EDSON HIROSHI WATANABE

APLICAÇÃO DE SOFTWARE ABERTO EM REDES INDUSTRIAIS

CURITIBA

2006

EDSON HIROSHI WATANABE

APLICAÇÃO DE SOFTWARE ABERTO EM REDES INDUSTRIAIS

Dissertação apresentada ao Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal do Paraná, como requisito parcial à obtenção do título de Mestre em Engenharia Elétrica.

Área de concentração: Sistemas Eletrônicos

Linha de pesquisa: Sistemas Abertos em Indústria

Orientador: Prof. Titular José Manoel Fernandes, PhD.

CURITIBA

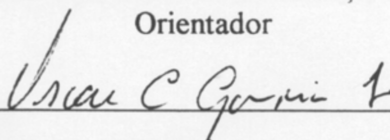
2006

APLICAÇÃO DE SOFTWARE ABERTO EM REDES INDUSTRIAIS

Edson Hiroshi Watanabe

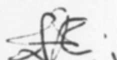
Dissertação aprovada como requisito parcial para obtenção do grau de Mestre no Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal do Paraná - UFPR.

Prof. José Manoel Fernandes, PhD.
Orientador

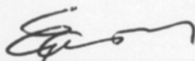


Prof. Oscar da Costa Gouveia Filho, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica - UFPR

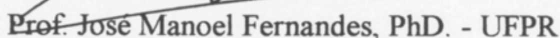
Banca Examinadora



Prof. Leandro dos Santos Coelho, Dr. – PUCPR



Prof. Evélio Martín García Fernández, Dr. – UFPR



Prof. José Manoel Fernandes, PhD. - UFPR

Curitiba, 04 de abril de 2006

Sinto muitas saudades de uma pessoa muito especial, que foi o meu primeiro orientador e apoiador. Ele dedicou a sua vida pela família, mas durante minha caminhada foi retirado subitamente do nosso convívio em 09.02.2004. Dedico este trabalho a ele, pois é o mínimo que posso fazer para dizer-lhe muito obrigado, meu Pai. “Porquanto o Senhor mesmo, dada a sua palavra de ordem, descerá dos céus, e os mortos em Cristo ressuscitarão primeiro”. **I Tessalonicenses 4:16**

Dedico este trabalho também a minha mãe que, desde minha infância, me ensinou que pela educação é que alcançaríamos a liberdade e dedicou sua vida para dar-nos um mínimo de educação, apesar de não ter tido oportunidade de estudar além do ensino fundamental. “Ensina a criança no caminho em que deve andar, e, ainda quando velho não desviará dele”. **Provérbios 22:6**

AGRADECIMENTOS

Agradeço, a Deus, pela vida que me preservou, orientações, sabedoria recebida em momentos de pouca inspiração. Nos momentos mais difíceis de minha vida, não deixou eu desistir, me incentivou e mostrou o Teu querer. Ele foi minha força em momentos de minha fraqueza. “Lâmpadas para os meus pés é a Tua Palavra e, luz para os meus caminhos”. **Sal. 119:105**

Agradeço ao professor José Manoel que confiou em mim, sempre me apoiou para que este trabalho chegasse ao seu final, passou por momentos difíceis de saúde física, mas graças a Deus se restabeleceu e está de volta em nosso meio, podendo continuar a transmitir os seus conhecimentos a muitos alunos ainda por vir. Neste período que convivemos juntos, aprendi a respeitá-lo e admirá-lo. Muito obrigado, meu orientador e amigo.

À minha esposa, Ana, a quem sou muito grato pelo apoio e muita ajuda recebida no transcorrer de toda caminhada nestes dois últimos anos. Cada dia é grande o meu respeito e admiração por ela, pela sua paciência e dedicação pela família, incentivou-me a não desistir, me apoiando em momentos difíceis passados.

Agradeço aos meus filhos Rodrigo, Guilherme e Daniel por tolerarem a falta do pai em algumas situações que se passaram em seus aprendizados. Vocês são bênçãos de Deus.

Agradeço aos professores da UFPR que passaram na minha trajetória até aqui, partilhando seus conhecimentos e orientações, e também de outras instituições pelos ensinos e apoios recebidos.

Agradeço ao meu amigo e camarada de longas datas, Clovis Lacava Lordello, companheiro desde a Siemens, estudioso, dedicado, foi um grande suporte com seus profundos conhecimentos de sistemas computacionais.

Agradeço às pessoas que passaram pela minha vida, nesse período, e deixaram suas contribuições.

“Vi novo céu e nova terra, pois o primeiro céu e a primeira terra passaram, e o mar não existe mais. Deus enxugará dos olhos toda lágrima, e a morte já não existirá, já não haverá mais luto, nem pranto, nem dor[...]”. Apocalipse 21:1-4

RESUMO

Nesta dissertação é proposta a utilização de produtos desenvolvidos com o conceito de sistemas abertos em redes industriais. Este trabalho objetiva apresentar um modelo aberto de comunicação em redes de sistemas de automação industrial e mostrar a sua implementação. Neste estudo é discutido o sistema industrial aberto baseado no sistema operacional *LINUX*, características de portabilidade e embarcabilidade do *LINUX*, tecnologia e arquitetura de rede focado no *Fieldbus*, protocolo e barramento industrial *CAN – Control Area Network.* Neste contexto, é proposta uma nova configuração para redes industriais com componentes abertos de redes factíveis de serem colocados em prática.

Palavras-chave: Sistemas Abertos. Sistema Embarcado. *LINUX. Filedbus. CAN*. Redes Industriais.

ABSTRACT

This work shows the use of the developed products with open system concept at the industrial network. The main objective of this work is to present an open model of communication in industrial automation systems networks and to show its implementation. This study will discuss the opened industrial system based in *LINUX* operating system, *LINUX* portability and embedded features, network technology and architecture focused in Fieldbus, Control Area Network (*CAN*) protocol and industrial bus and it will propose a new configuration for industrial networks with open components feasible networks to be used in practical application.

Keywords: Open System. Embedded System. Fieldbus. *LINUX*. *CAN*. Industrial Networks.

SUMÁRIO

1 INTRODUÇÃO **16**

2 TECNOLOGIA DE REDES INDUSTRIAIS **19**

2.1 ARQUITETURA E PROTOCOLO DE REDE INDUSTRIAL 19
2.2 PADRÃO DE REDE INDUSTRIAL - BARRAMENTO DE CAMPO OU FIELDBUS 22
2.2.1 Profibus 24
2.2.2 Interbus 25
2.2.3 DeviceNet 26
2.2.4 Ethernet Industrial 28

3 BARRAMENTO INDUSTRIAL *CAN* (*CONTROLLER AREA NETWORK*) **32**

3.1 ASPECTOS TÉCNICOS 32
3.2 HISTÓRICO 32
3.3 ESTRUTURA DA REDE CAN 34
3.4 CARACTERÍSTICAS GERAIS DO CAN 35
3.5 PRINCÍPIO DE FUNCIONAMENTO DA REDE CAN 38
3.5.1 Processo de Arbitragem não Destrutiva 40
3.6 PROTOCOLO DE COMUNICAÇÃO CAN E O PADRÃO ISO/OSI 42
3.7 INFORMAÇÕES DA CAMADA DE ENLACE 44
3.8 INFORMAÇÕES DA CAMADA FÍSICA 44
3.8.1 Representação do Bit 47
3.8.2 Sincronização do Bit 47
3.9 CARACTERÍSTICAS DO MEIO FÍSICO 49
3.10 TOLERÂNCIA À FALHAS NO BARRAMENTO 50
3.11 MÉTODOS DE DETECÇÃO E SINALIZAÇÃO DE ERRO 51
3.12 DETECÇÃO DE ERROS 51
3.13 A EVOLUÇÃO DO CAN 52

4 SISTEMAS ABERTOS **56**

4.1 LINUX E GNU IS NOT UNIX (GNU – GNU NÃO É UNIX) 59
4.2 SISTEMA OPERACIONAL LINUX 62
4.3 REQUISITOS DO SISTEMA PARA A INSTALAÇÃO DO LINUX 64
4.4 LINUX EM SISTEMA EMBARCADO 67
4.4.1 Restrições para Sistemas Embarcados 67
4.5 UMA BREVE VISÃO SOBRE ETAPAS DE PORTAGEM DO LINUX 69

5 ESTUDO DE CASO **73**

5.1 PROPOSTA DE UMA NOVA CONFIGURAÇÃO PARA REDES INDUSTRIAIS 73
5.2 LECU MASTER 75
5.3 LECU SLAVE 76
5.4 CLP PROPOSTO COM SISTEMA OPERACIONAL LINUX 78
5.5 CONFIGURAÇÕES DO SISTEMA 79

6 IMPLEMENTAÇÃO E TESTES DO CONCEITO PROPOSTO **83**

6.1 EVIDÊNCIA DO FUNCIONAMENTO DO PROTOCOLO CAN 84
6.2 TESTES DE TROCA DE PACOTES - GERÊNCIA DA REDE INDUSTRIAL PROPOSTA 85
6.3 ESQUEMAS DE MONTAGEM 87
6.4 PROCEDIMENTOS DOS TESTES 89
6.4.1 Configuração dos equipamentos utilizados nos testes 89
6.4.2 Instalação e preparação da rede LINUX 90
6.4.3 Depuração da rede LINUX 91
6.4.4 Instalação do pacote NET-SNMP 92
6.4.5 Configuração do pacote NET-SNMP 94

6.4.6 Criação de scripts de testes de troca de mensagens entre nós ... *96*
6.4.7 Roteiro de execução dos Testes e Resultados ... *102*

7 CONCLUSÕES ... **112**

REFERÊNCIAS ... **116**

APÊNDICES ... **120**

Apêndice A – Características de rede e gerência do *LINUX* ... **120**

Apêndice B – Informações complementares do barramento CAN ... **120**

Apêndice C - Protocolos de gerenciamento de redes ... **120**

Apêndice D - Listagem de códigos do arquivo de programas *Scripts* ... **120**

Apêndice E - Trabalho apresentado no ISA-2005 em Chicago - EUA ... **120**

A. CARACTERÍSTICAS DE REDE E GERÊNCIA DO *LINUX* ... **121**

A1. TCP/IP (TRANSMISSION CONTROL PROTOCOL/INTERNET PROTOCOL, PROTOCOLO DE CONTROLE DE TRANSMISSÃO/PROTOCOLO INTERNET) ... 121
A2. CAMADAS DO TCP/IP ... 121
A3. CAMADA DE REDE ... 122
A4. CAMADA INTER-REDE ... 123
A5. CAMADA DE TRANSPORTE ... 124
A6. CAMADA DE APLICAÇÃO ... 125
A7 MODELO ISO/OSI ... 126
A8 DEPURAÇÃO DE REDES TCP-IP ... 127

B. INFORMAÇÕES COMPLEMENTARES DO BARRAMENTO *CAN* ... **131**

B1. TIPOS DE MENSAGENS NO BARRAMENTO CAN ... 131
B1.1 Mensagem de Dados ... *131*
B.1.2 Mensagem Remota ... *137*
B.1.3 Mensagem de Erro ... *138*
B.1.4 Mensagem de Sobrecarga ... *139*
B2 CONTROLE DE ERRO NO BARRAMENTO CAN ... 139
B2.1 Erros ao Nível da Mensagem ... *139*
B2.2 Erros ao Nível do BIT ... *140*

C. PROTOCOLOS DE GERENCIAMENTO DE REDES ... **142**

C1. ORGANIZAÇÃO SNMP ... 143
C2 O PACOTE NET-SNMP ... 145
C3 OPERAÇÃO DE PROTOCOLOS SNMP ... 146
C4 FERRAMENTAS NET-SNMP ... 146

D. LISTAGEM DE CÓDIGOS DO ARQUIVO DE PROGRAMAS SCRIPTS ... **148**

D1. LISTA DO ARQUIVO TRIND.C GERADO PELO COMANDO MIB2C ... 148
D2. LISTA DO ARQUIVO TRIND.H GERADO PELO COMANDO MIB2C ... 154
D3. LISTA DO ARQUIVO TRIND.C ALTERADO ... 155
D4. LISTA DO ARQUIVO TRIND.H ALTERADO ... 168

E. TRABALHO APRESENTADO NO ISA-2005 EM CHICAGO - EUA ... **169**

LISTA DE FIGURAS

FIGURA 1 – EQUIPAMENTOS QUE COMPÕEM UMA REDE INDUSTRIAL. 20
FIGURA 2 – EQUIPAMENTOS QUE COMPÕEM NÓS INTELIGENTES. 21
FIGURA 3 - COMPONENTES DA REDE *CAN*. 34
FIGURA 4 – APLICAÇÃO DE *CAN* EM SISTEMAS INDUSTRIAIS. 39
FIGURA 5 - MODELO DAS CAMADAS *ISO/OSI*. 42
FIGURA 6 - CAMADAS DA REDE *CAN*. 43
FIGURA 7 – ESTRUTURA TÍPICA DE UM NÓ DA REDE *CAN*. 45
FIGURA 8 – SEGMENTOS DO *BIT TIME NOMINAL*. 48
FIGURA 9 – NÍVEIS DE TENSÃO DA REDE *CAN*. 50
FIGURA 10 - SISTEMAS ATUAIS. 53
FIGURA 11 - SISTEMAS FUTUROS. 54
FIGURA 12 – CONFIGURAÇÃO CONVENCIONAL. 80
FIGURA 13 – CONFIGURAÇÃO DISTRIBUÍDA. 81
FIGURA 14 – CONFIGURAÇÃO DISTRIBUÍDA. 85
FIGURA 15 - MONTAGEM DOS TESTES DESEJADOS. 87
FIGURA 16 - MONTAGEM DOS TESTES EXECUTADOS. 88
FIGURA 17 – EXEMPLO DE APLICAÇÃO. 96
FIGURA A1 – MODELO DE REDE *TCP/IP*. 122
FIGURA A2 – MODELO DE CAMADAS OSI E PROTOCOLO *TCP/IP*. 127
FIGURA B1 - CAMPOS DAS MENSAGENS DE DADOS. 132
FIGURA B2 - CAMPO DE ARBITRAGEM DO FORMATO *CAN* 2.0A. 132
FIGURA B3 - CAMPO DE ARBITRAGEM DO FORMATO *CAN* 2.0B. 133
FIGURA B4 - CAMPO DE CONTROLE *CAN* 2.A E *CAN* 2.B – QUADRO ESTENDIDO. 134
FIGURA B5 - CAMPO DE CONTROLE *CAN* 2.B – QUADRO PADRÃO. 135
FIGURA B6 - CAMPO DO CRC. 136
FIGURA B7 - CAMPO DE ACK. 137
FIGURA B8 - CAMPOS DA MENSAGEM REMOTA. 137
FIGURA B9 – MENSAGEM DE ERRO. 138
FIGURA B10 - MENSAGEM DE SOBRECARGA. 139
FIGURA C1 - ESTRUTURA LÓGICA DA *MIB*. 144

LISTA DE TABELAS

TABELA 1 – PADRÃO *IEEE 802*. 30
TABELA B1 - CODIFICAÇÃO DO NÚMERO DE BYTES DE MENSAGEM. 135
TABELA C1 – LISTA DE LINHA DE COMANDO NO PACOTE NET-SNMP. 147

LISTA DE ABREVIATURAS e SIGLAS

ARP – *Address Resolution Protocol*

ASN.1 – *Abstract Syntax Notation One*

ATM –*Asynchronous Transfer Mode*

CAN – *Controller Area Network*

CCITT - *Consultative Committee for International Telegraph and Telephone*

CLP – *Controlador Lógico Programável*

CNC – *Controle Numérico Computadorizado*

CPU – *Central Processor Unit*

CRC – Cyclic Redundancy Check

CSMA/CD with NDA - *Carrier Sense Multiple Access / Collision Detection with Non-Destructive Arbitration*

DIX Standart – *DEC-Intel-Xerox Standard*

DHCP – *Dynamic Host Configuration Protocol*

DNS – *Domain Name System*

FDDI – *Fiber Distributed Data Interconnection*

FreeBSD – *Free Berkeley Software Distribution*

FTP – *File Transfer Protocol*

FSF – *Free Software Foundation*

GNU – GNU IS NOT UNIX

GPL – *GNU Public License*

HTTP – *Hypertext Transfer Telnet Protocol*

ICMP – *Internet Control Message Protocol*

IEEE – *Institute of Electrical and Electronics Engineers*

IHM – *Interface Human Machine*

IP – *Internet Protocol*

ISO – *International Organization for Standardization*

KDE - *K Desktop Enviroment*

LAN – *Local Area Network*

MAC – Medium Access Control

LECU – *Linux Electronic Control Unit*

MIT – *Massachusetts Institute of Technology*

MMU – *Memory Management Unit*

MIB – *Management Information Base*

NAT – *Network Address Translator*

NET-SNMP – Network – Simple Network Management Protocol

NRZ - *Non Return to Zero*

ODVA – *Open DeviceNet Vendor Association*

OID – *Object Identifier*

OSI – Open System Interconnection

PID – Proportional Integral Derivative

PPP – Point to Point Protocol

Profibus *DP – Decentralized Peripherical*

Profibus *FMS – Field Message Specification*

Profibus *PA- Process Automation*

RARP – Reverse Address Resolution Protocol

RFC – *Request for Comments*

RTLINUX – Real Time LINUX

RM-OSI – Reference Model - *Open Systems Interconnection*

SCSI – Small Computer Systems Interface

SDCD – *Sistema Digital de Controle Distribuído*

SLIO – Serial Linked Input Output

SMTP – Simple Mail Transfer Protocol

SNMP – Simple Network Management Protocol

TCP/IP – *Transmission Control Protocol/Internet Protocol*

UDP – User Datagram Protocol

1 INTRODUÇÃO

Existe um consenso cada vez maior entre fornecedores e usuários de equipamentos e sistemas industriais sobre a necessidade da busca contínua de produtos com arquiteturas autônomas, independentes de fabricantes, que tenham alto desempenho, comprovados mecanismos de segurança e sejam tecnologicamente modernos e robustos. Estes produtos precisam: atender às novas exigências de controle, distribuição e armazenamento de informações; ter maior interoperabilidade entre plataformas de diferentes fabricantes; apresentar maior flexibilidade em manutenção e futuras atualizações.

Em sistemas de controle industriais convencionais, ainda é comum encontrar conexões interligando cada um dos pontos de monitoração até o centro de processamento de controle, como, por exemplo, os Controladores Lógicos Programáveis (*CLP*s), sendo necessária uma grande quantidade de cabos e tempo considerável de instalação, pois são feitas de cabos multi-vias. É intuito, deste trabalho, descrever as novas tecnologias de barramento e protocolos que vêm com uma importante função: suprir as necessidades da indústria em racionalizar o material gasto, reduzindo o tempo de instalação, facilitando a manutenção e, conseqüentemente, reduzindo o custo do projeto.

Dentre os novos conceitos tecnológicos, que surgiram recentemente, destaca-se a da utilização de tecnologias de sistemas abertos em empresas. As pesquisas recentes de mercado têm apontado que o sistema operacional *LINUX* (MDIC, 2005; PANIAGO, 2005) tem impulsionado mudanças no comportamento de muitas corporações. Tanto os fabricantes de equipamentos como os de *softwares* efetivamente consideram seriamente esta nova oportunidade em suas metas de vendas.

Muitos governos mundiais, tais como o da França, Alemanha e Rússia, adotaram os sistemas abertos como padrão para aquisição de produtos em seus departamentos federais. O governo brasileiro seguiu o mesmo caminho, estabeleceu um plano de ação denominado inclusão digital, para tornar a tecnologia acessível às classes desfavorecidas da sociedade. Através do uso de *softwares* livres as empresas desenvolvedoras de tecnologia, produzirão seus produtos com custos reduzidos, disseminando a idéia dos sistemas abertos.

Em termos de produtos industriais, pode-se ainda citar a tecnologia de barramento de campo, o protocolo *CAN*, como um dos aliados a estes novos conceitos, trazendo benefícios aos seus adeptos como redução de tempo de instalação de novas redes, redução na manutenção e custo de suporte. Na rede mundial *Internet,* podem ser encontradas empresas oferecendo produtos e serviços neste padrão de rede, reforçando o que foi escrito anteriormente.

O mundo caminha para uma vida mais participativa e globalizada de seus integrantes, em que as distâncias e barreiras são quebradas entre os povos cada vez mais rapidamente. Neste contexto, surge a idéia de desenvolvimento de tecnologia pelas comunidades em toda parte do mundo que se formam por interesses comuns. A idéia de sistemas abertos ganha a cada dia mais adeptos; o ser humano quer participar do desenvolvimento do mundo em que vive e ter o direito à liberdade. A indústria tem o desafio de alinhar-se a esses pensamentos vigentes na sociedade e com os seus interesses.

Este trabalho é dividido em capítulos. No capítulo 1, faz-se uma breve introdução sobre o tema tratado no trabalho é apresentado. O capítulo 2 descreve algumas tecnologias de redes industriais existentes no mercado, sobretudo protocolos *Fieldbus*, suas características e a sua evolução. O capítulo 3 aborda com profundidade as características funcionais do

protocolo *CAN*, pois ele faz parte de uma solução proposta neste trabalho. O capítulo 4 detalha os sistemas abertos, seu histórico, suas definições e órgãos reguladores. Neste capítulo, ainda, são discutidos os seus benefícios e mostradas as suas aplicações. São abordadas as características do *LINUX* como Sistema Operacional (GARRELS, 2005; MATTHEW, 1999), como embuti-lo (MOODY, 1998) em *hardwares* de computadores *PCs e também* em *hardwares* de aplicações específicos. No Capítulo 5, depois de conceituadas as mudanças no ambiente industrial, mostradas as novas tendências tecnológicas e o avanço dos sistemas abertos, é apresentado um estudo de caso com uma proposta de uma rede industrial, utilizando componentes abertos, o *LINUX* e o *CAN*. Neste capítulo é proposta a quebra de um paradigma, sugerindo que sistemas tradicionais como os *CLPs* industriais, os quais são equipamentos consolidados pelo mercado há muitos anos, sejam desafiados a usarem uma arquitetura aberta. No capítulo 6, são descritas as evidências e os testes de validações realizados em uma rede com arquitetura aberta; os procedimentos de testes e amostragem de resultados obtidos, os quais comprovam a viabilidade e a implementação deste modelo. Por fim, no Capítulo 7, seguem as conclusões deste trabalho e sugestões para futura pesquisa, deixando clara uma solução viável e desafiadora para as indústrias e motivando os leitores deste trabalho a buscarem novas alternativas na tomada de decisão. No apêndice, foram inseridos alguns temas que complementam os assuntos descritos em cada capítulo como: protocolo de rede *TCP/IP*, detalhes sobre o *CAN*, as listagens dos testes realizados e o trabalho apresentado sobre este tema na feira da ISA (*Instrumentation, Systems and Automation Society*) nos Estados Unidos da América em 2005.

2 TECNOLOGIA DE REDES INDUSTRIAIS

Neste capítulo são descritos, de modo simples, alguns dos barramentos mais utilizados nas indústrias.

2.1 ARQUITETURA E PROTOCOLO DE REDE INDUSTRIAL

Os sistemas de controle industriais tendem a tornarem-se complexos, com um grande número de variáveis, ações e controles. Conseqüentemente, um controle centralizado pode tornar-se caro, complexo e lento. Dividir o controle em partes menores, que possam ser controlados individualmente, passa a ser, então, uma solução atrativa por vários motivos: maior facilidade de desenvolvimento, operação, administração, confiabilidade do sistema como um todo (mau funcionamento de uma parte não implica necessariamente, no mau funcionamento de todo o sistema) e manutenção simplificada. Os Sistemas Digitais de Controle Distribuído – (SDCDs) são utilizados desde a década de 70 com sucesso, tendo como base a utilização de terminais remotos conectados aos dispositivos no campo e conectadas entre si a uma via de dados que, por sua vez, contém um elemento centralizador, que pode ser um *CLP*, um *PC* ou outro equipamento dedicado. A figura 1 ilustra esses equipamentos.

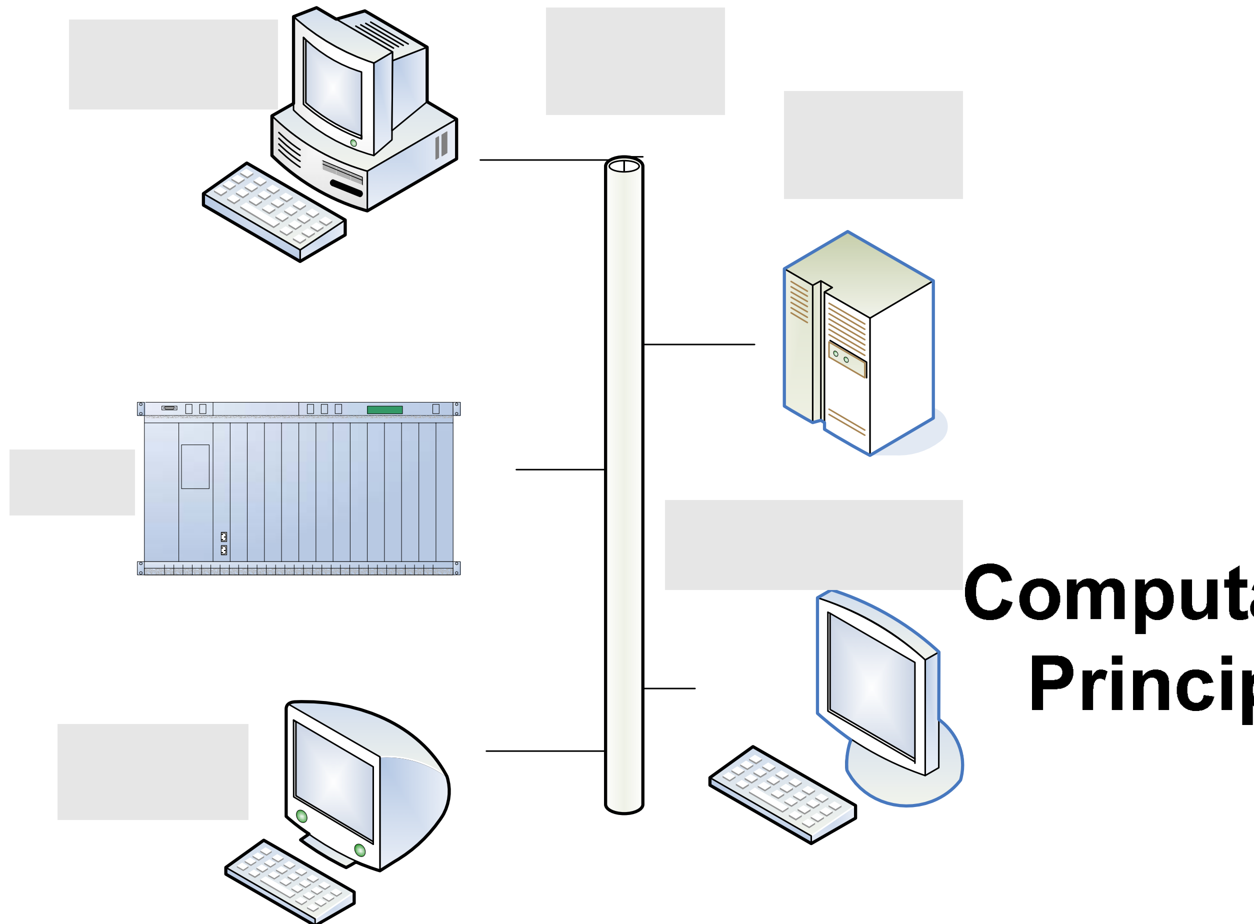


Figura 1 – Equipamentos que compõem uma Rede Industrial.
Fonte: O autor

Com a entrada da tecnologia de barramento de campo no mercado, o controle distribuído ganha uma nova alternativa, que é a utilização de dispositivos inteligentes. Esses dispositivos são dotados de alguma capacidade de processamento e interligados, através de um barramento, formando, assim, uma rede, possibilitando trocas de mensagens entre si e o controle do sistema de automação seja de responsabilidade da rede de dispositivos e não mais em um único elemento centralizador (*CLP*, *PC*, etc), originando o conceito de nós inteligentes, conforme apresentado na figura 2.

13 CLP

Figura 2 – Equipamentos que compõem nós inteligentes.
Fonte: Anônimo, 2004.

As interfaces elétricas tradicionais e outras medidas analógicas saem de cena e surgem as redes de dispositivos de campo que trafegam dados no formato digital, através de um protocolo que está sendo utilizado. A necessidade de existência do elemento centralizador fica por conta da supervisão e, conforme já descrito, não mais com a finalidade de controle. Os nós do sistema (dispositivos inteligentes conectados ao barramento de campo), tipicamente executam tarefas simples de sensoriamento, monitoração e atuação, mas, quando trabalham juntos, podem realizar tarefas complexas. Pode-se resumir a decisão de escolher uma solução em rede em relação a uma solução centralizada assim:

a) menos fios no sistema, o que torna a cabeamento mais simples e barato;

b) ligações curtas a sensores analógicos sensíveis a ruído elétrico, antes dos sinais serem convertidos em mensagens digitais "imunes" a ruído;

c) sistema flexível;

d) unidades funcionais podem ser adicionadas ou removidas de forma simples;

e) manutenção do sistema. A eletrônica de várias unidades pode ser idêntica, permitindo troca entre elas;

f) cada unidade pode ser desenvolvida e testada individualmente, de acordo com requisitos exigidos pelo sistema;

g) a partilha de dados entre as várias unidades pode eliminar redundância de informação.

Um ponto que poderia ser considerado desvantajoso, na solução em rede, é a necessidade obrigatória, de um sistema de gerenciamento dos elementos da rede mais elaborado, mais complexo e custoso monetariamente.

2.2 PADRÃO DE REDE INDUSTRIAL - BARRAMENTO DE CAMPO OU *FIELDBUS*

O barramento de Campo ou *Fieldbus,* também denominado de rede de barramento de entrada e saída (E/S), é um sistema de comunicação de dados bidirecional e serial que conecta dispositivos de campo e de controle.

O *Fieldbus* é um padrão aberto de barramento o qual permite que dispositivos de diferentes fabricantes sejam integrados em um só sistema, substituindo o cabeamento por um só barramento, formando uma rede de dispositivos, que podem ser acessados individualmente, utilizando mensagens padronizadas por um protocolo.

Um dos principais aspectos na definição de um barramento de campo é a sua padronização e disponibilização da tecnologia a vários fabricantes de forma a permitir interoperabilidade, ou seja, um dispositivo de campo pode ser substituído por um dispositivo similar com funcionalidade adicionada de um diferente fabricante na mesma rede de *Fieldbus* enquanto mantém uma determinada operação**.** A transmissão de uma informação de grandezas físicas (nível, pressão, temperatura, etc.) do dispositivo de medição a um *CLP* gera em

sistemas tradicionais no mínimo uma conversão Digital/Analógico (D/A) no dispositivo e uma conversão Analógico/Digital (A/D) no *CLP.* Estas conversões provocam erros que podem ser eliminados em um barramento de campo, em que a informação é transmitida digitalmente sem conversores. Portanto, o *Fieldbus* permite uma redução no trabalho do projeto, versatilidade na especificação do projeto, facilidade de instalação e manutenção física do sistema pela redução de conexões, redução de erros e, conseqüentemente, diminuição dos custos. Mas há um grande problema a ser enfrentado: a existência de inúmeros protocolos concorrentes. Dentre as mais utilizadas do mercado são citadas: *Fieldbus Foundation, Profibus, Interbus, DeviceNet* no padrão *Fieldbus* e o padrão *Ethernet Industrial.*

Pelo fato de não haver uma padronização global para a área de automação, cabe ao projetista analisar as diversas opções e procurar encontrar aquela que melhor atenda às necessidades do seu sistema.

Os barramentos de campo são classificados de acordo com o tipo de dispositivo a ser interconectado:

a) barramento de processo: Os instrumentos de processo que fornecem informações de grandezas físicas como pressão, temperatura, nível, vazão, etc. e utilizam um maior quantidade de *bits* (zeros e uns) para codificar a sua informação.

b) barramento de dispositivos: Os instrumentos que manipulam informações discretas, como atuadores *ON-OFF* (liga-desliga), sensores de posição (aberto-fechado), proximidade (longe-perto), etc.

Um sistema de automação pode conviver com diversos tipos de barramentos, conforme citados nas linhas a seguir:

a) barramento entre *CLPs*;

b) barramento entre *CLPs* e blocos remotos de E/S;

c) barramento entre instrumentos de processo;

d) barramento entre instrumentos discretos.

2.2.1 *Profibus*

Em 1986, grupos europeus de trabalho decidiram criar um padrão de comunicação com o intuito de interligar os diversos fabricantes de equipamentos e dispositivos industriais como *CLP*, *PC*, *CNC,* etc. para troca de informações entre eles. A base para esse trabalho foi o protocolo do fabricante Siemens denominado SINEC L2, e, em 1990, a maioria dos sistemas industriais podiam comunicar-se entre si, tornando-se uma das plataformas mais abertas do mundo (PROFIBUS, 1999).

O *Profibus* é um padrão aberto de barramento de campo para uma larga faixa de aplicações em automação de fabricação e processos. Ele destaca-se por atuar nos diversos níveis do processo industrial: ambiente de fábrica, processo e gerência. Oferece características diversas de protocolos de comunicações, tais como:

a) ***Profibus DP (Descentralized Peripherical):*** é o mais usado dentre os protocolos, ele é caracterizado pela velocidade, eficiência e baixo custo de conexão. Foi

projetado especialmente para comunicação entre sistemas de automação e periféricos distribuídos (PROFIBUS, 1999).

b) ***Profibus FMS (Field Message Specification)*:** é um protocolo de comunicação geral para as tarefas de comunicações solicitadas. *FMS* oferece muitas funções sofisticadas de aplicações para comunicação entre dispositivos inteligentes (PROFIBUS, 1999).

c) ***Profibus PA (Process Automation)*:** Este protocolo define os parâmetros e blocos de funções dos dispositivos de automação de processo, tais como transdutores de medidas, válvulas e *IHM* (*Interface Human Machine*) (PROFIBUS, 1999).

