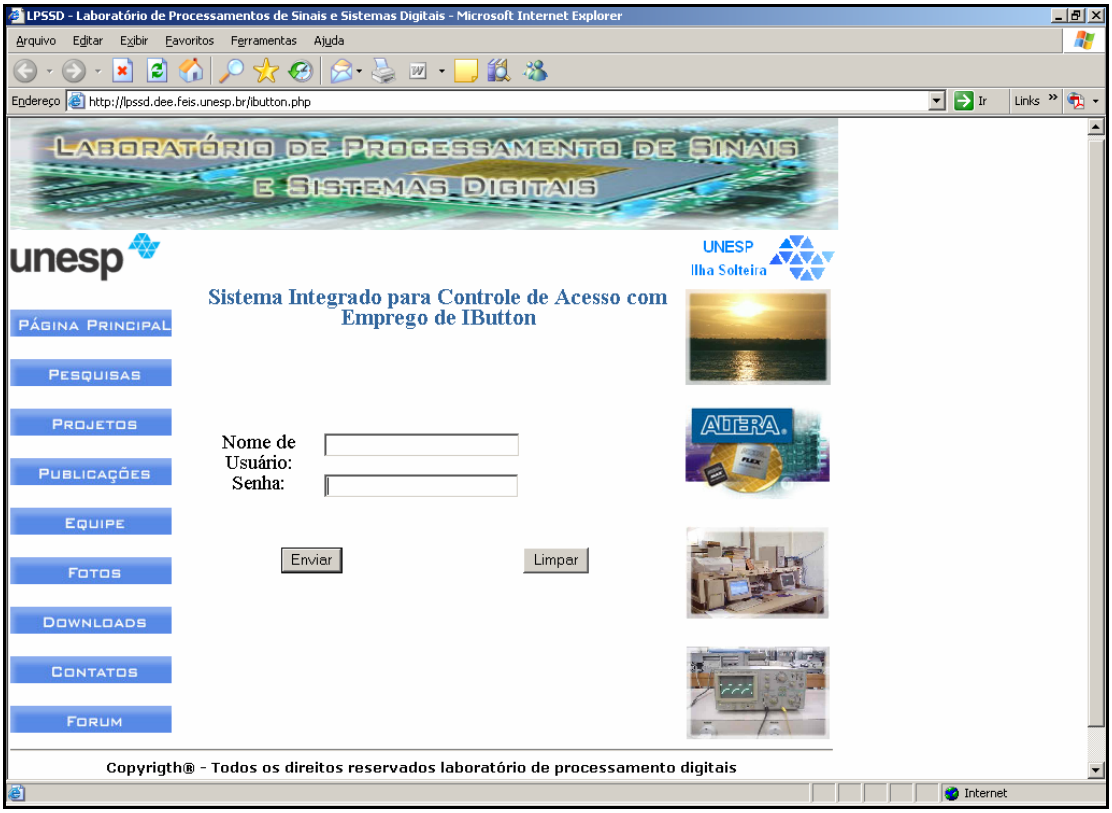


Figura 4.24 : Interface de acesso ao sistema via *WEB*.

Após a validação do nome de usuário e senha, o sistema apresenta os usuários que realizaram acesso ou se encontram no laboratório de pesquisa, informando a data de acesso, nome do usuário, o status (entrada ou saída) e o laboratório de pesquisa acessado.

4.4 Estrutura de Hardware Utilizada

Para a implantação física do sistema nas dependências do Laboratório de Processamento de Sinais e Sistemas Digitais, UNESP, Ilha Solteira, foi necessária a aquisição dos componentes apresentados na Tabela 4.7.

Tabela 4.7 : Componentes utilizados para a implantação do sistema.

Quantidade	Componente
2 Unidades	Interface de comunicação *1-wire* (*Blue Dot*) modelo DS1402D
1 Unidade	Adaptador serial modelo DS9097E
1 Unidade	Circuito de potência
1 Unidade	Fecho eletromagnético
15 Unidades	*iButton* modelo DS1990A
5 Metros	Cabo par trançado categoria 5
1 Unidade	*Microcontrolador Motorola M68HC08*

4.4.1 Interface de comunicação 1 - W*ire* (*Blue Dot*) modelo DS1402D

Para a comunicação dos *iButton´s* com o sistema foi utilizada a interface de comunicação *1-Wire* (*Blue Dot*) modelo DS1402D, a qual é ligada ao adaptador serial através de um conector modelo RJ-11. Na Figura 4.25 apresenta-se o *Blue Dot* utilizado.

Figura 4.25 : *Blue Dot* modelo DS1402D.

4.4.2 Adaptador serial modelo DS9097E

O *iButton* pode realizar a comunicação com o microcomputador por qualquer uma das interfaces de comunicação (paralela, serial ou *USB*) do microcomputador. O adaptador

escolhido para a implantação do sistema foi o modelo DS9097E, o qual é utilizado para a interface serial de 25 pinos. Com a utilização do microcontrolador, o uso do adaptador serial pode ser dispensado. Na Figura 4.26 apresenta-se o adaptador serial DS9097E.

Figura 4.26 : Adaptador serial modelo DS9097E.

4.4.3 Circuito de Potência

O fecho eletromagnético é acionado com um sinal emitido pela interface de comunicação paralela do microcomputador. Como a tensão de saída da interface paralela se mantém em torno de 5 V, e a tensão requerida para o acionamento do fecho eletromagnético é de 12 V, foi necessário utilizar um circuito de potência [25], o qual consiste em um relê de contato que é acionado pelo sinal emitido pela interface paralela do microcomputador.

Quando o relê é acionado, ele libera uma tensão de 12 V por um tempo determinado acionando o fecho eletromagnético com a tensão adequada. O circuito é composto por dois relês, um capacitor e um diodo. Na Figura 4.27 apresenta-se o esquemático do circuito de potência.

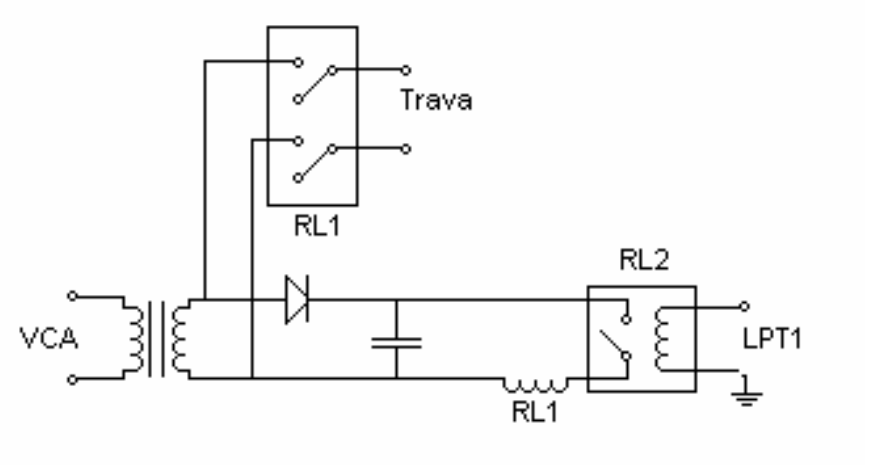


Figura 4.27 : Esquemático circuito de potência.

O sinal emitido pela interface de comunicação paralela aciona o relê dois (RL2, na Figura 4.27), que sequencialmente, aciona o relê um (RL1 na Figura 4.27), acionando o fecho eletromagnético. Na Figura 4.28 apresenta-se o circuito de potência.

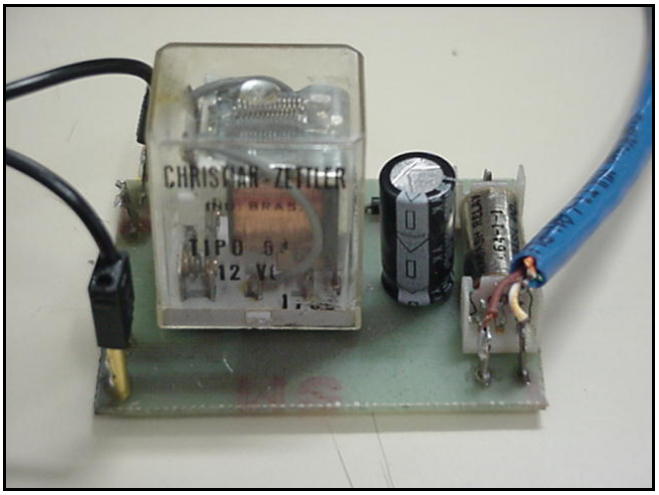


Figura 4.28 : Circuito de potência.

4.4.4 Fecho Eletromagnético

O fecho eletromagnético foi instalado no batente da porta do laboratório de pesquisa, onde o mesmo é ligado ao circuito de potência por meio de um do cabo par trançado nível 5. Na Figura 4.29 apresenta-se o fecho eletromagnético fixo no batente da porta de acesso. A maçaneta da porta foi travada internamente, não sendo possível a sua abertura com a mão.

Figura 4.29 : Fecho eletromagnético instalado na batente da porta do laboratório de pesquisa.

As características físicas do fecho eletromagnético consistem no uso de alimentação 12Vca, potência de 7,2 W, corrente nominal 0,7 A e resistência de 12,5 Ω a 20º C.

4.4.5 *iButton* DS1990A

O *iButton* utilizado para controle de acesso foi o modelo DS1990A [14] por ser o modelo mais simples e mais barato disponível pelo fabricante, todavia, pode ser utilizado qualquer modelo de *iButton* para uso no sistema, pois todos os modelo possuem um número de série único.

4.4.6 Microcontrolador *Motorola M68HC908*

Para a realização da leitura do número de *ROM* do *iButton*, foi utilizado um microcontrolador *Motorola M68HC908*, o qual foi programado para realizar a leitura do número de série no momento em que o usuário o toca na interface de comunicação *1-Wire* (*Blue Dot*).

A leitura é realizada através da utilização do protocolo de comunicação *1-Wire* (1-Fio), o qual se comunica com o microcomputador através da interface de comunicação paralela, onde a informação transmitida é dividida em pequenas partes (*bits*), que são enviadas ao dispositivo receptor uma após a outra (série).

4.4.6.1 Estrutura de Funcionamento do *Software*

O microcontrolador foi programado em linguagem *Assembler*, onde foram implementadas rotinas para a realização da leitura do número de *ROM* do *iButton*, utilizando o protocolo *1-Wire* e rotinas para o envio de informações pela interface de comunicação paralela do microcomputador. No fluxograma da Figura 4.30 apresenta-se a estrutura de funcionamento do *software* implementado no microcontrolador.

O dispositivo mestre envia um pulso de *reset* de 500 microssegundos para o dispositivo escravo (*iButton*), o qual, acusa a sua presença com o envio de um pulso de presença para o dispositivo mestre. Não sendo acusada a presença do dispositivo mestre, o programa permanece em *looping* constante até que se detecte a presença do dispositivo escravo.

Após a verificação da existência do dispositivo escravo, é inicializado o procedimento para a leitura do número de *ROM* do *iButton*, com o envio do comando *Read ROM* (33 hexadecimal), correspondendo a 8 *bits*, ou 8 pulsos enviados pelo dispositivo mestre ao dispositivo escravo. Esse processo é executado 64 vezes, controlado por um contador no programa, pois a cada leitura é adquirido o correspondente a 1 *bit* do número de *ROM* do *iButton*. A cada leitura de *bit*, o mesmo é armazenado em memória até que se obtenha o número completo.

Sequencialmente, após a etapa de leitura do número de *ROM*, o dispositivo mestre inicializa a rotina para envio do número de serie do *iButton bit* a *bit* pela interface de comunicação paralela do microcomputador, buscando cada *bit* armazenado em memória e enviando-os um a um, onde, para isso, o programa utiliza um contador, executando esse processo durante 64 vezes até que seja enviado o número de *ROM* por completo.