2.2.1.1 Características Técnicas

As características mais relevantes deste protocolo são:

a) *Profibus DP* e *FMS* utilizam como meio físico o padrão *RS-485*, com taxa de transmissão de 9 *kbps* a 12 *kbps*;

b) *Profibus PA* utiliza o padrão IEC 1158-2, comunicação com taxa de 31,25 *kbps*;

c) número máximo de estações em um segmento é de 32;

d) até 126 estações mestres e escravos no barramento.

2.2.2 *Interbus*

O Interbus é um dos mais conhecidos protocolos na área industrial, que utiliza o conceito *Fieldbus.* Os dispositivos de campo e de E/S são interligados igualmente e usados

em sistemas de controle. O barramento é composto por um cabo serial conectado às redes de sensores e atuadores para controlar dispositivos, equipamentos e células de produção. Ele também possibilita conectar-se aos sistemas de hierarquias superiores como os programas de gerenciamentos (INTERBUS, 2005).

2.2.2.1 Características técnicas

As características mais relevantes deste protocolo são:

a) topologia em anel - todos os dispositivos são integrados ativamente em um caminho de transmissão fechado. Cada dispositivo amplifica os sinais de entrada e envia-os ao seu destino, permitindo taxas de transmissões altíssimas em distâncias longas;

b) podem ser conectados até 512 dispositivos através de 16 níveis de rede;

c) taxa de transmissão de 500 *kbps*, comprimento da rede local de até 400m;

d) flexibilidade. O sistema mestre/escravo habilita conexão de até 512 dispositivos e 16 níveis de rede;

e) aplicações típicas: engenharia de processos, monitoração de sensores, e controle de atuadores.

2.2.3 *DeviceNet*

O Barramento DeviceNet (ODVA, 2004) foi desenvolvido pela *Allen Bradley,* em 1994, e, logo depois, tornou-se um protocolo aberto com a criação da *Open DeviceNet*

Vendors Association (ODVA). O *DeviceNet* é composto por padrões já consolidados no mercado como o barramento *CAN*, *softwares* de gerenciamento de redes e conta com uma rede ampla de distribuição do produto no mercado.

2.2.3.1 Características Técnicas

Seguem as características mais relevantes desse protocolo:

a) padrão aberto (ODVA);

b) topologia física em Barramento;

c) usa o protocolo *CAN* na camada de enlace;

d) linhas troncos e derivações;

e) conexão de até 64 elementos;

f) taxas de transmissão de 125 a 500 *kbps*, para distâncias de 500 a 100m respectivamente;

g) cabo de cobre com 5 fios;

h) alimentação (24V) e comunicação;

i) resistores de terminação de 121 Ω.

2.2.3.2 Mecanismos de aquisição de dados no *DeviceNet*

Os mecanismos de aquisição utilizados neste protocolo são:

a) *polling*: Mestre interroga o dispositivo ciclicamente;

b) cíclico : Um dispositivo analógico envia dados em intervalos pré-determinados;

c) mudança de Estado: Um dispositivo discreto envia dados quando ocorre uma mudança no seu estado.

2.2.4 *Ethernet* Industrial

A rede *Ethernet* passou por uma longa evolução, nos últimos anos, tornando-se a rede de melhor faixa e desempenho para uma variada gama de aplicações industriais. A Ethernet foi inicialmente concebida para ser uma rede de barramento *multidrop* (100Base-5) com conectores do tipo vampiro (*piercing*), mas este sistema mostrou-se de baixa praticidade. A evolução deu-se na direção de uma topologia estrela com par trançado. As velocidades da rede cresceram de 10 Mbps para 100 Mbps, e agora alcançam 1 Gbps (*IEEE802.3z* ou *Gigabit Ethernet*). A *Gigabit Ethernet* disputa com a tecnologia *ATM (Asynchronous Transfer Mode*) o direito de ser a espinha dorsal (*backbone*) das redes na empresa. A outra evolução dá-se no uso de um dispositivo de ligação dos elementos de rede denominado *HUBs* inteligentes com capacidade de comutação de mensagens e no uso de cabos *full duplex,* em substituição aos cabos *half duplex* mais comumente utilizados. Houve com estas mudanças uma significante redução na probabilidade de colisão de dados (SEIXAS JR, 2005).

2.2.4.1 Princípios básicos

Estas são algumas informações adicionais e os princípios básicos do protocolo Ethernet:

a) nos anos 70, a rede *Ethernet* nasceu dos laboratórios da empresa *Xerox*;

b) velocidade inicial: 2.94 Mbps;

c) utilização pelo comitê IEEE 802.3 da norma IEEE802.3 CSMA/CD (*Carrier Sense Multiple access with Collision Detection Access Method and Physical Layer Specifications);*

d) linha tronco com cabo grosso e tecnologia de derivações tipo *piercing*;

e) originou do padrão *DIX V1.0* (*Intel*, *Digital* e *Xerox)*, com a velocidade de 10 *Mbps*;

f) a tecnologia é denominada 802.3 e não Ethernet. Os quadros de informações definidos pela norma 802.3 CSMA/CD e DIX V2.0 são diferentes.

2.2.4.2 Melhoramentos da rede *Ethernet*

A rede *Ethernet* teve que receber várias modificações para se tornar mais adaptada ao ambiente industrial, como:

a) foram criados diversos novos padrões, conforme tabela 1:

Tabela 1 – Padrão *IEEE 802*.

Nome do Padrão	Função	Descrição
IEEE 802.1p	Priorização de mensagens	256 níveis de prioridade
EEE 802.12d	Redundância de *links*	Traz maior confiabilidade para a rede.
EEE 802.3x	*Full Duplex*	Comunicação bidirecional simultânea sobre *link* 10/100 Base-T em cabo categoria 5.
EEE 802.3z	*Gigabit Ethernet*	Uso como *backbone* corporativo. Afeta pouco a automação.

Fonte: (SEIXAS JR, 2005)

b) para reduzir o número de colisões e a conseqüente degradação de desempenho da rede *Ethernet*, o que a inviabilizava para algumas aplicações industriais, muitos melhoramentos foram realizados, tais como:

1) aumento da banda de 10 *Mbps* para 100 *Mbps*;

2) uso de *switches:*

3) ligar cada dispositivo a um *port* de um *switch;*

4) armazenar a mensagem antes de retransmití-la a outro nodo;

5) as colisões ficam reduzidas a um único nodo para transmitir e receber uma mensagem;

6) ligação *full duplex* entre o dispositivo e *switch*.

Para que a rede *Ethernet* se torne um padrão confiável também na área industrial, ainda existem algumas deficiências a serem vencidas, conforme listadas a seguir:

a) largura de banda é compartilhada e não dedicada:

1) compartilhamento necessita de arbitragem do barramento sem o conceito de prioridade;

2) compartilhamento resulta em colisões quando 2 (ou mais) dispositivos desejam transmitir seus dados simultaneamente;

3) colisões bloqueiam a rede e impedem outros dispositivos de transmitir seus dados.

b) mais dispositivos em um segmento aumenta a probabilidade de colisão;

c) *broadcast* de mensagens consome grande banda;

d) não existe como diferenciar o tráfego de alta e de baixa prioridade;

e) não existe como assegurar um caminho de baixo atraso para o tráfego de tempo real.

Finalizando este capítulo, pode-se concluir que as tecnologias de redes industriais estão em contínua evolução, uma vez que as empresas buscam definir padrões com perfis de redes mais seguras e de alto desempenho. Neste contexto, verificou-se o avanço dos sistemas abertos, como é o caso da família de protocolos *Fieldbus*. Foi visto, também, o padrão Ethernet, largamente instalado em redes comerciais, sofrendo adaptações e melhorias para ser utilizado também em redes industriais.

3 BARRAMENTO INDUSTRIAL *CAN* (*CONTROLLER AREA NETWORK*)

Para o desenvolvimento deste trabalho, foi escolhido, entre os barramentos de campo (*Fieldbus*), o barramento *CAN* para compor a arquitetura de nossa rede industrial. Foram avaliados vários protocolos e o *CAN* foi selecionado pela sua relevância no mercado e por possuir características importantes de controle de comunicação, as quais serão detalhadas na seqüência deste capítulo.

3.1 ASPECTOS TÉCNICOS

Neste capítulo, serão analisadas as principais propriedades do protocolo de comunicação *CAN* como os tipos e formatos de mensagens, as características elétricas , os mecanismos de prioridade de mensagens e detecção de erros (BOSCH, 1991).

3.2 HISTÓRICO

O protocolo *CAN* foi desenvolvido pela empresa Robert Bosch GmbH, na Alemanha, em 1986 (BOSCH, 1991), para aplicação na indústria automobilística, com o objetivo de simplificar os complexos sistemas de fios em veículos com sistemas de controle constituídos por múltiplos microcontroladores/microcomputadores para gestão do motor, sistema de freio, controle da suspensão, etc. A especificação base deste protocolo anuncia elevada taxa de transmissão, grande imunidade às interferências elétricas e capacidade de detectar erros.

A aplicação da tecnologia *CAN* tem se tornado cada vez mais popular para partilha de dados e controle em tempo real. Ao longo dos anos, o *CAN* evoluiu de aplicações dedicadas à indústria automobilística para uso industrial e produtos envolvendo microcontroladores,

suprimindo a necessidade de sistemas complexos de fios substituindo-os por um simples cabo, tornando a comunicação em rede simples, barata, robusta e sem ruído eletromagnético.

O protocolo de comunicação *CAN* descreve o método como a informação é transferida entre dispositivos e é assunto de padrões internacionais aprovados pela *ISO* - *International Standard Organization*. Esta organização aprovou-o como barramento padrão para redes de alta velocidade de transmissão de dados (de 125 kbps à 1Mbps), por meio da norma *ISO11898*. Já para taxas de transmissão menores (de 10 kbps à 125 kbps) foi estabelecida pela norma *ISO11519-2*. Ambas as normas estão em conformidade com o modelo de referência *OSI* - *Open Systems Interconnection* (define camadas para a estrutura de um sistema de comunicação), para as duas camadas inferiores: enlace (camada 2) e o físico (camada 1).

Com a aprovação da *ISO*, o *CAN* foi adotado pela indústria automobilística bem como por outros tipos de indústrias, devido à sua robustez e flexibilidade. A disponibilidade de circuitos integrados, colocados no mercado por vários fabricantes, incentiva a sua utilização devido ao seu baixo custo.

3.3 ESTRUTURA DA REDE *CAN*

A rede *CAN* é um barramento serial multi-mestre que possui os componentes mostrados na figura 3:

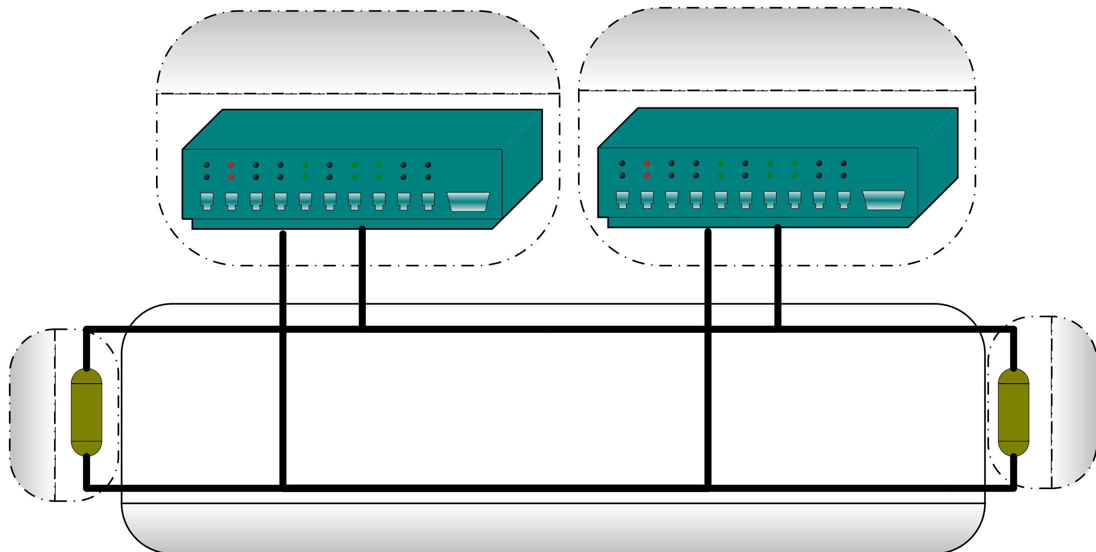


Figura 3 - Componentes da rede *CAN*.
Fonte: O autor

Unidade de Controle Eletrônico: Conhecido como nó da rede *CAN* é responsável pela aquisição e envio de dados da rede. Esta unidade pode ou não ser microprocessador, possui um controlador *CAN* (componente que acondiciona a norma) e um *transceiver* (componente transmissor/receptor para adequar o sinal digital para os níveis elétricos da rede);

Resistores Terminadores: Resistores colocados no final da rede para adequarem-se aos sinais elétricos da rede;

Cabos: É o meio físico para o transporte de dados, sendo geralmente cabo entrelaçado.

3.4 CARACTERÍSTICAS GERAIS DO *CAN*

O *CAN* é um protocolo de comunicação serial síncrono, que permite controle distribuído em tempo real, com elevado nível de segurança. O sincronismo entre os nós conectados à rede é feito em relação ao início de cada mensagem enviada ao barramento (evento que ocorre em intervalos de tempos conhecidos e regulares).

Este padrão define uma rede em barramento com capacidade multi-mestre, em que todos os nós podem pedir acesso ao meio de transmissão simultaneamente, ou seja, todos os nós podem tornar-se mestres em determinado momento e escravos em outro. A função de Mestre é requisitar e controlar atividades dentro da rede e a de Escravo é executar e repassar informações de processos controlados àquele que o controla ou requisita.

Este protocolo comporta também o conceito de *multicast*, isto é, permite que uma mensagem seja transmitida a um conjunto de receptores simultaneamente. Este procedimento agiliza o envio de informações dentro da rede.

Outro ponto forte deste protocolo é o fato de ser fundamentado no conceito *CSMA/CD with NDA* (*Carrier Sense Multiple Access / Collision Detection with Non-Destructive Arbitration*). Isto significa que todos os nós verificam o estado do barramento, analisando se outro nó está ou não enviando mensagens com maior prioridade. Caso isto seja percebido, o módulo, cuja mensagem tiver menor prioridade, cessará sua transmissão e o de maior prioridade continuará enviando sua mensagem deste ponto, sem ter que reiniciá-la.

O *CAN* utiliza o código de linha NRZ (*Non Return to Zero*), nele o nível lógico de cada *bit* (0 ou 1) permanece constante durante a sua duração. Este código apresenta uma boa

eficiência espectral, possibilitando um aproveitamento apropriado da largura de banda de transmissão.

A velocidade de transmissão dos dados é inversamente proporcional ao comprimento do barramento. A maior taxa de transmissão especificada é de 1 *Mbps*, considerando-se um barramento de 40 metros.

Nas redes *CAN*, não existe o endereçamento dos destinatários no sentido convencional. Ao invés disso, são transmitidas mensagens que possuem um identificador. Assim, um emissor envia uma mensagem a todos os nós *CAN* e cada um, por seu lado, decide, com base no identificador recebido, se deve ou não processar a mensagem. O identificador determina também a prioridade intrínseca da mensagem ao competir com outras pelo acesso ao barramento.

O *CAN* possibilita ligar em rede subsistemas inteligentes, tais como sensores e atuadores, cujas informações transmitidas por eles possuem tamanho de no máximo 8 *bytes,* sendo, no entanto, possível transmitir blocos maiores de dados recorrendo à segmentação.

O número de elementos num sistema *CAN* está, teoricamente, limitado pelo número possível de identificadores diferentes. Este número limite é, contudo, significativamente reduzido por limitações físicas do *hardware.* Existem no mercado de circuitos integrados *transceivers* que permitem ligar pelo menos 110 nós e, utilizando módulos de E/S adequados, é possível ter diversos sensores e atuadores por nó.

O *CAN* permite flexibilidade de configuração uma vez que podem ser adicionados novos nós a uma rede *CAN*, sem requerer alterações do *software* ou *hardware* dos demais nós,

se o novo nó for receptor, ou se o novo nó não necessitar da transmissão de dados adicionais. Isto é válido uma vez que o protocolo de transmissão de dados não requer endereços físicos para os componentes individuais, suporta o conceito de eletrônica modular e ao mesmo tempo permite recepção múltipla (*multicast*, *broadcast*) e sincronização de processos distribuídos.

Outra característica importante é o fato do controlador *CAN* de cada nó registrar os erros, avaliando-os estatisticamente, de forma a desencadear ações a eles relacionadas. Estas ações podem corresponder ao desligar, ou não, da estação que provoca os erros, tornando este protocolo eficaz em ambientes ruidosos.

Segue, na seqüência, características do *CAN* que colaboram na decisão de sua escolha pelos desenvolvedores de sistemas de automação industrial:

a) ser um padrão *ISO*;

b) considerável imunidade ao ruído;

c) capacidade multi-mestre;

d) capacidade *multicast*;

e) capacidade eficaz de detectar e sinalizar erros;

f) simplicidade;

g) retransmissão automática de mensagens "em espera" logo que o barramento esteja livre;

h) reduzido tempo de latência;

i) atribuição de prioridade às mensagens;

j) flexibilidade de configuração;

k) distinção entre erros temporários e erros permanentes dos nós;

l) elevadas taxas de transferência (1 *Mbit/s*);

m) redução de cabo a utilizar;

n) baixo preço;

o) *hardware* padrão.

3.5 PRINCÍPIO DE FUNCIONAMENTO DA REDE *CAN*

Para melhor compreensão do princípio de funcionamento da rede *CAN*, será abordada, a seguir, uma aplicação em sistemas industriais, conforme apresentado na figura 4.

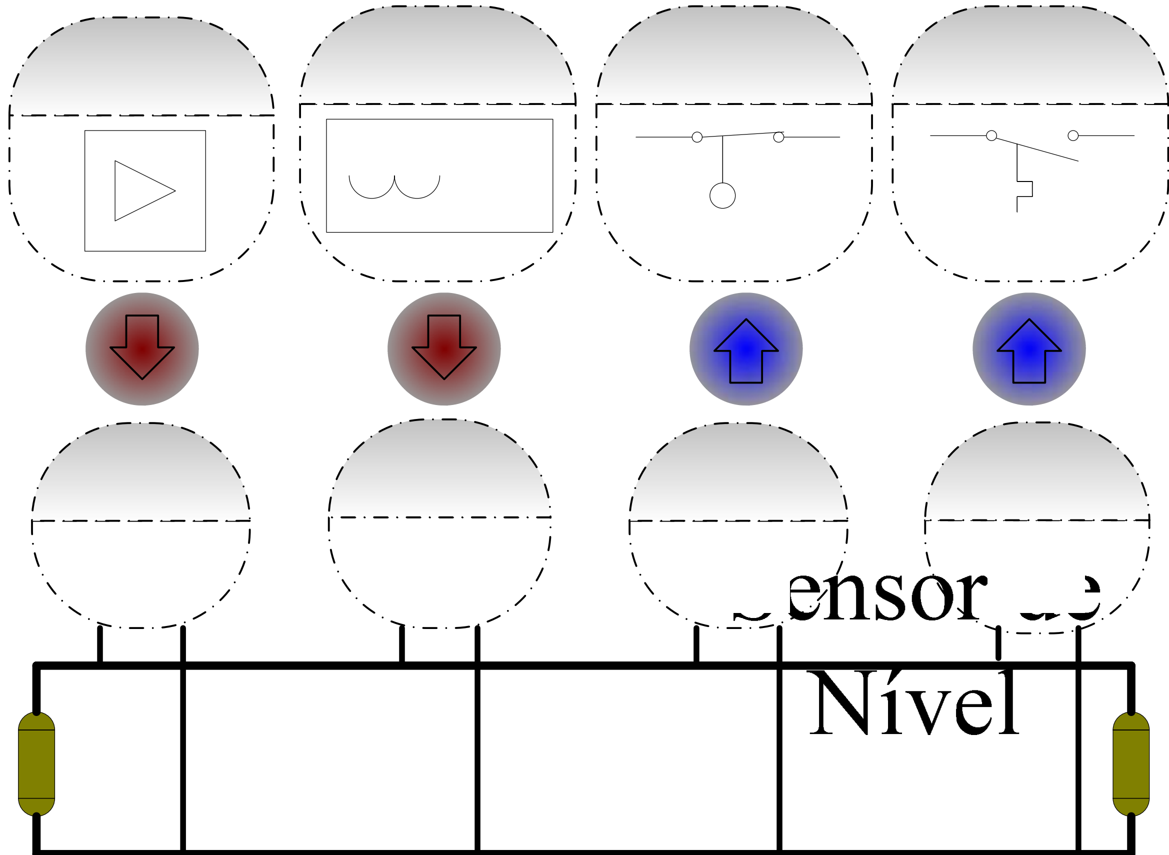


Figura 4 – Aplicação de *CAN* em Sistemas Industriais.
Fonte: O autor

A aplicação aborda um sistema de controle e monitoração do setor de secagem de uma empresa de transformação de grãos. Os grãos são transportados por uma esteira até um compartimento fechado (silo), ali eles são armazenados e expostos a uma temperatura pré-estabelecida, o nível de grãos no silo também é monitorado. Consideram-se assim quatro nós independentes: Um sensor de nível de silo de grãos (Nó A); sensor de temperatura do silo de grãos (Nó B); controlador do Motor A que controla a esteira de abastecimento do silo de grãos (Nó C); controlador do Motor B do ventilador do referido silo (Nó D).

O medidor de nível do silo informa constantemente à rede o nível atual do silo para que esta mensagem fique no barramento disponível a todos os nós conectados ao barramento. Somente o controlador do motor A (Nó C) trata a mensagem para que, em função do nível do silo, possa controlar o motor da esteira que abastece o silo.

Mensagem
Nível

O sensor de temperatura do silo envia ao barramento a temperatura do mesmo, o qual somente o controlador do motor B (Nó D) trata esta informação para controlar o motor B do ventilador.

O *CAN* utiliza um método de filtragem chamado *Frame Acceptance Filtering,* através de um identificador único, em toda rede, que, além de caracterizar o conteúdo da mensagem, estabelece a prioridade de mensagem de tal forma que somente o nó ou os nós, que utilizam a informação, e aceitam a mensagem.

Portanto, toda vez que um nó quiser enviar uma mensagem, deve enviá-la ao controlador *CAN* do próprio nó, onde a mensagem é construída e enviada ao barramento, assim que for autorizado o acesso a ele. E no instante em que este nó estiver no controle, todos os outros nós serão receptores da mensagem.

3.5.1 Processo de Arbitragem não Destrutiva

Nos sistemas de processamento em tempo real, a urgência da troca de mensagens pela rede pode ser significativamente diferente: uma grandeza (exemplo: nível do silo) deve ser transmitida com maior freqüência e menores atrasos do que outras que variem menos (ex. temperatura do silo). Portanto, para que os dados sejam processados em tempo real, estes devem ser transferidos por um meio físico que permita elevada taxa de transmissão e chamadas rápidas à alocação do barramento quando várias estações tentam transmitir simultaneamente.

No *CAN*, a prioridade com que uma mensagem é transmitida, dentro da rede, é especificada por um identificador dentro do quadro do protocolo. A prioridade das mensagens

é definida durante a fase de projeto do sistema, sob a forma de valores binários, em que o identificador de menor valor numérico tem maior prioridade. O identificador também é utilizado para arbitrar os pedidos de acesso ao barramento por parte dos nós concorrentes.

O *CAN* é uma rede *CSMA/CD with NDA* (*Carrier Sense Multiple Access / Collision Detection with Non-Destructive Arbitration*), ou seja: os nós atrasam a transmissão se o barramento estiver ocupado; quando a condição de barramento livre for detectada, qualquer nó pode iniciar a transmissão; os conflitos de acesso ao barramento são solucionados por comparação orientada *bit* a *bit* dos identificadores das mensagens, e funciona da seguinte forma:

a) enquanto transmite o identificador da mensagem de comunicação, cada nó monitora o barramento série;

b) se o *bit* transmitido for "recessivo", nível lógico '1', e for detectado um *bit* "dominante", nível lógico '0', o nó desiste da transmissão e inicia a recepção dos dados que chegam;

c) o nó, que transmite a mensagem com o menor identificador, ganha acesso ao barramento e continua a transmissão sem sofrer atraso.

Portanto, o acesso ao barramento obedece a uma prioridade, permitindo que a informação mais urgente seja atendida em primeiro lugar. Este processo é denominado arbitragem não destrutiva, porque a retransmissão automática de uma mensagem de comunicação é tentada após uma perda no processo de arbitragem, ou seja, a mensagem, que não foi enviada, não será destruída, mas armazenada até a próxima chance de envio.

3.6 PROTOCOLO DE COMUNICAÇÃO *CAN* E O PADRÃO *ISO/OSI*

A *ISO* estabeleceu um modelo de referência conhecido como Modelo de Referência *ISO/OSI* para comunicação de dados em sistemas de comunicação de rede (SOARES, 1995). Este modelo é representado por camadas, conforme apresentado na figura 5.

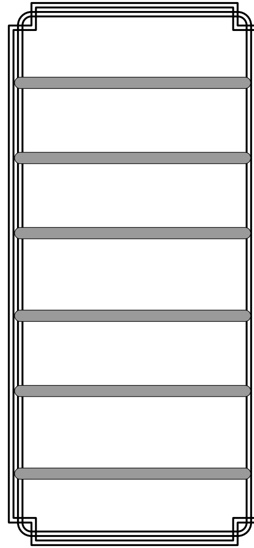


Figura 5 - Modelo das camadas *ISO/OSI*.
Fonte: O autor

Cada camada fornece serviço à camada superior de tal forma que a camada superior não precise preocupar-se em conhecer os detalhes da camada anterior.

A seguir, são abordados mais detalhes das duas últimas camadas, Enlace (nível 2) e Física (nível 1), onde o protocolo de comunicação *CAN* trabalha, conforme ilustrado na figura 6.

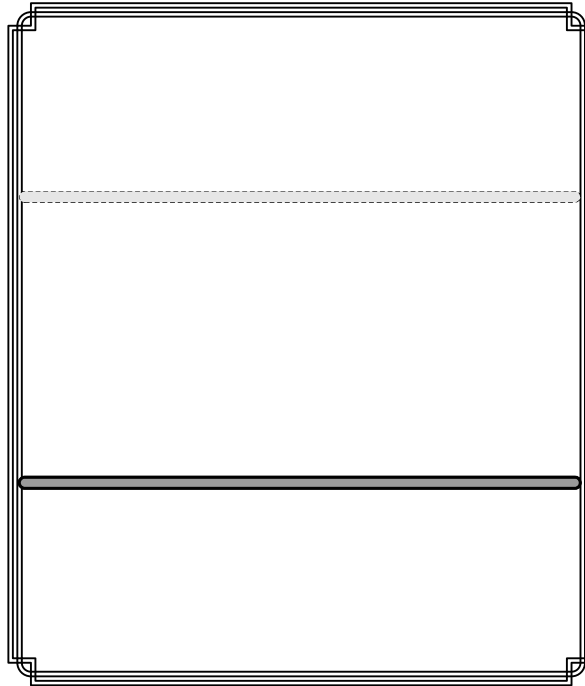


Figura 6 - Camadas da Rede *CAN*.
Fonte: O autor

A camada Física é responsável pela recepção e transmissão da mensagem pela rede e faz a verificação da sincronização de *bits.* Na transmissão, ela recebe informações da camada de Enlace: a mensagem encapsulada a ser transmitida, o identificador da mensagem e o número de *bytes* da mensagem. Na recepção, a camada física envia as informações recebidas pela rede para a camada de Enlace, e ela ao receber estes dados, realiza o desencapsulamento da mensagem, a detecção e sinalização de erros de mensagens e realiza a filtragem de aceitação das mensagens e, então, disponibiliza à camada superior a mensagem recebida e os erros encontrados (SOARES, 1995).

3.7 INFORMAÇÕES DA CAMADA DE ENLACE

Esta camada é o núcleo do protocolo *CAN* e possui 4 tipos de mensagens para realizar a comunicação entre diferentes nós da rede (para obter mais detalhes, consultar o apêndice B), que são:

a) **mensagem de dados**: São as mensagens com as informações do processo monitorado;

b) **mensagem remota**: É uma solicitação de uma informação que um nó faz à rede. O nó que tive a informação solicitada irá devolver a mensagem com os dados solicitados;

c) **mensagem de erro**: São as mensagens de erro que ocorreram no barramento;

d) **mensagem de sobrecarga**: São as mensagens que indicam que determinado nó está sobrecarregado de informações e solicita um atraso maior entre as transmissões, pois não conseguiu processá-las.

3.8 INFORMAÇÕES DA CAMADA FÍSICA

A Camada Física é responsável pela transferência de *bits* entre os nós de uma rede *CAN*. Esta camada define o modo como os sinais são transmitidos, parâmetros como temporização, codificação e sincronização das seqüências de *bits* a serem transmitidos (BOSCH, 1991). A estrutura típica de um nó da rede *CAN* é mostrada na figura 7.

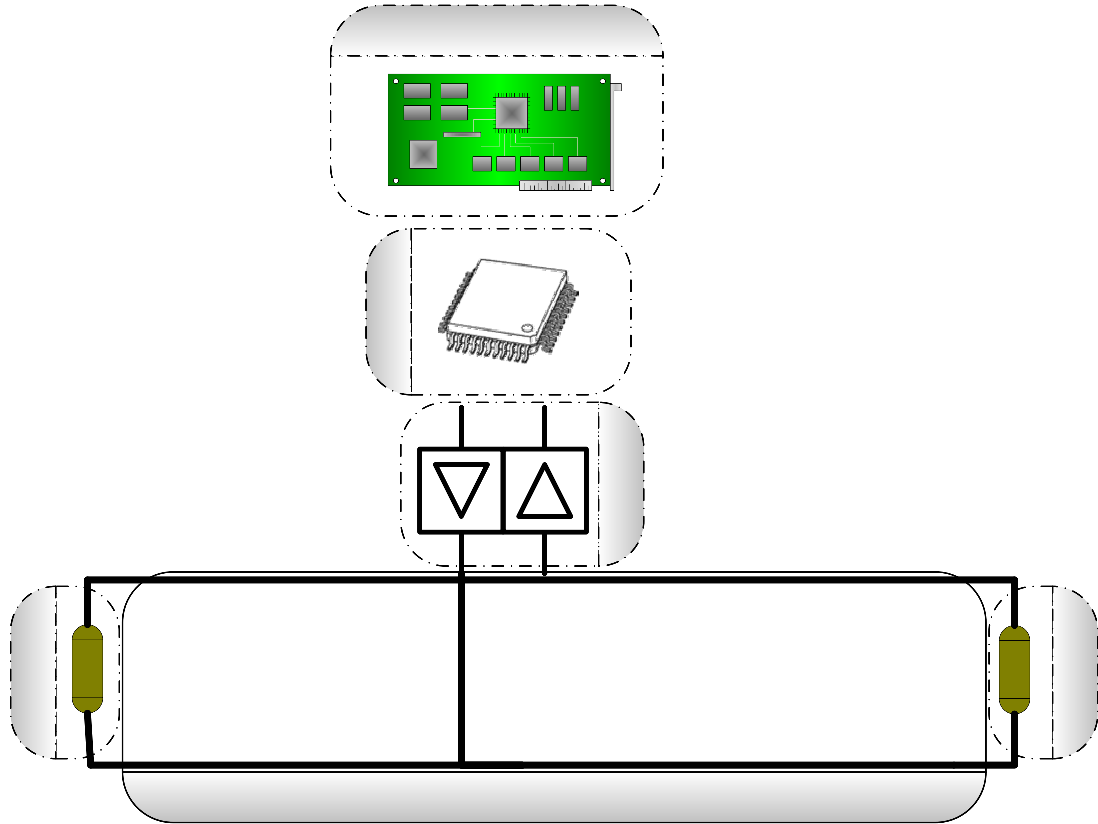


Figura 7 – Estrutura típica de um nó da rede *CAN*.
Fonte: O autor

A estrutura de um nó da rede *CAN* é composta pelos seguintes elementos:

a) **cartão processador:** é a placa de circuito impresso que contém o processador que utiliza ou gera informações, passando-as para o controlador *CAN*. Existem processadores que possuem controladores *CAN* incorporados;

b) **controlador *CAN*:** o Controlador *CAN* tem como função implementar as características da rede atendendo a 1ª. e a 2ª. Camada do padrão *ISO/OSI*. As principais funções são:

1) filtragem das mensagens;

2) encapsulamento da mensagem que será transmitida;

3) cálculo *CRC*;

4) inserir e apagar os "*Bit Stuff*";

5) sincronização do "*bit time*";

6) detecção e sinalização de erros;

7) gerar e detectar o *bit* de reconhecimento da mensagem;

8) disponibilizar a mensagem para o nó interessado na mesma.

b) **transceiver:** adapta o sinal digital para os padrões da rede, através de um transmissor e receptor. Além disso, o *transceiver* protege o controlador de eventuais sobre-tensões ou curto de rede. Ele apresenta algumas características importantes como:

1) taxa de transmissão acima de 1 Mbit/s;

2) proteção contra curto-circuito tanto para terra como para fonte;

3) proteção contra sobrecarga;

4) compatibilidade total com a Norma *ISO 11898-2*;

5) baixo consumo de corrente em modo de repouso.

d) **meio físico:** na especificação inicial feita por Robert Bosch, nenhum meio foi definido, permitindo diferentes opções para o meio de transmissão e níveis dos sinais (BOSCH, 1991). Essas propriedades foram posteriormente contempladas pelo padrão *ISO,* onde estão definidas características dos sinais. Com o *CAN,* é possível utilizar diversos meios físicos, tais como: par de fios entrelaçados, fibra ótica, rádio freqüência, etc. Atualmente, a maioria das aplicações utiliza um barramento diferencial a dois fios.

3.8.1 Representação do *Bit*

As mensagens da rede *CAN* são transmitidas por meio de uma seqüência de *bits*. O código de linha utilizado chama-se NRZ (*Non-Return-to-Zero*), em que o nível lógico do *bit* permanece constante sem variação, tanto para o nível lógico 0 quanto para 1, durante todo o intervalo de tempo de um *bit*.

3.8.2 Sincronização do *Bit*

O protocolo de comunicação *CAN* utiliza comunicação síncrona, tornando a transmissão mais eficiente, porém, é necessário um método de sincronização bastante sofisticado. O método de sincronização do *bit* é feito, além de um *start bit* no início da mensagem, um artifício de resincronização durante as alterações dos pulsos de borda dos quadros de mensagens (BOSCH, 1991).

O *bit rate* nominal é o número de *bits* por segundo, transmitido na ausência de resincronização por um transmissor ideal. O *bit time* nominal, que é composto de vários segmentos não sobrepostos, é calculado da seguinte forma:

bit time nominal = 1 / *bit rate nominal*

Os segmentos que compõem um *bit time* nominal são representados na figura 8 e podem ser assim descritos:

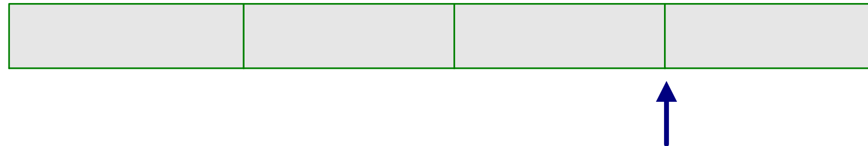


Figura 8 – Segmentos do *bit time nominal*.
Fonte: O autor

a) **segmento de sincronização:** Neste segmento, espera-se que a sincronização da rede ocorra através da borda de descida ou subida do *bit*;

b) **segmento de propagação:** É responsável pela compensação do atraso que a rede oferece durante o envio e recepção de sinais;

c) **segmento de fase 1 e 2:** São responsáveis pelo deslocamento do ponto de amostragem do sinal durante o processo de resincronização;

d) **amostragem do sinal:** É o ponto em que o sinal é amostrado.

O dimensionamento dos segmentos é feito em função das necessidades do sistema de comunicação. Os fatores que são levados em consideração para o cálculo dos segmentos da rede são: comprimento da rede, taxa de transmissão e o tempo de atraso.

O processo de sincronização ocorre no momento da alteração do estado lógico, passando de nível 0 para 1, ou vice-versa, durante a mensagem. Assim, existe a possibilidade de se ter vários *bits* no mesmo nível lógico, acarretando um erro de sincronismo. Para ajudar

no processo de sincronização há uma solução chamada "*Bit Stuff*" onde quando há 5 *bits* de mesmo estado lógico, insere-se outro *bit*, de nível lógico contrário aos anteriores, fazendo com que no máximo por um período de 5 *bits* não se terá alteração no nível lógico. Os nós receptores devem retirar os *bits* que foram incluídos na transmissão.

3.9 CARACTERÍSTICAS DO MEIO FÍSICO

A *ISO 11898-2* estabelece algumas características da rede física (BOSCH, 1991), sendo que as principais são:

a) taxa de transmissão acima de 1 *Mbit/s*;

b) comprimento máximo de 40 metros com taxa de transmissão de 1 *Mbit/s*;

c) impedância de linha de 120 Ω;

d) atraso de propagação de sinal de 5 ns/m.

No barramento *CAN,* a taxa de transmissão depende do comprimento de barramento e vice-versa. Esta limitação surge devido aos processos de arbitragem que necessitam que a onda se propague até o nó remoto e volte antes de ser amostrada. Para tamanhos de barramento maiores as recomendações, segundo a CiA (*CAN in Automation*), são:

a) 500 *kbit/s* para distâncias até 100 metros;

b) 250 *kbit/s* para distâncias até 250 metros;

c) 125 *kbit/s* para distâncias até 500 metros;

d) 50 *kbit/s* para distâncias até 1 km.

Se a distância do barramento for superior a 1 km, pode ser necessária a utilização de dispositivos repetidores (*repeater*) ou ponte (*bridge*). A faixa de variação de tensão nos nós é de 1,5 até 3,5 *Volts*.

O nível lógico 1 (recessivo) é conseguido através da tensão de aproximadamente 2,5 *Volts* entre os terminais e o nível lógico 0 (dominante) é conseguido quando o terminal *CAN*_H está com 3,5 *Volts* e o terminal *CAN*_L está com 1,5 *Volts*, conforme figura 9:

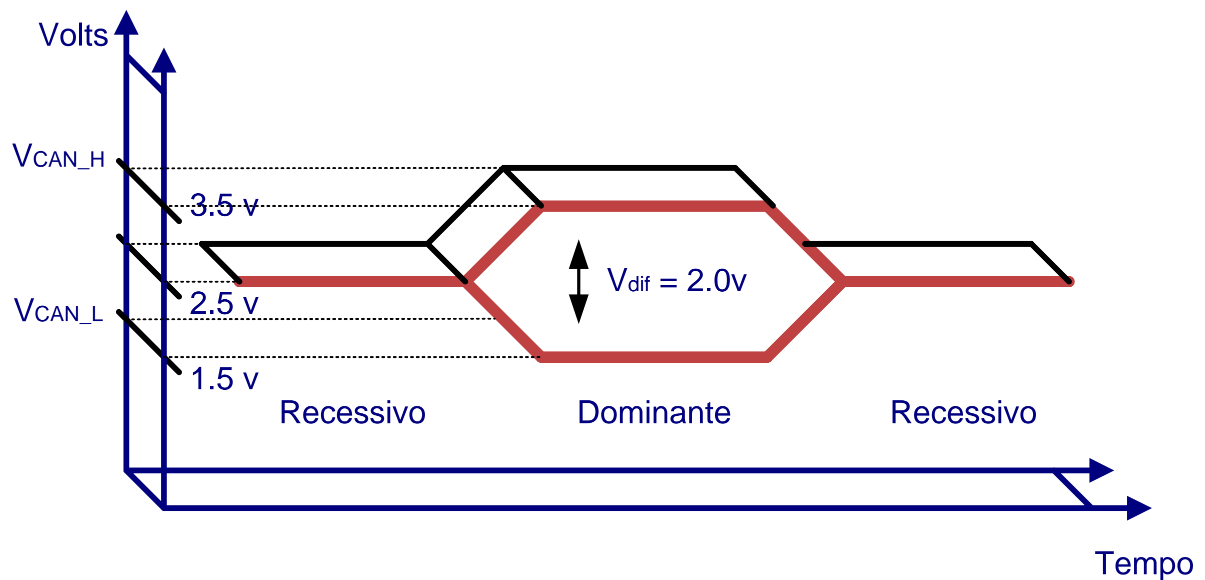


Figura 9 – Níveis de Tensão da Rede *CAN*.
Fonte: O autor

3.10 TOLERÂNCIA À FALHAS NO BARRAMENTO

Para os barramentos com 2 e 4 fios, a rede continuará funcionando num modo de segurança, quando houver algo de errado com os fios de dados *CAN_H e CAN_L*. Algumas

das condições de falha nas linhas de comunicação que permitem a continuidade das atividades da rede são:

a) curto do *CAN_H* (ou *CAN_L*) para *GND* (ou *VCC*);

b) curto entre os fios de dados *CAN_H* e *CAN_L*;

c) ruptura do *CAN_H* (ou *CAN_L*).

3.11 MÉTODOS DE DETECÇÃO E SINALIZAÇÃO DE ERRO

Numa rede *CAN,* há a garantia de que um quadro é simultaneamente aceito por todos os nós ou apenas por alguns. Assim, a consistência de dados é uma propriedade do sistema, alcançada através de conceitos de *multicast* e *error handling* (suporte de erros). Para implementar o segundo conceito, estão definidos diversos tipos de erros e respectivos mecanismos de detecção.

Ao contrário de outros sistemas de barramento, o protocolo *CAN* não utiliza mensagens de confirmação, mas detecta e assinala erros que ocorrem.

3.12 DETECÇÃO DE ERROS

Uma das grandes vantagens do *CAN* é a sua capacidade de adaptar-se às condições de falha, temporárias e/ou permanentes. Para a detecção de erros, o protocolo *CAN* implementa três mecanismos ao nível da mensagem e dois ao nível do *bit*. Mais detalhes consultar o Apêndice B.

3.13 A EVOLUÇÃO DO CAN

As figuras 10 e 11 ilustram a evolução da rede *CAN*. Os componentes controladores como o *SLIO(Serial Linked Input Output*) em conjunto com os circuitos de acionamento e os circuitos de potência, podem ser integrados em dispositivos modulares compactos, facilitando assim na instalação e manutenção. Nesta nova arquitetura, elimina-se o cabeamento local, diminui a quantidade de fios que podem falhar, possibilitando com isso, uma melhoria da eficácia do sistema total e também aumenta a robustez em termos de compatibilidade eletromagnética.

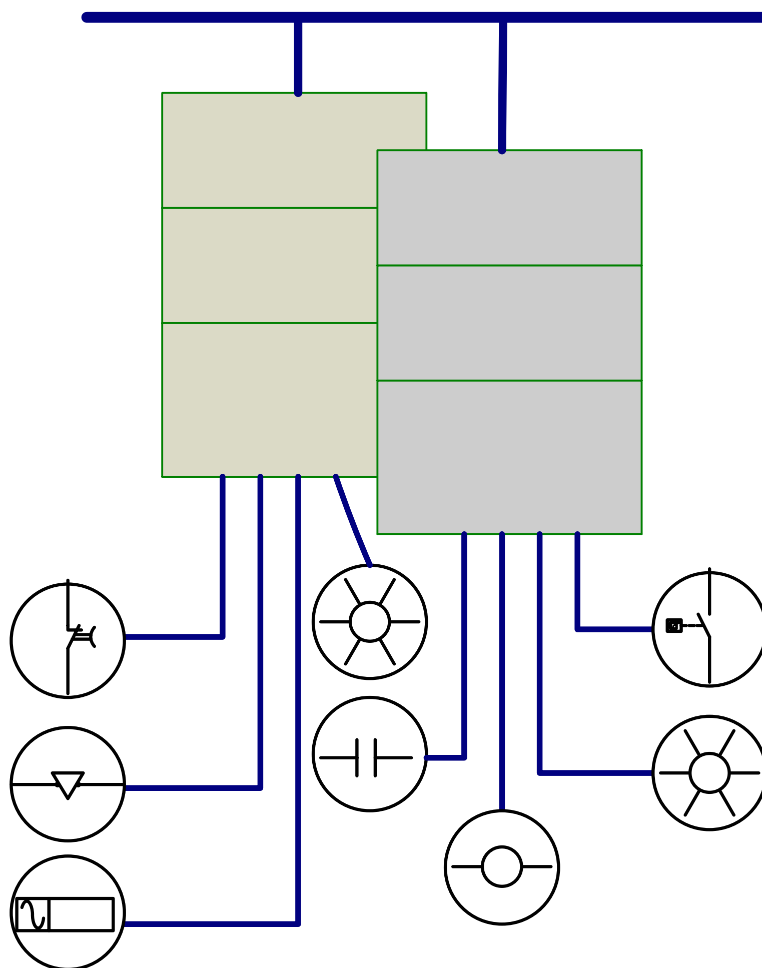


Figura 10 - Sistemas Atuais.
Fonte: O autor

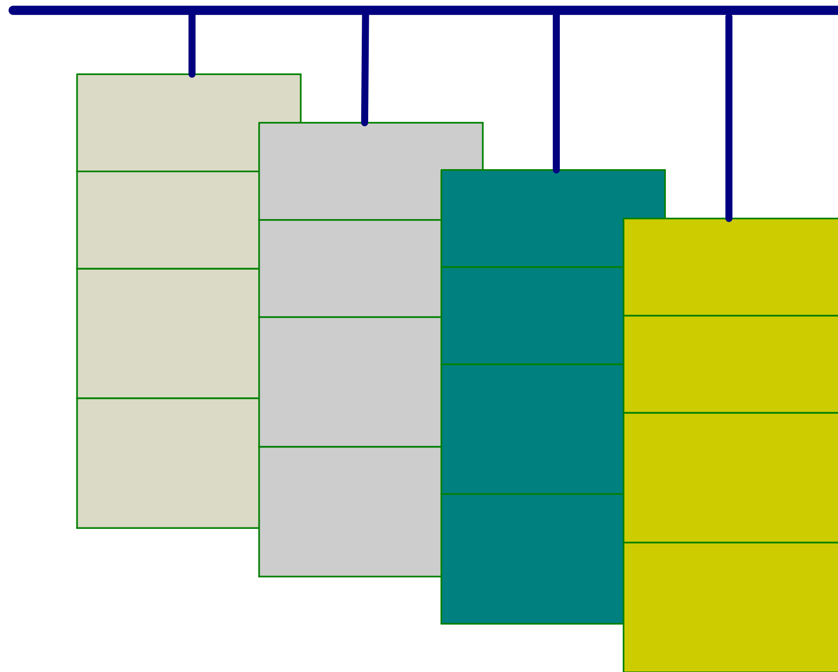


Figura 11 - Sistemas Futuros.
Fonte: O autor

Pode-se concluir este capítulo, apresentando os vários fatores que fazem o protocolo *CAN* ser aceito, por parte da indústria de automação. Isto se deve principalmente:

a) ao baixo custo;

b) à disponibilidade no mercado de componentes controladores, com o protocolo *CAN*;

c) à capacidade de Detecção de Erros;

d) ao meio de transmissão simples;

e) pela possibilidade de isolamento do nó com falhas;

f) ao elevado grau de capacidades de tempo real e controle distribuído;

g) à existência de *padrão*;

h) à fácil utilização;

i) à capacidade de funcionar em ambientes com condições elétricas adversas.

4 SISTEMAS ABERTOS

Dentro do contexto de sistemas abertos há duas definições importantes (MDIC, 2005):

a) *Software* Livre: Um *software* livre respeita as quatro liberdades essenciais que são:

1. liberdade para executar o programa para qualquer fim, em qualquer ponto e a qualquer tempo;
2. liberdade de estudar o funcionamento do programa e adaptá-lo às necessidades de quem estuda;
3. liberdade de redistribuição de cópias;
4. liberdade para melhorar o programa e publicar as melhorias.

b) Código aberto (*Open Source*): O código aberto tem os seguintes conjuntos de princípios:

1. distribuição livre, sem pagamento de *royalties* ou semelhante;
2. código fonte deve sempre estar aberto;
3. permitir modificações e trabalhos derivados;
4. garantir integridade autoral do código fonte;
5. não discriminar pessoas ou grupos;
6. não discriminar áreas de conhecimento, setores e atividades;

7. direitos de licença redistribuídos sem necessidade de licenças adicionais pelas partes;

8. a licença não deve ser ligada a um produto específico;

9. a licença não pode restringir outros *softwares* que são divulgados conjuntamente.

As idéias de *Software* Livre estão mais ligadas às questões de perpetuação das liberdades, enquanto os Códigos Abertos estão mais relacionados às questões práticas de produção e negócio (MDIC, 2005).

Uma grande vantagem de uma arquitetura aberta é que as pessoas podem ler, depurar, distribuir e trocar códigos, até que este *software* se torne estável e de boa qualidade. Este *software* será mais flexível e de melhor qualidade do que um *software* desenvolvido por canais convencionais, porque mais pessoas têm testado e em mais diferentes condições do que desenvolvidos em *software* proprietário.

No meio acadêmico e técnico, existe há mais de 20 anos a conscientização do uso de fontes abertas, porém, somente atualmente tem sido feito um grande esforço para convencer às pessoas voltadas ao mundo comercial sobre a importância do mesmo.

A opção por um sistema operacional aberto se justificaria pelas seguintes e principais razões:

a) ser considerado, por especialistas, de qualidade igual ou superior aos *softwares* pagos;

b) poder ser instalado em quantas máquinas forem necessárias sem ter que pagar pela licença;

c) permitir acesso ao código fonte, o que possibilita realizar programas de melhor qualidade e realizar alterações que visem adequar melhor o programa às necessidades do usuário;

d) oferecer variedade de programas em quase todos os campos de aplicação de *softwares*.

O que pode ser entendido por Sistemas Abertos em ambiente de automação industrial não é somente ter a característica de uma especificação pública e, conseqüentemente, gratuita, mas uma arquitetura que abrange as seguintes características (AZEVEDO, 2001):

a) **Interoperabilidade:** Possibilidade de conectar diferentes plataformas de *hardware* através de uma rede padrão;

b) **Modularidade:** Implementação de módulos de *software* com interface bem definida que permitem adição e remoção sem interferir em outros módulos;

c) **Escalabilidade:** Habilidade de rodar o mesmo *software* em centros de controle de diferentes tamanhos e funções;

d) **Portabilidade:** Implementação da mesma funcionalidade em diferentes plataformas de *hardware*.

Em resumo, um sistema aberto é aquele que deve ser interoperável e intercambiável, ou seja, os seus componentes podem ser adquiridos de um ou outro fornecedor, de acordo

com um critério definido, seja qualidade, preço, atualização ou outro motivo. O componente pode ser trocado por um da concorrência sem que haja necessidade de qualquer alteração no sistema, ou seja, partes que exercem funcionalidades específicas são compatíveis no que se diz respeito à forma como enviam e recebem informações da rede. Para tal, o protocolo necessita ser devidamente documentado e aberto, assim como devem existir mecanismos para garantir a compatibilidade de um determinado dispositivo antes que este seja disponibilizado no mercado.

Foi escolhido para este trabalho, como sistema operacional aberto, o *LINUX,* porque ele provê uma enorme infra-estrutura para o usuário como: ferramentas, fornecimento de manutenção, atualização, suporte e etc.

4.1 LINUX e GNU IS NOT UNIX (GNU – GNU não é UNIX)

O *LINUX* só tem esta grande força e penetração graças ao enorme trabalho desenvolvido por Richard Stallman (GNU, 2005), e também inúmeras outras pessoas. Ele foi o fundador do sistema operacional *GNU* e também da *GNU.org* que é uma organização criada para difundir o uso do *software* livre (GNU, 2005). Tudo iniciou quando Stallman começou o movimento pelo *software* aberto e livre, em 1984, na época em que começaram a aparecer os primeiros *softwares* comerciais. Stallman era um pesquisador no MIT (*Massachusetts Institute of Technology,* Instituto de Tecnologia de Massachusetts*)* e trabalhava na área de inteligência artificial. Certa vez, ele precisou alterar um código de um programa de controle de uma impressora que não funcionava conforme ele desejava. Logo, ficou indignado quando o fabricante da impressora negou o código fonte do programa para que pudesse fazer as modificações necessárias. O fabricante alegou que o código fonte era segredo comercial e não

podia ser cedido a terceiros. Então, Stallman iniciou um importante trabalho de criar versões abertas para todas as categorias de *software* existentes comercializadas sem acesso ao código fonte. Ele criou um compilador C, um editor de textos poderoso e popular denominado *Emacs (GNU, 2005).*

Mas, o fato mais importante que ele fez não foi a criação de vários programas poderosos e bem escritos, porque escrever programas e disponibilizar o código fonte para quem quer que seja poderia resultar justamente no contrário do que pretendia, ou seja, a liberdade no uso do *software*. Sendo o código aberto, nada impediria de alguém fazer algumas modificações, declarar ser proprietário e depois restringir o acesso. Portanto, para impedir que isso ocorresse, Stallman escreveu um documento que estabelece a forma, sob a qual programas de código fontes abertas podem ser distribuídos. Este documento especifica que o programa pode ser usado e modificado por quem quer que seja, desde que as modificações efetuadas sejam também disponibilizadas em código fonte. Este documento chama-se *GNU GPL* (*GNU General Public License*, Licença Pública Geral). Esta licença é mantida pela *FSF* (*Free Software Foundation*, Fundação para Software Livre) que também foi fundada por Stallman.

Pelo fato de qualquer sistema operacional conter um *kernel*, ou seja, o núcleo do sistema operacional, e pelo menos um conjunto mínimo de utilitários que fazem parte do projeto *GNU*, há os que defendem que o sistema deveria ser chamado sistema *GNU/LINUX*. A FSF também promove o uso e o desenvolvimento do sistema operacional *GNU* do conhecido *GNU/LINUX* (FSF, 2005).

Embora exista um grande número de implementações de *LINUX*, encontram-se muitas similaridades em diferentes distribuições, porque cada máquina *LINUX* é como uma caixa de

blocos de construção que se podem colocar juntos segundo as suas necessidades. A instalação do sistema é apenas o início de um longo período de relacionamento, pois o *LINUX* irá estimular a imaginação e criatividade do usuário que muito poderá realizar com a força do sistema.

Todas as ferramentas *GNU* são fontes abertas. Assim, elas podem ser instaladas em qualquer sistema e muitas distribuições oferecem pacotes pré-compilados de muitas ferramentas comuns o que facilita a instalação no sistema. Por outro lado, o *LINUX* vem com um conjunto completo de ferramentas de desenvolvimento que permite instalação de novos *softwares* provida de códigos fontes. Esta situação também permite instalar um software não pré-definido convenientemente para o seu sistema.

Estes são listados alguns exemplos de *softwares G*NU (GNU, 2005):

a) compiladores *GNU*: *C, C++*;

b) depurador *GNU*: *GDB*;

c) programa manipulador de imagem *GNU*: *Gimp;*

d) ambiente de área de trabalho *GNU*: *Gnome*;

e) editor de textoGNU: *Emacs*;

f) sistema relacional de banco de dados GNU: *SQL*.

4.2 SISTEMA OPERACIONAL LINUX

O *LINUX* é um sistema operacional derivado do *Unix,* feito para rodar em computadores pessoais. Ele possui a robustez, segurança e flexibilidade do *Unix*. O *LINUX* foi desenvolvido com uma arquitetura aberta (*open source*), o que dá mais liberdade para o desenvolvimento de *software*.

O *LINUX* suporta multitarefa real, memória virtual, bibliotecas dinâmicas, implementação da rede *TCP/IP* (ver detalhes no apêndice A), nomes de arquivos com até 255 caracteres e proteção entre processos (*crash protection*), e muitas outras funcionalidades (WELSH, 1997).

Um grande atrativo que o *LINUX* oferece é o fato de poder trabalhar tanto como servidor de aplicações quanto como estação de trabalho, sem que haja necessidade de grandes modificações no seu sistema.

Linus Torvalds desenvolveu originalmente o *LINUX* como um passatempo, visto que ele queria um sistema operacional que fosse semelhante a um *Unix*, com todas as suas funcionalidades e, ainda, que pudesse utilizá-lo num *PC* (WELSH, 1997).

A partir dessa idéia, Linus começou a trabalhar nesse que seria o futuro *kernel* do sistema operacional, que é chamado hoje de *LINUX*. Isso tudo aconteceu em meados de 1991, quando Linus cursava a faculdade de Computação na Finlândia, e, desde então, o seu uso não pára de crescer, tendo atualmente cerca de 10 milhões de usuários (WELSH, 1997).

Assim, surgiu o que seria o primeiro *kernel* utilizável do *LINUX. Kernel* é o código de programa que controla os recursos da máquina para as aplicações do usuário. Quando se fala de *LINUX*, normalmente se refere somente ao *kernel* do sistema.

Há distinção entre o *kernel* do *LINUX* e um sistema *LINUX*. O sistema *LINUX* inclui uma variedade de componentes, alguns escritos do zero, outros emprestados de projetos de desenvolvimentos e os demais criados em colaboração com outras equipes. Pode-se definir que o sistema *LINUX* é composto de três corpos: *kernel*, bibliotecas e utilitários do sistema (NEMETH, 2004).

Por *LINUX* ser um *software* de livre distribuição (PANIAGO, 2005), muitas pessoas e até mesmo empresas se empenham em organizar o *kernel* e mais uma série de aplicativos e manuais para que o sistema fique cada vez mais amigável.

A esse conjunto de aplicativos mais o *kernel* dá-se o nome de distribuição *LINUX*. Algumas distribuições *LINUX* são de tamanho variado (CA, 2004), dependendo da quantidade de aplicativos e a finalidade a que se propõem. Existem desde distribuições que cabem num disquete de 1,44 Mb até distribuições que ocupam vários CDs.

Entre os maiores distribuidores podem-se citar: *Conectiva*, *SuSE*, *Debian* e *Red Hat*. O que diferencia uma distribuição de outra é a maneira como são organizados e pré-configurados os aplicativos que cada uma contém. Algumas distribuições incluem ferramentas de configuração que facilitam a vida do administrador do sistema.

A ampla utilização do *LINUX* em ambiente acadêmico garante o desenvolvimento constante e sólido do sistema - com milhões de acadêmicos trabalhando em seu desenvolvimento em todo o mundo, o *LINUX* se destaca como um sistema seguro e bem organizado.

Seu ótimo desempenho de suas funcionalidades em máquinas servidoras e em sistemas dedicados e de grande porte faz do *LINUX* uma alternativa barata, eficaz e confiável - além de funcional em diversas plataformas.

4.3 REQUISITOS DO SISTEMA PARA A INSTALAÇÃO DO LINUX

Independente da versão do *LINUX*, em linhas gerais, o método usado para instalá-lo é o seguinte (WELSH,1997):

1) verificar se há espaço no *drive* para o *LINUX*. Caso seja um disco rígido "vazio", este item deve ser desconsiderado. Senão, deve-se redimensionar as partições atuais para criar espaço. A forma de reparticionar é obtida através da documentação do sistema operacional que está rodando;

2) inicializar o meio de instalação como o "*CD* de Iniciação" que irá conduzir na instalação do *LINUX;*

3) depois de reparticionar para alocar espaço para o *LINUX,* deve-se criar partições nestes espaços vazios;

4) criar os sistemas de arquivos para armazenar arquivos nas partições recentemente criadas;

5) instalar o *software* nos novos sistemas de arquivos.

Embora tenha sido descrita a forma de instalação, muitas distribuições possuem processos de instalações automatizadas.

Para instalação do *LINUX*, o fator mais importante, antes de instalá-lo, é o *hardware*. Desde que toda distribuição *LINUX* contenha o pacote básico e pode ser construído para encontrar quase todos os requerimentos (porque todos utilizam o *kernel Unix*), será necessário verificar se a distribuição rodará em seu *hardware*. Por exemplo, o *LinuxPPC* foi feito para rodar no *Macintosh* e outros *PowerPCs* e não roda num *PC* baseado em x86. Este *LinuxPPC* roda no novo Macs, mas não pode ser usado nas antigas tecnologias de barramento. Outro caso complicado é o *hardware* da *Sun*, uma vez que tanto o antigo, *SPARC CPU,* ou o mais recente, *Ultra Sparc*, requerem diferentes versões do *LINUX*.

Portanto, antes de instalar um *software*, é necessário conhecer as exigências de *hardware* para instalação do *LINUX*, conforme citadas a seguir (WELSH, 1997):

a) **placa-mãe e exigências da *CPU*:** Atualmente o *LINUX* suporta sistemas com uma *CPU Intel* 80386, 80486, *Pentium* ou *Pentium Pro*, bem como todas as variações neste tipo de *CPU* como: 386SX, 486SX, 486DX e 486DX2. Os processadores *AMD* e *Cyrix* que são "clones" sem serem da *Intel*, também funcionam com o *LINUX*. Este foi transportado para diversas arquiteturas diferentes da Intel como: *DEC Alpha*, *MIPS*, *Power PC*, *SPARC* e *Motorola 68k*. A placa-mãe do sistema terá que usar a arquitetura do *bus ISA*, EISA ou *PCI* que definem como o sistema irá comunicar-se com os periféricos e com os outros componentes no *bus* principal. É sugerido o *VLB* (*VESA Local Bus*) para os sistemas que utilizam uma arquitetura de *bus* local (para o vídeo e acesso do disco mais rápido);

b) **exigências da memória:** É necessário pelo menos 2MB de RAM, porém é recomendado que tenha 4 MB que rodará bem com todos os acessórios como o X *Window system*, o *Emacs*, etc. Oito *Megabytes* é mais do que suficiente para uso

pessoal, mas 16 MB ou mais poderá ser necessário, caso haja um fluxo intenso de usuários no sistema;

c) **exigências do controlador do disco rígido:** Há suporte do *kernel* para os controladores com padrão *XT* (8 *bits*) e *AT* (16 *bits*). O *LINUX* deverá suportar todos os controladores padrões: *MFM*, RLL, *IDE* e também SCSI;

d) **exigências do espaço no disco rígido:** A quantidade de espaço necessário no disco rígido dependerá muito da necessidade do sistema e de quanto *software* tem instalado. Pode-se executar um sistema completo em 10 ou 20 MB de espaço em disco. Se for preciso pacotes maiores como o *X Window System*, ou permissão para que mais usuários usem a máquina, será necessário mais espaço na ordem de 100 MB. Cada distribuição do *LINUX* geralmente vem com alguma instrução para ajudar a determinar a quantidade exata exigida;

e) **exigências do adaptador de vídeo e monitor:** O *LINUX* suporta todas as placas de vídeo padrões *Hercules*, *CGA, EGA, IBM* monocromática, Super *VGA* e monitores para interface *default* baseado em textos;

f) ***hardwares* diversos:** O *LINUX* suporta uma variedade grande de dispositivos opcionais e de fabricantes diversos, como: mouse, cd-rom, *drives* de fitas, impressoras, modens, placas de redes, e outros dispositivos poderão ser acrescentados a esta lista à medida que algum usuário sinta necessidade de usá-lo através do *LINUX* (RUBINI, 1999), bastando para isto a implementação de um programa de acesso ao dispositivo desejado, que poderá ser feito pelo próprio usuário, caso ele tenha habilidade para esta tarefa ou sugeri-la à alguma comunidades *LINUX* espalhadas no mundo.

4.4 LINUX EM SISTEMA EMBARCADO

Um dispositivo é definido como sistema embarcado (HORTON, 2004), quando este possui uma combinação de *hardware* e *software* para desempenhar uma função específica, interagindo continuamente com o ambiente ao seu redor através de sensores e atuadores. Normalmente possuem um programa proprietário que controla suas funcionalidades, com recursos reduzidos, não têm um sistema operacional como conhecemos, mas quando têm é um Sistema Operacional de Tempo Real onde os equipamentos e dispositivos têm necessidade de atender a requisições com tempo máximo de resposta. As aplicações embarcadas com requisitos de tempo real estão se tornando comuns, principalmente em sistema de controle de tráfego aéreo, defesa militar e em controle de processos industriais (SHIE, 2003).

Um sistema operacional embarcado é um sistema operacional para um sistema embarcado. Geralmente são bastante compactos e eficientes, não possuindo os recursos de um sistema operacional convencional que não serão utilizados no sistema embarcado.

Dispositivos embarcados são encontrados em vários lugares tais como: telefones celulares, *PDAs*, equipamentos de rede como roteadores, *drivers* de disco, leitores de *smart cards*, impressoras, *vídeo games*, microondas, geladeiras inteligentes, sistemas de controles industriais, equipamentos médicos e muitos outros.

4.4.1 Restrições para Sistemas Embarcados

Geralmente os dispositivos embarcados terão limitações, devido ao tamanho da sua configuração interna de *hardware*, podendo trazer ou não algumas restrições quanto as suas funcionalidade, tais como:

a) **portabilidade:** Uma característica importante de um sistema operacional embarcado é a sua portabilidade, pois necessita ser flexível o suficiente para ser portado para diferentes plataformas, trabalhando com diversos processadores como *Alpha, ARM, i386, Motorola PowerPC, SPARC, Motorola 68k, Hitashi SH3, VAX, MIPS*, etc. Estes sistemas são de modo geral bem modularizados para que possam ser configurados em função das necessidades da aplicação específica;

b) **limite de consumo de potência:** Como normalmente um sistema embarcado não tem uma fonte de energia contínua para alimentar o sistema, o processador deve ser bem dimensionado a fim de poder atender as necessidades sem desperdício de potência;

c) **memória:** Os sistemas operacionais embarcados possuem uma grande restrição em relação à memória, pois é preciso ser rápido e ter elevada capacidade. Como as aplicações embarcadas atualmente utilizam grande quantidade de memória, tem sido estudado as várias tecnologias de memória (*EEPROM, FRAM - ferromagnetic RAM, modelos flat-memory*, etc) para atender suas necessidades;

d) **temporais:** Os sistemas embarcados possuem uma característica chamada restrição temporal, ou seja, o resultado de um dado processo não depende apenas das respostas, mas também do instante em que elas foram dadas, determinando a política de escalonamento das tarefas (SILBERSCHATZ, 2004). Assim, uma tarefa não é executada somente porque está pronta para ser executada, mas sim porque tem prioridade máxima naquele momento;

e) **escalonamento:** Para realizar escalonamento de processos (SILBERSCHATZ, 2004) em sistemas embarcados, utiliza-se uma técnica denominada "Substituição de *Kernel*", ou seja, um segundo ***kernel*** é inserido no sistema operacional. Como exemplo tem-se o RT-LINUX - *Real-time LINUX*, onde o *LINUX* roda sob o controle do *kernel* de tempo real. Quando há uma tarefa de tempo real, o sistema operacional de tempo real assume o controle e executa o processo, mas quando não há tarefas de tempo real, o controle é devolvido ao *kernel* do *LINUX*. Esse modelo foi proposto pelos professores Victor Yodaiken e Michael Barabanov, do Departamento de Ciência da Computação, do *New Mexico Institute of Technology* (SHIE, 2003). Esta técnica apresenta um problema no caso de haver muitos processos grandes envolvidos, pois o *kernel* de tempo real não é preemptivo, podendo acarretar um grande retardo, prejudicando, assim, todo o sistema. Outra solução de escalonamento utilizada é o particionamento de funções entre *software* e *hardware*, em que uma função crítica como o escalonador pode ser implementada em *hardware* em um co-processador dedicado, visando aumento de desempenho ou atendimento de restrições temporais severas, especialmente em aplicações críticas, que apresentem um grande número de chaveamento de contexto.