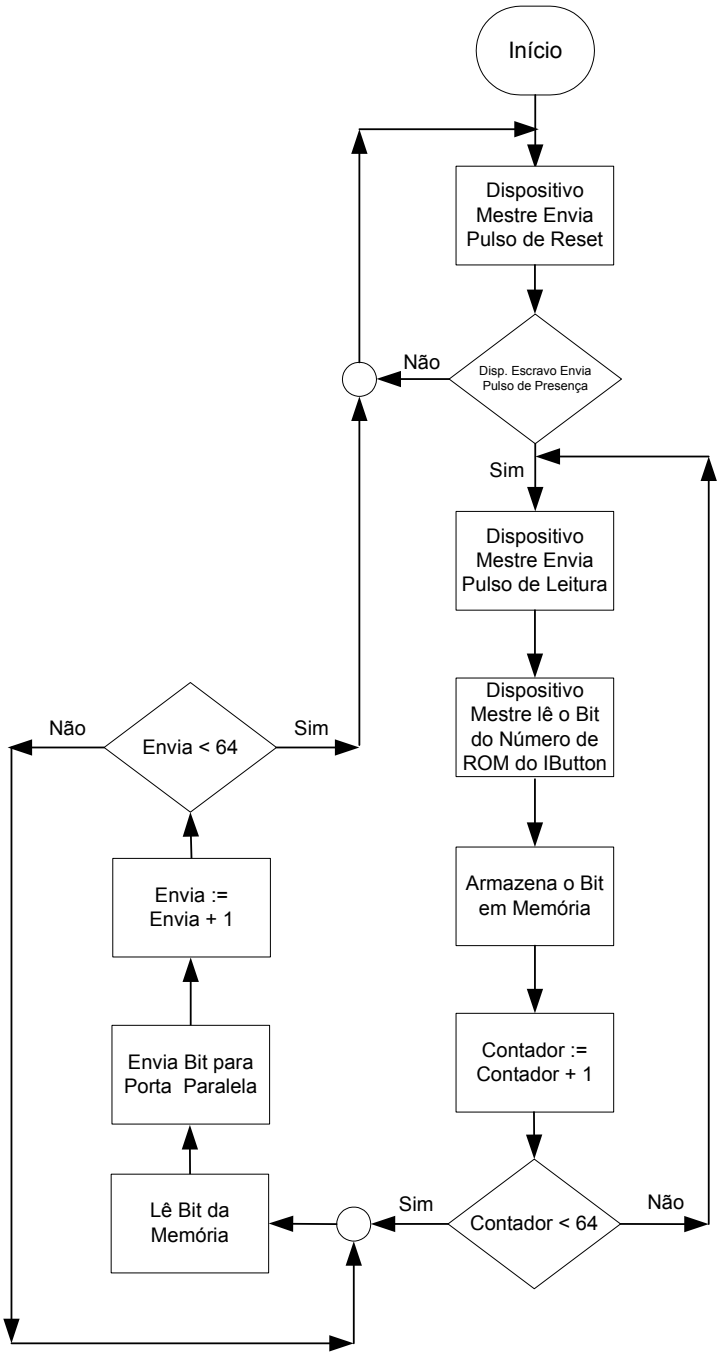


Figura 4.30 : Fluxograma de funcionamento do microcontrolador.

O *software* (módulo leitura), por sua vez, lê os *bits* um a um durante 64 vezes enviados pelo dispositivo mestre, armazenando-os em um vetor, onde logo após são inicializadas as rotinas para verificações de permissões de acesso do usuário.

4.5 Implantação do Sistema

Após a fase de testes de bancada, o sistema foi instalado no laboratório de Processamento de Sinais e Sistemas Digitais, Departamento de Engenharia Elétrica – FEIS-UNESP, controlando o acesso dos alunos as dependências do laboratório de pesquisa. O sistema foi instalado em um microcomputador *Intel Pentium 4* com 2.4 *Ghz*, 256 *Mb* de memória *RAM* e sistema operacional *Windows XP*. Foram instalados os módulos de leitura administrador, e o sistema gerenciador de banco de dados *Interbase Client* versão 6.0. O *Interbase Server* versão 6.0 e o banco de dados do sistema foram instalados em um microcomputador AMD 450 *Mhz*, 128 *Mb* de memória *RAM*, com sistema operacional *Linux Slackware 9.0*, localizado na rede local.

O sistema dispõe de um leitor (*Blue Dot*) localizado no lado externo e interno do laboratório de pesquisa, mostrado na Figura 4.31, onde os usuários inserem seu *iButton* no momento da entrada e saída das dependências do laboratório de pesquisa

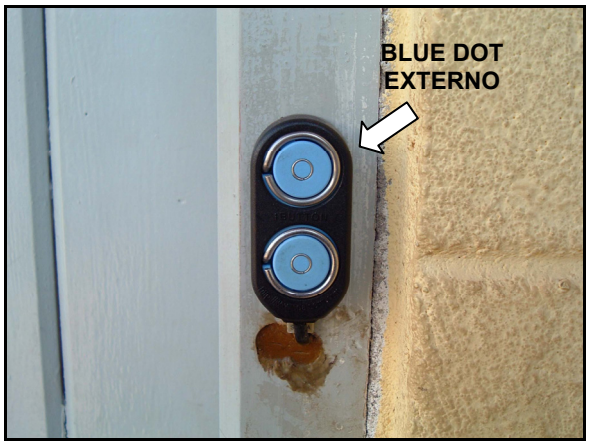


Figura 4.31 :*Blue Dot* localizado no lado externo do laboratório.

A estrutura física de instalação do sistema foi estruturada de modo a disponibilizar uma chave de emergência para uma eventual falta de energia, sendo a mesma, fixa no lado interno do laboratório. Para as ligações físicas entre o microcomputador, leitores e adaptadores foi utilizado o cabo par trançado categoria 5. Dessa forma, foi utilizado um fio para o sinal

positivo proveniente dos "*Blue Dot´s*" e um para cada terra, além de dois fios para cada pólo do fecho eletromagnético. Na Figura 4.32 apresenta-se a instalação física interna, contendo a chave de emergência, o adaptador serial e o *Blue Dot* interno.

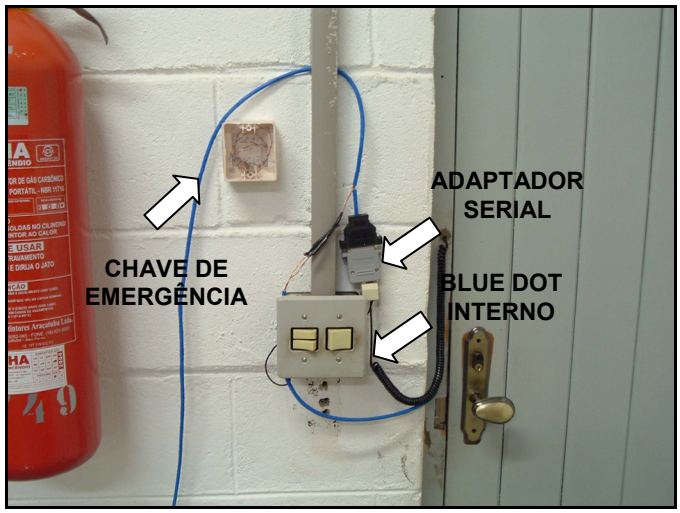


Figura 4.32 :Instalação física interna.

Capítulo 5 – Metodologia de Avaliação e Desempenho do *Software*

5.1 Introdução

Nesse capítulo são abordados os principais conceitos teóricos aplicados para a realização de teste de *software*, procurando realizar a identificação e eliminação de erros que persistem, representando a última revisão da especificação, projeto e codificação.

5.2 Teste, Qualidade e Desempenho do Software.

O processo de teste de *software* [27] é um elemento crítico da garantia de qualidade de *software*, e representa a ultima revisão de especificação, projeto e codificação. O teste de *software* é uma atividade minuciosa que visa encontrar possíveis erros de projeto e falha de estruturas lógicas. Cerca de 50% do tempo gasto no desenvolvimento de um *software* e 50% de seu custo total são gastos com testes de programas e sistemas em desenvolvimento. Um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto [28].

A qualidade de *software* [27] é uma atividade aplicada ao longo de todo o processo de engenharia de *software*, abrangendo métodos e ferramentas de análise, projeto e codificação e testes, revisões técnicas formais que são aplicadas durante cada fase de engenharia de *software*, estratégias de testes de múltiplas fases, controle da documentação e das mudanças feitas nela [29].

Os fatores que afetam a qualidade do *software* podem ser descritos em dois grupos amplos:

- Fatores que podem ser medidos diretamente (erros e unidades de tempo);

- Fatores que podem ser medidos apenas indiretamente (usabilidade e manutenibilidade).

Para todos esses casos, deve haver medição, onde se deve comparar o *software* (documentos, programas e dados) com algum outro dado e chegar a uma indicação de qualidade.

5.2.1 Fundamentos de Teste de *Software*

Durante a etapa de testes [27], o engenheiro de *software* tem a função de criar uma série de casos de testes, com a intenção de "demolir" o *software* que ele construiu. A atividade de teste de *software* é um processo da engenharia de *software* que pode ser vista como um processo destrutivo, em vez de construtivo.

A atividade de teste exige que o programador/desenvolvedor descarte noções preconcebidas de perfeição durante a fase de implementação de *software*, procurando encontrar de todas as maneiras possíveis erros no *software*.

5.2.2 Objetivos da Atividade de Teste

O objetivo principal da atividade de testes [30] é projetá-los para que descubram sistematicamente diferentes classes de erros e façam-no com uma quantidade de tempo e esforço mínimos. Se a atividade de teste for conduzida com sucesso, ela descobrirá erros no *software*.

A atividade de teste [30] demonstrará que as funções do *software* aparentemente estão trabalhando de acordo com as especificações, que os requisitos de desempenho aparentemente foram cumpridos. Contudo, os dados compilados quando a atividade de testes é levada a efeito, proporcionam uma boa indicação da confiabilidade e alguma indicação da qualidade do *software* como um todo.

Deve-se levar em consideração que a atividade de teste não pode mostrar a ausência de *bugs*, ela só pode mostrar se defeitos de *software* estão presentes [31].

5.3 Fluxo de Informações de Teste

O fluxo de informações da atividade de teste [30] segue o padrão descrito na Figura 5.1, onde duas classes de entradas são fornecidas ao processo de teste, a primeira é chamada de configuração de *software*, que inclui uma especificação de requisitos de *software*, uma especificação de projeto e o código fonte. A segunda classe de entrada é a chamada configuração de teste, que inclui um plano de procedimento de teste, as ferramentas de teste, que serão usadas e casos de teste e seus resultados esperados.

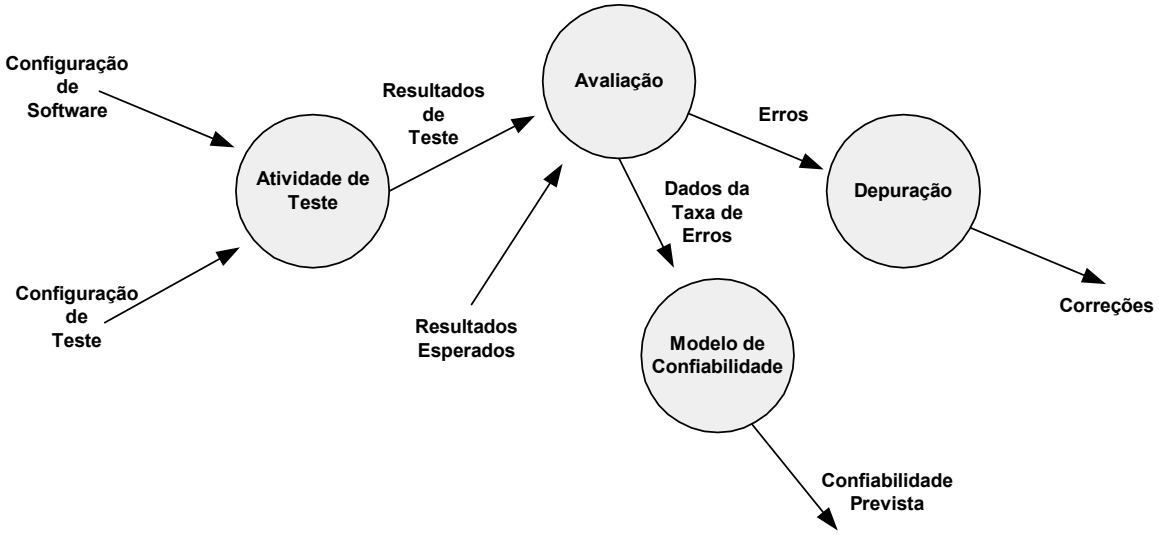


Figura 5.1 : Fluxo de informações da atividade de teste de *software*.

Quando a seqüência de testes é realizada [30], e todos os resultados são avaliados, os resultados de testes são comparados com os resultados esperados. Quando dados errôneos são identificados, infere-se a presença de erro, e inicia-se o processo de depuração.

À medida que os resultados de teste são reunidos e avaliados, uma indicação qualitativa de qualidade e da confiabilidade do *software* começa a vir à tona [30]. Se erros graves por

ventura venham a surgir, a qualidade e a confiabilidade do *software* são comprometidas, sendo necessário à realização de testes adicionais. Contudo, se as funções do *software* estiverem funcionando corretamente e os erros encontrados forem de fácil correção, pode-se chegar a duas conclusões: a qualidade e a confiabilidade do *software* são aceitáveis ou os testes são inadequados para revelar erros graves [31].

Se a fase de testes não identificar erro algum, provavelmente os testes realizados não foram elaborados corretamente e possíveis erros estão escondidos no *software*. Provavelmente esses erros serão descobertos por usuários e corrigidos pelo programador durante a fase de manutenção [29].

5.4 Projeto de Casos de Teste

Qualquer produto trabalhado por engenharia pode ser testado de duas maneiras [30]: conhecendo-se a função específica que um produto projetado deve executar e conhecendo-se o funcionamento interno de um produto, testes podem ser realizados para garantir que o funcionamento interno do produto tenha um desempenho de acordo com as especificações do projeto inicial e que os componentes internos foram colocados à prova.

No processo de testes realizados nesse trabalho, foi adotado o teste de caixa branca, analisando as principais estruturas lógicas do *software*.

5.5 Teste de Caixa Branca (White Box)

O teste de caixa branca consiste em um conjunto de testes que usa a estrutura de controle do projeto procedimental para obter casos de teste.

Com esse método, o engenheiro de *software* pode gerar casos de teste que garantam que todos os caminhos independentes dentro de um módulo tenham sido exercitados pelo menos uma vez, exercitem todas as decisões lógicas para valores falsos ou verdadeiros, executem

todos os laços em suas fronteiras e dentro de seus limites operacionais e exercitem as estruturas de dados internas para garantir sua validade [31].

5.5.1 O Teste de Caminho Básico

Esse método possibilita obter uma medida da complexidade lógica de um projeto procedimental e use essa medida como guia para definir um conjunto básico de caminhos de execução. Os casos de teste derivados para exercitarem o conjunto básico tem a garantia de executar cada instrução do programa pelo menos uma vez durante a atividade de teste [30].

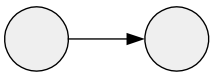
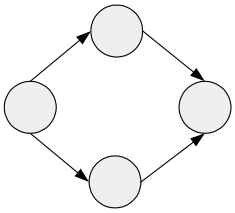
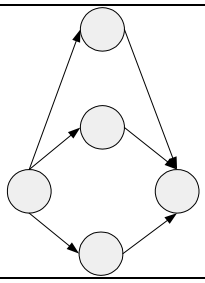
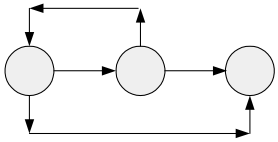
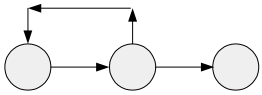
5.5.1.1 Notação de Grafo de Fluxo

Antes que o teste do caminho básico possa ser aplicado, uma notação simples para a representação do fluxo de controle, denominada grafo de fluxo, ou grafo de programa deve ser introduzida. Essa notação descreve o fluxo de controle lógico, onde cada construção estruturada tem um símbolo de grafo correspondente.

Critérios baseados em fluxo de controle utilizam apenas características de controle de execução do programa, como comandos ou desvios, para determinar quais estruturas estão sendo utilizados e quais podem ser eliminadas [30].

Para facilitar a análise do fluxo de controle, pode ser utilizada uma ferramenta para a geração de grafos de fluxo. Todo programa tem uma seqüência finita de comandos da linguagem de programação. Esses blocos de comandos são representados por círculos no grafo de fluxo de controle. Um bloco de comando representa uma seqüência de um ou mais comandos, tendo a característica de que, sempre que o comando inicial for executado, todos os demais comandos do bloco também serão [30]. A notação utilizada para representar grafos de fluxo de controle é apresentada na Tabela 5.1.

Tabela 5.1 : Notação para grafos de fluxo de controle.

Seqüência	
	
Estruturas de Condição	
If	***Case***
	
Estruturas de Repetição	
While	***For / Until***
	

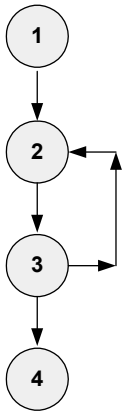
Nessa notação é definido que cada círculo, denominado nó, representa uma ou mais instruções do programa e cada arco (seta), denominada ramo, representa um fluxo de controle [30]. Os critérios de cobertura de fluxo de controle mais conhecidos são:

- Todos – Nós: todos os nós (comandos) devem ser executados pelo menos uma vez, onde dessa forma, a execução do programa deve abranger cada vértice do grafo de fluxo de controle pelo menos uma vez;
- Todos – Arcos: todos os arcos (ou decisões) devem ser executados pelo menos uma vez, ou seja, exige que a execução do programa passe pelo menos uma vez em cada aresta do grafo de fluxo de controle;
- Todos – Caminhos: todos os caminhos possíveis em um grafo de fluxo de controle são executados pelo menos uma vez, requerendo assim, que todos os caminhos possíveis do programa sejam executados (impraticável, pois o número de caminhos

é muito grande). Um caminho é executável se existe um conjunto de valores que possa ser atribuído às variáveis de entrada do programa e que causa a execução desse caminho.

Na Tabela 5.2, apresenta-se um exemplo de critérios de cobertura de grafo de fluxo de controle, no qual se podem verificar os critérios: Todos-Nós, onde cada nó (círculo) deve ser executado pelo menos uma vez {(1 – 2 – 3 – 4)}; Todos-Arcos, que requer a cada execução de cada arco pelo menos uma vez (1 – 2 – 3 – 2 – 3 – 4); Todos-Caminhos, onde cada caminho deve ser executado pelo menos uma vez (1 – 2 – 3 – 2 – 3...2 – 3 – 4). Nesse último caminho, pode-se confirmar a citação anterior de impraticável, pois existe um número de caminhos muito grande [30].

Tabela 5.2 : Critérios de cobertura de grafo de fluxo de controle.

<table>
<tr>
<td>
(1) Procedure TCalcFator.BotaoCalcular (TObject: Sender);
(1) Var VConta, VFat, VCalc : Integer;
(1) Begin
(1) VFat := StrToInt(Edit1.Text);
(1) VCalc := 0;
(2) For VCaonta := 1 To VFat Do
(3) VCalc := VCalc + (VFat * VConta);
(4) ShowMessage (‘O fatorial é ’ + IntToStr(VCalc);
(4) End;
</td>
<td>


</td>
</tr>
<tr>
<td>
Todos – Nós: {1 – 2 – 3}

Todos – Arcos: {1 – 2 – 3 – 2 – 3 – 4}
</td>
<td>
Todos – Caminhos:
{(1 – 2 – 3 – 4)}
(1 – 2 – 3 – 2 – 3 – 4)
(1 – 2 – 3 – 2 – 3...2 – 3 – 4)
</td>
</tr>
</table>

5.5.1.2 Complexidade Ciclomática

A complexidade ciclomática é uma métrica de *software* que proporciona uma medida quantitativa da complexidade lógica de um programa. O valor computado da complexidade ciclomática define o número de caminhos independentes do conjunto básico de um programa

e nos oferece um limite máximo para o número de testes que deve ser realizado para garantir que todas as instruções sejam executadas pelo menos uma vez [30].

Um caminho independente é qualquer caminho através do programa que introduza pelo menos um novo conjunto de instruções de processamento, ou uma nova condição. Quando estabelecido em termos de um grafo de fluxo, um caminho independente deve incluir pelo menos um ramo que não tenha sido atravessado antes que o caminho seja definido [29].

A complexidade ciclomática tem suas bases na teoria dos grafos e nos proporciona uma métrica de *software* extremamente útil. A complexidade pode ser verificada através de três procedimentos[30]:

1. O número de regiões do gráfico de fluxo corresponde a complexidade ciclomática;
2. A complexidade ciclomática, V(G), para um fluxo de grafo G é definida como V(G) = E – N + 2, onde E é o número de ramos do grafo de fluxo e N, o número de nós do grafo de fluxo;
3. A complexidade ciclomática, V(G), para um grafo de fluxo G é definida como V(G) = P + 1, onde P é o número de nós predicativos contidos no grafo de fluxo G.

O valor para V(G) nos oferece um limite máximo para o número de caminhos independentes que constitui o caminho básico e, por implicação, um limite máximo do número de testes que deve ser projetado e executado para garantir a cobertura de todas as instruções do programa [30]. Na Figura 5.2 apresenta-se o grafo de fluxo, indicando Ramo, Nó e Região.

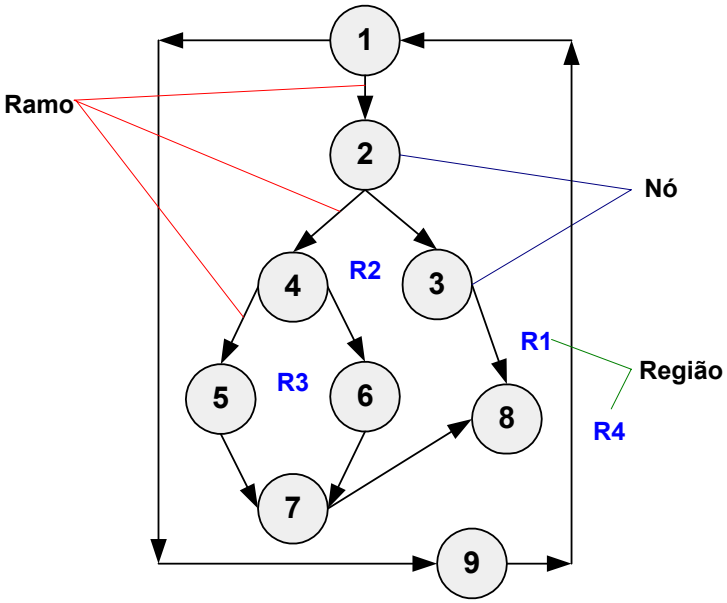


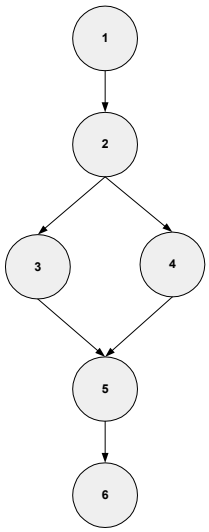
Figura 5.2 : Grafo de fluxo indicando ramo, nó e região.

5.5.1.3 Gerando Casos de Teste

O método de teste de caminho básico pode ser aplicado a um código fonte detalhado. A derivação de casos de teste é definida como uma série de passos, descritos nos itens abaixo [30]. Na Tabela 5.3 apresenta-se o código fonte e o grafo de fluxo de controle.

1. Através do código fonte, deve ser traçado um grafo de fluxo.

Tabela 5.3 : Grafo de fluxo de controle.