4.5 UMA BREVE VISÃO SOBRE ETAPAS DE PORTAGEM DO LINUX

LINUX está ganhando popularidade em sistemas embarcados e muitos fornecedores comerciais de equipamentos estão especializando-se em portá-los em seus produtos. Os sistemas operacionais proprietários embarcados implicam em custos, que por si só já é uma desvantagem e fica particularmente pior nas constantes migrações de plataformas e/ou novas

versões de processadores. O mesmo ocorre com os *drivers* dos periféricos que fazem parte do *hardware* como um todo, que em alguns casos são cobrados à parte. Neste contexto quando pensa-se em *LINUX*, por estar disponível sob licença *GPL,* em alguns casos, torna-se uma boa opção, pois atrai o interesse das próprias empresas desenvolvedoras de processadores e periféricos em geral, como uma opção para atingir mais rapidamente o mercado, que portam os *drivers* em seus produtos, trazendo inúmeras vantagens para seu desenvolvimento. As versões mais recentes do *LINUX* são portáveis (MOODY, 1998; HORTON, 2004) para uma variedade de opções de arquiteturas, plataformas e processadores e apresentam-se em crescimento. Esta é a seqüência de tarefas para portar o *LINUX* numa placa:

a) estabelecer o *CDK* (*Cross Development Kit*) ou o *SDK* (*System Development Kit*) que consistem nas ferramentas para compilar o *kernel* que se quer;

b) definir o código de iniciação (conhecido como código de *Boot*) cuja principal função é efetuar alguns *checks* simples no sistema e carregar o *kernel* que se quer rodar e passar o controle para esse *kernel*;

c) implementar as funções dependendentes da arquitetura, plataforma e *CPU*;

d) implementar os códigos para os dispositivos que compõem o *hardware* da placa.

O trabalho de portagem, de modo geral, pode consumir muitos homens-hora, de forma que, na prática, a grande maioria dos fornecedores de solução baseada em *LINUX* se coloca no mercado, agregando algo de novo ao conjunto de soluções que já estão disponíveis, seja ela uma nova arquitetura, um remodelamento, uma nova *CPU* ou uma nova plataforma. Atualmente já existem arquiteturas bem definidas para as famílias: *PowerPC, Intel x86,*

SPARC, ARM, entre outras. O tamanho e a configuração necessária do *hardware* necessário pode variar muito conforme a aplicação que irá rodar no sistema. Por exemplo, num único *MPC860* com 1 MB de memória *flash* e 4MB de memória RAM pode ser suficiente para um *kernel* pequeno e em alguns casos, até mesmo CPU's de 16 *bits* podem ser usadas. A configuração mínima da placa absolutamente necessária é:

a) CPU, Flash, RAM, Relógios e circuitos auxiliares;

b) 1 porta console;

c) 1 porta serial ou 1 interface de rede.

Pode-se definir os passos básicos ou seqüências de desenvolvimento para portar o *LINUX* em um sistema como segue:

a) seqüência de inicialização do sistema (*Boot)*;

b) inicialização *CPU*/plataforma;

c) criação dos dispositivos requeridos;

d) montagem do sistema de arquivos (*File Systems)* ;

e) *scripts* para estabelecer ambiente de desenvolvimento (*run-time environment)* e disparar aplicações desejadas.

O tempo médio de implementação pode variar entre algumas semanas até vários meses de desenvolvimento, dependendo da complexidade do produto.

Conclui-se neste capítulo, que o usuário é o beneficiado com o uso de um sistema aberto, principalmente se ele tiver habilidades de projetista de sistemas. Com regras claras, direitos e deveres bem estabelecidos, qualquer um está livre de utilizar, modificar e criar novas funcionalidades para os sistemas disponíveis no mercado. O suporte pode vir através de empresas especializadas ou pelas comunidades de usuários deste tipo de sistemas. Os sistemas abertos devem sempre manter as suas características fundamentais que o fizeram conhecidos: interoperabilidade, modularidade, escalabilidade e portabilidade. Quanto aos sistemas embarcados, cada vez mais empresas fabricantes de equipamentos estão adotando este procedimento, por exemplo, portagem do *LINUX*, para criar sistemas mais enxutos e para aplicações específicas, como a de um eletrodoméstico com controle eletrônico embutido. Para isto, não são necessárias as configurações de *hardwares* tão complexas como se imagina.

5 ESTUDO DE CASO

Nos capítulos anteriores deste trabalho, foram vistas as evoluções das arquiteturas das redes industriais com o desenvolvimento de novos barramentos que trouxeram benefícios para os seus usuários: empresas e desenvolvedores de sistemas. Foram também abordadas as características do sistema operacional *LINUX,* um sistema estável e robusto que possui as funcionalidades para atuar em qualquer ambiente de trabalho. Neste capítulo, todos estes conhecimentos tecnológicos são compilados para propor uma nova rede industrial com componentes que compõem a família de soluções com Sistemas Abertos.

5.1 PROPOSTA DE UMA NOVA CONFIGURAÇÃO PARA REDES INDUSTRIAIS

As principais exigências cada vez mais requisitadas pela maioria das empresas atuais são: manterem suas informações importantes armazenadas em seus servidores de alta velocidade e estarem conectadas em rede para que os usuários deste sistema busquem as informações para o seu trabalho e acessem a rede mundial *Internet*. As interações entre usuários e servidores são cada vez mais intensas e velozes, sobretudo o item segurança deve ser de alta prioridade nas trocas de informações entre seus componentes. É neste ambiente que o sistema operacional *LINUX* tem-se destacado e ganho adeptos, consagrando-se e tornando-se um dos sistemas operacionais mais utilizados no mundo. Um dos objetos deste trabalho é mostrar sua viabilidade também na indústria.

A idéia é fazer com que a rede *CAN* seja localizada dentro da rede *LINUX* a qualquer instante e existam trocas de mensagens eficientes entre os elementos que compõem uma rede industrial. Para este propósito é utilizado o módulo *LECU* (*LINUX Electronic Control Unit*, Unidade de Controle Eletrônico *LINUX*) que é um módulo eletrônico, projetado para suportar

as caraterísticas de um ambiente industrial, responsável pelo controle de parâmetros do processo e também da rede. Ele é composto de interfaces de E/S digitais e analógicas conforme a sua aplicação. Para aumentar o seu desempenho, flexibilidade e conectividade na rede, o módulo é capacitado com o sistema operacional *LINUX*. Os aplicativos de supervisão farão a monitoração simultânea dos pontos espalhados pela área industrial, agilizando o tempo de atuação dos algoritmos de controle. O módulo *LECU* tem duas configurações: *MASTER* e *SLAVE*. O *MASTER* tem a função de interrogar cada *SLAVE*, receber deles o estado do ponto de E/S (sensores, atuadores, motores, etc) controlado. Por outro lado, os SLAVEs têm a função de controlar os pontos de E/S e repassar as informações solicitadas pelo MASTER ou espontaneamente, caso haja algum problema.

O protocolo *IP* (*Internet Protocol*) é a parte central do *LINUX* como sistema de mensagem, cujo propósito é ter os pacotes roteados e enviados através da rede do *LINUX*, assim como para ser localizado através da camada de Rede. Para ter o protocolo *CAN* integrado à rede, deve ser criada uma família de endereços para definir o endereçamento do *CAN* dentro da rede *LINUX*. Na camada de Rede, onde se cuidará da localização dos elementos controlados e do transporte das mensagens, será planejada a implementação de um processo que cuidará de encaminhar os pacotes para uma saída pré-estabelecida. Para simplificar tal desenvolvimento, a rede *CAN* será vista como uma única sub-rede, dentro do núcleo do *LINUX*, evitando, assim, dividir essa rede em sub-redes. Este encaminhamento funcionará no dispositivo *LECU MASTER* que executará também a função de *Network Address Translation* (*NAT)*, ou seja, conversor de endereçamento para fornecer serviços de acesso entre as redes gerais *IP* com os dispositivos *LECU SLAVEs*. Isto significa que o *LECU MASTER* deverá estar presente em tal configuração de rede. O mesmo processo controlará a

criação/destruição de grupos de *Multicast CAN*, unindo todos os *LECU SLAVEs* que cooperam para o mesmo resultado (SOARES, 1995).

5.2 LECU MASTER

É um módulo eletrônico para fins de controle, projetado com um microcontrolador capaz de suportar o sistema operacional *LINUX* completo para oferecer recursos múltiplos às aplicações de monitoração e controle de pontos espalhados pela fábrica. Ele é gerenciável remotamente por um Sistema Supervisor para aquisição de informações de nós da rede industrial. O *LECU MASTER* possui multiprotocolo, além do protocolo *CAN* para trocar pacotes de informações entre os *LECU SLAVEs* e ele mesmo, o protocolo *TCP/IP* para comunicar com o Sistema Supervisor, podendo este ser conectado através do padrão de rede *Ethernet*. Um dos maiores benefícios oferecidos por este módulo é a interoperabilidade e a monitoração multitarefa, agilizando a varredura dos pontos controlados, atuando e informando com rapidez o resultado obtido.

O *LECU MASTER* é projetado com quantidade limitada de interfaces de E/S digitais e analógicas, caso o sistema em questão requeira uma estrutura com poucos pontos para monitoração.

O *MASTER,* além de monitorar todos os *SLAVEs,* tem a função também de detectar e corrigir o processo em uma área de atuação pré-definida, trocando, a todo instante informações com o sistema supervisor.

Características do módulo:

a) sistema operacional *LINUX* completo;

b) interface de janelas *KDE*;

c) aplicações de controle e monitoramento: Algoritmo *PID* e *Software* de Gerência;

d) multiprotocolo: *CAN* e *TCP/IP*;

e) interface padrão de rede *Ethernet*;

f) interface padrão de comunicação *CAN*;

g) interfaces de E/S digitais;

h) interfaces conversora analógica/digital e digital/analógica;

i) gabinete e estrutura elétrica com padrão industrial.

5.3 LECU SLAVE

Similarmente ao módulo *LECU MASTER*, é um módulo eletrônico de controle capacitado com o sistema operacional *LINUX* compacto para oferecer recursos específicos às aplicações de monitoração e controle requisitados pelos pontos de coletas de informações do processo. Ele é gerenciado pelo *LECU MASTER* ou por um *CLP*, conectado através de um barramento *CAN*, para aquisição de informações de cada nó da rede industrial. O *LECU* possui multiprotocolo, além do protocolo *CAN* para trocar pacotes de informações entre os *LECU SLAVEs,* os *MASTERs* ou *CLP*s. Possui, ainda, uma interface *Ethernet* para conexão via protocolo *TCP/IP* com um computador através de uma placa de rede, o qual executará uma configuração e manutenção local. O conjunto de *Hardware* e *Software* que compõe este

módulo executa a leitura de entradas, o processamento das informações e a atuação através das saídas. Um dos benefícios oferecidos por este módulo é a monitoração multitarefa, agilizando a varredura dos pontos monitorados informando com mais rapidez o resultado obtido. A quantidade de interfaces de E/S digitais e analógicas expansíveis pode ser projetada conforme a necessidade da aplicação.

As características do módulo *LECU SLAVE* são:

a) sistema operacional *LINUX* compacto;

b) aplicações de controle e monitoramento: algoritmo *PID* e *software* de gerência;

c) Multiprotocolo: *CAN* e *TCP/IP*;

d) interface padrão de rede *Ethernet*;

e) interface padrão de comunicação *CAN*;

f) E/S digitais;

g) E/S analógicas;

h) gabinete e estrutura elétrica com padrão industrial.

5.4 *CLP* PROPOSTO COM SISTEMA OPERACIONAL LINUX

Os *CLP*s são utilizados pela sua estrutura física robusta, confiabilidade e quantidade de pontos monitoráveis. Neste trabalho, são propostas alterações na sua estrutura operacional, mantendo as características funcionais que o consagraram.

Todas as características já conhecidas do *CLP* podem ser encontradas facilmente em páginas eletrônicas de fabricantes ou em materiais acadêmicos disponibilizados na rede mundial *Internet*. Este trabalho propõe a portagem do sistema operacional *LINUX* para torná-lo ágil e robusto nas funcionalidades ligadas à rede. O *CLP* adquire, assim, a possibilidade de executar facilmente as tarefas simultâneas e em tempo real, caso a aplicação necessite, dando, assim, ao sistema condições de reduzir o tempo de respostas ao processo monitorado. As funções agregadas às características típicas do *CLP* são:

a) sistema operacional *LINUX* completo;

b) segurança em rede;

c) multitarefa;

d) multiusuário;

e) multiprotocolo de comunicação, principalmente o *TCP/IP* para possíveis aplicações com tráfego de imagens e sons;

f) características de sistema aberto: interoperabilidade, modularidade, escalabilidade, e portabilidade.

5.5 CONFIGURAÇÕES DO SISTEMA

As configurações propostas, que serão descritas a seguir, foram originadas através dos pontos discutidos neste e em capítulos anteriores e poderão ser melhoradas na medida em que novas funcionalidades sejam requeridas pela aplicação. Elas poderão ser facilmente implementadas pelas versatilidades dos componentes do sistema.

A primeira configuração apresentada na figura 12 é uma rede industrial convencional, utilizando um *CLP*, um sistema de supervisão para gerenciamento de toda atividade do sistema de automação e *LECUs SLAVEs* para monitoração dos pontos de E/S do processo. Todos estes elementos possuem o Sistema Operacional *LINUX* embarcado. As principais características dos sistemas abertos, citados no capítulo 3, são os grandes benefícios que o desenvolvedor e usuários de soluções industriais poderão perceber. As fortes interações de rede, que o *LINUX* proporciona, traz rapidez, e conseqüentemente, agilidade na troca de informações entre todos os elementos da rede e também um bom desempenho em segurança de acesso com configuração de senhas, criptografia (codificação de informações) nas trocas de informações e *Firewall* (*software* de filtragem de pacotes).

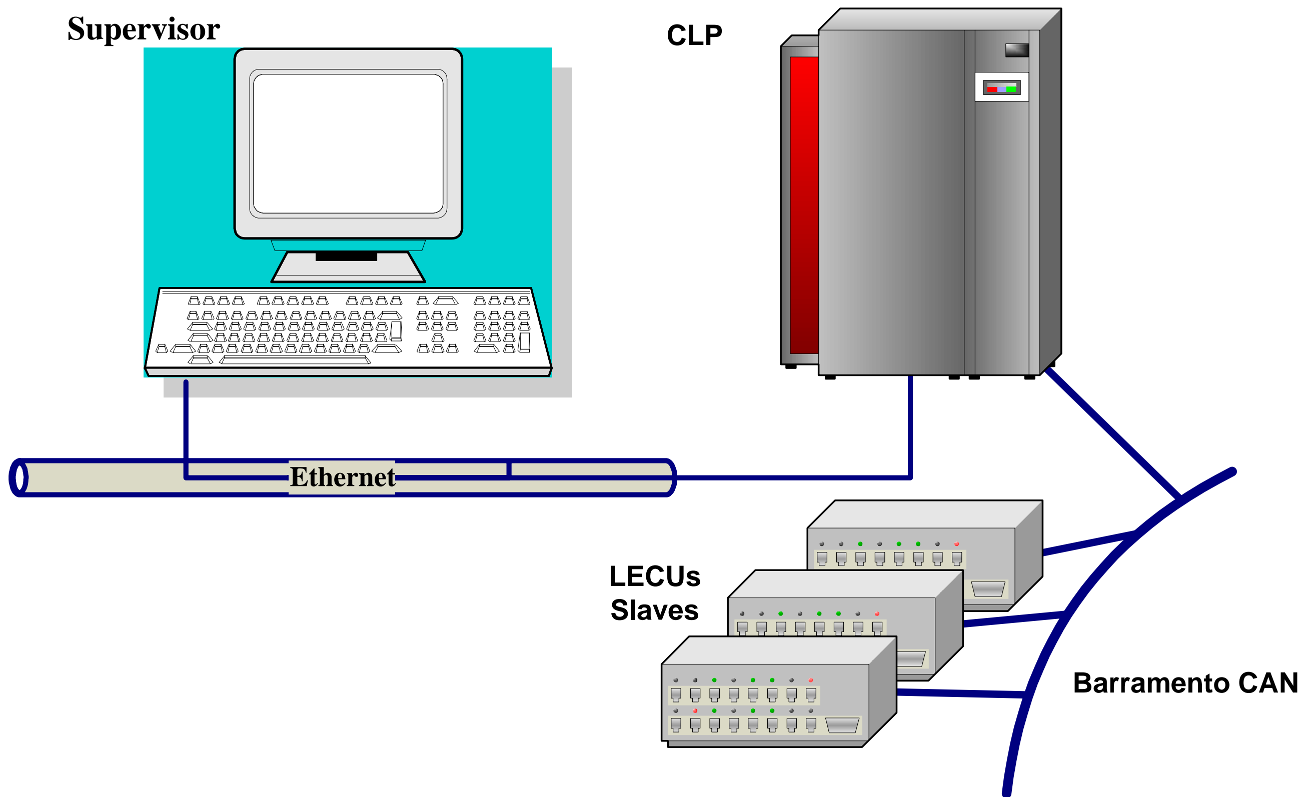


Figura 12 – Configuração Convencional.
Fonte: O autor

A segunda configuração apresentada na figura 13 trata de uma arquitetura completa baseada em componentes abertos. É composta de um sistema supervisor, *LECUs MASTERs* e *LECUs SLAVEs*. No *Desktop,* computador de mesa, do supervisor roda um sistema de gerência baseado no *LINUX* com sistemas de janelas e menus que facilita a navegação pelos elementos da rede, visualização de estados, monitoração dos alarmes e geração de relatórios.

Esta configuração proporciona muitos benefícios, pois herda todas as propriedades de um sistema aberto, bem como do protocolo de barramentos de campo *CAN* como sistema operacional *LINUX*. Estes são alguns tópicos importantes:

a) fortes interações de rede no *LINUX*;

b) tempo reduzido na aquisição de dados entre todos elementos da rede devido à característica multitarefa do *LINUX*;

c) economia no tempo de instalação /manutenção utilizando barramento *CAN entre LECU MASTER* e *LECU SLAVE;*

d) bom desempenho em segurança de acesso com configuração de senhas, criptografia nas trocas de informações e *Firewall*.

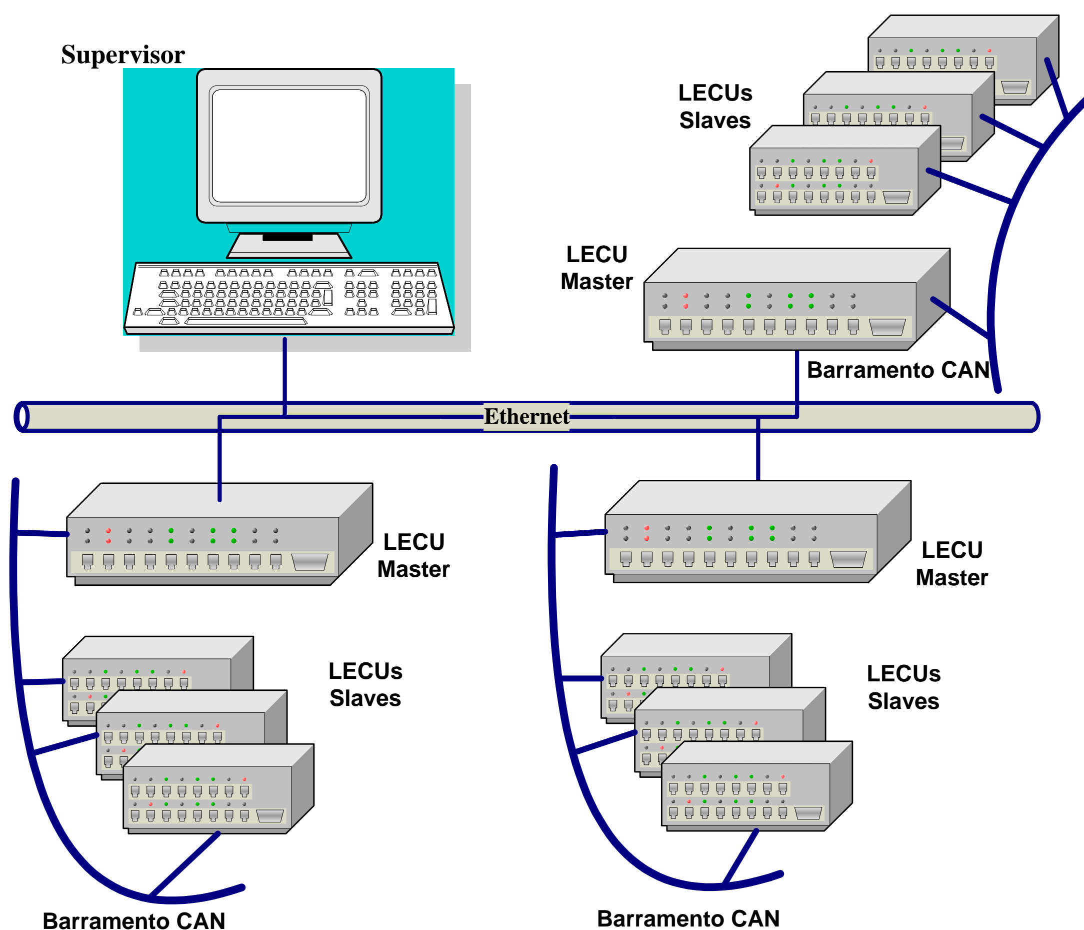


Figura 13 – Configuração Distribuída.
Fonte: O autor

Conclui-se, no final deste capítulo, que as redes industriais existentes atualmente podem ser viabilizadas com elementos de sistemas abertos. Os elementos utilizados em sistemas convencionais, que utilizam arquiteturas com microprocessadores, e que possuam os pré-requisitos para portagem do sistema operacional *LINUX* podem tornar-se dispositivos inteligentes, aumentando, assim, a capacidade e o desempenho da rede industrial.

6 IMPLEMENTAÇÃO E TESTES DO CONCEITO PROPOSTO

Neste capítulo, são mostradas as evidências e os testes realizados que comprovam a viabilidade técnica das soluções propostas, utilizando arquitetura aberta em aplicações industriais. Todos os componentes descritos, neste trabalho, já são utilizados comercialmente, o que se quer mostrar é o uso combinado deles em uma possível aplicação nas indústrias, como é o caso do sistema operacional *LINUX*. Ele já é conhecido pelo seu bom desempenho em tarefas críticas, a sua robustez em suas características de rede e pela sua arquitetura no gerenciamento de processos internos, como já descrito no capítulo referente ao tema. O desafio do *LINUX* é estabelecer-se como um padrão industrial, devido à resistência natural, nesta área, não aceitam facilmente as inovações tecnológicas. Entretanto, trabalhos como este pode contribuir para mostrar a este mercado quão bom são os produtos e soluções em arquitetura aberta.

Os testes mostram que a arquitetura proposta é viável, conseqüentemente, ela tornar-se-á uma realidade na indústria, à medida que mais aplicações começarem a surgir no mercado e mostrarem os seus benefícios. O barramento *CAN* hoje é um padrão aberto adotado na área automotiva, como sistema embarcado em veículos, e, mais recentemente, na área industrial, em países europeus principalmente na Alemanha. Este protocolo tem ganho adeptos ao seu uso, devido aos mecanismos seguros de controle dos pacotes de dados que trafegam no barramento.

Devido ao reduzido recurso financeiro, na elaboração deste trabalho, o projeto não pôde ser totalmente executado na prática, pois era preciso adquirir cartões eletrônicos com interface *CAN*, denominados *PCI-CAN*. Um cartão custava na época dos testes US$ 180,00, o qual significava, em Reais, R$ 1.300,00, incluindo-se as taxas de importação. Seriam

necessárias, pelo menos, três placas para montagem de uma rede. Porém, trabalhos estão publicados em páginas da *Internet*, sobre a utilização de interfaces *CAN* em indústrias e por pesquisadores de diversas empresas (BOSCH, 1991; TEXAS, 1999) no mundo todo, evidenciando o seu funcionamento, através de testes e de suas características técnicas. Com estas evidências práticas realizadas por várias pessoas ao redor do mundo (JANAKIRAMAN, 2003), pode-se ter a garantia e a segurança do seu funcionamento também na arquitetura proposta no capítulo anterior. Concluindo, assim, através destes trabalhos que este padrão pode ser viável também na indústria.

Com isso, concentram-se esforços, testando as conexões de rede na parte do *LINUX*. Foram utilizadas conexões *TCP-IP* entre os equipamentos Supervisor e *LECUs MASTERs*.

6.1 EVIDÊNCIA DO FUNCIONAMENTO DO PROTOCOLO *CAN*

A tecnologia de barramento *CAN* está consolidada no mercado industrial e existem inúmeros fabricantes de componentes espalhados no mundo de controladores e *transceivers* neste padrão. Apesar desse fato, não foi possível testar a rede proposta, na prática, de modo completo, conforme mencionado anteriormente. Uma tentativa foi com uma placa da *Sprectrum Digital,* cedida pela Sociedade Educacional de Santa Catarina - SOCIESC. Esta placa é composta por um componente eletrônico de altíssima integração, chamado *Digital Signal Processor* (*DSP*) de modelo TMS320LF2407, da fabricante *Texas Instruments* (JANAKIRAMAN, 2003; TEXAS, 2005). A figura 14 ilustra a face dos componentes do referido cartão. Os testes foram realizados, mas não foram relatados, pois esta placa não suporta interagir com o Sistema Operacional *LINUX*. Foram contactados os fornecedores desta placa, porém não foi obtido sucesso para utilizá-la nos testes.

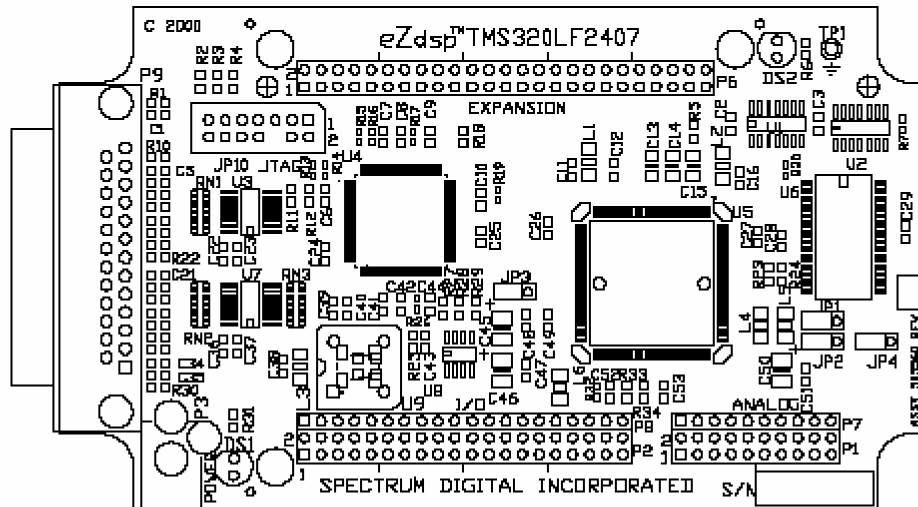


Figura 14 – Configuração Distribuída.
Fonte: Sprectrum Digital, 2005.

6.2 TESTES DE TROCA DE PACOTES - GERÊNCIA DA REDE INDUSTRIAL PROPOSTA

Na figura 15, é mostrado um esquema de montagem do teste desejado com todos os elementos que compõem a rede proposta: Sistema Supervisor, os *LECUs MASTERs* e os *SLAVEs* todos interligados entre si via *HUB*. As funções desempenhadas por cada elemento da rede estão descritas a seguir:

a) **nós I e II:** Estes nós são formados pelas unidades *LECUs MASTER*, os quais têm funções de controle e monitoração dos pontos de E/S, e dos *LECUS SLAVES*. Outra função executada por esta unidade é a de gerência de todos os nós que compõem a sua rede. Ela possui o mapeamento dos nós e as funções que cada um deverá desempenhar, e pode, inclusive, tomar decisões sobre o processo através da execução do algoritmo de controle do processo. O sistema supervisor pode atuar sobre qualquer nó dentro da rede através da unidade *LECU MASTER*, realizando,

entre outras funções, alterações de limites de medições e tempos de comutação dos atuadores. Os *LECUs SLAVEs* reportam ao *LECU MASTER* quando ocorre alguma anomalia no processo monitorado. Então, o *LECU MASTER,* em posse destas informações, tem o poder de tomar decisões, pois possui o algoritmo de controle de sua rede. Depois, ele reporta ao sistema supervisor as ações tomadas e as atualizações dos estados dos nós por ele gerenciado através de *Traps* (mensagens espontâneas).

b) **nós III e IV:** Estes nós são formados pelas unidades *LECU SLAVE*, os quais têm a função de registrar as variações nas grandezas eletromecânicas, tais como: temperatura, pressão, peso, etc., nos pontos de monitorações dentro da área industrial. O controlador *LECU SLAVE,* depois de tratar estas grandezas, envia os sinais de entrada e verifica se estão dentro dos limites estabelecidos. Caso estejam fora do limite, é enviada esta informação à unidade *MASTER* que tomará decisões com base nas informações do processo. A unidade ainda terá funções de controle de atuadores como, por exemplo, controle de uma válvula de vazão, as quais serão controladas localmente. Nos testes realizados as unidades *LECU SLAVE*, permaneceram desativadas.

c) **supervisor:** O mapa de toda a rede do processo controlado é visualizado através de software de supervisão, em que todos os pontos são mostrados, sinalizados por cores e representados por blocos de figuras. Operação de escrita e leitura, mudanças de parâmetros são algumas das tarefas executados por este centro de gerência. Qualquer mudança nos nós da rede é reportada ao sistema supervisor para atualização imediata dos estados dos pontos monitorados pelos *LECUs*

SLAVEs. É através do supervisor que o operador do sistema interage com todo o processo. O supervisor efetua as configurações e as atualizações de algoritmos de controle nos *LECUs* e tem total controle dos componentes da rede.

6.3 ESQUEMAS DE MONTAGEM

A montagem para os testes programados inicialmente seriam realizados na configuração mostrada na figura 15, composta pelos elementos de redes descritas no capítulo 6.2.

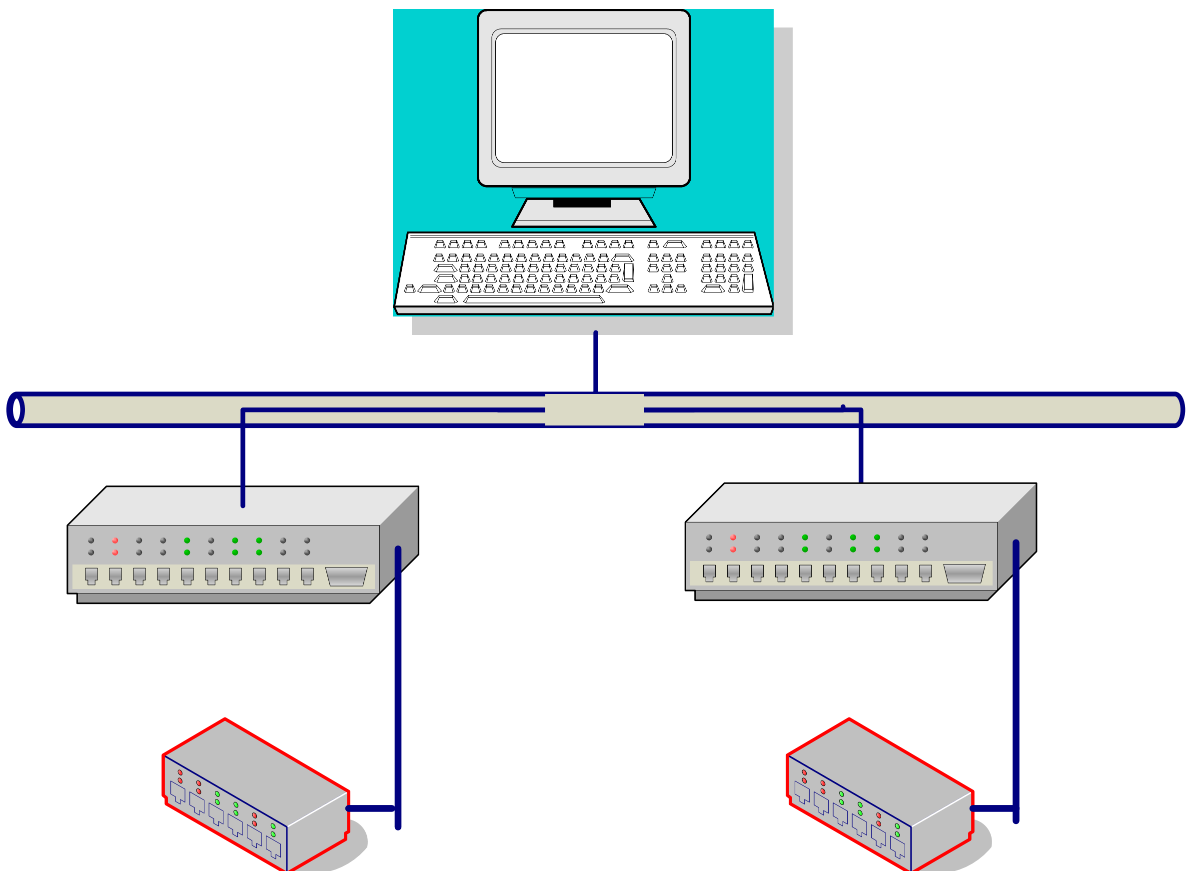


Figura 15 - Montagem dos Testes Desejados.
Fonte: O autor

Os testes foram realizados em uma configuração menor, conforme a figura 16, devido à impossibilidade de aquisição das placas PCI-CAN. Nesta configuração, foram utilizados um equipamento Gerente, que representa o Supervisor, e duas máquinas Agentes que representam os *LECUs MASTERs*. O Gerente é constituído por um computador da linha *PC,* modelo *Desktop*, com monitor de vídeo, mouse e teclado, fazendo parte de sua configuração. O computador gerente tem o *LINUX* como sistema operacional e endereço IP: 10.1.1.3. Os Agentes (A e B) são dois equipamentos da linha *PC* modelo *Thin Client*, os quais não possuem discos rígidos em sua configuração básica. Os Agentes (A e B) possuem o sistema operacional *LINUX* embarcado em cada um deles com os endereços *IPs*: 10.1.1.4 e 10.1.1.5 respectivamente. A interligação de todos estes equipamentos entre si, formando uma rede foi feita através de um *HUB*.