(1) Procedure Acess.Entrada (TObject: Sender); **(1)** Begin **(2)** If (Edit1.Text = 'Administrador') Then **(3)** ShowMessage ('Administrador'); **(4)** Else **(4)** ShowMessage ('Usuário'); **(5)** MenuShow; **(6)** End;	

2. Determinação da complexidade ciclomática do grafo de fluxo resultante.

Número de Ramos (N) = 6
Número de Nós (R) = 6
V (G) = N – R + 2
V (G) = 6 – 6 + 2
V (G) = **2** *(Complexidade Ciclomática)*

3. Determinar um conjunto de caminhos linearmente independentes.

Caminho 1: (1 – 2 – 3 – 5 – 6)
Caminho 2: (1 – 2 – 4 – 5 – 6)

4. A preparação dos casos de teste, apresentados na Tabela 5.4, forçam a execução de cada caminho no conjunto básico.

Tabela 5.4 : Casos de teste do teste de caminho básico.

Caso de teste caminho 1 Edit1.Text = Entrada Valida, onde Edit1.Text = 'Administrador' Resultados Esperados = Mensagem "Administrador" = Abertura do formulário de menu
Caso de teste caminho 2 Edit1.Text = Entrada Valida, onde Edit1.Text < > 'Administrador' Resultados Esperados = Mensagem "Usuário" = Abertura do formulário de menu

Cada caso de teste é executado e comparado com os resultados esperados. Assim que todos os casos de teste são executados, o analista pode certificar-se de que todas as instruções do programa foram executadas pelo menos uma vez [30].

Vale salientar, que alguns caminhos independentes não podem ser testados isoladamente, ou seja, a combinação de dados exigidos para atravessar o caminho pode não ser conseguida no fluxo normal do programa. Nesse caso, esses caminhos são testados como parte de outro caminho de teste [30].

Capítulo 6 – Critérios de Testes Aplicados no Sistema

Nesse capítulo são apresentados os testes realizados nas principais rotinas do *software* que realiza a leitura e validação dos *iButton´s*. Foram realizados testes estruturais (*White Box*), procurando enfocar os principais trechos do código fonte do sistema.

6.1 Testes Estruturais - Teste do Caminho Básico

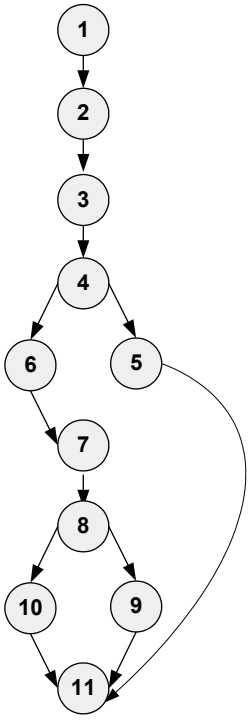
6.1.1 Código Fonte - Procedimento de Verificação da Existência de Adaptador e Porta Serial.

O primeiro procedimento avalia o trecho de programa responsável pela verificação da existência do adaptador e o número da interface de comunicação serial do microcomputador em que se encontra instalado. Descreve-se a seguir a estrutura de testes para essa rotina.

1. A partir do código fonte do programa a ser testado, é criado o grafo de fluxo do controle, apresentados na Tabela 6.1.

Tabela 6.1 : Código fonte e grafo de fluxo de controle.

```
(1) Procedure TFPrincipal.FormCreate(Sender: TObject);
(1) Begin
(2)   SetupDone := FALSE;
(3)   RetValue :=
ReadProFile('DefaultPortNum',@RetStr,200);
(4)  If (RetValue = -1) Then
(5)   Begin
(5)     ShowMessage ('PORTA NÃO SELECIONADA
        <VERIFIQUE  TMEX>');
(5)     Halt;
(5)   End
(6)  Else
(6)     NumeroPorta := StrToInt(StrPas(RetStr)) ;
(7)RetValue:=ReadProFile('DefaultPortType);
(8)  If (RetValue  = -1) Then
(9)   Begin
(9)     ShowMessage ('PORTA NÃO SELECIONADA
<VERIFIQUE  TMEX>');
(9)     Halt;
(9)   End
(10)  Else
(10)  Begin
(10)     TipoPorta:= StrToInt(StrPas(RetStr)) ;
(10)     Get_Version(@ztbuf);
(10)     NumIndice := 0;
(10)     Application.OnIdle := Leitura;
(10)   End;
(11) End;
```



2. Cálculo da complexidade ciclomática do grafo de fluxo de controle resultante.

Número de Ramos (R) = 12

Número de Nós (N) = 11

V (G) = R – N + 2

V (G) = 12 – 11 + 2

V (G) = 3 (*Complexidade Ciclomática*).

3. Conjunto básico de caminhos linearmente independentes.

Caminho 1 = (1 – 2 – 3 – 4 – 6 – 7 – 8 – 9 – 11);

Caminho 2 = (1 – 2 – 3 – 4 – 6 – 7 – 8 – 10 – 11);

Caminho 3 = (1 – 2 – 3 – 4 – 5 – 11).

4. Casos de teste que forcem a execução de cada caminho no conjunto básico.

Caso de teste do caminho 1:

SetupDone = entrada com valor "Falso"

RetValue = entrada válida (Adquire nº. da porta serial), onde RetValue < > -1

<u>*Resultados Esperados*</u>: RetValue = entrada válida (Adquire nº. do tipo de porta serial)

: RetValue = entrada válida, onde RetValue < > -1

: Chamada da rotina de leitura.

Caso de teste do caminho 2:

SetupDone = entrada com valor "Falso";

RetValue = entrada válida (Adquire nº. da porta serial), onde RetValue < > -1

<u>*Resultados Esperados*</u>: RetValue = entrada inválida (não adquire nº. do tipo de porta serial), onde RetValue = -1;

: Mensagem "Porta não selecionada – Verifique TMEX";

: Aborta Programa.

Caso de teste do caminho 3:

SetupDone = entrada com valor “Falso”

RetValue = entrada inválida (não adquire nº. da porta serial), onde RetValue = -1

Resultados Esperados: Mensagem “Porta não selecionada – Verifique TMEX”

: Aborta Programa.

6.1.2 Código Fonte - Procedimentos de Verificação de Permissão de Acesso.

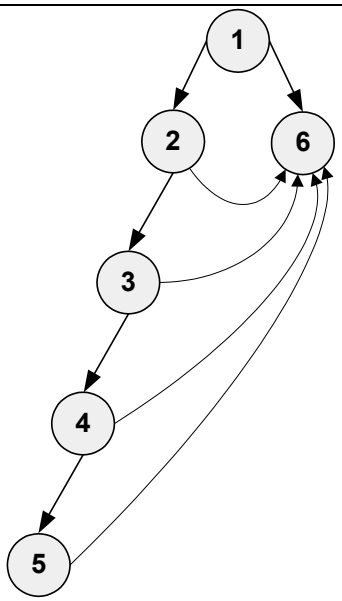
As rotinas apresentadas a seguir, têm a função de verificar se o usuário possui permissão de acesso no domingo, verificando horário, período permitido, data inicial e data final. Se todas as condições forem satisfeitas, o sistema realiza a abertura da porta, emitindo um sinal pela interface de comunicação paralela do microcomputador. O algoritmo verifica todas as possibilidades de configurações de acesso propostas no cadastro de permissões de acesso. A seguir, serão apresentados os testes realizados nos respectivos trechos de código fonte.

6.1.3 Verificação de Permissão de Acesso – Rotina 1

Essa rotina verifica se o usuário possui permissão de acesso no domingo e o intervalo de tempo (hora inicial e hora final permitida). Caso todas as condições forem satisfeitas, o sistema envia um sinal pela interface de comunicação paralela do microcomputador, acionando o fecho eletromagnético a permitindo o acesso do usuário as dependências do laboratório de pesquisa. A seguir, apresenta-se a estrutura de testes para essa rotina.

1. A partir do código fonte do programa a ser testado, é criado o grafo de fluxo do controle, apresentado na Tabela 6.2.

Tabela 6.2 : Código fonte e grafo de fluxo de controle.

(1) If (IBQuery2.FieldByName('Domingo').AsBoolean=True) Then **(2)** If ((time>=IBQuery2.FieldByName('HrIni_Dom').AsDateTime) and(time<=IBQuery2.FieldByName('HrFin_Dom').AsDateTime)) then **(3)** If (IBQuery2.FieldByName('Periodo').AsBoolean=True) Then **(4)** If ((date >= IBQuery2.FieldByName('Per_Ini').AsDateTime) and (date <= IBQuery2.FieldByName('Per_Fin').AsDateTime)) Then **(5)** Begin **(5)** VarPino:= 3; **(5)** IOPort1.PortAddress :=$378; **(5)** IOPort1.PortData:= VarPino; **(5)** IOPort1.Write; **(5)** Sleep(700); **(5)** IOPort1.Reset; **(5)** MessageBeep(1); **(6)** End;	

2. Cálculo da complexidade ciclomática do grafo de fluxo de controle resultante.

Número de Ramos (R) = 9

Número de Nós (N) = 6

V (G) = R – N + 2

V (G) = 9 – 6 + 2

V (G) = 5 (*Complexidade Ciclomática*).

3. Conjunto básico de caminhos linearmente independentes.

Caminho 1 = (1 – 2 – 3 – 4 – 5 – 6);

Caminho 2 = (1 – 2 – 6);

Caminho 3 = (1 – 2 – 3 – 6);

Caminho 4 = (1 – 2 – 3 – 4 – 6);

Caminho 5 = (1 – 6);

4. Casos de teste que forcem a execução de cada caminho no conjunto básico.

Caso de teste do caminho 1:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada válida, com valor “True”;

((time >= IBQuery2.FieldByName('HrIni_Dom').AsDateTime) and (time <= IBQuery2.FieldByName('HrFin_Dom').AsDateTime)) = entrada válida, onde a hora atual (time) esta compreendida entre os valores armazenados nas variáveis “HrIni_Dom” e “HrFin_Dom”;

(IBQuery2.FieldByName('Periodo').AsBoolean=True) = entrada inválida, onde período, recebe valor “True”;

((date >= IBQuery2.FieldByName('Per_Ini').AsDateTime) and (date <= IBQuery2.FieldByName('Per_Fin').AsDateTime)) = entrada válida, onde a data atual esta compreendida entre os valores armazenados nas variáveis “Per_Ini” e “Per_Fin”

Resultados Esperados: Abertura da porta (envio de sinal pela porta paralela);

: Armazena informações de acesso no banco de dados.

Caso de teste do caminho 2:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada válida, com valor “True”;

((time >= IBQuery2.FieldByName('HrIni_Dom').AsDateTime) and (time <= IBQuery2.FieldByName('HrFin_Dom').AsDateTime)) = entrada inválida, onde a hora atual (time) não esta compreendida entre os valores armazenados nas variáveis “HrIni_Dom” e “HrFin_Dom”;

Resultados Esperados: Finaliza estrutura de condição IF;

Caso de teste do caminho 3:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada válida, com valor “True”;

((time >= IBQuery2.FieldByName('HrIni_Dom').AsDateTime) and (time <= IBQuery2.FieldByName('HrFin_Dom').AsDateTime)) = entrada válida, onde a hora atual (time) esta compreendida entre os valores armazenados nas variáveis “HrIni_Dom” e “HrFin_Dom”;

(IBQuery2.FieldByName('Periodo').AsBoolean=True) = entrada inválida, onde período, recebe valor “False”;

Resultados Esperados: Finaliza estrutura de condição IF;

Caso de teste do caminho 4:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada válida, com valor “True”;

((time >= IBQuery2.FieldByName('HrIni_Dom').AsDateTime) and (time <= IBQuery2.FieldByName('HrFin_Dom').AsDateTime)) = entrada válida, onde a hora atual (time) esta compreendida entre os valores armazenados nas variáveis “HrIni_Dom” e “HrFin_Dom”;

(IBQuery2.FieldByName('Periodo').AsBoolean=True) = entrada inválida, onde período, recebe valor “True”;

((date >= IBQuery2.FieldByName('Per_Ini').AsDateTime) and (date <= IBQuery2.FieldByName('Per_Fin').AsDateTime)) = entrada inválida, onde a data atual não esta compreendida entre os valores armazenados nas variáveis “Per_Ini” e “Per_Fin”

Resultados Esperados: Finaliza estrutura de condição IF;

Caso de teste do caminho 5:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada inválida, com valor “False”;

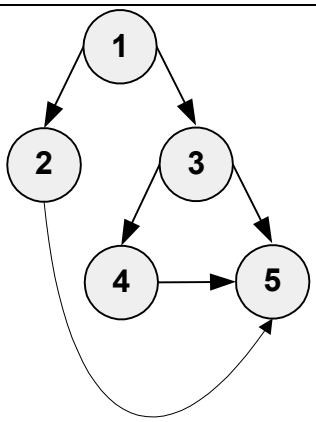
Resultados Esperados: Finaliza estrutura de condição IF;

6.1.4 Verificação de Permissão de Acesso – Rotina 2

Essa rotina verifica se o usuário possui permissão de acesso no domingo e se o intervalo de tempo (hora inicial e hora final permitida) foi excedido. A seguir, apresenta-se a estrutura de testes para essa rotina.