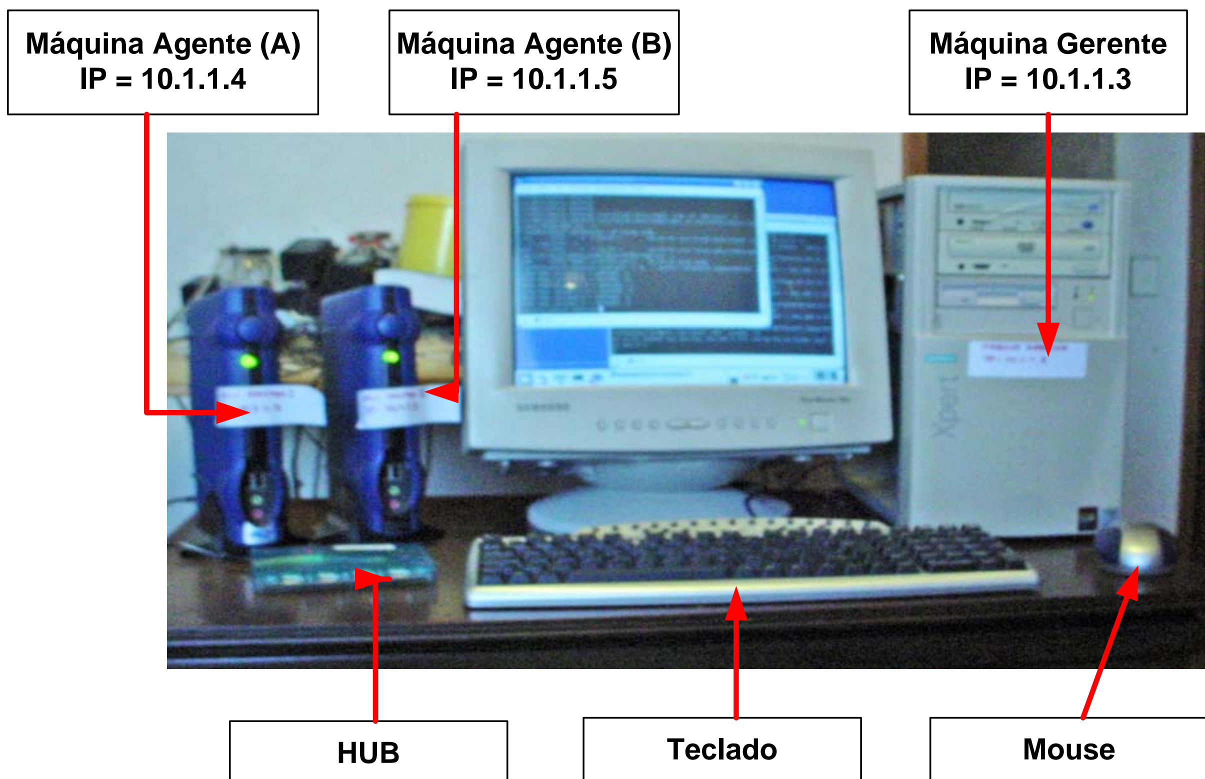


Figura 16 - Montagem dos Testes Executados.
Fonte: O autor

6.4 PROCEDIMENTOS DOS TESTES

Para a verificação do funcionamento da rede mostrada na figura 16, na primeira fase, foram utilizados os próprios recursos do *LINUX* para gerência de rede, como os pacotes: *ping*, *tcpdump* e *scripts* (NEMETH, 2004). Na segunda fase, utilizou-se o protocolo de gerenciamento de redes denominado *SNMP* (*Simple Network Management Protocol*). Este protocolo auxilia a verificação de todos os nós instanciados na rede, dando a idéia geral das condições de seus estados e simulando os pontos E/S do sistema controlado.

Os procedimentos de testes são compostos pelas seguintes etapas:

a) instalação e preparação da rede *LINUX*;

b) depuração da rede *LINUX*;

c) instalação do pacote *NET-SNMP*;

d) configuração do pacote *NET-SNMP*;

e) criação de scripts de testes de troca de mensagens entre os nós;

f) roteiro de execução dos testes e Resultados.

6.4.1 Configuração dos equipamentos utilizados nos testes

a) Computador Gerente:

- o *CPU Pentium* II;

- Clock de 400 MHz;
- 96 *Mbits* de memória de dados *RAM*;
- 40 *Gbits* de Disco Rígido.

b) Computador Agente:

- CPU AMD;
- Clock de 266 MHz;
- 128 *Mbits* de memória de dados RAM;
- 256 *Mbits* de memória de programa *FLASH*;

c) *HUB*:

- 4 Portas Ethernet.

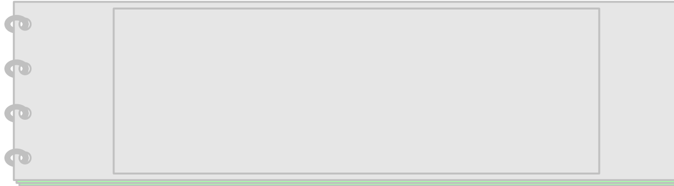
6.4.2 Instalação e preparação da rede *LINUX*

O Ambiente de Desenvolvimento foi o *LINUX Mandrake Community 10.1* – versão *linux-2.6.8.1-mdk*.

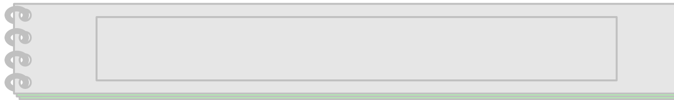
Após carregar a versão linux-2..6.8.1-mdk, através dos discos de instalação do *LINUX Mandrake Community,* foram realizados os seguintes passos:

1) após o término da instalação do pacote *LINUX*, o sistema foi reinicializado e conectado como *root* (conta de administrador);

2) o cursor foi posicionado no diretório de configuração, através do comando *cd /usr/src/linux-2.6.8.1*. O comando *MAKE* foi executado para compilar todos os elementos do sistema operacional *LINUX* para gerar uma versão no novo ambiente instalado;



3) foram carregados todos os programas gerados em seus diretórios padrões, através do comando *MAKE MODULES INSTALL*:



6.4.3 Depuração da rede *LINUX*

1) o comando *ping* foi utilizado para verificar o estado individual de cada máquina e depois foi testado o segmento de rede, direcionando o *ping* para outra máquina, mudando apenas o endereço IP (10.1.1.3, 10.1.1.4 ou 10.1.1.5).

2) foi utilizado, em conjunto, o comando *ping* e o comando *netstat –i*, para mostrar o estado da interface de rede.

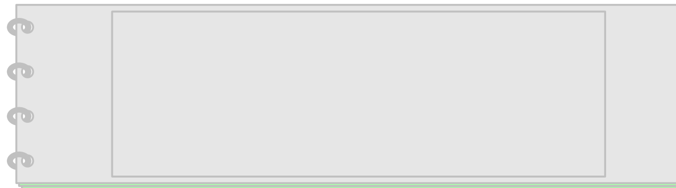
cd /u

make

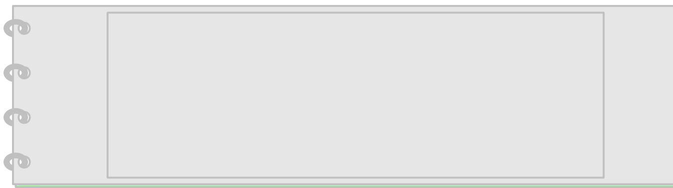
6.4.4 Instalação do pacote *NET-SNMP*

A versão do pacote *NET-SNMP* utilizado foi o *net_snmp-5.3.pre1*. Após fazer um *download* do referido pacote, através da *Internet* (NET-SNMP, 2005), foram realizados os seguintes passos:

1) o programa *NET-SNMP* foi instalado no diretório fonte do usuário. Para isto, o cursor foi posicionado neste diretório através do comando *cd /usr/src*. Depois, foi executado o comando *TAR*, que tem como finalidade compactar muitos arquivos em um único arquivo de armazenamento. Para maiores detalhes sobre os parâmetros deste comando, consultar (WELSH, 1997).



2) após o término do processo de instalação, o cursor foi posicionado no diretório de instalação, através do comando: cd /usr/src/net-snmp-5.3.pre1. Depois, o pacote foi configurado, através do comando ./configure:



Durante a execução da configuração, aparecem várias perguntas ao usuário, que foram respondidas da seguinte forma:

a) default version of SNMP to use (3): Tecla 2;

b) nos testes realizados foi especificada a versão do protocolo SNMP como versão 2C.

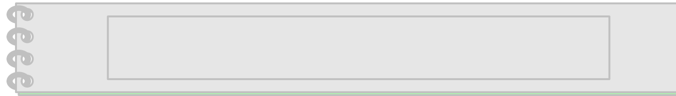
c) system Contact Information (root@): Tecla Return;

d) system Location (Unknown): Tecla Return;

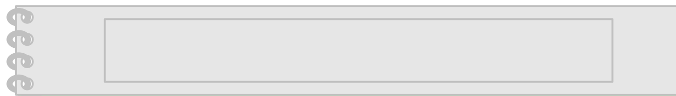
e) location to write logfile (var/log/snmpd.log): Tecla Return;

f) location to write persistent information (var/net_snmp): Tecla Return.

3) para que os arquivos do *NET-SNMP* fiquem em conformidade com o ambiente do sistema em questão, foi compilado todo o pacote, através do comando *MAKE*:



4) após o término da compilação dos arquivos do *NET-SNMP*, os novos arquivos gerados devem ser acomodados nos seus diretórios padrões, através do comando *MAKE INSTALL*:



Os diretórios pré-estabelecidos na elaboração deste projeto, para o desenvolvimento dos testes foram os seguintes:

a) diretório base do pacote *NET-SNMP*: /usr/src/net-snmp-5.3.pre1;

b) diretório para as fontes (*.c e *.h) das *MIBs*: /usr/src/net-snmp-5.3.pre1/agent/mibgroup;

c) diretório das fontes (*.txt) das *MIBS*: : /usr/src/net-snmp-5.3.pre1/mibs;

d) diretórios das fontes (*.txt) das *MIBS* instalados na compilação e que são acessados pelo gerente *SNMP* : /usr/local/share/snmp/mibs;

e) diretórios das bibliotecas snmp: /usr/local/lib;

f) diretório dos agentes executáveis daemons snmpd e snmptrapd: /usr/local/sbin;

g) Diretório dos comandos (snmpget, snmpset, snmptrap, snmptranslate,...): /usr/local/bin;

h) diretório do arquivo de configuração snmpd.conf: /etc/snmp;

i) diretório do arquivo de configuração snmpd.conf que será copiado durante a compilação: /usr/local/share/snmp;

j) diretório do arquivo de configuração da *MIB* snmp.conf: /usr/local/share/snmp;

k) diretórios do arquivo de configuração do *Trap* snmptrapd.conf: /etc/snmp; /usr/local/etc/snmp; /var/net-snmp.

6.4.5 Configuração do pacote *NET-SNMP*

O pacote *NET-SNMP* possui um arquivo de configuração denominado snmpd.conf. Este arquivo define todas as suas ações, como quais as *MIBs* a serem mostradas, o nível de

acesso de variáveis: leitura e ou de escrita, elementos de rede e etc. O arquivo snmpd.conf é um arquivo geral baseado no arquivo EXAMPLE.conf (localizado no diretório /usr/src/net-snmp-5.3.pre1) com as modificações de acordo com as necessidades do projeto. Há também uma ajuda disponível *on line* (# man snmpd.conf).

Os campos mais significativos estão descritos a seguir:

6.4.6 Criação de *scripts* de testes de troca de mensagens entre nós

Para cada novo projeto, é necessária a geração de novas bases de informações (*MIB*), correspondente às necessidades do sistema proprietário. O apêndice C explica a organização das *MIBs*. Na seqüência deste trabalho, são descritos detalhadamente as etapas de geração das bases de informações.

6.4.6.1 Criação e Instalação de uma Nova *MIB*

Utilizou-se o exemplo da figura 4, do capítulo 3, como base de uma aplicação, onde se tem conforme a figura 17 a seguir: um sensor de nível de silo de grãos, sensor de temperatura do silo de grãos, um controlador do Motor que controla a esteira de abastecimento do silo de grãos e um controlador do Motor B do ventilador do referido silo. Neste sistema, existe ainda a possibilidade de envio de alarmes espontâneos, caso seja detectada alguma anormalidade no processo monitorado, os *LECUs MASTERs* serão avisados e estes repassarão as informações para o sistema supervisor.

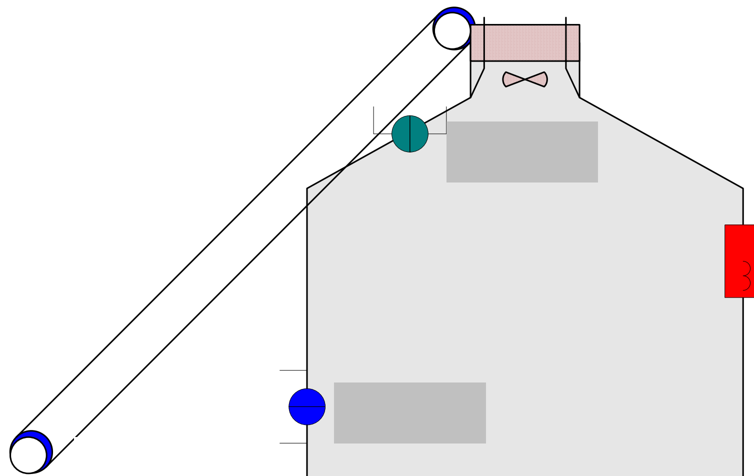


Figura 17 – Exemplo de Aplicação.
Fonte: O autor

A dinâmica de funcionamento do exemplo é o seguinte: o nível do silo é constantemente monitorado, igualmente a temperatura. Caso o nível de grãos no silo baixe de um valor estipulado, o *LECU* acionará o motor que controla a esteira que abastece o silo, até que o nível de grãos volte à normalidade. Simultaneamente, caso a temperatura do silo aumente além do limite estabelecido, o motor do ventilador será ativado, até que a temperatura volte ao seu valor normal.

Os seguintes passos foram seguidos:

1) foi criada uma *MIB* no diretório /usr/src/net snmp-5.3.pre1/mibs denominada TRIND-MIB.txt (Teste de Rede INDustrial-MIB) com os seguintes objetos:

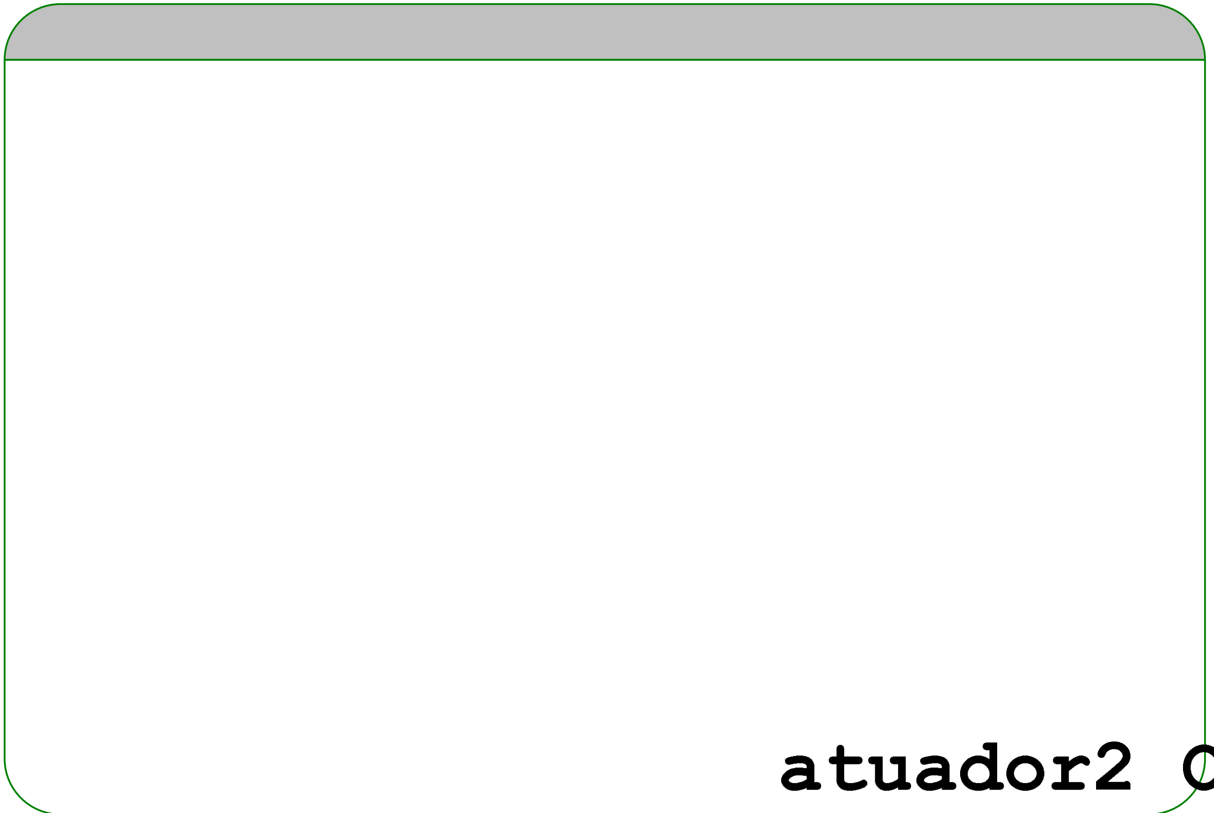
a) 2 sensores (sensor1.0 e sensor2.0) simulando sinais de entrada de pontos de leituras de temperatura e nível;

b) 2 atuadores (atuador1.0 e atuador2.0) simulando sinais de saída de controle de relés, para controle de motores;

c) 2 alarmes (alarme1.0 e alarme2.0) sinalizando situações espontâneas de erros como detecção de extrapolação da faixa de medidas;

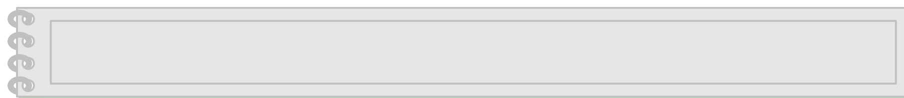
A seguir é mostrado o conteúdo do arquivo TRIND-MIB.txt, que representa a seqüência das definições de uma *MIB*.

```
TRIND-MIB DE
IMPORTS
    MODULE-I
    OBJECT I
    OBJECT-T
    FROM SNM
trind OBJECT

sensor1 OBJE
     SYNTAX
     MAX-ACC
     STATUS
     DESCRIP
::= { trind
```

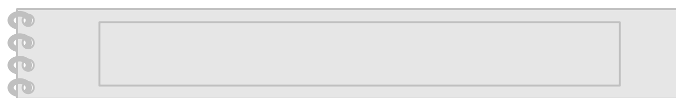


```
atuador2 OBJ
SYNTAX
MAX-ACC
STATUS
DESCRIP
{ trind

alarme1 OBJE
SYNTAX
MAX-ACC
STATUS
DESCRIP
::= { trind

alarme2 OBJE
```

2) Para que as ferramentas *SNMP* reconheçam as suas informações contidas na *MIB* do arquivo TRIND-MIB.txt, foi necessário criar um arquivo snmp.conf. Isto foi feito através do comando *echo* a seguir:



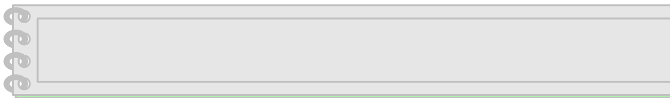
3) Depois, foram exportadas todas as *MIBs* (inclusive a criada TRIND-MIB), através do comando *EXPORT MIB = ALL,* para que todos possam reconhecê-los:



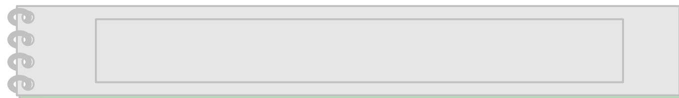
4) Para que as ferramentas SNMP conseguissem executar leitura e escritas nas variáveis das *MIBs* do projeto foi necessário gerar os arquivos fontes *.h e *.c, os quais correspondem a códigos de programa. Foi necessário instalar o suporte *SNMP* via Perl (linguagem de programação) para rodar o programa mib2c e gerar os fontes *.h e *.c das novas *MIBs.* A seguir, estão os passos para instalação do pacote *PERL*.

```
# cd /usr/src/net-snmp-5.3.pre1/perl/SNMP

# perl Makefile.PL

# make

# make test

# make install
```

5) Após terminado a instalação do pacote *PERL*, o próximo passo foi gerar os arquivos *.h e *.c do nosso projeto para gerar o Agente. O cursor deve ser posicionado no diretório */mibgroup* que é o diretório de desenvolvimento do Agente:



O arquivo da *MIB*:TRIND-MIB foi compilado via mib2c:



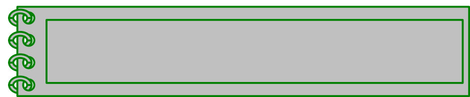
Com o comando mib2c, foram gerados dois arquivos denominados trind.c e trind.h no diretório: /usr/src/net-snmp-5.3.pre1/agent/mibgroup/

O Apêndice D1 lista o arquivo trind.c gerado pelo mib2c.

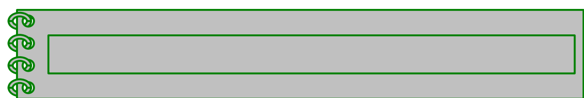
O Apêndice D2 lista o arquivo trind.h gerado pelo mib2c.

6. O Apêndice D3 lista os arquivos trind.c e trind.h com as alterações em função das necessidades do projeto.

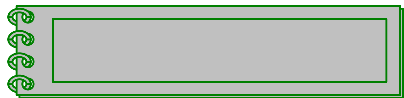
7. Foi configurada a compilação de um novo *daemon* snmpd para que as novas *MIBs* façam parte do pacote e sejam vistos pelo Agente. O cursor foi posicionado no diretório de instalação do NET-SNMP:



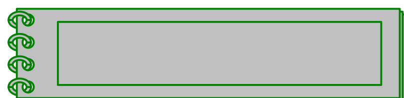
8. Uma nova configuração é executada para incluir a *MIB* do projeto, trind:



9. O novo processo do Agente *daemon snmpd* é gerado com a inclusão de novas *MIBs* através do comando *MAKE*:



10. O último passo é instalar o novo *daemon snmpd* e os arquivos que compõem o pacote NET-SNMP em seus respectivos diretórios, através do comando *MAKE INSTALL*:



Esta etapa de preparação do ambiente de testes consumiu muitas horas de trabalho do projeto, pois os documentos encontrados na *Internet* referentes ao pacote NET-SNMP, não foram suficientemente esclarecedores para instruir um desenvolvedor, na criação de uma *MIB* de teste e também de gerar um programa de acesso às variáveis da *MIB*. Dedicou-se muito tempo em pesquisa, análise, adaptações de rotinas de programas e geração de uma documentação.

6.4.7 Roteiro de execução dos Testes e Resultados

Após ter executado todos os passos do capítulo anterior, tem-se a certeza da criação de um ambiente de testes com todos os elementos necessários para rodar o Gerente e o Agente SNMP.

Este capítulo descreve a execução de todos os testes em conformidade com os procedimentos pré-estabelecidos. Os testes serviram para mostrar o funcionamento da rede na camada de gerenciamento. A montagem utilizada foi mostrada na figura 16 e a estrutura de

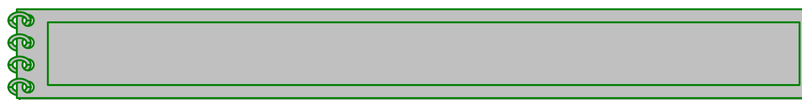
gerenciamento é o esquema Gerente/Agente com trocas de informações através do protocolo SNMP. Para obter mais detalhes sobre o ambiente de gerência, consultar o Apêndice C.

Foram criados processos denominados *snmpd* para rodar tanto na máquina Gerente como nos Agentes. Na seqüência, é executado um conjunto de testes que demonstra a troca de informações entre gerente (máquina 10.1.13) e agentes (máquinas 10.1.1.4 e 10.1.1.5).

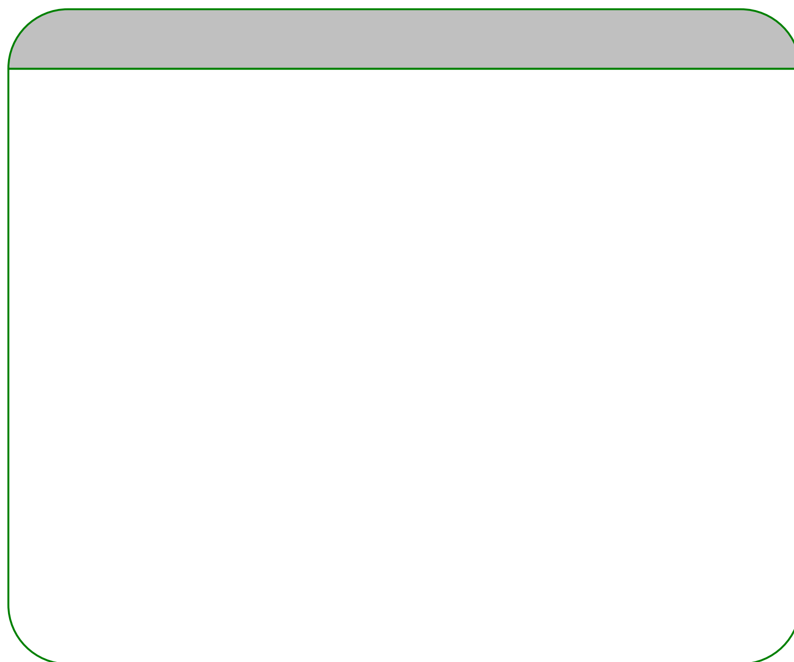
6.4.7.1 Testes de troca de mensagens dentro das máquinas localmente

Nesta fase, foram realizados testes com o objetivo de gerar resultados, que serviram de parâmetros para os testes da fase seguinte. Os testes verificaram o funcionamento correto do ambiente e seus componentes. Foram testados em todas as máquinas individualmente como máquina local (*localhost*):

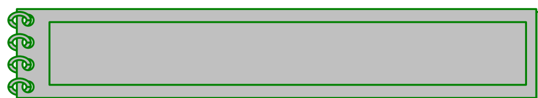
1) Para verificar se a *MIB*: *TRIND-MIB* está realmente carregada, utilizou-se o comando *snmptranslate* indicando a *MIB* do projeto (TRIND_MIB) e o ramo que ele se encontra (*experimental)*, conforme a estrutura lógica da *MIB*, ver figura C1:



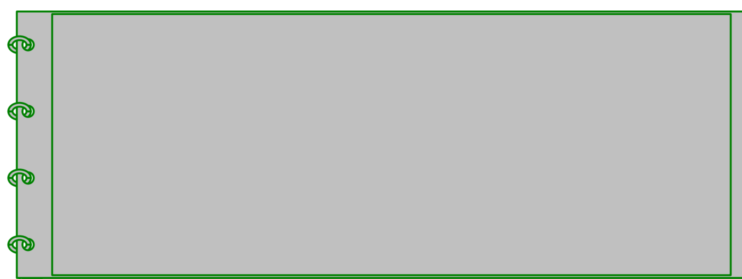
O Agente consultou a *MIB* do projeto e respondeu positivamente, conforme mostra a listagem a seguir:



2) O comando seguinte mostra como iniciar um processo. Ele é feito, iniciando o *daemon snmpd* com os seus parâmetros pré-definidos. Para maiores detalhes dos parâmetros deste comando verificar (NEMETH, 2004):

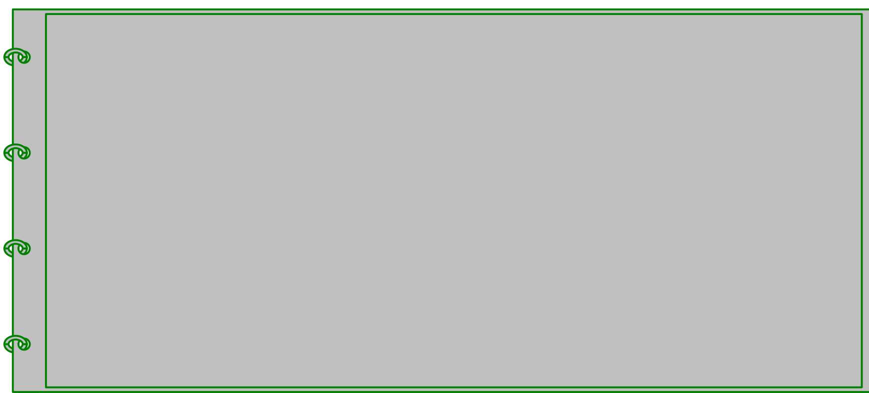


3) Foi verificado se o agente respondeu a *MIB: TRIND-MIB,* utilizando o comando *snmpwalk*, e o resultado foi o seguinte:



```
+--experime
   |
   +--trinc
      |
      +-- -
      |
      +-- -
      |
      +-- -
      |
      +-- -
      |
```

4) Testes de leituras foram executados na *MIB:TRIND-MIB* nas duas variáveis sensores: sensor1.0 e sensor2.0, através do comando *SNMPGET*. O resultado da leitura é mostrado a seguir, com seus valores inteiros iguais a zero.



5) Testes de escritas são executados na *MIB:TRIND-MIB* nas variáveis sensor1.0 e sensor2.0. Estes testes retornam erros, pois as variáveis sensores são somente de leituras, conforme esperado. Comando utilizado SNMPSET:

get -n

-MIB

snmpget -n

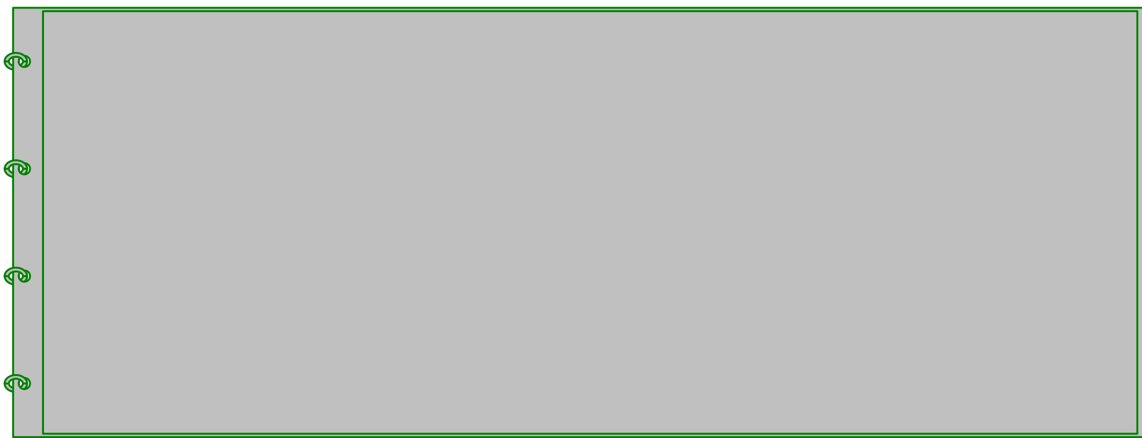
6) Testes de leituras e escritas foram executados na *MIB*:TRIND-MIB nas variáveis atuadores e foram realizados com sucessos, pois estas são variáveis de leitura e escrita. Foram utilizados comandos utilizados *snmpget* e *snmpset*:

```
# snmpset -m TRIND-MIB -v 2c -c public localhost atuador1.0 i 3
 TRIND-MIB::atuador1.0 = INTEGER: 3

# snmpget -m TRIND-MIB -v 2c -c public localhost atuador1.0
TRIND-MIB::atuador1.0 = INTEGER: 3

# snmpset -m TRIND-MIB -v 2c -c public localhost atuador2.0 i 4
TRIND-MIB::atuador2.0 = INTEGER: 4

# snmpget -m TRIND-MIB -v 2c -c public localhost atuador2.0
TRIND-MIB::atuador2.0 = INTEGER: 4
```

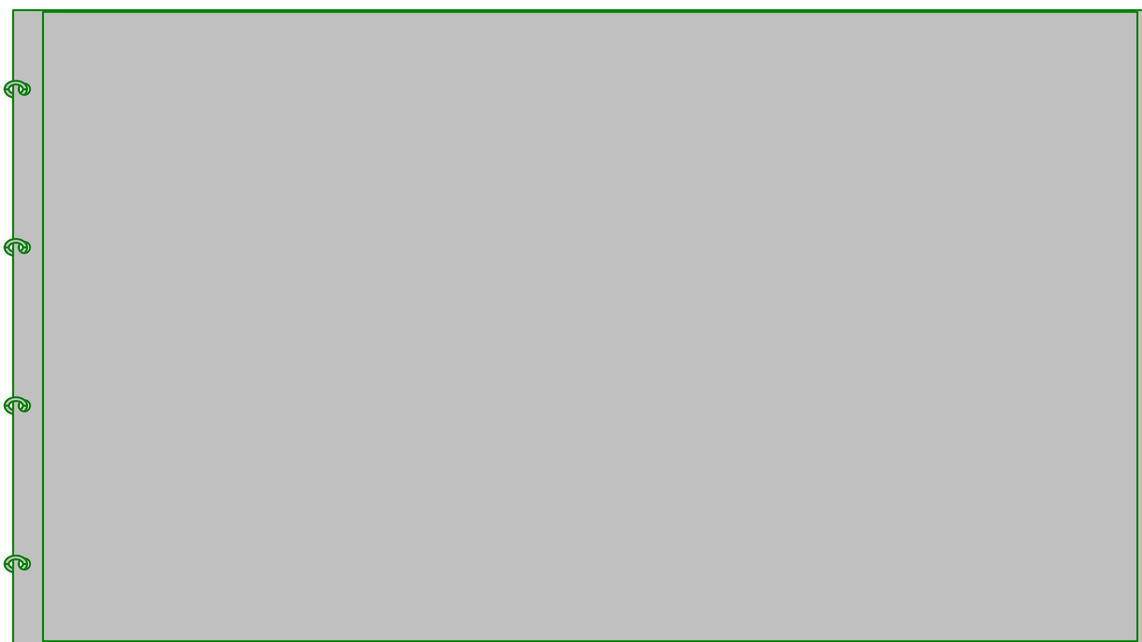
7) As variáveis alarmes não podem ser escritas com valores diferentes de 1, pois são utilizados somente para gerar *traps* através do agente. Assim, fez-se um teste para verificar o resultado ao escrever um valor correto e ou incorreto na variável:

a) neste teste, foi escrito um valor diferente de 1 na variável alarme1.0, e a máquina local retornou a mensagem “erro valor ilegal”, pois a variável não permite a atribuição de outro valor a não ser 1. Na seqüência foi escrito o valor 1 na mesma, e a máquina aceitou sem indicação de erro. Executando-se a leitura da variável alarme1.0, foi retornado o valor 0, pois a leitura desta variável, sempre resultará

em valor 0, independente do que for escrito. Isto é uma característica desta variável.



b) variável alarme2.0, idem da variável alarme1.0:



6.4.7.2 Testes de troca de mensagens entre as máquinas

Os testes realizados, neste capítulo, têm a função de verificar a interconectividade entre os elementos de rede e sua configuração, conforme ilustrado na figura 16. Para tanto, os seguintes endereços *IPs* foram definidos:

a) máquina Gerente: 10.1.1.3

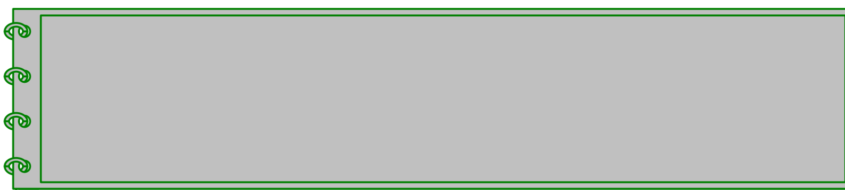
b) máquina Agente-A: 10.1.1.4

c) máquina Agente-B: 10.1.1.5

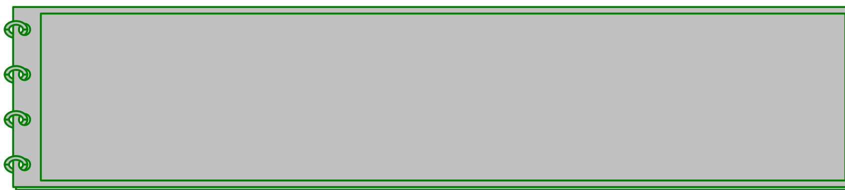
Os testes de trocas de mensagens foram realizados exaustivamente entre as máquinas interligadas em rede, igualmente ocorridas no modo local. O roteiro dos testes e os resultados são descritos a seguir:

1. Leitura das variáveis SENSORES:

a) através do comando *snmpget*, a máquina Gerente 10.1.1.3 leu a variável sensor1.0 da máquina Agente 10.1.1.4 e o resultado obtido foi: sensor1.0 = 0;



b) através do comando *snmpget*, a máquina Gerente 10.1.1.3 leu a variável sensor2.0 da máquina Agente 10.1.1.5 e o resultado obtido foi: sensor2.0 = 0.



2) Escrita e leitura de variáveis ATUADORES:

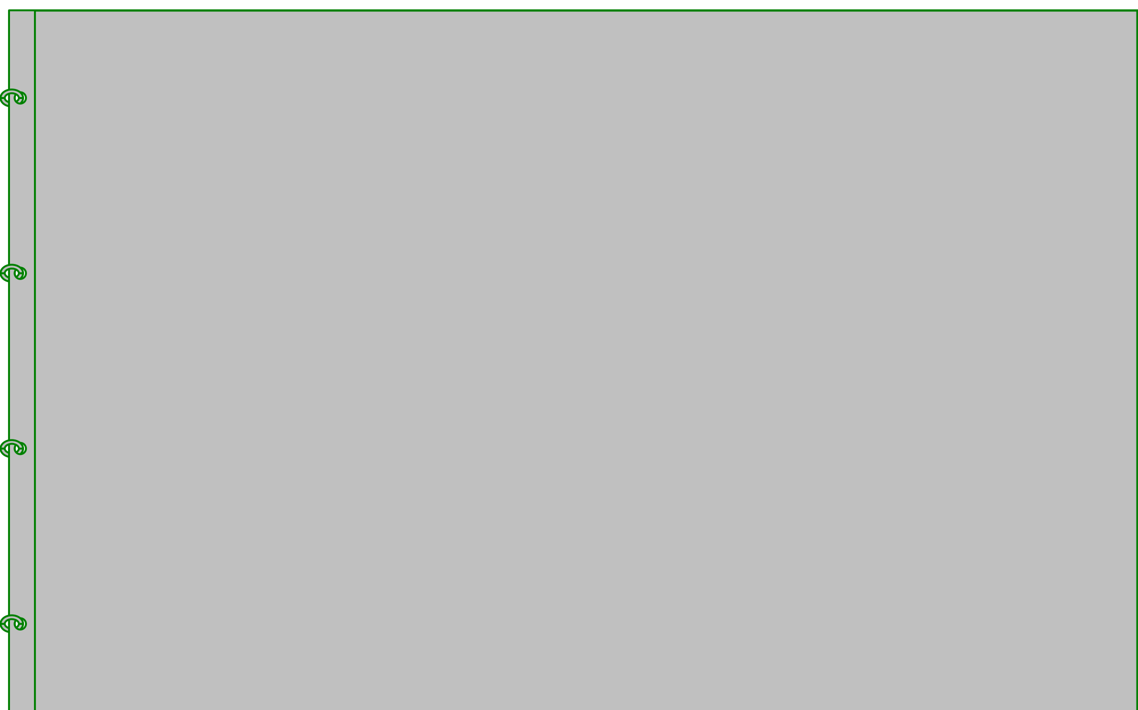
a) o Gerente 10.1.1.3 realizou escrita, através do comando *snmpset,* nas variáveis atuador1.0 do Agente 10.1.1.4 e atuador2.0 do Agente 10.1.1.5;

b) em seguida, fez a leitura das respectivas variáveis, o Gerente 10.1.1.3 executou a leitura através do comando *snmpget* na variável atuador1.0 do Agente 10.1.1.4 e na variável atuador2.0 do Agente 10.1.1.5.

snmpget -

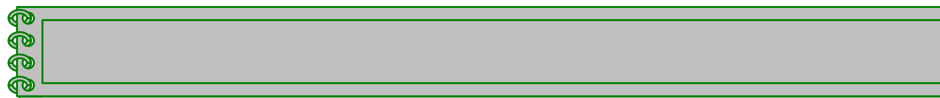
c) seguem as seqüências de comandos enviados e os resultados obtidos destes testes:

TRIND-MI

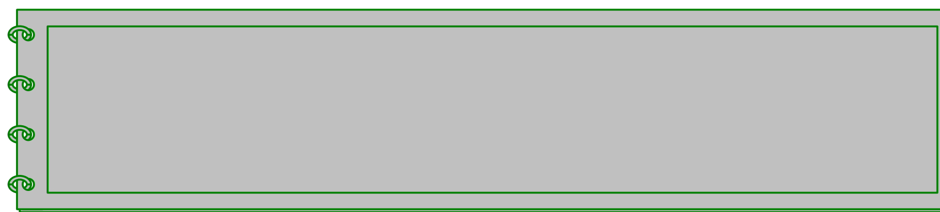


3) Existe a possibilidade do Agente enviar uma mensagem espontânea para notificar o Gerente de alguma anomalia ocorrida em pontos monitoradas por ele. Para observar o funcionamento deste teste, usou-se o comando *trap* do protocolo *SNMP,* como é descrito a seguir:

a) para gerar um *TRAP,* deve-se enviar um comando *snmptrap* para o gerente desejado. O *TRAP* sinaliza que ocorreu um erro em um ponto monitorado, representado por uma variável. Neste teste, foi utilizada a variável alarme1.0, o qual já havia sido carregado com valor 1, indicando que ocorreu um erro. No Agente 10.1.1.4, executou-se o comando *snmptrap,* atribuindo a ele a variável alarme1.0 como parâmetro, conforme segue:

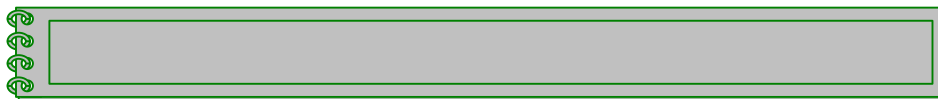


b) a resposta monitorada pelo gerente 10.1.1.3, através do comando *tcpdump,* indica a hora da ocorrência do alarme (22:33:36 hs), o endereço *IP* do agente que gerou o *trap* (10.1.1.4) e o endereço *IP* do gerente destino (10.1.1.3):

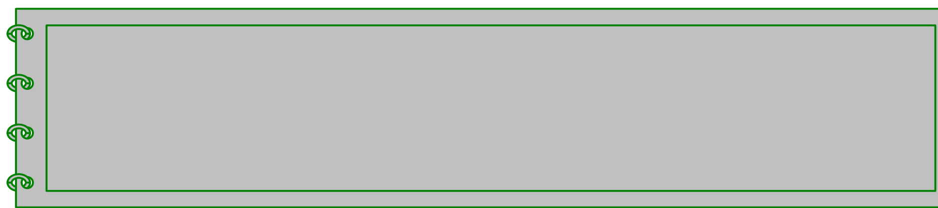


4) Foi repetido o mesmo processo do item anterior para o Agente 10.1.1.5 para a variável alarme2.0:

a) executou-se o comando *snmptrap* no agente 10.1.1.5, atribuindo a ele a variável alarme2.0, conforme segue:



b) a resposta monitorada pelo gerente 10.1.1.3 através do comando *tcpdump* foi:



Com as informações obtidas pelo *trap*, o software supervisor poderá usá-las para gerenciar problemas na rede.

Concluindo, mostrou-se, neste capítulo, o planejamento, a execução de um roteiro programado de testes na camada de Gerência e o relato dos resultados de cada teste com a sua avaliação. Pode-se verificar o bom andamento dos testes, através da análise dos resultados obtidos: as comparações dos resultados nos testes locais e em rede foram semelhantes, os quais provam que as trocas de mensagens entre as máquinas ocorreram com sucesso.

snmptrap –

Os resultados obtidos confirmaram o sucesso na implementação e na monitoração de uma rede em ambientes que combinam elementos de sistemas abertos.

7 CONCLUSÕES

Pelos estudos realizados dos sistemas de controle de processos existentes atualmente, como soluções nas indústrias, nota-se uma tendência para os próximos anos, dos avanços dos sistemas e *softwares* abertos, que atualmente já ganharam força entre usuários domésticos, governos, comércio e certamente ganharão adeptos nas indústrias. Quebrou-se o paradigma do modo de distribuição de produtos. Antes o que era aberto e de graça era sinônimo de baixa qualidade e sem suporte técnico. Através do sucesso da experiência de Linus Torvalds com o *LINUX* e da formação da comunidade *GNU* com suas regras claras, descobriu-se um novo horizonte para o rápido desenvolvimento dos sistemas abertos. É, nesse contexto, que este trabalho se encaixa, propondo uma nova configuração para redes industriais com componentes abertos e factíveis de serem colocados, hoje, em prática. Este trabalho contribui para a visualização desta nova revolução industrial, decorrente dos sistemas e produtos, utilizando o conceito de Sistemas Abertos na indústria.

Iniciou-se este trabalho, analisando a evolução dos sistemas de controle industriais, comparando as soluções proprietárias que muitas indústrias ainda utilizam, motivadas pela aparente segurança que a empresa desenvolvedora oferece de seu projeto ou sistema.

Foram abordadas as tecnologias emergentes de redes industriais, suas evoluções das conexões multi-vias para as conexões a dois fios e seriais, por meio de barramentos. Dentre estes tipos de conexões, foi escolhido para este trabalho o *CAN,* pelas suas boas características que o consagraram, como: alta velocidade, mecanismo de controle de colisão e tratamento de erro. Este barramento é utilizado em sistemas embarcados de veículos e também algumas experiências em redes industriais com sucesso.

Na seqüência do trabalho, dedicou-se um capítulo aos sistemas abertos, dando ênfase aos produtos de altíssima qualidade, existentes no mercado atual, e o sistema operacional *LINUX* encabeça a lista de produtos bem sucedidos. Grandes empresas, no mundo dos negócios, já aderiram e continuam aderindo a ele. Estão instalando em seus servidores informações valiosíssimas, pois confiam na segurança de acesso ao sistema a um pacote de programas desenvolvido por muitas pessoas espalhadas pelo mundo.

Por fim, uma arquitetura aberta de rede foi proposta, no trabalho, com elementos desenvolvidos neste padrão, como: o Sistema Operacional LINUX, os elementos inteligentes de rede *LECUs MASTERs* e *SLAVES*, o barramento *CAN* e um Sistema Supervisório.

Para ilustrar e validar este estudo, foram realizados testes, atingindo-se assim, o objetivo de apresentar um modelo aberto de comunicação em redes de sistemas de automação industrial e mostrar a sua implementação. Foi verificada e evidenciada a viabilidade prática da utilização de componentes de sistemas abertos em uma rede, especificamente em redes industriais com as devidas modificações dos elementos para este ambiente, conforme citados em vários capítulos deste trabalho. Foram testados: o sistema operacional *LINUX* e as suas características de rede e segurança; a rede que interliga o supervisório e as máquinas *LECUs*; as funções de gerência do *LECU MASTER*. Ao longo do desenvolvimento do trabalho, foi-se descobrindo muitos *softwares* abertos que serviram de auxílio para a execução dos testes práticos como: *NET-SNMP* e outros *softwares* utilitários de apoio.

O *LINUX* possui muitas particularidades e o usuário iniciante precisa gastar muitas horas para o seu aprendizado e utilização, pois se trata de um sistema com inúmeras funcionalidades como por exemplo: sistema, rede e segurança, os quais devem ser adequadas para o uso especifico de cada projeto. Na verificação do funcionamento da rede de teste, após

a sua montagem, foram encontrados problemas, que foram solucionados após o estudo mais detalhado do sistema operacional *LINUX*.

Os elementos de rede *LECUs MASTER E SLAVE* foram simulados com o uso de computadores compactos com o sistema operacional *LINUX* já embarcado neles. Este procedimento não alterou o resultado dos testes caso fossem utilizados *hardwares* proprietários, utilizando microcontroladores e periféricos desenvolvidos exclusivamente para serem os elementos *LECUs*, pois o importante foi a simulação idêntica de funcionalidades em ambos os casos. Foram observadas durante a realização dos testes, que as trocas de informações entre os elementos de rede, eram feitas de modo eficiente e seguro, comprovando uma das principais características do sistema operacional *LINUX,* que é a robustez no gerenciamento das conexões de rede.

Para o gerenciamento dos elementos da rede, foi utilizado o pacote *NET-SNMP*, que é um sistema baseado no protocolo *SNMP* e disponível para uso livre de qualquer pessoa que queira elaborar um teste de gerenciamento sem gastar muito tempo. Apesar de esta ser a proposta, foram encontrados problemas na configuração e escrita de programas de testes, pois as informações encontradas na página eletrônica do pacote muitas informações não são tão triviais para serem utilizados na montagem dos elementos necessários para o seu funcionamento. Após muitas tentativas e erros, as dificuldades foram vencidas e este trabalho contribui com informações que facilitarão os novos usuários do pacote *NET-SNMP*. Todos os conhecimentos adquiridos sobre este pacote estão descritos ao longo deste trabalho escrito para serem transmitidos aos leitores e motivá-los a usarem os conceitos dos sistemas abertos em suas aplicações, tanto particulares como profissionais, como incentivo para o desenvolvimento contínuo neste caminho que não tem mais volta.

A evolução natural deste trabalho é a sua continuidade com o desenvolvimento prático dos conceitos descritos e a elaboração dos módulos *LECUs*, definindo um hardware específico para o MASTER e o SLAVE, e um Sistema Supervisor mais elaborado em *LINUX,* utilizando janelas para ser mais amigável ao usuário.

Futuramente, alguém que se interesse em dar continuidade ao tema deste trabalho, poderá utilizar, além de versões mais elaborados do programas *MIB Browsers,* outros *softwares* para *LINUX* existentes no mercado, que são mais bem elaborados para monitoramento e controle de uma rede industrial. como é o caso, por exemplo, do *LabView,* da empresa *National Instruments*.

REFERÊNCIAS

ANS.1 - USO E ESPECIFICAÇÃO FORMAL DE PROTOCOLOS. Disponível em: http://www.cic.unb.br/ extensao/c003/asn1_ufrgs.pdf. Acesso em: 06.fev.2005.

ARNETT, Mathew, Desvendando o *TCP/IP*, Rio de Janeiro: Campus, 1995.

AZEVEDO, G. P.; OLIVEIRA FILHO, Ayru L.. IEEE Computer Applications in Power, pp. 27-32, October 2001.

BOSCH, CAN Specification Version 2.0, 1991, Disponível em: <http://wiki.daimi.au.dk/oodist/files/can2spec.pdf.> Acesso em: 11.abr.2004.

COMPUTER ASSOCIATES, Exploiting the Potential of Linux, White Paper, May 2004. Disponível em: <http://www.ebcvg.com/pdf/dl/exploiting_the_potential_of_linux.pdf.> Acesso em: 02.fev.2005.

FREE SOFTWARE FOUDATION. Site com artigos sobre regras de distribuição livre de programas de computadores. Disponível em: <www.fsf.org/>. Acesso em: 16.jan.2005.

GARRELS, Machtelt. Introduction to Linux: A Hands on Guide, Version 1.17 20050301 Edition, 03, 2005. Disponível em: <http://www.tldp.org/LDP/intro-linux/intro-linux.pdf>. Acesso em: 19.jun.2005.

GNU – The Project. Site com artigos sobre regras de distribuição livre de programas de computadores. Disponível em: <www.gnu.org/gnu/thegnuproject.html>. Acesso: 16.jan.2005.

HORTON, David. Pocket Linux Guide, Revision 3.0, 2004−11−02, Revised by: DH, Modified code example per author. Disponível em: <http://www.tldp.org/LDP/Pocket-Linux-Guide/Pocket-Linux-Guide.pdf>. Acesso em: 05.dez.2004.

INTERBUS - Technology For The New Millenium. Interbus Basics. Disponível em: <www.interbusclub.com>. Acesso: 19.jan.2005.

JANAKIRAMAN, HAREESH, SPRA890A, Programming Examples for the 24x/240xA CAN, Texas Instrumens, January 2003, Revised: April 2003. Disponível em: <http://focus.ti.com/lit/an/spra890a/spra890a.pdf>. Acesso em: 07.ago.2005.

MATTHEW, Neil; STONES Richard, Beginning Linux Programming (Linux Programming Series), 2nd ed. Birmingham: Wrox Press, 1999. ISBN: 1861002971.

MDIC-MINISTÉRIO DO DESENVOLVIMENTO, INDÚSTRIA E COMÉRCIO EXTERIOR. O Impacto do Software Livre e de Código Aberto na Indústria de Software do Brasil, Brasília 2005. Disponível em:< http://www.softex.br/media/pesquisa-swl.pdf>. Acesso em: 16.jan.2005.

MOODY, Paul. One approach to an embedded Linux, Revised draft: 1.1b, 1998.05.01. disponível em:<http://old.lwn.net/lwn/1998/0430/miniHOWTO.html, http://www.eklektix.com/lwn/1998/0430/miniHOWTO.html> Acesso em: 16.jan.2005.

NEMETH, Evi; SNYDER, Garth; HEIN, Trent R.. Manual Completo do Linux: Guia do Administrador, Tradução de Ariovaldo Griesi. São Paulo: Pearson Makron Books, 2004.

NET-SNMP. Um pacote de programa de gerência. Disponível em:< www.net-snmp.org>. Acesso em: 15.mai.2005.

ODVA. DeviceNet Technical Overview - PUB0026R1, Disponível em: <http://www.odva.org>/. Acesso em : 10.set.2004.

PANIAGO, Carlos Fernando Assis. Software livre para desktop. Disponível em: <http://www.cnpm.embrapa.br/>. Acesso em: 15.mai.2005.

PROFIBUS Brochure, Technical Description - Order-No. 4.002, September 1999. Disponível em: <http://www.dia.uniroma3.it/autom/Reti_e_Sistemi_Automazione/PDF/Profibus%20Technical%20Overview.pdf>. Acesso em: 19.jan.2005

RUBINI, Alessandro. LINUX Device Drivers. São Paulo: Market Books do Brasil, 1999.

SEIXAS JUNIOR, Constantino. Ethernet Industrial, UFMG, Departamento de Engenharia Eletrônica. Disponível em: <http://www.cpdee.ufmg.br/~seixas/PaginaII/Download/DownloadFiles/Aula%20IEC%2061131-3.pdf>. Acesso em: 06.jan.2005.

SHIE, Erlich. Custom Linux: A Porting Guide, Porting LinuxPPC to a Custom SBC, Revision 2.1 2003-03-08. Disponível em: <http://www.tldp.org/LDP/cpg/Custom-Porting-Guide.pdf>. Acesso em: 23.jan.2005.

SILBERSCHATZ, Abraham; GALVIN, Peter e GAGNE, Greg. Sistemas Operacionais com Java, Tradução de Daniel Vieira. Rio de Janeiro: Campus, 2004.

SOARES, Luiz Fernando Gomes; LEMOS, Guido; COLCHER, Sergio. Redes de Computadores: das LANs, MANs e WANs às Redes ATM. Rio de Janeiro: Campus, 1995.

STALLINGS, William. SNMP, SNMPv2, SNMPv3 and RMON1 and 2. 3rd ed. Massashusetts: Addison-Wesley, 1999.

TEXAS, INSTRUMENTS. SPRU024E, *TMS320C2x/C2xx/C5x - Optimizing C Compiler User's Guide,* August 1999. Disponível em: <http://focus.ti.com/lit/ug/spru024e/spru024e.pdf>. Acesso em: 07.ago.2005.

WELSH, Matt; KAUFMAN, Lar. Dominando o Linux. Rio de Janeiro: Ciência Moderna, 1997.

APÊNDICES

Apêndice A – Características de rede e gerência do *LINUX*

Apêndice B – Informações complementares do barramento CAN

Apêndice C - Protocolos de gerenciamento de redes

Apêndice D - Listagem de códigos do arquivo de programas *Scripts*

Apêndice E - Trabalho apresentado no ISA-2005 em Chicago - EUA

A. CARACTERÍSTICAS DE REDE E GERÊNCIA DO *LINUX*

A1. TCP/IP (*Transmission Control Protocol/Internet Protocol*, Protocolo de Controle de Transmissão/Protocolo *Internet*)

É um conjunto de protocolos utilizado em muitos sistemas operacionais, dentre eles, no *LINUX/UNIX*, sendo o nome herdado dos dois protocolos mais importantes: o *TCP* e o *IP* (ARNETT, 1995). Diz-se que o grande sucesso técnico da Internet deve-se, em grande parte, ao *design* elegante e flexível do *TCP/IP* e ao fato de ser parte da família de protocolos aberta e não proprietária (NEMETH, 2004).

O *TCP* é um protocolo orientado a conexões, que facilita a conversação entre dois programas como uma ligação telefônica. O *TCP* fornece entrega confiável, *full-duplex*, controle de fluxo, correção de erros entre os processos nos dois *hosts* e controle de congestionamento.

O protocolo IP, que é a base da estrutura de comunicação da *Internet*, é um protocolo que direciona pacotes de dados de uma máquina à outra.

Os protocolos *TCP/IP* podem ser utilizados em redes simples como uma ligação ponto-a-ponto ou em rede de pacotes bem complexos. Exemplos: *Ethernet*, barramentos SCSI, PPP, X.25, *Token-Ring*, ligações telefônicas discadas, enlaces de satélites, etc.

A2. CAMADAS DO *TCP/IP*

O *TCP/IP* é dividido em camadas conforme ilustra a figura A1:

Figura A1 – Modelo de rede *TCP/IP*.
Fonte: O autor

A3. CAMADA DE REDE

A camada de Rede é a responsável pelo envio de datagramas construídos na camada Inter-rede e pelo mapeamento entre um endereço de identificação de nível Inter-rede para um endereço físico ou lógico do nível de Rede.

Exemplos de protocolos nesta camada são:

a) protocolos com estrutura de rede própria (X.25, *Frame-Relay*, ATM);

b) ethernet: Este protocolo tem como objetivo transferir *frames* (quadros) para máquinas que estão na mesma rede;

c) PPP (*Point to Point Protocol*): Este protocolo tem como função transportar todo o tráfego entre dois dispositivos de rede através de uma conexão física única;

d) *Token-Ring* (Anel de "*Token*"); É um protocolo da *internet* que opera na camada física (ligação de dados) do modelo OSI conforme a figura A1.2. É utilizado numa topologia em anel em que as estações devem aguardar a sua recepção para

transmitir. A transmissão dá-se durante uma pequena janela de tempo, e apenas por quem detém o *token*;

e) FDDI (*Fiber Distributed Data Interconnect*): É uma implementação *TokenRring,* baseada nas especificações sobre fibras como mídia física. Em oposição as *Token Ring* simples (caminho único), FDDI usa dois caminhos para obter melhores resultados;

f) protocolos de Nível Físico (V.24, X.21).

As máquinas, neste nível, possuem uma identificação única chamada endereço MAC (*Medium Access Control*) ou endereço físico que distingue uma máquina de outra, possibilitando o envio de mensagens específicas para cada uma delas.

A4. CAMADA INTER-REDE

A comunicação na camada Inter-rede entre máquinas é feita através do protocolo IP. A identificação de cada máquina e a rede, onde esta está situada, é feita através do endereço IP. Se houver endereçamento nos níveis inferiores, é realizado um mapeamento para possibilitar a conversão de um endereço IP em um endereço deste nível.

Os protocolos existentes nesta camada são:

a) IP (*Internet Protocol*): Protocolo de transporte de dados não confiável, sem conexão;

b) ICMP (*Internet Control Message Protocol*): Este protocolo é um protocolo de suporte para o IP, que fornece controle e Relatório de erros;

c) ARP (*Address Resolution Protocol*): Este protocolo descobre qual endereço *Ethernet* (endereço de *hardware* – físico) utilizar para conversar com um determinado endereço *Internet*;

d) RARP (*Reverse Address Resolution Protocol*): A função deste protocolo é inversa ao ARP, ou seja, converte um endereço de *hardware* (físico) em um endereço *Internet*.

A função mais importante desta camada é realizada pelo protocolo IP que é a própria comunicação inter-redes. Este protocolo realiza a função de roteamento, através do transporte de mensagens entre redes, decidindo a melhor rota que uma mensagem deve seguir, através da estrutura de rede para chegar ao destino.

A5. CAMADA DE TRANSPORTE

Esta camada possui dois protocolos que são:

a. UDP (*User Datagram Protocol*);

b. *TCP* (*Transmission Control Protocol*).

Esses protocolos realizam as funções de transporte de dados fim-a-fim, ou seja, considera-se apenas a origem e o destino da comunicação, sem se preocupar com os elementos intermediários.

O protocolo UDP realiza a multiplexação, a fim de que várias aplicações possam acessar o sistema de comunicação de forma coerente.

O protocolo *TCP* realiza, além da multiplexação, uma série de funções para tornar a comunicação entre origem e destino mais confiável. Ele tem as seguintes funções: o controle de fluxo, o controle de erro, a sequenciação e a multiplexação de mensagens.

A camada de transporte oferece para o nível de aplicação um conjunto de funções e procedimentos para acesso ao sistema de comunicação, de modo a permitir a criação e a utilização de aplicações de forma independente da implementação. As interfaces *socket* ou TLI (ambiente *Unix*) e Winsock (ambiente *Windows*) fornecem um conjunto de funções-padrão para permitir que as aplicações possam ser desenvolvidas independentemente do sistema operacional no qual rodarão.

A6. CAMADA DE APLICAÇÃO

Os protocolos da camada de aplicação fornecem serviços de comunicação ao sistema ou ao usuário. Pode-se separar os protocolos de aplicação em protocolos de serviços básicos e protocolos de serviços para o usuário:

1) Protocolos de serviços básicos são os que fornecem serviços para atender as próprias necessidades do sistema de comunicação *TCP/IP*. A seguir são relacionados alguns exemplos:

a. DNS (*Domain Name System*): É um esquema de gerenciamento de nomes, hierárquico e distribuído. O DNS define a sintaxe dos nomes usados na *Internet*, regras para delegação de autoridade na definição de nomes, um banco de dados distribuído que associa nomes a atributos (entre eles o endereço IP) e um algoritmo distribuído para mapear nomes em endereços;

b. DHCP (*Dynamic Host Configuration Protocol*): Este protocolo fornece configuração e endereço IP automaticamente para todos os *hosts* solicitantes.

2) Exemplos de protocolos de serviços para o usuário:

a. FTP (*File Transfer Protocol*): Controla os procedimentos de transferência de arquivos de um computador remoto para o computador local;

b. HTTP (*Hipertext Transfer Telnet Protocol*): Este protocolo é utilizado para transferência de dados (acesso a arquivos, programas e diretórios) na *World Wide Web* (www);

c. SMTP (*Simple Mail Transfer protocol*): Protocolo que envia mensagens no sistema de correio eletrônico na arquitetura *Internet TCP/IP*;

d. SNMP (*Simple Network Management Protocol*) : Este protocolo permite que uma rede baseada em *TCP-IP* troque informações administrativas como configuração e estados dos *hosts*.

A7 MODELO ISO/OSI

O modelo de camadas ISO/OSI descreve um sistema ideal de comunicação entre computadores, através de sete camadas de serviços com uma função específica. A figura A2 ilustra o modelo de camadas OSI e a correspondência com as camadas do protocolo *TCP/IP*:

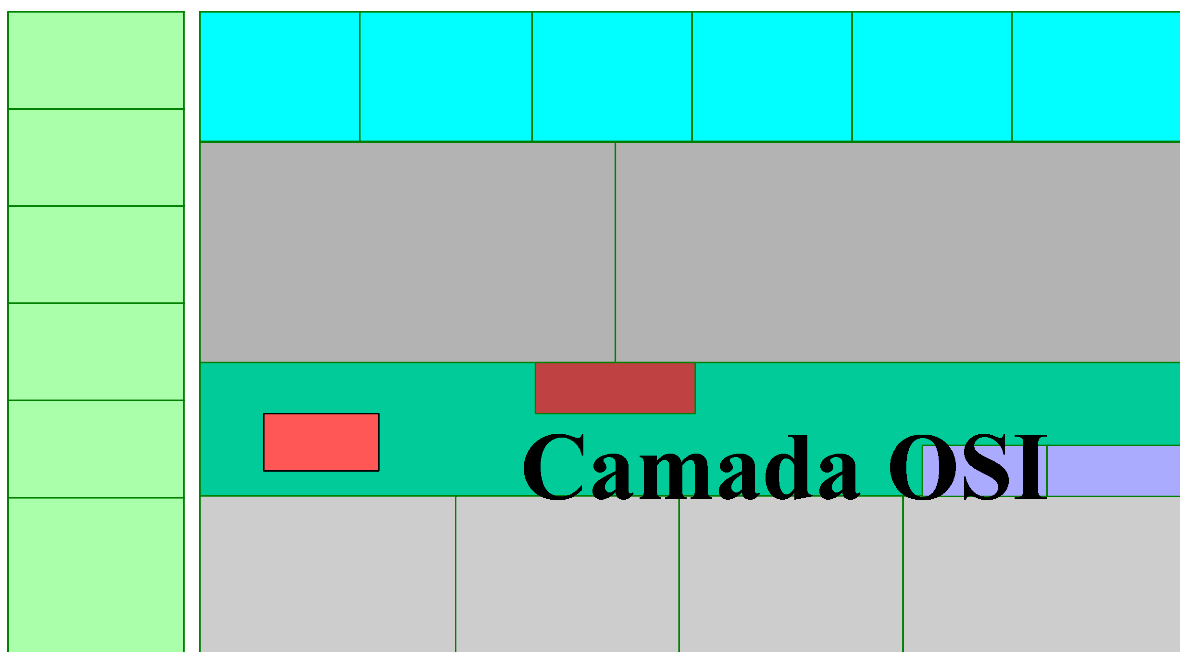


Figura A2 – Modelo de camadas OSI e protocolo *TCP/IP*.
Fonte: O autor

A8 DEPURAÇÃO DE REDES *TCP-IP*

A maioria das ferramentas de depuração geram informações de baixo nível, de forma que se torna necessário conhecer as principais idéias do *TCP/IP* e o funcionamento para poder utilizá-las. A seguir, são explanadas algumas ferramentas úteis para iniciar a verificação de comunicação da rede: *ping*, *traceroute*, *netstat* e *tcpdump* (NEMETH, 2004).

Comando *PING*: Este comando muito simples é extremamente útil, pois verifica o estado de *hosts* individuais e testa segmentos de redes.

Ele envia um pacote denominado *ECHO-REQUEST* a um *host* de destino (que pode ser ele mesmo) e espera para ver se o *host* responde. Se o *ping* não funcionar, é quase certo que nada mais sofisticado funcionará a menos que esteja sendo interferido pela utilização do

firewall (*software* de filtragem de pacotes). Para evitar este inconveniente, recomenda-se desabilitar o *firewall* durante o período da depuração.

A maioria das versões *ping* executa um *loop* infinito, a menos que seja fornecido o argumento de contagens de pacotes. Para sair, ao estar satisfeito com o resultado, basta digitar o caractere de interrupção (normalmente é o <Ctrl-C>).