1. A partir do código fonte do programa a ser testado, é criado o grafo de fluxo do controle, apresentados na Tabela 6.3.

Tabela 6.3 : Código fonte e grafo de fluxo de controle.

(1) If (IBQuery2.FieldByName('Domingo').AsBoolean = 'True')and **(1)** (time < IBQuery2.FieldByName('HrIni_Dom').AsDateTime) Then **(2)** Begin **(2)** IBQuery1.SQL.Add ('insert into "TBEvento"; **(2)** End; **(3)** Else If (IBQuery2.FieldByName('Domingo').AsBoolean = 'True')and(time>IBQuery2.FieldByName('HrFin_Dom').AsDateTime) Then **(4)** Begin **(4)** IBQuery1.SQL.Add ('insert into "TBEvento"; **(5)** End;	

2. Cálculo da complexidade ciclomática do grafo de fluxo de controle resultante.

Número de Ramos (R) = 6

Número de Nós (N) = 5

V (G) = R – N + 2

V (G) = 6 – 5 + 2

V (G) = 3 (*Complexidade Ciclomática*).

3. Conjunto básico de caminhos linearmente independentes.

Caminho 1 = (1 – 2 – 5);

Caminho 2 = (1 – 3 – 4 – 5);

Caminho 3 = (1 – 3 – 5).

4. Casos de teste que forcem a execução de cada caminho no conjunto básico.

Caso de teste do caminho 1:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada válida, com valor "True" e ((time < IBQuery2.FieldByName('HrIni_Dom').AsDateTime) = entrada válida, onde a hora atual (time) é menor que o valor armazenado nas variável "HrIni_Dom";

Resultados Esperados: Porta não é aberta (não é enviado sinal pela porta paralela);

: Armazena informações de tentativa de acesso no banco de

dados;

: Envia *e-mail* ao administrador informando a tentativa de acesso.

Caso de teste do caminho 2:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada inválida, com valor “False” e ((time < IBQuery2.FieldByName('HrIni_Dom').AsDateTime) = entrada inválida, onde a hora atual (time) é maior que o valor armazenado na variável “HrIni_Dom”; IBQuery2.FieldByName('Domingo').AsBoolean = entrada válida, com valor “True” e time> IBQuery2.FieldByName('HrFin_Dom').AsDateTime = entrada válida, onde a hora atual (time) é maior que o valor armazenado na variável “HrFin_Seg”

Resultados Esperados: Porta não é aberta (não é enviado sinal pela porta paralela);

: Armazena informações de tentativa de acesso no banco de dados;

: Envia *e-mail* ao administrador informando a tentativa de acesso.

Caso de teste do caminho 3:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada inválida, com valor “False” e ((time < IBQuery2.FieldByName('HrIni_Dom').AsDateTime) = entrada inválida, onde a hora atual (time) é maior que o valor armazenado na variável “HrIni_Dom”; IBQuery2.FieldByName('Domingo').AsBoolean = entrada inválida, com valor “False” e time> IBQuery2.FieldByName('HrFin_Dom').AsDateTime = entrada inválida, onde a hora atual (time) é menor que o valor armazenado na variável “HrFin_Seg”;

Resultados Esperados: Finaliza If.

6.1.5 Verificação de Permissão de Acesso – Rotina 3

Essa rotina verifica se o usuário possui permissão de acesso no domingo, o intervalo de tempo (hora inicial e hora final permitida), se o usuário possui permissão de acesso por

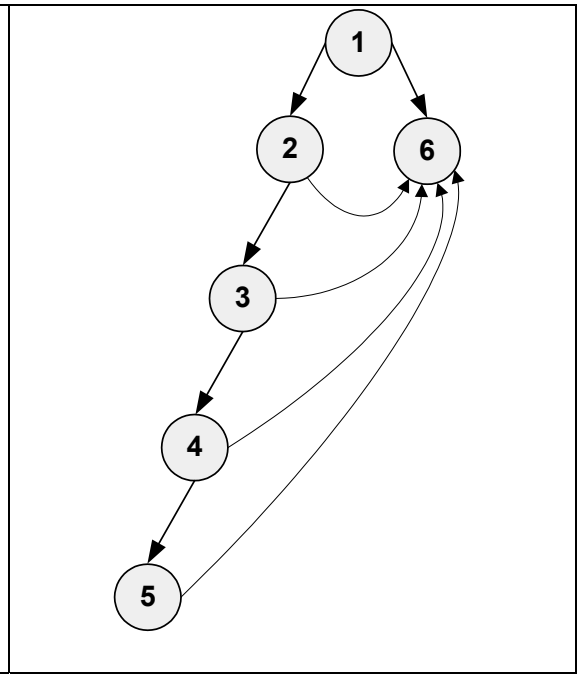
período (data inicial e data final) permitido. A rotina verifica se a condição de período é inválida, ou seja, a data inicial e data final não são válidas.

Caso todas as condições forem satisfeitas, o sistema não permite o acesso do usuário as dependências do laboratório de pesquisa, registrando a tentativa de acesso no banco de dados do sistema e enviando um *e-mail* para o administrador. A seguir, apresenta-se a estrutura de testes para essa rotina.

1. A partir do código fonte do programa a ser testado, é criado o grafo de fluxo do controle, apresentados na Tabela 6.4.

Tabela 6.4 : Código fonte e grafo de fluxo de controle.

```
(1) If (IBQuery2.FieldByName('Domingo').AsBoolean=True) Then
(2) If ((time>=IBQuery2.FieldByName('HrIni_Dom').AsDateTime)
and(time<=IBQuery2.FieldByName('HrFin_Dom').AsDateTime)) then
(3) If (IBQuery2.FieldByName('Periodo').AsBoolean=True) Then
(4) If ((date >= IBQuery2.FieldByName('Per_Ini').AsDateTime =
False) and (date <= IBQuery2.FieldByName('Per_Fin').AsDateTime =
False)) Then
(5)     Begin
(5)         VarPino:= 3;
(5)         IOPort1.PortAddress :=$378;
(5)         IOPort1.PortData:= VarPino;
(5)         IOPort1.Write;
(5)         Sleep(700);
(5)         IOPort1.Reset;
(5)         MessageBeep(1);
(6)     End;
```



2. Cálculo da complexidade ciclomática do grafo de fluxo de controle resultante.

Número de Ramos (R) = 9

Número de Nós (N) = 6

V (G) = R – N + 2

V (G) = 9 – 6 + 2

V (G) = 5 (*Complexidade Ciclomática*).

3. Conjunto básico de caminhos linearmente independentes.

Caminho 1 = (1 – 2 – 3 – 4 – 5 – 6);

Caminho 2 = (1 – 2 – 6);

Caminho 3 = (1 – 2 – 3 – 6);

Caminho 4 = (1 – 2 – 3 – 4 – 6);

Caminho 5 = (1 – 6);

4. Casos de teste que forcem a execução de cada caminho no conjunto básico.

Caso de teste do caminho 1:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada válida, com valor “True”;

((time >= IBQuery2.FieldByName('HrIni_Dom').AsDateTime) and (time <= IBQuery2.FieldByName('HrFin_Dom').AsDateTime)) = entrada válida, onde a hora atual (time) esta compreendida entre os valores armazenados nas variáveis “HrIni_Dom” e “HrFin_Dom”;

(IBQuery2.FieldByName('Periodo').AsBoolean=True) = entrada inválida, onde período, recebe valor “True”;

((date >= IBQuery2.FieldByName('Per_Ini').AsDateTime) and (date <= IBQuery2.FieldByName('Per_Fin').AsDateTime)) = entrada válida, onde a data atual esta compreendida entre os valores armazenados nas variáveis “Per_Ini” e “Per_Fin”

Resultados Esperados: Abertura da porta (envio de sinal pela porta paralela);

: Armazena informações de acesso no banco de dados.

Caso de teste do caminho 2:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada válida, com valor “True”;

((time >= IBQuery2.FieldByName('HrIni_Dom').AsDateTime) and (time <= IBQuery2.FieldByName('HrFin_Dom').AsDateTime)) = entrada inválida, onde a hora

atual (time) não esta compreendida entre os valores armazenados nas variáveis “HrIni_Dom” e “HrFin_Dom”;

Resultados Esperados: Finaliza estrutura de condição IF;

Caso de teste do caminho 3:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada válida, com valor “True”;

((time >= IBQuery2.FieldByName('HrIni_Dom').AsDateTime) and (time <= IBQuery2.FieldByName('HrFin_Dom').AsDateTime)) = entrada válida, onde a hora atual (time) esta compreendida entre os valores armazenados nas variáveis “HrIni_Dom” e “HrFin_Dom”;

(IBQuery2.FieldByName('Periodo').AsBoolean=True) = entrada inválida, onde período, recebe valor “False”;

Resultados Esperados: Finaliza estrutura de condição IF;

Caso de teste do caminho 4:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada válida, com valor “True”;

((time >= IBQuery2.FieldByName('HrIni_Dom').AsDateTime) and (time <= IBQuery2.FieldByName('HrFin_Dom').AsDateTime)) = entrada válida, onde a hora atual (time) esta compreendida entre os valores armazenados nas variáveis “HrIni_Dom” e “HrFin_Dom”;

(IBQuery2.FieldByName('Periodo').AsBoolean=True) = entrada inválida, onde período, recebe valor “True”;

((date >= IBQuery2.FieldByName('Per_Ini').AsDateTime) and (date <= IBQuery2.FieldByName('Per_Fin').AsDateTime)) = entrada inválida, onde a data atual não esta compreendida entre os valores armazenados nas variáveis “Per_Ini” e“Per_Fin”

Resultados Esperados: Finaliza estrutura de condição IF;

Caso de teste do caminho 5:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada inválida, com valor “False”;

Resultados Esperados: Finaliza estrutura de condição IF;

6.1.6 Verificação de Permissão de Acesso – Rotina 4

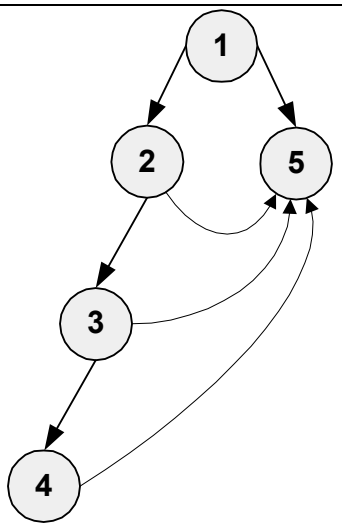
Essa rotina verifica se o usuário possui permissão de acesso no domingo, o intervalo de tempo (hora inicial e hora final permitida), se o usuário possui permissão de acesso por período (data inicial e data final) permitido. A rotina verifica se a condição de período é inválida, ou seja, a data inicial e data final não são válidas.

Caso todas as condições forem satisfeitas, o sistema não permite o acesso do usuário as dependências do laboratório de pesquisa, registrando a tentativa de acesso no banco de dados do sistema e enviando um *e-mail* para o administrador. A seguir apresenta-se a estrutura de testes para essa rotina.

1. A partir do código fonte do programa a ser testado, é criado o grafo de fluxo do controle, apresentados na Tabela 6.5.