Sintaxe: ***ping*** *nome do host*

% *ping localhost*

Exemplo:

O exemplo demonstra que o servidor 10.1.1.4 está funcionando e conectado à rede com as seguintes informações:

a) endereço IP (*Internet Protocol, Protocolo Internet*) do host: 10.1.1.4;

b) número de seqüência ICMP (*Internet Control Message protocol*, Protocolo de Mensagens de Controle *Internet*). Se houver descontinuidade na sequência ICMP, indica pacotes perdidos. Conforme já visto, embora o IP não garanta uma entrega de pacotes, uma rede saudável deve perder poucos dados. Para descobrir a causa

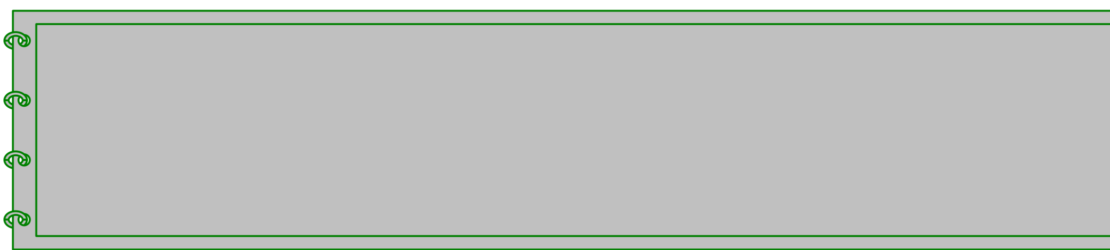
do desaparecimento de pacotes, deve ser executado o comando *TRACEROUTE* (descrito no item seguinte);

c) tempo de viagem de ida e volta: este tempo dá uma visão do desempenho global de uma rota através de uma rede. O tempo de verificação dos pacotes trafegarem na rede, podem variar cerca de décimos ou centésimos de milisegundos, sem demonstrar problemas na rede. Essa é a forma de funcionamento do IP e do LINUX. O comando *ping* não distingue uma falha de rede de uma falha de um servidor, apenas indica que algo está errado e a resposta garante somente que uma máquina está ligada;

d) quando uma rede não tem problemas, o *ping* revela se um *host* está ativo ou não. Mas, quando se sabe que o *host* remoto está ativo e funcionando, o *ping* dá informações importantes sobre a viagem de ida e volta bem sucedida, significando que todas as redes que se encontram entre a fonte e o destino estão funcionando bem, pelo menos numa primeira abordagem.

Comando *TRACEROUTE*: Este comando descobre a seqüência de *gateways* (roteadores de alta capacidade), através da qual um pacote IP, viaja para alcançar o seu destino.

Sintaxe: ***traceroute*** nome *do host* (pode ser especificado de forma simbólica ou numérica)



Este comando mostra quais *gateways* estão envolvidos na conexão e o tempo de viagem de ida e volta para cada *gateway*.

Comando *NETSTAT*: Este comando fornece várias informações sobre o estado do software de rede como: estado de conexão de rede (através do comando com a seguinte opção: *netstat* –a), estatística de interface (*netstat* –s), informações de roteamento e tabelas de conexão (*netstat* –r) e informações de configuração de interface (*netstat* –i).

Comando *TCPDUMP*: Este comando, muito utilizado como analisador de rede, escuta a primeira interface de rede que ele encontra. O *tcpdump* imprime a saída dos cabeçalhos dos pacotes na interface de rede conforme opção desejada (NEMETH, 2004).

B. INFORMAÇÕES COMPLEMENTARES DO BARRAMENTO *CAN*

As informações foram baseadas na especificação do *CAN* versão 2.0 (BOSCH, 1991).

B1. TIPOS DE MENSAGENS NO BARRAMENTO *CAN*

Os tipos de mensagens que trafegam no barramento *CAN* levando as informações entre os nós são:

a) mensagem de Dados;

b) mensagem Remota;

c) mensagem de Erro;

d) mensagem de Sobrecarga.

B1.1 Mensagem de Dados

Existem dois formatos de quadros de mensagens de Dados no protocolo *CAN*:

a) ***CAN* 2.0A** – A especificação 2.0A trata as mensagens com identificador de 11 *bits*, que permite ter até 2048 mensagens numa rede sob este formato, podendo caracterizar uma limitação em determinadas aplicações;

b) ***CAN* 2.0B** – A especificação 2.0B designa as mensagens com identificador de 11 *bits* de quadro padrão e identificador de 29 *bits* de quadros estendidos. Uma rede constituída de formato de quadros estendidos pode ter aproximadamente 537 milhões de mensagens. Neste formato, não há limites em quantidade de

mensagens, mas o que pode ser observado em alguns casos é que os 18 *bits* adicionais, no identificador, aumentam o tempo de transmissão de cada mensagem, o que pode caracterizar um problema em determinadas aplicações que trabalhem em tempo-real (problema conhecido como *overhead*).

A mensagem de dados é composta por sete campos para ambos os formatos, conforme a figura B1:

Início da Mensagem SOF	Campo de Arbitragem	Campo de Controle	Campo de Dados	Campo do CRC	Campo de ACK	Fim da Mensagem

Figura B1 - Campos das Mensagens de Dados.
Fonte: O autor

1. **Início da Mensagem (SOF - *Start Of Frame*):** Indica o início da mensagem, compostas por um único *bit* dominante (nível lógico ‘0’);

2. **Campo de Arbitragem:** Este campo é constituído pelos *bits* identificadores da mensagem. Conforme já mencionado, há 2 formatos: 11 *bits* (*CAN* 2.0A) e 29 *bits* (*CAN* 2.0B);

A figura B2, mostra o campo de Arbitragem do formato *CAN* 2.0A:

11 Bits Identificadores	RTR

Figura B2 - Campo de Arbitragem do Formato *CAN* 2.0A.
Fonte: O autor

Neste campo, tem-se, além dos 11 *bits* identificadores o *bit* RTR:

a) ***bit* RTR (*Remote Transmission Request*):** Este *bit* informa se a mensagem é de dados ou remota. Nas mensagens de dados, este *bit* é “dominante”(nível lógico ‘0’) e mensagem remota “recessivo”, nível lógico ‘1’.

A figura B3, mostra o campo de Arbitragem do formato *CAN* 2.0B:

Figura B3 - Campo de Arbitragem do formato *CAN* 2.0B.
Fonte: O autor

O campo, neste padrão do formato *CAN* 2.0B – quadro estendido -, tem-se 29 *bits* identificadores, divididos em dois grupos, sendo o primeiro de 11 *bits* e o segundo de 18 *bits*. Além destes, tem-se os seguintes *bits*:

a) ***bit* SRR (*Substitute Remote Request*):** Este *bit* é recessivo (nível lógico ‘1’) e substitui o *bit* RTR do formato 2.0A;

b) ***bit* IDE (*Identifier Extension*):** Este *bit* é recessivo (nível lógico ‘1’) e identifica que a mensagem é de formato de quadro estendido;

c) ***bit* RTR (*Remote Transmission Request*):** Este *bit* informa se a mensagem é de dados ou remota como no formato 2.0A. Nas mensagens de dados, este *bit* é dominante (nível lógico ‘0’) e mensagem remota recessivo (nível lógico ‘1’).

3. **Campo de Controle:** Este campo é constituído por seis *bit*s, informando o número de bytes da mensagem. O campo de Controle do formato *CAN* 2.A e *CAN* 2.B estendido é ilustrado na figura B4:

Figura B4 - Campo de Controle *CAN* 2.A e *CAN* 2.B – quadro estendido.
Fonte: O autor

a) ***bit* r0 e r1:** São *bits* de reserva que devem ser dominantes (nível lógico ‘0’);

b) ***bits* DLC3...0 (Data Lenght Code):** Conjunto de *bits* que informa o número de bytes da mensagem, conforme a tabela B1:

Tabela B1 - Codificação do número de bytes de mensagem.

Número de Bytes	DLC3	DLC2	DLC1	DLC0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0

Fonte: O autor

O campo de Controle do formato *CAN*2.B - quadro padrão é ilustrado na figura B5:

Figura B5 - Campo de Controle *CAN* 2.B – quadro padrão.
Fonte: O autor

a) **IDE (*Identifier Extension*):** Este *bit* é dominante (nível lógico ‘0’) e identifica que a mensagem é de formato de quadro padrão;

b) ***bit* r0:** É um *bit* de reserva que deve ser dominante (nível lógico ‘0’);

c) ***bits* DLC3...0 (*Data Lenght Code*):** Conjunto de *bits* que informa o número de bytes da mensagem, conforme a tabela B1.

4. **Campo de Dados:** São os dados a serem transmitidos de zero até oito bytes.

A ordem de transmissão geralmente é crescente: byte0, byte1, byte2, byte3, byte4, byte5, byte6 e byte7.

5. **Campo do CRC (*Cyclic Redundancy Ciclic*):** Este campo é utilizado para verificar a integridade da mensagem. A figura B6 ilustra o formato do mesmo:

Seqüência CRC															Delimitador CRC
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

Figura B6 - Campo do CRC.
Fonte: O autor

a) **Seqüência CRC:** É formada por 15 *bits* com a função de armazenar o código de redundância cíclica, o qual se pode verificar a integridade da mensagem. Este valor é comparado a cada receptor e havendo qualquer diferença é gerada uma mensagem de erro. O cálculo da seqüência CRC é feito tendo em conta uma polinomial geradora;

b) **Delimitador CRC:** Este *bit* indica o fim deste campo, através de um *bit* recessivo (nível lógico ‘1’).

6. **Campo de ACK (*Acknowledge*):** Este campo é utilizado pelos nós receptores para indicar que uma mensagem foi recebida. É composto por dois *bits*, conforme a figura B7:

ACK	Delimitador ACK

Figura B7 - Campo de ACK.
Fonte: O autor

a) **ACK:** Este *bit* é enviado recessivo (nível lógico '1') pelo transmissor. Os nós, que receberem corretamente a seqüência CRC, enviam o reconhecimento, alterando o *bit* para dominante (nível lógico '0');

b) **Delimitador ACK:** Este *bit* indica o fim do campo ACK, através de um *bit* recessivo (nível lógico '1').

7. **Fim da Mensagem:** Este campo é utilizado para indicar o final de mensagem, e é composto de 7 *bits* recessivos (nível lógico '1').

B.1.2 Mensagem Remota

É utilizada quando um nó receptor de dados solicita a transmissão de uma mensagem de dados com o mesmo identificador. Ela é constituída por seis campos, conforme a figura B8:

Início da Mensagem SOF	Campo de Arbitragem	Campo de Controle	Campo de CRC	Campo de ACK	Fim da Mensagem

Figura B8 - Campos da Mensagem Remota.
Fonte: O autor

Os campos que constituem uma mensagem remota são idênticos aos de dados, exceto quando existem campos de dados e o *bit* RTR (*Remote Transmission Request*) que indica que a mensagem é remota é recessivo (nível lógico '1').

B.1.3 Mensagem de Erro

Uma mensagem de erro, cujo formato está ilustrado na figura B9, é transmitida por qualquer nó, quando é detectado um erro no barramento, e é constituída por dois campos distintos:

Superposição do *Flag* de Erro		**Delimitador de Erro**
***Flag* de Erro**		

Figura B9 – Mensagem de Erro.
Fonte: O autor

a) ***Flag* de Erro:** São seis *bits* consecutivos, sendo chamados *Flags* de Erro Ativo quando são todos dominantes (nível lógico '0'), e chamados *Flags* de Erro Passivo, quando todos são recessivos (nível lógico '1');

b) **Delimitador de Erro:** São oito *bits* consecutivos recessivos (nível lógico '1') que indicam o final da mensagem;

c) Um nó "ativo", ao detectar uma condição de erro, sinaliza-o, através da transmissão dos *Flags* de Erro Ativo. O formato destes *flags* viola a lei da inserção de *bits* e, então, os demais nós identificam um erro de "*bit stuff*", pois seis *bits* consecutivos de mesmo nível lógico acabam por provocar este tipo de erro nos demais nós. Então, no segundo intervalo, os nós que identificaram o erro tentam enviar os *Flags* de Erro Passivo, podendo assim esta mensagem completar os doze *bits*.

B.1.4 Mensagem de Sobrecarga

Quando um receptor requer um atraso da próxima mensagem de dados, permite o processamento da informação pelos nós sobrecarregados. O formato da mensagem de sobrecarga é igual ao da mensagem de erro, conforme figura B10:

Superposição do *Flag* de Sobrecarga		**Delimitador de Sobrecarga**
***Flag* de Sobrecarga**		

Figura B10 - Mensagem de Sobrecarga.
Fonte: O autor

a) ***Flag* de Sobrecarga:** São seis *bits* consecutivos, sendo chamados *Flags* de Sobrecarga que são todos dominantes (nível lógico ‘0’) e seguidos de até seis *bits* dominantes (nível lógico ‘0’) de outros nós;

b) **Delimitador de Erro:** São oito *bits* consecutivos recessivos (nível lógico ‘1’) que indicam o final da mensagem.

B2 CONTROLE DE ERRO NO BARRAMENTO *CAN*

B2.1 Erros ao Nível da Mensagem

Existem três tipos de erros possíveis:

a) **Erro de Redundância Cíclica (CRC - *Cyclic Redundancy Check*):** Funciona como um checksum. O módulo transmissor calcula um valor, em função dos *bits* da mensagem e o transmite juntamente com ela. Os módulos receptores recalculam

este CRC e verificam se este é igual ao transmitido com a mensagem. Se não forem coincidentes, é assinalado um erro de CRC;

b) **Erro de Formato:** Os módulos receptores analisam o conteúdo de alguns *bits* da mensagem recebida. Estes *bits* (seus valores) não mudam de mensagem para mensagem e são determinados pelo padrão *CAN;*

c) **Erro na checagem de Reconhecimento:** Os módulos receptores respondem a cada mensagem íntegra recebida, escrevendo um *bit* dominante no campo ACK de uma mensagem resposta que é enviada ao módulo transmissor. Caso esta mensagem resposta não seja recebida (pelo transmissor original da mensagem), significará que, ou a mensagem de dados transmitida estava corrompida, ou nenhum módulo a recebeu. Toda e qualquer falha acima mencionada, quando detectada por um ou mais módulos receptores, fará com que estes coloquem uma mensagem de erro no barramento, avisando toda a rede de que aquela mensagem continha um erro e que o transmissor deverá reenviá-la.

B2.2 Erros ao Nível do *BIT*

Existem dois tipos de erros possíveis:

a) **Erro de Monitoramento:** Após a escrita de um *bit* dominante, o módulo transmissor verifica o estado do barramento. Se o *bit* lido for recessivo, significa que existe um erro no barramento;

b) **Erro de *Bit*:** Apenas cinco *bits* consecutivos podem ter o mesmo valor (dominante ou recessivo). Caso seja necessário transmitir seqüencialmente seis ou mais *bits* de

mesmo valor, o módulo transmissor inserirá, imediatamente após cada grupo de cinco *bits* consecutivos iguais, um *bit* de valor contrário. O módulo receptor ficará encarregado de, durante a leitura, retirar este *bit*, chamado de *Bit Stuff*. Caso uma mensagem seja recebida com pelo menos seis *bits* consecutivos iguais, algo de errado terá ocorrido no barramento.

Apesar dos mecanismos de detecção de erros descritos serem eficientes, pode ocorrer de algum nó estar com problemas e levar ao bloqueio do funcionamento da rede. Por isso, existem, ainda, mecanismos não detalhados nesta dissertação, que, através de avaliações estatísticas da situação dos nós de toda rede, permitem distinguir entre erros esporádicos e erros permanentes a localização dos nós com problemas.

C. PROTOCOLOS DE GERENCIAMENTO DE REDES

Os protocolos para gerenciamento de redes permitem, de uma forma padronizada, verificar um dispositivo para descobrir sua configuração, estado e conexão de rede.

O protocolo para gerenciamento mais utilizado com o *TCP/IP* (ARNETT, 1995) é o SNMP (*Simple Network Management Protocol*) (STALLINGS, 1999).

O SNMP define um espaço de nomes hierárquicos de dados de administração e uma maneira de ler e gravar os dados em cada um dos nós (NEMETH, 2004). A complexidade do protocolo reside acima da camada do protocolo nas convenções para construção do espaço de nomes e formatação dos itens de dados dentro de um nó. O SNMP precisa de um programa servidor cliente ("gerente") para fazer uso dele. Embora a nomenclatura seja contrária a intuição, tem-se a seguinte função:

a) SERVIDOR SNMP (agente): Representa aquilo que se está gerenciando e tem como funções principais: responder às solicitações enviadas pelo gerente e enviar automaticamente informações de gerenciamento ao gerente, quando previamente programado;

b) CLIENTE SNMP (gerente): Representa a estação de gerenciamento que permite o envio e a obtenção de informações de gerenciamento junto aos agentes.

C1. ORGANIZAÇÃO SNMP

Os recursos que são gerenciados são denominados objetos, e a coleção destes objetos é denominada *MIB* (*Management Information Base*). Estes arquivos de texto estruturado descrevem os dados que podem ser acessados via SNMP.

As *MIBs* contêm descrições de variáveis de dados específicos, referenciadas através de nomes conhecidos como OIDs (*Object Identifiers*).

A *MIB* SNMP básico para *TCP/IP* (*MIB*-I) define o acesso a dados de gerenciamento comuns como informações sobre o sistema, suas interfaces, etc.

Atualmente tem-se a *MIB*-II (definido pela RFC 1213) que, além de ser uma revisão da *MIB* I, ela se apresenta mais completa, fornecendo informações sobre um determinado equipamento gerenciado.

A *MIB* é organizada em forma de diretório de arquivos de disco em uma estrutura de árvore. Porém, existe um ponto (.) como caracter separador e cada nó recebe um número em vez de um nome. Para facilidade de referência, os nós também recebem nomes textuais similarmente à associação de nomes de *hosts* e endereços IP. Exemplo: O OID que se refere ao tempo em que o sistema fica ativo é: 1.3.6.1.2.1.1.3 (nome textual: *iso.org.dod.internet.mgmt.mib-2.system.sysUpTime*) (NEMETH, 2004). Se a forma numérica do identificador terminar com zero, significa que o objeto é a única instância existente.

A árvore hierárquica da figura C1 definida pela ISO representa a estrutura lógica da *MIB*, que indica o identificador numérico e o nome de cada objeto.

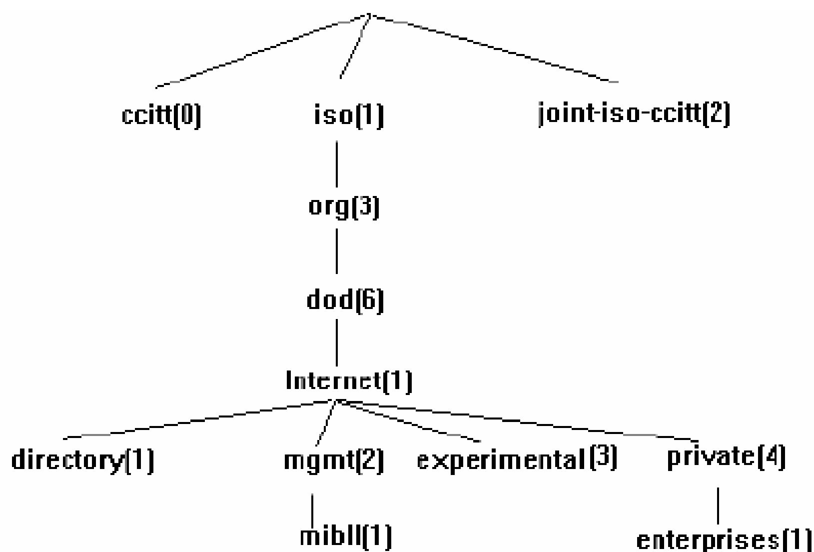


Figura C1 - Estrutura Lógica da *MIB*.
Fonte: O autor

O nó raiz da árvore possui três subníveis que são: o nó 0 que é administrado pela CCITT (*Consultative Committe for International Telegraph and Telephone*), o nó 1 que é administrado pela ISO (*International Organization for Standartization*) e o nó 2 que é administrado em conjunto pela CCITT e pela ISO.

Sob o nó da ISO, fica o nó que pode ser utilizado pelas outras instituições como o org (3). Abaixo dele, fica o dod (6) que pertence ao departamento de defesa dos EUA. Sob este nó, fica a comunidade *Internet*, e abaixo deste nó, tem-se entre outros, os nós: *management* (2), *experimental* (3) e *private* (4).

Sob o nó do *management* (2), ficam as informações de gerenciamento. É sob este nó, que está o nó da *MIB* II (1).

Sob o nó *experimental* (3), estão as *MIBs* experimentais como a *MIB* que foi testada neste projeto (TRIND-MIB.txt).

Os tipos de dados básicos das variáveis SNMP das *MIBs* são definidos pelo padrão ASN.1 (*Abstract Syntax Notation One*) (ANS.1, 2005): inteiros, *strings* e nulo. Estes tipos podem ser combinados para comporem outros tipos de dados como: *OBJECT IDENTIFIER* (Identificador de Objeto – uma *string* de números derivados de uma árvore nomeada), *ENUMERATED* (enumerado - um conjunto de limitado de inteiros com significado associado), *BOOLEAN* (boleana - um inteiro com valores true (1) ou *false* (0), etc.

C2 O PACOTE NET-SNMP

A distribuição NET_SNMP é uma implementação gratuita autorizada para o LINUX que inclui um agente SNMP, algumas ferramentas em linhas de comando e uma biblioteca para desenvolvimento de aplicações que entendem SNMP.

O gerente envia comandos aos agentes, solicitando informações de variáveis de um objeto gerenciado ou modificando o valor de uma determinada variável.

O agente busca informações sobre o *host* local e o serve aos gerentes SNMP que repassa à aplicação que as solicitou.

Para se ter uma instalação padrão, é necessário ter memória, disco, processos, CPU e *MIBs* para realizar estatísticas de interfaces de rede. Utilizando o recurso do agente de executar um comando *LINUX* qualquer e retornar a saída gerada pelo comando na forma de uma resposta SNMP, pode-se monitorar qualquer variável em seu sistema com SNMP.

C3 OPERAÇÃO DE PROTOCOLOS SNMP

Há quatro operações SNMP básicas: *get*, *get-next*, *set* e *trap*. O detalhamento destes comandos e de outros será mostrado na tabela C1.

a) *Get* e *Set* são operações básicas para leitura e gravação de uma hierarquia *MIB*, assim como ler e escrever o conteúdo de tabelas;

b) *Trap* é uma notificação assíncrona do servidor (agente) e não solicitada pelo cliente (gerente) que informa a ocorrência de um evento como falha ou a recuperação de um *link* de rede ou mesmo problemas de roteamento e de autenticação.

C4 FERRAMENTAS NET-SNMP

Os comandos listados na tabela C1 podem ser instalados e compilados do lado do cliente do pacote NET-SNMP (NEMETH, 2004).

Tabela C1 – Lista de linha de comando no pacote NET-SNMP.

COMANDO	FUNÇÃO
Snmpdelta	Monitora, ao longo do tempo, alterações em variáveis SNMP
Snmpdf	Monitora o espaço em disco num host remoto via SNMP
Snmpget	Obtém o valor de uma variável SNMP por meio de um agente
Snmpgetnext	Obtém a próxima variável na seqüência
Snmpset	Configura uma variável SNMP num agente
Snmptable	Obtém uma tabela das variáveis SNMP
Snmptranslate	Pesquisa e descreve OIDs na hierarquia MIB
Snmptrap	Gera um alerta de captura
Snmpwalk	Percorre um MIB iniciando num dado OID

Fonte: O autor

D. LISTAGEM DE CÓDIGOS DO ARQUIVO DE PROGRAMAS SCRIPTS

D1. LISTA DO ARQUIVO TRIND.C GERADO PELO COMANDO MIB2C

trind.c (Auto Gerado pelo MIB2C)

```
/*
* Note: this file originally auto-generated by mib2c using
* : mib2c.int_watch.conf,v 1.3 2005/05/03 14:38:11 dts12 Exp $
*/

#include <net-snmp/net-snmp-config.h>
#include <net-snmp/net-snmp-includes.h>
#include <net-snmp/agent/net-snmp-agent-includes.h>
#include "trind.h"

/*
* The variables we want to tie the relevant OIDs to.
* The agent will handle all GET and (if applicable) SET requests
* to these variables automatically, changing the values as needed.
*/

long sensor1 = 0; /* XXX: set default value */
long sensor2 = 0; /* XXX: set default value */
long atuador1 = 0; /* XXX: set default value */
long atuador2 = 0; /* XXX: set default value */
long alarme1 = 0; /* XXX: set default value */
long alarme2 = 0; /* XXX: set default value */

/*
* Our initialization routine, called automatically by the agent
* (Note that the function name must match init_FILENAME())
*/
```

trind.c (Auto Gerado pelo MIB2C) (cont.)

```
void
init_trind(void)
{
netsnmp_handler_registration *reg;
netsnmp_watcher_info *winfo;

static oid sensor1_oid[] = { 1,3,6,1,3,20,1 };
static oid sensor2_oid[] = { 1,3,6,1,3,20,2 };
static oid atuador1_oid[] = { 1,3,6,1,3,20,3 };
static oid atuador2_oid[] = { 1,3,6,1,3,20,4 };
static oid alarme1_oid[] = { 1,3,6,1,3,20,5 };
static oid alarme2_oid[] = { 1,3,6,1,3,20,6 };

/*
* a debugging statement. Run the agent with -Dtrind to see
* the output of this debugging statement.
*/
DEBUGMSGTL(("trind", "Initializing the trind module\n"));


/*
* Register scalar watchers for each of the MIB objects.
* The ASN type and RO/RW status are taken from the MIB
definition,
* but can be adjusted if needed.
*
```

trind.c (Auto Gerado pelo MIB2C) (cont.)

```
*
* In most circumstances, the scalar watcher will handle all
* of the necessary processing. But the NULL parameter in the
* netsnmp_create_handler_registration() call can be used to
* supply a user-provided handler if necessary.
*
* This approach can also be used to handle Counter64, string-
* and OID-based watched scalars (although variable-sized
writeable
* objects will need some more specialised initialisation).
*/
DEBUGMSGTL(("trind",
"Initializing sensor1 scalar integer. Default value = %d\n",
sensor1));
reg = netsnmp_create_handler_registration(
"sensor1", NULL,
sensor1_oid, OID_LENGTH(sensor1_oid),
HANDLER_CAN_RONLY);
winfo = netsnmp_create_watcher_info(
&sensor1, sizeof(long),
ASN_INTEGER, WATCHER_FIXED_SIZE);
if (netsnmp_register_watched_scalar( reg, winfo ) < 0 ) {
snmp_log( LOG_ERROR, "Failed to register watched sensor1" );
}
```

trind.c (Auto Gerado pelo MIB2C) (cont.)

```
DEBUGMSGTL(("trind",
"Initializing sensor2 scalar integer. Default value = %d\n",
sensor2));
reg = netsnmp_create_handler_registration(
"sensor2", NULL,
sensor2_oid, OID_LENGTH(sensor2_oid),
HANDLER_CAN_RONLY);
winfo = netsnmp_create_watcher_info(
&sensor2, sizeof(long),
ASN_INTEGER, WATCHER_FIXED_SIZE);
if (netsnmp_register_watched_scalar( reg, winfo ) < 0 ) {
snmp_log( LOG_ERROR, "Failed to register watched sensor2" );
}

DEBUGMSGTL(("trind",
"Initializing atuador1 scalar integer. Default value = %d\n",
atuador1));
reg = netsnmp_create_handler_registration(
"atuador1", NULL,
atuador1_oid, OID_LENGTH(atuador1_oid),
HANDLER_CAN_RWRITE);
winfo = netsnmp_create_watcher_info(
&atuador1, sizeof(long),
ASN_INTEGER, WATCHER_FIXED_SIZE);
if (netsnmp_register_watched_scalar( reg, winfo ) < 0 ) {
snmp_log( LOG_ERROR, "Failed to register watched atuador1"
);
}
```

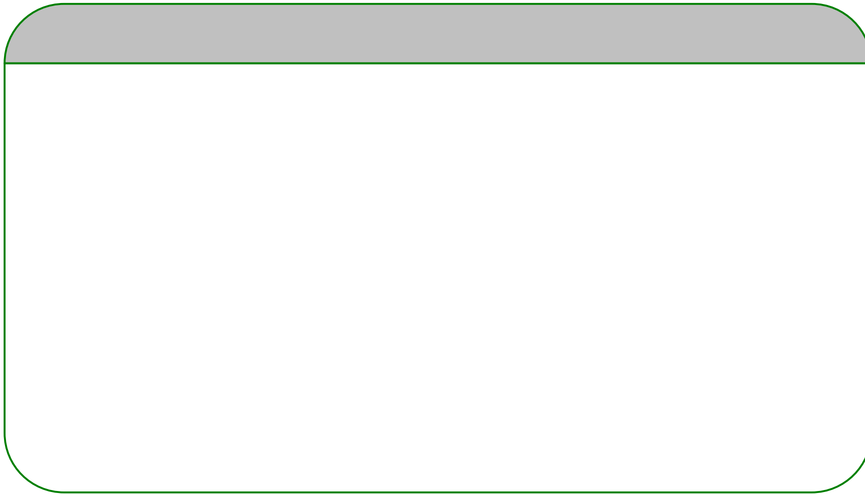
trind.c (Auto Gerado pelo MIB2C) (cont.)

```
DEBUGMSGTL(("trind",
"Initializing atuador2 scalar integer. Default value = %d\n",
atuador2));
reg = netsnmp_create_handler_registration(
"atuador2", NULL,
atuador2_oid, OID_LENGTH(atuador2_oid),
HANDLER_CAN_RWRITE);
winfo = netsnmp_create_watcher_info(
&atuador2, sizeof(long),
ASN_INTEGER, WATCHER_FIXED_SIZE);
if (netsnmp_register_watched_scalar( reg, winfo ) < 0 ) {
snmp_log( LOG_ERROR, "Failed to register watched atuador2"
);
}

DEBUGMSGTL(("trind",
"Initializing alarme1 scalar integer. Default value = %d\n",
alarme1));
reg = netsnmp_create_handler_registration(
"alarme1", NULL,
alarme1_oid, OID_LENGTH(alarme1_oid),
HANDLER_CAN_RWRITE);
winfo = netsnmp_create_watcher_info(
&alarme1, sizeof(long),
ASN_INTEGER, WATCHER_FIXED_SIZE);
if (netsnmp_register_watched_scalar( reg, winfo ) < 0 ) {
snmp_log( LOG_ERROR, "Failed to register watched alarme1" );
}
```

trind.c (Auto Gerado pelo MIB2C) (cont.)

```
DEBUGMSGTL(("trind",
"Initializing alarme2 scalar integer. Default value = %d\n",
alarme2));
reg = netsnmp_create_handler_registration(
"alarme2", NULL,
alarme2_oid, OID_LENGTH(alarme2_oid),
HANDLER_CAN_RWRITE);
winfo = netsnmp_create_watcher_info(
&alarme2, sizeof(long),
ASN_INTEGER, WATCHER_FIXED_SIZE);
if (netsnmp_register_watched_scalar( reg, winfo ) < 0 ) {
snmp_log( LOG_ERROR, "Failed to register watched alarme2" );
}


DEBUGMSGTL(("trind",
"Done initalizing trind module\n"));
}
```

D2. LISTA DO ARQUIVO TRIND.H GERADO PELO COMANDO MIB2C



trind.

```
/*
* Note: this file original
* : mib2c.int_watch.co
*/
#ifndef TRIND_H
#define TRIND_H

/* function declarations
void init_trind(void);
```

D3. LISTA DO ARQUIVO TRIND.C ALTERADO

trind.c

```
#include <net-snmp/net-snmp-config.h>
#include <net-snmp/net-snmp-includes.h>
#include <net-snmp/agent/net-snmp-agent-includes.h>

/* inclusão do nosso header */
#include "trind.h"

unsigned char *var_trind();

/*
* The variables we want to tie the relevant OIDs to.
* The agent will handle all GET and (if applicable) SET requests
* to these variables automatically, changing the values as needed.
*/

static int sensor1 = 0; /* XXX: set default value */
static int sensor2 = 0; /* XXX: set default value */
static int atuador1 = 0; /* XXX: set default value */
static int atuador2 = 0; /* XXX: set default value */
static int alarme1 = 0; /* XXX: set default value */
static int alarme2 = 0; /* XXX: set default value */

static int aux = 0; /* auxiliar para localização da entry da estrutura OID */

static oid trind_oid[] = { 1,3,6,1,3,20};

struct variable1 trind_variables1 [] =
{
#define SENSOR1 1
#define SENSOR2 2
#define ATUADOR1 3
#define ATUADOR2 4
#define ALARME1 5
#define ALARME2 6
```

trind.c (cont.)

```
{ SENSOR1, ASN_INTEGER, RONLY, var_trind, 1,{1}},
{ SENSOR2, ASN_INTEGER, RONLY, var_trind, 1,{2}},
{ ATUADOR1, ASN_INTEGER, RWRITE, var_trind, 1,{3}},
{ ATUADOR2, ASN_INTEGER, RWRITE, var_trind, 1,{4}},
{ ALARME1, ASN_INTEGER, RWRITE, var_trind, 1,{5}},
{ ALARME2, ASN_INTEGER, RWRITE, var_trind, 1,{6}},
};


/*
* Our initialization routine, called automatically by the agent
* (Note that the function name must match init_FILENAME())
*/
void
init_trind(void)
{
/*
* * Register ourselves with the agent to handle our mib tree.
* * The arguments are:
* * descr: A short description of the mib group being loaded.
* * var: The variable structure to load.
* * (the name of the variable structure defined above)
* * vartype: The type of this variable structure
* * theoid: The OID pointer this MIB is being registered underneath.
*/
REGISTER_MIB("trind", trind_variables1, variable1, trind_oid);

}
```

trind.c (cont.)

```
unsigned char *
var_trind(struct variable *vp,
oid *name,
size_t *length,
int exact,
size_t *var_len,
WriteMethod **write_method)
{
static long long_ret;


if (header_generic(vp,name,length,exact,var_len,write_method)
== MATCH_FAILED )
return NULL;

switch(vp->magic) {

case SENSOR1:
/*
aux = SENSOR1;
*write_method = write_trind;
*/
return (unsigned char *) &sensor1;

case SENSOR2:
/*
aux = SENSOR2;
*write_method = write_trind;
*/
return (unsigned char *) &sensor2;

case ATUADOR1:
aux = ATUADOR1;
*write_method = write_trind;
return (unsigned char *) &atuador1;
```

trind.c (cont.)

```
case ATUADOR2:
aux = ATUADOR2;
*write_method = write_trind;
return (unsigned char *) &atuador2;

case ALARME1:
/*
* This object is essentially "write-only".
* It only exists to trigger the sending of a trap.
* Reading it will always return 0.
*/
aux = ALARME1;
*write_method = write_trindtrap1;
return (u_char *) &alarme1;
case ALARME2:
/*
* This object is essentially "write-only".
* It only exists to trigger the sending of a trap.
* Reading it will always return 0.
*/
aux = ALARME2;
*write_method = write_trindtrap2;
return (u_char *) &alarme2;
default:
ERROR_MSG("");
}
return NULL;
}
```

trind.c (cont.)

```
int
write_trind(int action,
u_char * var_val,
u_char var_val_type,
size_t var_val_len,
u_char * statP, oid * name, size_t name_len)
{
/*
* Define an arbitrary maximum permissible value
*/
#define MAX_EXAMPLE_INT 255
static long intval;
static long old_intval;

switch (action) {
case RESERVE1:
/*
* Check that the value being set is acceptable
*/
if (var_val_type != ASN_INTEGER) {
DEBUGMSGTL(("trind", "%x not integer type", var_val_type));
return SNMP_ERR_WRONGTYPE;
}
if (var_val_len > sizeof(long)) {
DEBUGMSGTL(("trind", "wrong length %x", var_val_len));
return SNMP_ERR_WRONGLENGTH;
}
```

trind.c (cont.)

```
intval = *((long *) var_val);
if (intval > MAX_EXAMPLE_INT) {
DEBUGMSGTL(("trind", "wrong value %x", intval));
return SNMP_ERR_WRONGVALUE;
}
break;

case RESERVE2:
/*
* This is conventially where any necesary
* resources are allocated (e.g. calls to malloc)
* Here, we are using static variables
* so don't need to worry about this.
*/
break;

case FREE:
/*
* This is where any of the above resources
* are freed again (because one of the other
* values being SET failed for some reason).
* Again, since we are using static variables
* we don't need to worry about this either.
*/
break;

case ACTION:
/*
* Set the variable as requested.
* Note that this may need to be reversed,
* so save any information needed to do this.
*/
```

trind.c (cont.)

```
switch (aux)
{
case ATUADOR1:
old_intval = (long)atuador1;
atuador1 = (int)intval;
break;
case ATUADOR2:
old_intval = (long)atuador2;
atuador2 = (int)intval;
break;
}
break;
case UNDO:
/*
* Something failed, so re-set the
* variable to its previous value
* (and free any allocated resources).
*/

switch (aux)
{
case ATUADOR1:
atuador1 = (int)old_intval;
break;

case ATUADOR2:
atuador2 = (int)old_intval;
break;
}
break;
```

trind.c (cont.)

```
case COMMIT:
/*
* Everything worked, so we can discard any
* saved information, and make the change
* permanent (e.g. write to the config file).
* We also free any allocated resources.
*
* In this case, there's nothing to do.
*/
break;

}
return SNMP_ERR_NOERROR;
}


int
write_trindtrap1(int action,
u_char * var_val,
u_char var_val_type,
size_t var_val_len,
u_char * statP, oid * name, size_t name_len)
{

#define MAX_EXAMPLE_INT 255
static long intval;
static long old_intval;
```

trind.c (cont.)

```
DEBUGMSGTL(("trind", "write_exampletrap entered: action=%d\n",
action));
switch (action) {
case RESERVE1:
/*
* The only acceptable value is the integer 1
*/
if (var_val_type != ASN_INTEGER) {
DEBUGMSGTL(("trind", "%x not integer type", var_val_type));
return SNMP_ERR_WRONGTYPE;
}
if (var_val_len > sizeof(long)) {
DEBUGMSGTL(("trind", "wrong length %x", var_val_len));
return SNMP_ERR_WRONGLENGTH;
}

intval = *((long *) var_val);
if (intval != 1) {
DEBUGMSGTL(("trind", "wrong value %x", intval));
return SNMP_ERR_WRONGVALUE;
}

break;

case RESERVE2:
/*
* No resources are required....
*/
break;
```

trind.c (cont.)

```
case FREE:
/*
* ... so no resources need be freed
*/
break;

case ACTION:
/*
* Having triggered the sending of a trap,
* it would be impossible to revoke this,
* so we can't actually invoke the action here.
*/

break;

case UNDO:
/*
* We haven't done anything yet,
* so there's nothing to undo
*/
break;

case COMMIT:
/*
* Everything else worked, so it's now safe
* to trigger the trap.
* Note that this is *only* acceptable since
* the trap sending routines are "failsafe".
* (In fact, they can fail, but they return no
* indication of this, which is the next best thing!)
*/
```

trind.c (cont.)

```
DEBUGMSGTL(("trind", "write_exampletrap sending the trap\n",
action));

send_easy_trap(SNMP_TRAP_ENTERPRISESPECIFIC, 99);

DEBUGMSGTL(("trind", "write_exampletrap trap sent\n", action));
break;

}
return SNMP_ERR_NOERROR;
}

int
write_trindtrap2(int action,
u_char * var_val,
u_char var_val_type,
size_t var_val_len,
u_char * statP, oid * name, size_t name_len)
{

#define MAX_EXAMPLE_INT 255
static long intval;
static long old_intval;

DEBUGMSGTL(("trind", "write_exampletrap entered: action=%d\n",
action));
switch (action) {
case RESERVE1:
/*
* The only acceptable value is the integer 1
*/
if (var_val_type != ASN_INTEGER) {
DEBUGMSGTL(("trind", "%x not integer type", var_val_type));
return SNMP_ERR_WRONGTYPE;
}
```

trind.c (cont.)

```
if (var_val_len > sizeof(long)) {
DEBUGMSGTL(("trind", "wrong length %x", var_val_len));
return SNMP_ERR_WRONGLENGTH;
}

intval = *((long *) var_val);
if (intval != 1) {
DEBUGMSGTL(("trind", "wrong value %x", intval));
return SNMP_ERR_WRONGVALUE;
}

break;

case RESERVE2:
/*
* No resources are required....
*/
break;

case FREE:
/*
* ... so no resources need be freed
*/
break;

case ACTION:
/*
* Having triggered the sending of a trap,
* it would be impossible to revoke this,
* so we can't actually invoke the action here.
*/

break;
```

trind.c (cont.)

```
case UNDO:
/*
* We haven't done anything yet,
* so there's nothing to undo
*/
break;

case COMMIT:
/*
* Everything else worked, so it's now safe
* to trigger the trap.
* Note that this is *only* acceptable since
* the trap sending routines are "failsafe".
* (In fact, they can fail, but they return no
* indication of this, which is the next best thing!)
*/

DEBUGMSGTL(("trind", "write_exampletrap sending the trap\n",
action));

send_easy_trap(SNMP_TRAP_ENTERPRISESPECIFIC, 99);

DEBUGMSGTL(("trind", "write_exampletrap trap sent\n", action));
break;

}
return SNMP_ERR_NOERROR;
}
```

D4. LISTA DO ARQUIVO TRIND.H ALTERADO

trind.h

```
#ifndef TRIND_H
#define TRIND_H

config_add_mib(TRIND-MIB)

int
write_trind(int action,
u_char * var_val,
u_char var_val_type,
size_t var_val_len,
u_char * statP, oid * name, size_t name_len);


int
write_trindtrap1(int action,
u_char * var_val,
u_char var_val_type,
size_t var_val_len,
u_char * statP, oid * name, size_t name_len);

int
write_trindtrap2(int action,
u_char * var_val,
u_char var_val_type,
size_t var_val_len,
u_char * statP, oid * name, size_t name_len);


void
init_trind(void);

unsigned char *
var_trind(struct variable *vp,
oid *name,
size_t *length,
int exact,
size_t *var_len,
WriteMethod **write_method);


#endif /* TRIND_H */
```

E. TRABALHO APRESENTADO NO ISA-2005 EM CHICAGO - EUA

OPEN SOFTWARE APPLICATIONS THAT CAN BE USED IN INDUSTRIAL CONTROLS

Edson Watanabe
Author
Federal University of Parana
Brazil
ehwatanabe@uol.com.br

José Manoel Fernandes
Author and Supervisor
Federal University of Parana
Brazil
fernandez@ufpr.eletrica.br

KEYWORDS
Linux, GNU, CAN, PLC, Fieldbus, Open Software, Embedded.

ABSTRACT

The goal of this paper is to provide an open model for both implementation and communications for industrial automation systems. This document will overview open operating systems used in Industrial automation along with several open fieldbus networking standards. This presentation shows that an open operating system like Linux can provide portability and embedded features and allow easy licensing options for both developers and end users. Furthermore, standard open network technology and architecture focused in Fieldbus, Control Area Network (CAN) protocol and industrial bus are increasing in demand and will continue to be an easy to support for industrial automation users.

INTRODUCTION

Today, most automation system suppliers and their users are looking for the latest, more interoperable architectures, that provide high performance and highly available controls and furthermore consider security. The systems provided today must be highly flexible and able to control the entire plant, transfer information, and provide places to store history and information. Also, many industries are asking for greater flexibility to modify without shutting down, and the ability to network easily to existing and future control components. In the past, Distributed Control Systems (DCS) and Programmable Logic Controllers (PLC) would have been the only means to apply many control applications. These older systems were networked together and then an engineer would have to manually route each point via a separate dbase (time and money$$). In addition, a great amount of time would be necessary to install and set up special cables and hardware (HW) to connect both systems to work as one. By using the proposed technology of open networks, the new fieldbuses and protocols provide a common functionality that will provide an easy transportable solution to any suppliers product. The biggest advantage is that it supplies the necessities of the industry but limits the cost to the users. Consequently standardized networks reduce the installation time, training, and world-wide acceptance of the technology which can only promote installation of new networks and help in reducing maintenance and support costs.

INDUSTRIAL NETWORKS TECNOLOGY – FIELDBUS

The Fieldbus or Input/Output (I/O) bus network is a digital serial two-way communication bus that connects control and field devices. Fieldbus is an open standard bus that allows devices of different manufacturers to be integrated in only one system. The devices and data can be communicated with using standardized messages wthin the fieldbus protocol. The fieldbus devices have microprocessor-based communications capabilities that allow process variables to carry a data quality signal which can recognize errors faster. As a result, plant operators are notified of abnormal conditions or can be notified by the system to do preventive maintenance. This can provide a more efficient plant and allow operations to make better decisions. One of the main aspects in

the definition of fieldbus is the interoperability; one fieldbus device can be replaced by a similar device with added functionality from a different supplier on the same fieldbus network while maintaining specified operations. Fieldbus reduces complexity of the project and provides the needed versatility in most projects. Also, fieldbus provides easy installation and reduces physical maintenance of the system which provides costs benefits. Overall there are many open protocols which are hard to choose from. For example: CAN, Fieldbus Foundation, CEBus, WorldFIP, BitBus, ProfiBUS, Lonworks, BACNet, X10, EIB, CAB, etc... Since there is little global standardization for the automation area, it is left to the designer to analyze the several options and to find out the best one that fits his needs.

INDUSTRIAL NETWORKS ARCHITECTURE AND PROTOCOL

As the control systems become more complex, with large amount of variables, applications and interlocks, it can become expensive, complex and even cause unacceptably system performance. One would consider dividing the control into smaller parts that can individually controlled. This method of partitioning the control becomes an attractive solution since is allows simplier development, and a easier to maintain, operate and administrate with a more predictable performance. The Distributed Control System (DCS) which has been used since the 1970s in many process applications provides a data concentrator console that is connected to automation controllers that connect with remote I/O modules. These remote I/O module are hardwired to the devices in the field and connected between the controller with a proprietary communications networks that are still being used today. With the concept of Fieldbus in the market, the distributed control concept can be used in conjunction with intelligent devices. These fieldbus devices with processing capability, allow the devices in the fieldbus network to talk to one another, apply system control, and also transfer data to an operator HMI (Human Machine Interface) console. By using these fieldbus devices, the control is moved away from the controller and into the actual field devices which is even more distributed. These devices do not use the old analog signals of 4-20 mA and these fieldbus networks only pass through given the digital format, depending upon the protocol that is being used. The necessity of the HMI and controllers is for supervision and tuning of the system, and since control is managed by the instruments, the controllers will do less control.

GENERAL CHARACTERISTICS OF THE CAN (CONTROLLER AREA NETWORK) PROTOCOL

The CAN was developed by Robert Bosch in 1986 for application in the automobile industry with the intent to simplify the complex wire systems in vehicles. While simplifying wiring systems, multiple microcontrollers are coordinated to manage the engine control the braking systems, the suspension systems, etc. The CAN is a synchronous serial communications protocol that supports distributed real-time control that gives the following features: a high level of security, a high speed of transmission rate, a great immunity to the electric interferences and cost-effective data bus for multi-master and real-time applications. In addition to automotive applications, the CAN protocol is suitable to be used as general data bus for industrial process control applications. Most older connection techniques of point-to-point that is widely accepted in most industrial control systems does not provide easy sharing and multiplexing of data on a common bus like CAN. CAN is totally interoperable and is a fully documented standard protocol where information is transferred between devices and is subject to international standards approved by International Standard Organization (ISO). With the approval of the ISO, CAN has been adopted by the automobile industry as well as many other types of industries, mainly due to its robustness and flexibility. The availability of low cost pre-packaged communication modules that are sold by many manufacturers encourages utilization of CAN. It is a system with capabilities of multi-master based communications, which means that when the communications bus is free any node may commit access to the network. The unit with the message of higher priority to be transmitted gains bus access. This protocol also provides the ability to transmit one message simultaneously to several receivers. This method of data transfer is called multicasting messages. Another strong point of this protocol is the fact of being based on the concept Carrier Sense Multiple Access/Collision Detection with Non-Destructive Arbitration (CSMA/CD with NDA). This means that all nodes verify the state of the bus, each node analyzes if another node is or not sending messages with bigger priority. In case that this is perceived, the module whose messages have lower priority will cease its transmission until the messages with higher priority are sent. This protocol has another note-worthy function which is called Non Return to Zero (NRZ), where each bit (0 or 1) is transmitted by a value of specific and constant voltage during its duration. This method presents a low spectral density, making it possible to utilization the entire width of the transmission band. The speed of transmission of the data is inversely

proportional to the length of the communications bus. The largest acceptable degradation of CAN signal is experienced when the transmission is at 1Mbps and the length is at 40 meters. In CAN networks, the addressing of the destination addresses of the conventional system does not exist, nevertheless, messages are transmitted by identification. Thus, a sender sends a message to all nodes on the CAN network and each one decides by its identification, if it should process the message. The identification also determines the priority of the message when competing with others for the access to the bus. The CAN protocol makes it possible to connect the net intelligent subsystems, such as sensory and actuators, where the transmitted information has a size of maximum 8 bytes, being however possible to transmit blocks data bigger through the use of segmentation. The number of elements in a CAN system is theoretically limited for the possible number of different identifications. This number has limitations that is significantly reduced for physical limitations of the hardware. The CAN allows for flexible configuration that can add new nodes onto a CAN network without requiring alterations to the software (SW) or the hardware and it allows the new node to be receiver or to be the transmitter of the data. In order to achieve the assurance that the data transfer takes place, powerful measures for error detection, signalling and self-checking are implemented in every CAN node.

OPENS SYSTEMS

The CAN was developed to provide, among others similar protocols, interoperability. So it is a perfect match for Open Systems that are used in many industrial systems.The big advantage of defining an Open architecture would be enabling the people to develop sub systems and later merge them together to form a commissionable system with little integration. The idea behind this development effort is that with more diversified individuals using CAN would allow better versatility that is better tested than a similar proprietary solution that is not used as much as a widely accepted standard such as CAN. The benefit of open architectures is the overall attention of diversified users that are focused on different application context, which can lead to a wider and deeper test bench. As an alternative to the private companies that provide a non-standard solution to the market we hope that this proposal might migrate venders and users into considering a solution built onto open architectures. There is, nowadays, an overwhelming claim for that approach into several different market segments and the time will come that the market will demand this more powerful and reliable approach. Some of advantages that come up into such analysis are:

- Users of an entirely open systems are already testifying the quality of such systems;
- This technology lowers the costs associated to licensing the software yet reduces the costs of the services and the maintainability of the end user;
- improvements to the systems are possible into systems that beyond open systems are also open source;
- more people get involved, more solutions comes up with some smart solutions gathered throughout the whole architecture.

Beyond interoperability, there are also other very important characteristics for an OpenOS that brings usefullness in industrial automation systems, they are:

- **Interoperability**: see above;
- **Modularity:** it allows solutions start from very simple one and be increased as necessary as the plant becomes bigger;
- **Scalability:** there is Open Operating System (OOS) sistems deployed in appliances as small as a RJ-45 connector as big as application and network servers gathering up to 512 processors into a single rack;
- **Portability:** some OOS are structured in such a way to make it easy to be ported into very different alectrical architectures from different families and even different chip manufactures.

For this study it was choosen a very representative of this class – Linux – as the mature stage it has reached since this creation, the overall support we can be received easily, where most brand new HW devices are released with a first drive to support it into linux kernel.

LINUX OPERATING SYSTEM

At first, it has been created, derived from Unix, to run into the personal computers at that time and since then it has experienced a grow rate very impressive. Linux provides features that are equivalent to the most up-to-date proprietary offerings of Operating System (OS). Among those, we can mention:

- memory managment – to provide virtual memory for processes running into user space;

- real time support for embedded and time-critical applications;
- multi-processing and portable operating system interface (POSIX) threads;
- inter-processing communications using files and/or pipes;
- several internal synchonization resources like semaphores, message queues and shared memory;
- sockets
- a large set of libraries and development support;
- others...

A very interesting point is: it goes from very small deployment up to big servers gathering a lot of processors together. In our case it is really important because it will integrate the very end of the CAN network at the production ground up to the big PC that will configure the overall network.

LINUX AND GNU IS NOT UNIX (GNU) – A CLOSELY MATCH

Linux has this power because of enormous work developed for Richard Stallman and a lot of people. Everything initiated when Richard started the movement the open free software in 1984, at the time that it had started to appear first softwares commercial. Richard was a researcher in the Massachussets Institute of Technology (MIT) and worked in the area of artificial intelligence. He established Free Software Foundation (FSF) and wrote a document that establishes the form under which programs of code of open sources can be distributed. This document specifies that the program can be used and modified for who wants that is, since that the effected modifications also is available in code source. This document calls Gnu Public License (GPL). A great tool and majority utility of development that are not specifically Linux, are part of project GNU. For the fact of any system kernel must contain one and at least one minimum set of utility, some people defend that the system would have to be called GNU/Linux system.

LINUX AND THE EMBEDDED REAL TIME OPEN SYSTEM SOFTWARE (OSS)

Embedded systems are those which has a specific HW and SW designed to reach a specific purpose and, because of that, has some constraints regarding the time spent between a input event has triggered and the system outputs results shows up – this is called real time. A device is considered embedded when it has a combination of HW and SW to perform a specific function and interacts continuously with the environment around it through sensors and actuators. The developer of this type of system should know beyond the HW and SW programming also about technologies of data acquisition and actuators, notion of supervision and process control and system customization. Embedded devices are found in some places such as: cellular telephones, Personal Digital Assistant (PDA), equipment of networks(ie.routers, switches), intelligent readers of smart cards, printers, video games, microwaves, refrigerators, medical systems of industrial controls, equipment and many others. The overall system for these types of devices must be sufficiently compact and efficient, implementing specific and dedicated resources to the hardware and in the layer of application.

LINUX: OVERVIEW IN PORTING IT INTO A EMBEDDED PLATFORM

Linux is getting more and more popular as it drives embedded systems like modems, routers and others. The proprietary embedded systems simply cost more, and in most of the cases require fees to allow usage for each unit. The most recent linux kernel can fit in several families of processors as well as different platforms. In order to have linux kernel running in a new platform, we may split the task into 5 different branches:

- to mount a complete Cross Development Kit (CDK) system that will enable to build the complete target system into a different host system;
- to define means by which the code get loaded into the embedded system and get the processor control at the most priviledged level – the literature call that a Boot Loader;
- to implement the architecture dependent functions, it means the Central Processor Unit (CPU) and platform procedures. This correspond to the code under arch/* and include/asm/* tree branches of the kernel source tree;

- to implement the coding for the HW devices (net, char, block, ttys) when the kernel source tree does not have that yet (just the case for brand new HW devices and/or families);
- to test the overall system build;

The development work may take a few hundred of men-hours up to several thousands of them, depending on the pre-developed code that is a available. Keep in mind that it is possible to choose HW components for which there is code available already in the literature – and this is the exact reason that a lot of companies provides the drivers for such opened OS. Currently there is suppport for the following CPU families: PowerPC, x86, Sparc, Arm, among others. The size of a complete software build for a CPU family may span a program size of few kilobytes up to gigabytes, but for a embedded system with no human interface (keyboard, display, mouse, whatever...) it may be built into 1megabytes space of program memory 4 megabytes of volatile memory and 16 kilobytes of non-volatile memory to hold the configuration.

The minimal HW support to host a linux kernel would be:

- CPU;
- non volatile memory (normally Flash memory + NVRAM);
- volatile memory (Random Access Memory (RAM));
- clocks for system and communication;
- console port;
- communication ports;

Among the main functionalities that must be ported to the target platform/processor are:

- interrupt controller;
- pre-emptive kernel functions;
- virtual memory in systems with Memory Management Unit (MMU);
- threads functions, processes functions, semaphores, atomic operations, queue managment;

PROPOSAL OF A NEW CONFIGURATION FOR INDUSTRIAL NETWORKS

The idea is to make CAN network reachable into linux network which then can be used with industrial automation. The Linux Electronic Control Unit (LECU) is an electronic device responsible for executing a specific control, made up basically of digital and analog input/output interfaces. To increase its performance, flexibility and conectivity and its actuation in the network, it is embedded into the Linux operating system. Control applications will run in the device while scanning practically all points simultaneously within the plant network, improving significantly the performance time of the control algorithms. This module has two configurations, Master and Slave, the first type to build in a total distributed network and the second to build in a traditional network connecting to the PLC equipments. The Internet Protocol (IP) is the core for the linux as messaging system, and as matter of fact, every protocol inside linux should be merged into an IP stack in order to have their packets routed and forwarded through the linux network as well as be reached through the socket layer. In order to have CAN integrating into this network, it must be defined an address family and protocol family according (AF_CAN and PF_CAN) as the key way of integrating CAN into linux. As a high level layer we plan to implement a top-level 'cand' (CAN daemon) that will create a CAN socket of AF_CAN family in order to control the CAN connections through the CAN network. To simplify such development the whole CAN network will be looked as a single whole subnet inside the linux kernel avoiding further splitting of such network into subnets. Such function will run into the LECU Master that will also implement Network Address Translation (NAT)/Masquerading to provide gateway services between the general IP networks with the LECU slave devices. It will mean that the LECU Master is a MUST into such network. The same daemon will manage the creation/destruction of CAN multicast groups, gathering all the LECU slaves that interconnect and exchange data as needed. As the matter of low level, basically the idea is to implement the CAN link layer in order to glue the IP layer of linux proceeding the appropriate Link Layer Interface.

LECU MASTER

This control module is embedded with complete Linux operating system to offer multiple resources to the scan and control applications that will run in it. It is remotely managed for supervisory software responsible for information acquisition of each node located in the industrial network. The LECU is supported for multiprotocol, CAN to change information packages between the LECU Slaves and Master and Transmission Control Protocol/ Internet Protocol (TCP/IP) for communication with the supervisory software being able to be connected through the Ethernet standard. A great advantage is its multitask monitoring speeding the scan of the monitored points updating results at a much faster rate. The Master even monitoring all the Slaves has greater functionality to detect and correct the process better than most systems sold today.

Characteristics of the module:

- Complete LINUX Operating System;
- KDE Windows Interface;
- Proportional, Integral, Derivative (PID) Application;
- Multiprotocol: CAN, TCP/IP and others;
- DeviceNet communication Interface;
- Ethernet Interface;
- Digital Input/Output Interfaces;
- Analog/Digital and Digital/Analog Converting Interfaces;
- Industrial Standard.

LECU SLAVE

This control module is embedded with compact Linux operating system to offer the specific resources to the scan and control applications that will run in it to monitor and control all points located throughout the plant. It is managed by the LECU Master or a PLC connected through a DeviceNet interface for information acquisition of each node located in the industrial network. The LECU Slave is supported for multiprotocol interfaces, CAN to change information packages between the LECU Slaves and Masters and or PLCs, and is supported with TCP/IP interface to connect with the remote system for configuration and maintenance being able to be connected through the Ethernet standard. This module is composed by hardware and software that executes the reading of the inputs, information processing and act on the outputs. The advantage to this solution is the high speed scanning of its multitask monitoring.

Characteristics of the module:

- Compact Linux Operating System;
- Multitask;
- Multiprotocol: CAN, TCP/IP and others;
- DeviceNet communication Interface;
- Interface of Ethernet network;
- Digital Input and Output;
- Analog Input and Output;
- Industrial Standard.

PLC IMPROVED WITH LINUX OPERATING SYSTEM

Keeping all the known features of the PLC industrial equipment, the porting of LINUX operating system will be a plus to improve its capability to scan and act on the monitored pointes. Linux will give to the PLC all open system characteristics: interoperability, modurability, scalability and portability and will increase its performance in executing its tasks.

Aggregated functions:

- Security in network;
- Multitask;
- Multiuser;

- Multiprotocol Communication Interfaces;

SYSTEM CONFIGURATION

The first configuration presents a traditional dedicated industrial network with a central logic controller, supervisory system and intelligent devices, all these network elements ported with Linux operating system. The main characteristic in this configuration is the improvement offered by the open systems philosophy to some commercially available industrial equipment as a PLC and the facility to create and manage the system. Reduction of the information exchange time between all the elements of the network and its controllers.

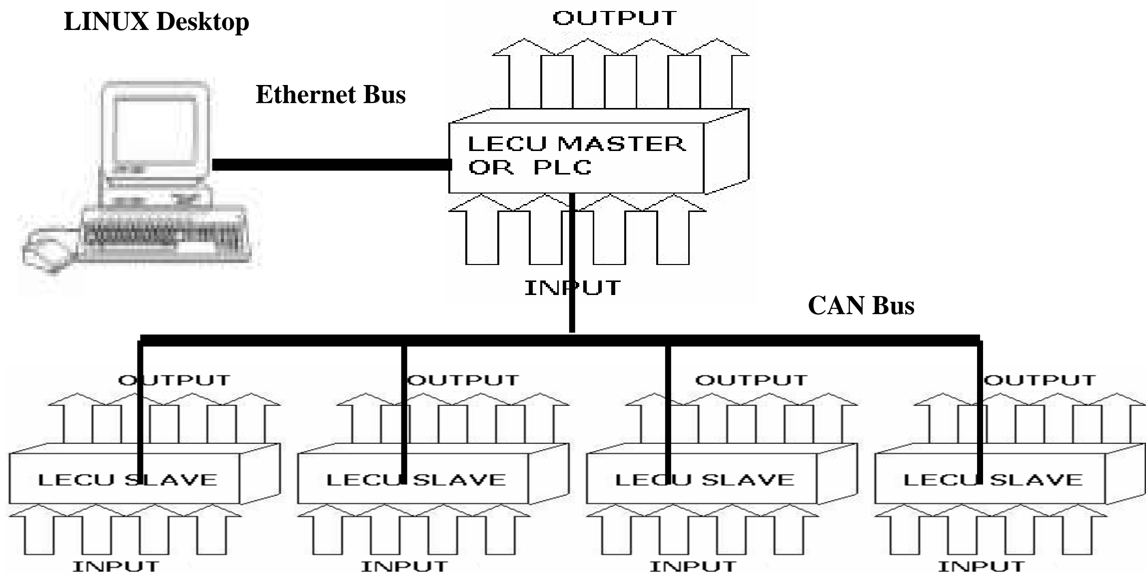


Fig. 1 – DEDICATED CONFIGURATION

The second configuration presented here deals with an complete new architecture based in opened components, in desktop run a management system based on LINUX environment with windows and menus to facilitate the navigation for the elements on the network, visualization of states, alarms monitoring and reports generation. The LECU Master communicates with the manager through TCP/IP protocol and the LECU Slave through the CAN protocol. This configuration provides many advantages, therefore it inherits all the properties of an opened system as well as of the CAN field bus and of all LINUX operating system network and functional characteristics.

Below is related some important topics:

- Aquisition data time reducing between all elements of the network due the characteristic multitask of the Linux;
- Reduced cost due to a lowered installation time while using CAN Bus between LECU Master and LECU Slave.

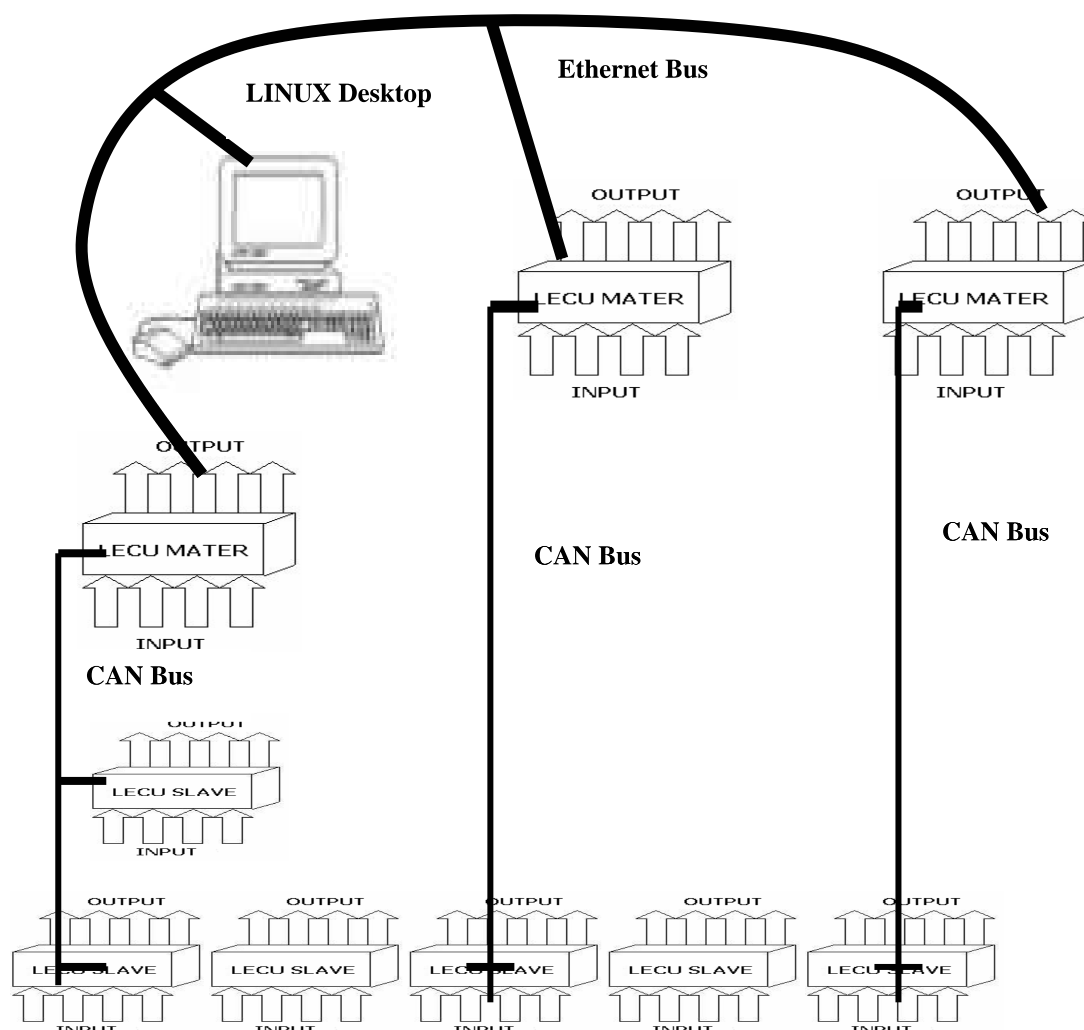


Fig. 2 – Distribuited Configuration

CONCLUSION AND PROPOSED FUTURE WORK

Throughout the recent studies on existing process control systems, a market trend is being noticed that should continue throughout the next few years to include wide usage of open software standards. This great evolution in this area will provide advances of open software applications that already actually gained widespread adoption among domestic users, governments, commerce and will also gain adoption throughout the automation industry. The great paradigm in the way of products distribution will be broken, before what it was opened and free were synonymous of low quality and without technical support will eventually change into high quality and acceptance. Through the success of the experience of Linus Tovald with the Linux and the formation of GNU foundation with clear rules, it was discovered a new horizon for fast development of the open systems. In this context that this work could contribute, it is proposed that a new configuration for industrial networks with open strategies is feasible and practical today. This work will also extend into the visualization of this new industrial revolution resulting from systems and products in this area using this Open Systems concept. The natural evolution of this work is its continuity with the practical development of the modules LECU and the Supervisory System in LINUX.

REFERENCES

Garrels, Machtelt, Introduction to Linux: A Hands on Guide, Version 1.17 20050301 Edition, 03, 2005

Shie, Erlich, Custom Linux: A Porting Guide, Porting LinuxPPC to a Custom SBC,
Revision 2.1 2003-03-08

Robert Bosch GmbH, CAN Specification Version 2.0, 1991, Postfach 30 02 40, D-70442 Stuttgart

Matthew, Neil; Stones Richard, Beginning Linux Programming (Linux Programming Series), Second Edition, Wrox Press, 2000

Nemeth, Evi; Snyder, Garth; Hein, Trent R., Manual Completo do Linux: Guia do Administrador, Translated by: Griesi, Ariovaldo, Sao Paulo: Pearson Makron Books, 2004