Tabela 6.5 : Código fonte e grafo de fluxo de controle.

```
(1) If (IBQuery2.FieldByName('Domingo').AsBoolean=True) Then
(2) If ((time>=IBQuery2.FieldByName('HrIni_Dom').AsDateTime)
and(time<=IBQuery2.FieldByName('HrFin_Dom').AsDateTime)) then
(3) If (IBQuery2.FieldByName('Periodo').AsBoolean=True) Then
(4)     Begin
(4)        VarPino:= 3;
(4)         IOPort1.PortAddress :=$378;
(4)         IOPort1.PortData:= VarPino;
(4)         IOPort1.Write;
(4)         Sleep(700);
(4)         IOPort1.Reset;
(4)         MessageBeep(1);
(5)     End;
```



2. Cálculo da complexidade ciclomática do grafo de fluxo de controle resultante.

Número de Ramos (R) = 7

Número de Nós (N) = 5

V (G) = R – N + 2

V (G) = 7 – 5 + 2

V (G) = 4 (*Complexidade Ciclomática*).

3. Conjunto básico de caminhos linearmente independentes.

Caminho 1 = (1 – 2 – 3 – 4 – 5);

Caminho 2 = (1 – 2 – 5);

Caminho 3 = (1 – 2 – 3 – 5);

Caminho 4 = (1 – 5).

4. Casos de teste que forcem a execução de cada caminho no conjunto básico.

Caso de teste do caminho 1:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada válida, com valor “True”;

((time >= IBQuery2.FieldByName('HrIni_Dom').AsDateTime) and (time <= IBQuery2.FieldByName('HrFin_Dom').AsDateTime)) = entrada válida, onde a hora atual (time) esta compreendida entre os valores armazenados nas variáveis “HrIni_Dom” e “HrFin_Dom”;

(IBQuery2.FieldByName('Periodo').AsBoolean=True) = entrada inválida, onde período, recebe valor “True”;

<u>*Resultados Esperados*</u>: Abertura da porta (envio de sinal pela porta paralela);

: Armazena informações de acesso no banco de dados.

Caso de teste do caminho 2:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada válida, com valor “True”;

((time >= IBQuery2.FieldByName('HrIni_Dom').AsDateTime) and (time <= IBQuery2.FieldByName('HrFin_Dom').AsDateTime)) = entrada inválida, onde a hora atual (time) não esta compreendida entre os valores armazenados nas variáveis “HrIni_Dom” e “HrFin_Dom”;

Resultados Esperados: Finaliza estrutura de condição IF;

Caso de teste do caminho 3:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada válida, com valor “True”;

((time >= IBQuery2.FieldByName('HrIni_Dom').AsDateTime) and (time <= IBQuery2.FieldByName('HrFin_Dom').AsDateTime)) = entrada válida, onde a hora atual (time) esta compreendida entre os valores armazenados nas variáveis “HrIni_Dom” e “HrFin_Dom”;

Resultados Esperados: Finaliza estrutura de condição IF;

Caso de teste do caminho 4:

IBQuery2.FieldByName('Domingo').AsBoolean = entrada inválida, com valor “False”;

Resultados Esperados: Finaliza estrutura de condição IF;

6.2 Demais Testes Realizados

Depois da aplicação de algumas metodologias de teste de *software*, propostas pela análise estruturada, foram aplicados outros procedimentos para análise de desempenho do *software*. Os itens a seguir descrevem os demais testes realizados.

6.2.1 Teste de Desempenho por Tempo

Nesse procedimento de teste adotado, foram inseridas no código fonte do programa procedimentos para aferir o tempo de acesso às principais rotinas do sistema que envolve o acesso ao banco de dados, verificação de permissões de acesso e tempo da emissão de sinal

pela interface de comunicação paralela do microcomputador. A seguir, são descritas as principais rotinas verificadas.

6.2.1.1 Leitura do Número de *ROM* do *iButton*

Nessa etapa de teste, foi inserido no código fonte do *software*, uma rotina para a verificação do tempo de leitura do número de *ROM* do *iButton* no momento em que o usuário toca o dispositivo na interface de comunicação *1-Wire (Blue Dot)*. Os resultados obtidos são apresentados na Tabela 6.6.

Tabela 6.6 : Tempo de leitura do número de *ROM* do *iButton*.

Número de Testes Realizados	Tempo (milesegundos)
1	0:2
2	0:4
3	0:3
4	0:1
Média	**0:25**

6.2.1.2 Verificação de Cadastro de Usuários no Banco de Dados

Durante essa etapa de teste, foi avaliado o tempo gasto para a obtenção do número de *ROM* do *iButton*, até o acesso ao banco de dados do sistema para verificação se o usuário encontra-se cadastrado. Os resultados obtidos consideram o acesso ao banco de dados, localizados no servidor, através da rede local. Na Tabela 6.7 apresenta-se os resultados obtidos.

Tabela 6.7 : Tempo de acesso ao banco de dados.

Número de Testes Realizados	Tempo (milesegundos)
1	0:921
2	0:832
3	0:948
4	0:893
Média	**0:8985**

6.2.1.3 Verificação para Usuário com Acesso Irrestrito

Durante esse teste, foi avaliado o tempo gasto pelo sistema para a obtenção do número de *ROM* do *iButton*, verificação se o usuário é cadastrado no banco de dados do sistema, verificação se possui permissões de acesso, verificação se a permissão é do tipo "Irrestrito", até o envio de um sinal pelo *software* por meio da interface de comunicação paralela do microcomputador, e consequentemente, a abertura da porta. Na Tabela 6.8 apresenta-se os resultados obtidos.

Tabela 6.8 : Resultados obtidos para verificação de usuário irrestrito.

Número de Testes Realizados	Tempo (seg/mileseg)
1	1:162
2	1:502
3	1:653
4	1:563
Média	**1:47**

6.2.1.4 Verificação de Permissões Semanais

Nesse teste, foi verificado o tempo gasto pelo sistema para a obtenção do número de *ROM* do *iButton*, verificação se o usuário é cadastrado no banco de dados do sistema, verificação se possui permissões de acesso, e se essa permissão é semanal (dia da semana, hora inicial, hora final), até o envio de um sinal pelo *software* por meio da interface de comunicação paralela do microcomputador, e consequentemente, a abertura da porta. Na Tabela 6.9 apresenta-se os resultados obtidos.

Tabela 6.9 : Resultados obtidos para verificação de permissões semanais.

Número de Testes Realizados	Tempo (seg/mileseg)
1	1:042
2	1:512
3	1:092
4	1:612
Média	**1:3145**

7 Discussão

O desempenho do sistema, levando-se em consideração os testes de tempo realizados, com a inserção de rotinas de tempo nos principais trechos do programa e os testes reais, no momento em que o usuário coloca o *iButton* na interface de comunicação *"Blue Dot"*, até a validação e verificação das permissões de acesso do usuário no servidor de banco de dados e a emissão de um sinal pela interface de comunicação paralela do microcomputador para a abertura da porta é considerado satisfatório para o usuário.

No microcomputador responsável em gerenciar o acesso dos alunos, onde se encontra instalado o módulo leitura, pôde-se observar que durante determinadas operações realizadas, como a execução de aplicativos que requerem mais recursos de processamento, como a varredura do sistema pelo antivírus, o desempenho do sistema diminui consideravelmente, tornando o tempo de espera da abertura da porta maior. Dessa forma, sugere-se que o microcomputador responsável em realizar o controle de acesso seja dedicado, ou que aplicativos que requerem maior capacidade de processamento sejam executados em momentos que o acesso ao laboratório de pesquisa esteja menos intenso, ou mesmo, o uso de um microcomputador com uma arquitetura de *hardware* mais eficiente.

Como o servidor de banco de dados encontra-se instalado nas dependências do laboratório de pesquisa, dentro da própria rede interna do laboratório, no caso de uma eventual queda da rede da instituição, a rede interna do laboratório não é afetada, permanecendo o sistema conectado ao banco de dados. Todavia, se o sistema fosse instalado em uma amplitude maior, como realizar o gerenciamento de vários laboratórios, e por se tratar de um sistema distribuído, uma eventual falha da rede local, paralisaria o funcionamento do sistema.

A instalação do *software* utilizando o sistema operacional *Windows XP*, *Windows 2000* e *Windows NT*, necessita da instalação adicional de um *software* para liberar o acesso de

leitura e escrita na interface paralela do microcomputador, pois esses sistemas operacionais bloqueiam o acesso a esse recurso.

Para evitar paradas súbitas no sistema quando eventualmente ocorrerem quedas de energia, aconselha-se o uso de *no-breaks* no microcomputador servidor e no microcomputador que ira hospedar o módulo leitura.

Em algumas arquiteturas de microcomputadores, durante a execução do *POST* (*Power on Self Test*), uma checagem de *hardware* que a *ROM Bios* executa todas as vezes que o microcomputador é ligado, um dos procedimentos do teste é emitir um sinal pela interface paralela para checar seu funcionamento. Nesse caso, em uma eventual queda de energia, quando a corrente elétrica for restabelecida e o microcomputador religar-se automaticamente, o sinal emitido durante a execução do *POST* acionaria o circuito de potência, consequentemente acionando o fecho eletromagnético e abrindo a porta. Tal problema pode ser sanado desabilitando-se nas configurações do *setup* o religue automático após uma queda de energia.

8 Conclusões

No desenvolvimento do projeto pode-se observar que o sistema possui vastas aplicações, podendo ser empregado tanto no âmbito da área acadêmica como na área comercial. O uso do ambiente visual possibilita uma maior interatividade do usuário com o sistema, tornando mais fácil a sua utilização.

A utilização do *iButton* no lugar da chave convencional, apresentou um resultado bastante satisfatório, levando-se em conta os demais meios de identificação existentes no mercado, como o código de barras e o cartão magnético, pois o *iButton* é um dispositivo de identificação barato, altamente resistente a diversos ambientes, podendo ser utilizado como um chaveiro no dia a dia, não sendo possível realizar cópias, como no uso das chaves tradicionais.

O sistema encontra-se há dez meses instalado nas dependências do Laboratório de Processamento de Sinais e Sistemas Digitais, UNESP, campus de Ilha Solteira, gerenciando o acesso dos usuários as dependências do laboratório de pesquisa. Desde a sua instalação até a presente data o sistema opera em sua versão 3.0.2 (módulo leitura e módulo administrador), pode-se observar que o sistema não apresenta falhas que comprometam a segurança do laboratório de pesquisa, atendendo a todos os objetivos propostos no início do projeto.

A utilização desse sistema possibilita o acesso dos discentes aos laboratórios de pesquisa fora dos horários normais de aula, sem a necessidade de funcionários para supervisionar a entrada e a saída, pois todos os acessos realizados são armazenados no banco de dados do sistema.

Pode-se observar a satisfação dos usuários do laboratório com a utilização do *iButton* no lugar da chave tradicional, tornando mais prático e rápido seu acesso. Em casos isolados, ocorreram eventos em que usuários perderam seu *iButton*, onde o procedimento adotado foi

somente o bloqueio do número de série do *iButton* no sistema, não havendo a necessidade de realizar a troca de fechaduras.

Pelos relatórios fornecidos no sistema, é possível realizar auditorias constantes, realizando um acompanhamento individual e coletivo dos usuários que utilizam as dependências do laboratório.

Espera-se que novas atualizações e ferramentas sejam no futuro, inseridas no contexto do sistema, procurando descobrir novos métodos para inibir ou até mesmo coibir furtos que ocorrem diariamente dentro das instituições de ensino.

9 Sugestões para trabalhos futuros

Como sugestões para trabalhos futuros para o desenvolvimento e aperfeiçoamento do *software*, propõem-se:

1. Aperfeiçoamento do acesso ao sistema via *WEB*, com a implementação de opções de escolha de laboratório de pesquisa por administrador de laboratório, acesso a relatórios on-line, abertura da porta, visualização de usuários que se encontram no interior do laboratório de pesquisa com a utilização de uma *WEBCAM*;
2. Criação de grupos de usuários (administradores, docentes, discentes, etc.), com permissões de acesso e perfis pré-estabelecidos;
3. Emprego de microcontroladores com rádio freqüência, diminuindo consideravelmente a utilização de fios na implantação física do sistema;
4. Adaptação no sistema, em facultar ao administrador escolher outros tipos de dispositivos de identificação além do *iButton*, como cartão magnético, cartão de proximidade e código de barras;
5. Acesso do usuário ao laboratório de pesquisa, com entrada de dados (senha de acesso pessoal) por teclado numérico, localizado no lado externo ao laboratório de pesquisa, no caso de ausência do *iButton* por parte do usuário;
6. Abertura remota da porta do laboratório por telefone, mediante o fornecimento de senha de acesso do administrador.

10 Referências

[1] SAKAMOTO, J. M.S.; SILVA, A.C.R.; OLIVEIRA, L. C. O. Um Novo Sistema de Controle de Acesso para Laboratórios Interdisciplinares. **International Conference on Enginnering and Computer Education**, Santos SP, p.1-4, 2003.

[2] DUQUE, M. A. C.; PAULA A.; DUQUE, T, F. **Sistema de Control de Aceso con iButton.** Local, Santafé de Bogotá,158f. Tesis (Ingeniero Electrónico) - Pontifícia Universidad Javeriana. Facultad de Ingeniería. Departamento de Electrónica. Fev.2000.

[3] BOREKI,G.; ZIMMER, A. **Sistema de controle de acesso por IButton com verificação biométrica por geometria da mão**. UNICENP – Centro Universitário Positivo, p.1-9.

[4] ZALUD, B. **Down with Rekeying**. Security. ABI/Inform Global, Jun.2003. p.12.

[5] O´LEARY, T. **Door & Electronic Access Control Make-Over**. Locksmith Ledger International. Career and Technical Education, Mar. 2005.p.10.

[6] ORION SEGURANÇA, Ronda, <**http://www.orionseguranca.com.br/orion/rondaelt.html**>. Acesso em: 21 de janeiro de 2005.

[7] ATI TELECOMUNICAÇÕES. Sistema de gerência de segurança de acesso físico Disponível em: <**http://www.ati.com.br/Portugues/tuor_sgs.html**>. Acesso em: 15 de outubro de 2005.

[8] CANTÙ, M. Programação orientada a objetos no Delphi. In:___.**Dominando o Delphi 5 – A Bíblia**, São Paulo, Makron Books , 2000, p. 39-71.

[9] ALVES, W.P. Definição da base de dados. In:___.**Aplicações Avançadas para Banco de Dados.**1.ed. São Paulo: Érica, 2002. p. 185-186.

[10] ALVES, W.P. Introdução ao Interbase 6. In:___.**Aplicações Avançadas para Banco de Dados.**1.ed.São Paulo:Érica, 2002. p. 18-20.

[11] BLUE, T.; KASTER J.et.al. Visão geral do desenvolvimento de banco de dados em Delphi. In:___.**Desenvolvendo Banco de Dados em Delphi.** São Paulo, Makron Books, 1997. p. 1-3.

[12] DALLAS SEMICONDUCTOR. What is a iButton . Disponível em: <**http://www.ibutton.com/ibuttons/index.html**>. Acesso em: 10 de fevereiro de 2004.

[13] DALLAS SEMICONDUCTOR. iButton Standards. <**http://www.maxim-ic.com/appnotes.cfm/appnote_number/937** >. Acesso em: 10 de fevereiro de 2004.

[14] DALLAS SEMICONDUCTOR. iButton Products: iButton´s. <**http://www.ibutton.com/products/ibuttons.html** >. Acesso em: 10 de fevereiro de 2004.

[15] DALLAS SEMICONDUCTOR. Reading and writing iButton´s via serial interfaces. Disponível em: < **http://www.maxim-ic.com/appnotes.cfm/appnote_number/512**>. Acesso em: 10 de fevereiro de 2004.

[16] DALLAS SEMICONDUCTOR. iButton Products: Readers and Adapters. <**http://www.ibutton.com/products/readers.html**>. Acesso em: 10 de fevereiro de 2004.

[17] DALLAS SEMICONDUCTOR. iButton TMEX Runtime Environment Reference Manual Version 3.11 . <**http://www.pdfserv.maxim-ic.com/en/an/tm320rte.pdf** >. Acesso em: 25 de fevereiro de 2004.

[18] CANTÙ, M. Object Pascal avançado. In:___.**Dominando o Delphi 5 – A Bíblia**, São Paulo, Editora Makron Books, 2000, p. 71-74.

[19] KORTH, H.F.; SILBERSCHATZ, A. Introdução a banco de dados. In:___. **Sistema de banco de dados**. 2.ed. São Paulo: Makron Books, 1993. p. 1-19.

[20] KORTH, H..F.; SILBERSCHATZ, A. Bancos de dados distribuídos. In:___. **Sistema de banco de dados**. 2.ed. São Paulo: Makron Books, 1993. p. 513-519.

[21] EMS HITECH. What is IB Manager ?, <**http://www.ems-hitech.com/ibmanager**>. Acesso em: 25 de setembro de 2004.

[22] ALVES, W.P. Criação da base de dados. In:___.**Aplicações Avançadas para Banco de Dados.**1.ed.São Paulo: Érica, 2002. p. 193-196.

[23] BLUE, T.; KASTER J.et.al. Os componentes de acesso aos dados. In:___.**Desenvolvendo Banco de Dados em Delphi.** São Paulo, Makron Books, 1997. p. 13-15.

[24] WELLING, L.,THOMSON, L. Desenvolvimento WEB. In:___. **PHP e MYSQL**.3.ed. São Paulo:Campus, 2003.p. 676.

[25] SAKAMOTO, J. M.S. Relatório de Pesquisa. **Sistema baseado em iButton para controle de acesso e medida de temperaturas**. UNESP - Faculdade de engenharia de Ilha Solteira, p.13-16, 2002.

[26] PEREIRA, F. Protocolo 1-Wire. In:___.**PIC - PROGRAMAÇÃO EM C.** 3.ed.São Paulo: Érica, 2003. p. 283-291.

[27] PRESSMAN, R.S. Garantia de Qualidade de Software. In:___. **Engenharia de Software**.3.ed. São Paulo:Makron Books,1995.p. 723-770.

[28] HETZEL,W. Como Testar Necessidades Técnicas e Funcionais. In:___.**Guia Completo ao Teste de Software.**1.ed.Rio de Janeiro:Campus,1987. p. 38-42.

[29] HETZEL,W. Testes de Programa – Testes em Pequena Escala. In:___.**Guia Completo ao Teste de Software.**1.ed.Rio de Janeiro:Campus,1987. p. 50-93.

[30] PRESSMAN, R.S. Técnicas de Teste de Software. In:___. **Engenharia de Software**.3.ed. São Paulo:Makron Books,1995.p. 786-822.

[31] GONÇALVES, K. V. **Teste de Software em Aplicações de Banco de Dados Relacional**. Local, Campinas, 46f. Tese (Mestrado em Engenharia de Computação) – UNICAMP, Universidade Estadual de Campinas. Instituto de Computação.

Apêndice A: Base de Dados

A base de dados do sistema integrado é composta por um conjunto de tabelas relacionadas, a qual representa o arquivo físico de dados, armazenada em dispositivos periféricos, disponíveis para consulta e atualização pelo usuário.

As tabelas representam toda a estrutura de armazenamento de dados (arquivos) do sistema, no qual os dados são armazenados em linhas (registros) e colunas (campos). A seguir serão descritas todas as tabelas a serem utilizadas pelo sistema.

O sistema desenvolvido opera em conjunto com o sistema gerenciador de banco de dados relacional *Interbase*, este, instalado em um microcomputador localizado na rede local, responsável pelo armazenamento e gerenciamento de todo o banco de dados do sistema.

As tabelas foram geradas utilizando o programa aplicativo *IB Manager 3* [17], uma ferramenta de alto desempenho para administração de bases de dados em sistemas gerenciadores de banco de dados *Interbase*. O programa dispõe de uma interface gráfica de fácil utilização para manutenção de bases de dados, gerenciamento de tabelas, construção de comandos em *SQL*, administração de usuários e outros privilégios, impressão, etc.

Apêndice B: Comandos *SQL* utilizados para a criação das tabelas do banco de dados do sistema.

Abaixo são apresentados os comandos em *SQL* utilizados para a criação das tabelas utilizadas no sistema. Nessas tabelas, são descritos os tipos de dados de cada campo, seu tamanho e definição de chave primária. Elas são utilizadas para armazenamento de todos os dados utilizados pelo sistema. Descreve-se a seguir cada item da tabela, gerado pelo programa aplicativo *IB Manager*:

- ✓ *Table*: especifica o nome dado a tabela;
- ✓ *Fields*: campos da tabela
  - o *Name*: indica o nome atribuído aos campos da tabela;
  - o *Type:* tipo de dado declarado podendo ser:
    - *Blob*: tipo de dado binário, usado para armazenamento de gráficos ou sons;
    - *Character*: usado para armazenamento de *strings* com tamanho de dados fixo (1 até 32767 caracteres);
    - *Date*: usado para armazenamento de datas;
    - *Decimal*: tipo de dado numérico inteiro;
    - *Double Precision*: tipo de dado real, 64 bits, com precisão de até 15 casas decimais;
    - *Float*: tipo de dado real, 32 bits, com precisão de até 7 casas decimais;
    - *Integer*: tipo de dado numérico positivo ou negativo (-2.147.483.648 a 2.147.483.648).

- *Numeric*: tipo de dado numérico, usado para armazenar números reais com ponto decimal fixo. O usuário especifica o tamanho, podendo ser de 1 a 15 dígitos;
- *Smallint*: tipo de dado numérico, usado para armazenar valores inteiros entre -32.768 a 32.767;
- *Varchar*: caracteres alfanuméricos (1 a 32767 caracteres)

  - *Domain*: especifica o domínio do dado;
  - *Not Null*: indica se o campo declarado pode assumir ou não um valor nulo;

- ✓ *Objects, that depend on table* <nome da tabela>: objetos dependentes da tabela <nome da tabela>
  - *Name*: nome do objeto;
  - *Type*: tipo do objeto;
  - *Field*: campo do objeto;
- ✓ *Constraints*: especifica os relacionamentos, chaves estrangeiras e primárias:
  - *Constraint Name*: nome das chaves primárias e estrangeiras;
  - *Type*: tipo (chave primária ou estrangeira);
  - *On Field*: nome do campo (ligação);
- ✓ *FK Table*: nome da tabela a qual esta relacionada (tabela estrangeira);
- ✓ *Update Rule*: tipo da regra;
- ✓ *FK Field*: nome do campo (chave estrangeira);
- ✓ *Delete Rule*: restrições de regras;
- ✓ *Índices*: nome dos campos índices;
  - *Index Name*: nome do índice;
  - *On Field*: nome do campo (ligação);

- *Unique*: se o campo é único;
- *Active*: se o campo esta ativo;
- *Sorting*: modo de ordenação.

APENDICE C: Tabela de Usuários do Sistema

```
/* Tabela: TBUsuario */
CREATE TABLE "TBUsuario" (
    "Cd_Usu" SMALLINT NOT NULL,
    "IBT_Usu" VARCHAR (16) CHARACTER SET ASCII COLLATE ASCII,
    "Nome_Usu" CHAR (50) CHARACTER SET WIN1251 COLLATE WIN1251,
    "Sexo_Usu" VARCHAR (15) CHARACTER SET ASCII COLLATE ASCII,
    "End_Usu" VARCHAR (50) CHARACTER SET ASCII COLLATE ASCII,
    "Compl_Usu" VARCHAR (15) CHARACTER SET ASCII COLLATE ASCII,
    "Cidade_Usu" CHAR (20) CHARACTER SET ASCII COLLATE ASCII,
    "Estado_Usu" CHAR (2) CHARACTER SET ASCII COLLATE ASCII,
    "CEP_Usu" INTEGER,
    "Tel_Fixo_Usu" INTEGER,
    "Tel_Cel_Usu" INTEGER,
    "EMail_Usu" VARCHAR (50) CHARACTER SET ASCII COLLATE ASCII,
    "Foto_Usu" BLOB sub_type 2 segment size 1,
    "Funcao_Usu" CHAR (20) CHARACTER SET ASCII COLLATE ASCII,
    "Cd_Lab" SMALLINT,
    "Status_ES" SMALLINT);
/* Primary keys definition */
ALTER TABLE "TBUsuario" ADD CONSTRAINT "PK_TBUsuario" PRIMARY KEY
("Cd_Usu");
/* Trigger: "AI_TBUsuario_Cd_Usu" */
CREATE TRIGGER "AI_TBUsuario_Cd_Usu" FOR "TBUsuario" ACTIVE
BEFORE INSERT POSITION 0
```

```
AS
BEGIN
  IF (NEW."Cd_Usu" IS NULL) THEN
     NEW."Cd_Usu" = GEN_ID("TBUsuario_Cd_Usu_GEN", 1);
END
```

Apêndice D: Comandos *SQL* utilizados para a criação da tabela de laboratórios do sistema.

/* Tabela: TBLabo */

```
CREATE TABLE "TBLabo" (
"Cd_Labo" SMALLINT NOT NULL,
"Nome_Labo" VARCHAR (80) CHARACTER SET WIN1251 COLLATE WIN1251,
"Resp_Labo" VARCHAR (80) CHARACTER SET WIN1251 COLLATE WIN1251,
"Email_Labo" VARCHAR (35) CHARACTER SET WIN1251 COLLATE WIN1251,
"Username_Labo" VARCHAR (8) CHARACTER SET WIN1251 COLLATE WIN1251,
"Senha_Labo" VARCHAR (8) CHARACTER SET WIN1251 COLLATE WIN1251,
"Acesso_Adm" CHAR (6) CHARACTER SET WIN1251 COLLATE WIN1251);
```

/* Primary keys definition */

```
ALTER TABLE "TBLabo" ADD CONSTRAINT "PK_TBLabo" PRIMARY KEY
("Cd_Labo");
```

Apêndice E: Comandos *SQL* utilizados para a criação da tabela de permissões do sistema.

/* Tabela: TBRest */

```
CREATE TABLE "TBRest" (
"Cd_Rest" SMALLINT NOT NULL,
"IBT_Usu" VARCHAR (16) CHARACTER SET WIN1251 COLLATE WIN1251,
"Segunda" VARCHAR (10),
"HrIni_Seg" TIME,
"HrFin_Seg" TIME,
"Terca" VARCHAR (10),
"HrIni_Ter" TIME,
"HrFin_Ter" TIME,
"Quarta" VARCHAR (10) CHARACTER SET WIN1251 COLLATE WIN1251,
"HrIni_Qua" TIME,
"HrFin_Qua" TIME,
"Quinta" VARCHAR (10) CHARACTER SET WIN1251 COLLATE WIN1251,
"HrIni_Qui" TIME,
"HrFin_Qui" TIME,
"Sexta" VARCHAR (10) CHARACTER SET WIN1251 COLLATE WIN1251,
"HrIni_Sex" TIME,
"HrFin_Sex" TIME,
"Sabado" VARCHAR (10) CHARACTER SET WIN1251 COLLATE WIN1251,
"HrIni_Sab" TIME,
"HrFin_Sab" TIME,
"Domingo" VARCHAR (10) CHARACTER SET WIN1251 COLLATE WIN1251,
```

```
"HrIni_Dom" TIME,
"HrFin_Dom" TIME,
"Irrestrito" VARCHAR (10) CHARACTER SET WIN1251 COLLATE WIN1251,
"Periodo" VARCHAR (10) CHARACTER SET WIN1251 COLLATE WIN1251,
"Per_Ini" DATE,
"Per_Fin" DATE,
"TrintaDias" VARCHAR (10) CHARACTER SET WIN1251 COLLATE WIN1251,
"Cd_Labo" SMALLINT);
/* Primary keys definition */
ALTER TABLE "TBRest" ADD CONSTRAINT "PK_TBRest" PRIMARY KEY
("Cd_Rest");
```

Apêndice F: Comandos *SQL* utilizados para a criação da tabela de eventos do sistema.

/* Tabela: TBEvento */

```
CREATE TABLE "TBEvento" (
"IBT_Usu" VARCHAR (18) CHARACTER SET ASCII COLLATE ASCII,
"Hora_Evento" TIME,
"Data_Evento" VARCHAR (12),
"StatusEvento" SMALLINT,
"Tipo_Evento" VARCHAR (12) CHARACTER SET WIN1251 COLLATE WIN1251,
"Cd_Lab" SMALLINT);
```

Apêndice G: Comandos *SQL* utilizados para a criação da tabela de e-mail do sistema.

/* Tabela: TBEmail */

```
CREATE TABLE "TBEmail" (
"Cd_Email" VARCHAR (5) CHARACTER SET WIN1251 NOT NULL COLLATE
WIN1251,
"DestinatMail" VARCHAR (50) CHARACTER SET WIN1251 COLLATE WIN1251,
"AssuntoMail" VARCHAR (60) CHARACTER SET WIN1251 COLLATE WIN1251,
"TextoMail" VARCHAR (100) CHARACTER SET WIN1251 COLLATE WIN1251,
"RemetMail" VARCHAR (50) CHARACTER SET WIN1251 COLLATE WIN1251,
"NomeRemetMail" VARCHAR (50) CHARACTER SET WIN1251 COLLATE WIN1251,
"AutUsuarioMail" VARCHAR (20) CHARACTER SET WIN1251 COLLATE WIN1251,
"AutSenhaMail" VARCHAR (20) CHARACTER SET WIN1251 COLLATE WIN1251,
"ServerHostMail" VARCHAR (30) CHARACTER SET WIN1251 COLLATE WIN1251,
"ServerPortMail" VARCHAR (5));
/* Primary keys definition */
ALTER TABLE "TBEmail" ADD CONSTRAINT "PK_TBEmail" PRIMARY KEY
("Cd_Email");
```

Apêndice H: Comandos *SQL* utilizados para a criação da tabela de administrador do sistema.

/* Tabela: TBAdmin */

```
CREATE TABLE "TBAdmin" (
"Cod_Admin" SMALLINT,
"Nome_Admin" VARCHAR (60) CHARACTER SET WIN1251 COLLATE WIN1251,
"Email_Admin" VARCHAR (60) CHARACTER SET WIN1251 COLLATE WIN1251,
"Username_Admin" VARCHAR (8) CHARACTER SET WIN1251 COLLATE WIN1251,
"Senha_Admin" VARCHAR (8) CHARACTER SET WIN1251 COLLATE WIN1251);
```

Apêndice I: Comandos *SQL* utilizados para a criação da tabela de status de eventos do sistema.

/* Tabela: TBStatusEvento*/

```
CREATE TABLE "TBStatusEvento" (
"Cd_Status" SMALLINT,
"Desc_Status" CHAR (100) CHARACTER SET WIN1251 COLLATE WIN1251);
```

Anexo A: Rotina TMEX - TMExtendedStartSession.

Essa chamada API atribui como argumento o número da porta “PortNum” e o tipo de porta “PortType” da Microlan a ser usada. Ela retorna o número da sessão e se a sessão foi estabelecida, o 0 indica que a MicroLan esta ocupada com uma outra sessão. Uma sessão com valor menor que 0 indica que não existe driver compatível para o tipo de porta. É recomendado que a chamada API TMGetTypeVersion seja utilizado para verificar o tipo de porta. A manipulação dessa sessão deve ser usada primeiro antes da função TMEndSession. A manipulação da sessão é eficiente se nenhum outro aplicativo tentar iniciar uma nova sessão na MicroLan.

Função Microsoft Windows TMEX DLL :

Linguagem Pascal TMExtendedStartSession(short PortNum, short PortType, void far *Reserved);

Essa função retorna:

>0 : porta determinada, valor de sessão retornado;

=0 : porta não determinada;

<0 : Sessão TMEX retornou código de erro.

Anexo B: Rotina TMEX – Get_Version.

Essa função API retorna em uma string a identificação da versão do driver TMEX. Tenha certeza que o buffer possua 80 bytes, pois dessa forma o número de identificação pode ser armazenado na string.

Função Microsoft Windows TMEX DLL:

short far pascal Get_Version(char far *ID_buf);

A função retorna:

1: String é armazenada na variável ID_buf;

0: Erro, não pode ser obtido a identificação.

Anexo C: Rotina TMEX – TMSetup.

Essa função API deve ser chamada antes de qualquer outra função TMEX. Nenhuma outra chamada a funções API funcionará corretamente na MicroLan até que essa função seja chamada. É aconselhável que ela seja chamada no começo da aplicação para inicializar a MicroLan realizando a verificação da integridade física da MicroLan.

Note que a execução dessa função reinicia todas as partes da MicroLan especificada.

Função Microsoft Windows TMEX DLL:

short far pascal TMSetup(long session_handle);

A função retorna:

0: Falha de Setup;

1: Setup OK;

2: Setup OK but MicroLan esta em curto circuito;

3: MicroLan não existe;

4: Comando TMSetup não suportado;

-200: Sessão inválida.

Anexo D: Rotina TMEX – TMRom.

Essa chamada API transfere os dados da ROM padrão entre os oito bytes internos fornecidos pelo driver para o vetor ‘ROM” de oito inteiros declarado no início do programa aplicativo. A direção da transferência de dados é especificada pelo primeiro valor inteiro do vetor. Se o primeiro inteiro é zero, então os oito bytes do buffer interno são transferidos nas oito variáveis inteiras do vetor. Se o primeiro inteiro é diferente de zero, então os oito bytes menos significativos para o buffer interno de oito bytes. Essa função é utilizada no programa aplicativo para obter os dados do número de ROM do dispositivo que foram encontrados com as funções TMFirst e TMNext.

Função Microsoft Windows TMEX DLL:

short far pascal TMRom(long session_handle, void far *state_buffer, short far *
ROM);

A função retorna:

1: Número de ROM lido ou armazenado;

<0: Erro na MicroLan.

Anexo E: Rotina TMEX – TMFirst.

Essa chamada API procura o primeiro dispositivo multi-drop na MicroLan especificado com o ‘session_handle’. O algorítmo de procura da ROM é usado para procurar o número de série único (ROM) na MicroLan. Os dados padrões da ROM que são encontrados são armazenados no buffer interno de oito bytes. O buffer interno de oito bytes pode ser lido usando fazendo chamada à API TMRom.

Função Microsoft Windows TMEX DLL:

short far pascal TMFirst(long session_handle, void far *state_buffer);

A função retorna:

0: dispositivo não encontrado;

1: encontrado primeiro dispositivo na MicroLan;

<0: erro de rede.

Anexo F: Rotina TMEX – TMNext.

Essa chamada API procura o próximo dispositivo na MicroLan especificado com o ‘session_handle’. O algorítmo de procura da ROM é usado para procurar o próximo número de registro único (ROM) de dados na MicroLan.

Os dados padrões que são encontrados são armazenados em um buffer interno de oito bytes. O buffer interno de oito bytes pode ser lido fazendo chamada a API TMRom. Após o último dispositivo encontrado na MicroLan, a próxima chamada a API TMNext irá retornar zero. Quando o TMNext retorna o valor zero, o algorítmo de procurara reinicializar e a próxima chamada a API TMNext sera equivalente a uma chamada a API TMFirst.

Função Microsoft Windows TMEX DLL:

short far pascal TMNext(long session_handle, void far *state_buffer);

A função retorna:

0: dispositivo não encontrado;

1: próximo dispositivo na MicroLan encontrado;

<0: erro de rede.

Anexo G: Rotina TMEX – TMEndSession.

A chamada a essa API finaliza a sessão na MicroLan que é designada pelo manipulador de sessões ‘session_handle’. Ele retorna o valor um se a sessão especificada pelo ‘session_handle’ foi finalizada com sucesso, e zero se a sessão estabelecida não é valida.

Função Microsoft Windows TMEX DLL:

short far pascal TMEndSession(long session_handle);

A função retorna:

0: session_handle realize leitura inválida;

1 => session ended, session_handle no longer valid

<0 => TMEX Session Error Return Codes

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo