



**UNIVERSIDADE FEDERAL DE OURO PRETO – UFOP
ESCOLA DE MINAS – EM
COLEGIADO DO CURSO DE ENGENHARIA DE
CONTROLE E AUTOMAÇÃO - CEC AU**



**MANUAL DE MONTAGEM DE UM CLUSTER BEOWULF
SOB A PLATAFORMA GNU/LINUX.**

**MONOGRAFIA DE GRADUAÇÃO
EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO**

DANNY AUGUSTO VIEIRA TONIDANDEL

Ouro Preto, 2008

DANNY AUGUSTO VIEIRA TONIDANDEL

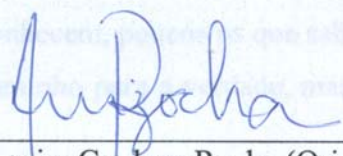
**MANUAL DE MONTAGEM DE UM CLUSTER BEOWULF SOB A
PLATAFORMA GNU/LINUX.**

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção de Grau em Engenheiro de Controle e Automação.


Orientador: Dr. Luiz Joaquim Cardoso Rocha

**Ouro Preto
Escola de Minas – UFOP
Dezembro/2008**

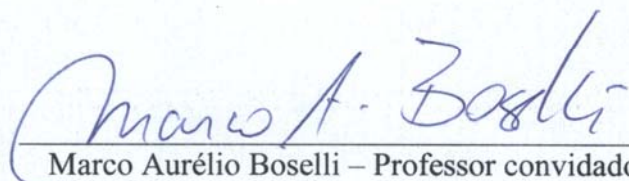
Monografia defendida e aprovada, em 16 de julho de 2008, pela comissão avaliadora constituída pelos professores:



Luiz Joaquim Cardoso Rocha (Orientador)



Valério Silva Almeida – Professor Convidado



Marco Aurélio Boselli – Professor convidado

AGRADECIMENTOS

Tenho muito a agradecer: a Deus pela vida, a minha querida mãe, pelo amor incondicional, apoio e suporte material, sem a qual minha luta não teria sentido. Aos meus queridos colegas e amigos, companheiros de farra e estudos, pelos momentos inesquecíveis. Aos amores, possíveis e impossíveis, que vivi na UFOP. Aos professores pela ciência, aos funcionários pela paciência.

Guardarei comigo todas as lembranças, sabendo que aqui aprendi muito mais do que métodos e fórmulas: guardo a convicção de que a verdade não é patrimônio de ninguém em particular e que diante dela os intelectuais se exaltam, mas os sábios se curvam.

Aprendi que são muitos os que conhecem, poucos os que sabem e raros os que amam. E que o estudo, aliado ao trabalho, é o caminho para a verdade, mas o amor é a luz que nos permite contemplá-la.

EPIGRAFE

“No dizer de Etienne Gilson, tudo pode ser constatado cientificamente, menos o princípio de que tudo pode ser constatado cientificamente.”

Venâncio Barbieri.

SUMÁRIO

LISTA DE FIGURAS	vii
I INTRODUÇÃO	1
1.1 Justificativas.....	1
1.2 Metodologia	2
1.3 Estrutura do trabalho.....	3
II BREVE HISTÓRIA DO <i>LINUX</i>	5
III <i>CLUSTERS</i> E PROCESSAMENTO PARALELO	7
IV <i>MPI</i> E <i>PVM</i>	10
4.1 <i>MPI - Message Passing Interface</i>	10
4.2 <i>PVM - Parallel Virtual Machine</i>	10
4.3 Princípio de funcionamento	12
V AS DISTRIBUIÇÕES <i>GNU/LINUX</i>	16
VI <i>HARDWARE</i>	18
6.1 <i>Hardware</i> e equipamento básico sugerido.....	18
6.2 Instalação física.....	19
VII INSTALAÇÃO DO SISTEMA OPERACIONAL	21
7.1 Como obter os arquivos de instalação?.....	21
7.2 Testando a distribuição	22
7.3 Enfim, instalando o <i>Linux</i> !.....	23
7.4 Contas de usuário.....	25
7.5 Rede	26
VIII A MÁQUINA VIRTUAL	30
8.1 Os pacotes <i>RPM</i>	30
8.2 Instalando os pacotes <i>rsh-client</i> , <i>rsh-server</i>	30
8.3 Editando os arquivos de configuração do <i>RSH</i>	31
8.4 Instalando o pacote <i>PVM - Parallel Virtual Machine</i>	34
8.5 Criando o arquivo <i>.rhosts</i>	34
8.6 Configurando o arquivo <i>.bashrc</i>	36
8.7 Desabilitando o <i>Firewall</i> pessoal	37
8.8 Adicionando membros ao grupo <i>PVM</i>	38
8.9 Criando a máquina Virtual.....	39
XIX O PROGRAMA “CONVERSA”	41
9.1 O ambiente <i>kdevelop</i>	41
9.2 Programa mestre “conversa.cpp”	42
9.3 Programa escravo “escravo1.cpp”	44
9.4 Máquina Virtual	45
X APLICAÇÃO: CÁLCULO DA CONSTANTE π	47
10.1 O cálculo de π	47
10.2 Programa mestre “calcular.cpp”	48
10.3 Programa “calcularescravo.cpp”	50
XI ANÁLISE DE DESEMPENHO.....	53
11.1 Medida de desempenho com o <i>benchmark</i> “calcular”	53
11.2 Precisão e convergência.....	56

XII CONSIDERAÇÕES FINAIS.....	58
XIII REFERÊNCIAS BIBLIOGRÁFICAS	60
ANEXO A - DESCRIÇÃO DAS FUNÇÕES DA BIBLIOTECA <i>PVM</i>	63
A.1 Classificação	63
ANEXO B - COMANDOS DO CONSOLE <i>PVM</i>	64
B.1 Listagem dos comandos	63

LISTA DE FIGURAS

Figura 2.1 - Tux: O pingüim símbolo do <i>Linux</i>	5
Figura 3.1 - Cluster de videogames <i>PlayStation3</i> [®] da Unicamp.....	7
Figura 3.2 - Cluster <i>WIGLAF</i>	8
Figura 3.3 - Cluster Simulador do Espaço da <i>NASA</i>	8
Figura 4.1 - PVM na capa da Revista <i>Linux Journal</i>	11
Figura 4.2 - Programa <i>hello.c</i>	14
Figura 4.3 - Programa <i>hello_other.c</i>	15
Figura 6.1 - Equipamento sugerido para montagem do <i>Cluster Beowulf</i>	18
Figura 6.2 - Esquema físico de Instalação de um <i>Cluster Beowulf</i>	19
Figura 7.1 - Distribuições <i>GNU/Linux</i> mais populares.....	21
Figura 7.2 - <i>DiskDrake</i> , o gerenciador de partições de disco do <i>Mandriva</i>	24
Figura 7.3 - Alternando para superusuário com o comando “ <i>su</i> ”.....	25
Figura 7.4 - Gerenciando contas de usuário.....	26
Figura 7.5 - Escolhendo a conexão <i>Ethernet</i>	27
Figura 7.6 - Selecionando interface de rede.....	27
Figura 7.7 - Definindo a opção de número <i>IP</i>	28
Figura 7.8 - Definindo nome e domínio.....	28
Figura 7.9 - Definindo configurações de inicialização.....	29
Figura 8.1 - Criando o arquivo <i>.rhosts</i> com o editor “ <i>vi</i> ”.....	35
Figura 8.2 - Desabilitando <i>firewall</i> pessoal.....	37
Figura 8.3 - Gerenciando usuários do sistema.....	38
Figura 8.4 - Editando grupo <i>pvm</i>	39
Figura 8.5 - Adicionando usuários ao grupo <i>pvm</i>	39
Figura 8.6 - Criando a Máquina Virtual.....	40
Figura 9.1 - Instalando compiladores e ambiente de desenvolvimento <i>kdevelop</i>	41
Figura 9.2 - Ambiente <i>kdevelop</i>	42
Figura 9.3 - Programa “ <i>conversa.cpp</i> ”.....	43
Figura 9.4 - Copiando executáveis para a biblioteca do <i>pvm</i>	44
Figura 9.5 - Programa “ <i>escravo1.cpp</i> ”.....	45
Figura 9.6 - Executando o programa “ <i>conversa</i> ”.....	46
Figura 10.1 - Área de integração para o cálculo de <i>pi</i>	47
Figura 10.2 - Programa “ <i>calcular.cpp</i> ”.....	48
Figura 10.3 - Programa “ <i>calcularescravo.cpp</i> ”.....	50
Figura 10.4 - Copiando executáveis-escravo para a biblioteca <i>pvm</i>	51
Figura 10.5 - Localizando o programa “ <i>calcular</i> ”.....	52
Figura 10.6 - Executando o programa “ <i>calcular</i> ”.....	52
Figura 11.1 - Desempenho do programa paralelo versus programa serial.....	54
Figura 11.2 - Região na qual o programa paralelo começa a ser mais rápido.....	55
Figura 11.3 - Convergência do erro no cálculo de <i>pi</i>	56
Figura 11.4 - Ampliação da figura 11.3.....	57
Figura B.1 - Comandos do console <i>pvm</i>	65

RESUMO

É apresentado um levantamento dos aspectos necessários à montagem e configuração de um sistema de processamento paralelo: “*Cluster Beowulf*”; que consiste basicamente em dois ou mais computadores conectados em rede, que realizam uma computação concorrente, ou seja, aplicações ou outras tarefas, criando a ilusão de que o trabalho é feito por apenas uma máquina, chamada de “computador virtual”. Essa configuração proporciona uma série de vantagens como alto desempenho e baixo custo de implantação, sendo possível ainda atingir o desempenho de um “supercomputador”, utilizando-se várias máquinas conectadas umas às outras, sem limite de nós. Sua aplicação vai desde a meteorologia, astronomia, cálculos numéricos avançados, até a renderização de gráficos para a indústria cinematográfica. Objetiva-se gerar um tutorial auto-explicativo, indo desde o hardware básico necessário, instalação e configuração do sistema operacional Linux, rede, bibliotecas de distribuição do processamento, criação da máquina virtual (PVM), até o desenvolvimento de uma aplicação bastante utilizada em teste e validação de sistemas paralelos: o cálculo da constante π (“pi”).

Palavras-chave: *Clusters, Cluster Beowulf, Linux, processamento paralelo, PVM, Máquina Virtual.*

ABSTRACT

It is presented a study concerning the necessary steps in order to assembly a parallel processing system: “Cluster Beowulf”, which is a process whereby a set of computers connected by a network are used collectively in order to solve a single large problem (concurrent computation), tasks or applications, creating the illusion that the job is done by a single one system, called “Virtual Machine”. This arrangement brings a series of advantages such as high performance and low cost, being possible to achieve a supercomputer-like performance, juts by using a great number of interconnected machines, with no restrictions to the number of nodes. Its application field involves since the meteorology, astronomy, advanced numerical calculation, until graphical rendering for the cinematographic industry. It is proposed a “how-to” manual, comprehending the necessary hardware, installation and configuration of the Linux operating system, network, message passing libraries, virtual machine (PVM) creation, and the development of a largely used application in test and validation of distributed systems: the constant π (“pi”) calculation.

Keywords: *Clusters, Cluster Beowulf, Linux, parallel processing, PVM, Virtual Machine.*

I INTRODUÇÃO

1.1 Justificativas

O mundo está em constante mudança: a explosão da microeletrônica e da informática nas últimas décadas propiciou um horizonte cada vez mais amplo para o crescimento do planeta como um todo. Ciência e tecnologia avançam a passos largos, impulsionadas pela grande capacidade dos sistemas de computação atuais, permitindo a verificação de teorias e a construção de modelos computacionais cada vez mais complexos. Da mesma forma, a indústria se aproveita desses frutos tecnológicos no intuito de aumentar sua produção, com demandas cada vez maiores (MORIMOTO, 2003).

Muitos computadores (e programas) atuais ainda utilizam a arquitetura de *Von Neumann* (em série), ou seja, apenas uma memória de dados e instruções, onde os programas são executados de forma estritamente seqüencial. Como o tempo de tráfego das informações em um circuito eletrônico é muito pequeno mas não é nulo, significa que existe um limite físico de velocidade para os processadores e que infelizmente está próximo de ser alcançado. Como então prosseguir com a evolução dos computadores? A principal resposta vem da analogia entre um processador e o cérebro humano: sabe-se que a velocidade de um sinal elétrico dentro de um *CI*¹ é muito maior do que um impulso nervoso transmitido pelos neurônios do cérebro. Então, de onde vem a superioridade do cérebro visto que o neurônio é muito mais lento que um circuito eletrônico? A resposta é óbvia: por que o cérebro possui centenas de bilhões de neurônios trabalhando em paralelo.

Neste contexto, pode-se pensar em supercomputadores², que possuem milhares de processadores paralelos, como solução para esses desafios, dada sua gigantesca velocidade e capacidade de processamento. No entanto, o alto custo de aquisição destes equipamentos pode beirar a cifra de alguns milhões de dólares. Muitos se perguntariam: o que fazer então? Uma possível solução surgiu nos laboratórios do *CESDIS – Center of Excellence in Space Data and Information Sciences*, da *NASA* – em 1994, quando engenheiros e pesquisadores tiveram a idéia de utilizar microcomputadores comuns, os populares PC's, acessíveis a qualquer um para realizarem tal tarefa. Bastava “apenas” fazer com que vários deles, ligados em rede,

¹ CI: Circuito Integrado.

² Lista dos 500 mais rápidos supercomputadores do planeta: www.top500.org/

funcionassem como uma só máquina – um “computador virtual” –, também chamado de “*CLUSTER*”³(BUENO, 2002).

Um *Cluster* é basicamente um sistema que compreende dois ou mais computadores (denominados *nós*) ligados em rede que executam um processamento paralelo, ou seja, aplicações ou outras tarefas, sendo que o primeiro deles foi batizado com o nome de “*BEOWULF*”⁴ e acabou tornando-se nome genérico para todos os *Clusters* que se assemelhavam à configuração do original. Há também *clusters* projetados sob outras plataformas, inclusive comerciais, com denominações diferentes do primeiro.

Beowulf inaugurou um período de ouro para o *Linux* na área da supercomputação, justamente por permitir que grande poder computacional fosse alcançado com *hardware* disponível comercialmente. É o que se chama de “*out of the shelf*” - direto das prateleiras. O projeto pioneiro contava primeiramente com 16 máquinas 486, hardware comum e de fácil acesso, executando a plataforma *GNU/Linux*[®].

O campo de aplicações de um *Cluster Beowulf* e *clusters* em geral é imenso: meteorologia, astrofísica, exploração de petróleo, otimização, pesquisa operacional, biotecnologia (Figura 3.1), tolerância à falhas, simulações de mercado financeiro, inteligência artificial, servidores de internet, tratamento de imagens, jogos e até renderização de efeitos visuais para o cinema (BROWN, 2006).

1.2 Metodologia

É proposto um tutorial comentado que possibilita a montagem e utilização de um sistema de processamento paralelo, chamado *Cluster Beowulf*, englobando as etapas e procedimentos necessários para tal, desde sua montagem física até o desenvolvimento de uma aplicação simples: “o cálculo da constante π (pi)”, bastante utilizada para validação em sistemas paralelos, além de atestar o funcionamento do mesmo. Têm-se portanto, um manual fundamentado nas seguintes etapas:

- Discussão e detalhamento do *hardware* básico utilizado;
- Instalação física do *Cluster Beowulf*;

³ Lista dos 500 mais rápidos Clusters no planeta: <http://clusters.top500.org/>

⁴ Referência ao primeiro livro impresso da língua inglesa: “A Lenda de Beowulf”, um poema épico medieval sobre a batalha de um único guerreiro contra os monstros do mar.

- Discussão sobre as distribuições *GNU/Linux*, testes de utilização, instalação do sistema operacional, configuração das contas de usuário e rede;
- Montagem da máquina virtual, instalação e configuração dos pacotes necessários para o processamento paralelo;
- Testes de comunicação entre mestre e escravos, com o programa “*conversa*”;
- Desenvolvimento da aplicação para cálculo de π (pi) em linguagem *C++*;
- Análise do sistema, comparação do *Cluster Beowulf* com um sistema “*monoprocessado*” (programa em série);
- Considerações finais;

1.3 Estrutura do trabalho

Nos capítulos 1 ao 3 é mostrada uma breve introdução aos sistemas paralelos, tipos de *Clusters*, aplicações do processamento paralelo na área da supercomputação e uma breve história do sistema operacional *Linux*.

No capítulo 4, englobam-se alguns conceitos relativos aos conjuntos de softwares e bibliotecas de trocas de mensagens, *MPI* e *PVM*, responsáveis pela distribuição do processamento em sistemas paralelos, como o “*Cluster Beowulf*”.

Dos capítulos 5 ao 7, é mostrada uma discussão sobre as distribuições *GNU/Linux* e o hardware utilizado, assim como o detalhamento das etapas de instalação, configuração do sistema operacional e rede. Posteriormente, no capítulo 8, é listada uma seqüência de ações que permitirão a montagem da “máquina virtual”, com o detalhamento dos pacotes de *software* específicos da biblioteca *PVM*, servidores de acesso remoto (*rsh*), criação e edição de arquivos essenciais para a comunicação entre as unidades de processamento e a montagem da máquina virtual propriamente dita, com a execução do software *PVM*.

O primeiro teste de comunicação é contemplado no capítulo 9, com a apresentação do programa “*conversa*”, onde mestre e escravos na rede formadora do *Cluster* trocam mensagens entre si.

Já no capítulo 10, apresenta-se o desenvolvimento de uma aplicação bastante utilizada na validação e teste de sistemas paralelo: a integração numérica para o cálculo da constante *pi*.

Em sequência, no capítulo 11, parte-se para uma análise de desempenho (*benchmarking*) do *Cluster Beowulf* em comparação a um computador em série (apenas um processador), atestando sua funcionalidade na utilização para fins práticos.

Finalmente, no capítulo 12, são feitas algumas considerações finais a respeito do trabalho de elaboração de um manual e da programação paralela como um todo, com a intenção de contribuir com a disseminação do conhecimento e entendimento sobre o tema específico da montagem de sistemas paralelos.

II BREVE HISTÓRIA DO LINUX

Basta apenas aparecer o desenho de um pingüim gordinho e sentado em qualquer lugar (figura 2.1) para que logo qualquer pessoa com algum conhecimento em informática o associe ao *Linux*: o *Tux* é a imagem que se tornou símbolo deste sistema operacional (ALECRIM, 2004).



Figura 2.1 - Tux: O pingüim símbolo do Linux.
FONTE: (ALECRIM, 2004).

O sistema *Linux* tem sua origem no *Unix*, um sistema operacional multitarefa e multiusuário que tem a vantagem de “rodar” em uma grande variedade de computadores. É dividido em duas partes: a primeira é o *kernel*, que é o núcleo do sistema responsável pela comunicação com o hardware; a segunda é composta pelos programas e serviços que dependem do *kernel* para seu funcionamento (SILVA, 2006).

Em meados da década de 60, a *Bell Telephone Labs.* da *AT&T*, juntamente com a *General Electric* e o projeto *MAC* do MIT⁵ (*Massachusetts Institute of Technology*), desenvolvem o sistema operacional *Multics*. Porém, como ele não atinge seu propósito inicial, logo o Laboratório *Bell* retira-se do projeto (WELSH; KAUFMAN, 1995).

Por causa de um jogo chamado *Space Travel* usado como passatempo durante o projeto *Multics*, dois engenheiros de software da *AT&T*, Ken Thompson e Dennis Richie, que não tinham mais acesso ao sistema, resolveram criar um sistema operacional rudimentar que possibilitasse a portabilidade do jogo em um computador PDP-7, que já não era mais utilizado. Desta forma nasceu o sistema operacional chamado *Unics*, como trocadilho ao *Multics* e que de alguma forma, passou a ser escrito posteriormente como *Unix*.

Já no início da década de 70, o *Unix* é reescrito em linguagem C pelo próprio criador da linguagem, Dennis Ritchie, fazendo com que seu uso dentro da *AT&T* crescesse tanto que um grupo de suporte interno para o sistema acabou sendo criado. Eles forneciam cópias do código fonte para fins educacionais em universidades.

⁵ Massachussets Institute of Technology: web.mit.edu

Em 1983, *Richard Stallman*, um cientista do *MIT* lança o projeto *GNU* (*GNU's not Unix*) que tinha a pretensão de criar um sistema operacional do tipo *Unix* gratuito, em defesa de muitos programadores que haviam contribuído para o aprimoramento do *Unix* e consideravam injusto que a *AT&T* e outros se apoderassem do fruto deste trabalho. No ano seguinte, o projeto *GNU* é iniciado oficialmente, e para efeito de organização, *Stallman* e outros criam a *Free Software Foundation* (*FSF*), uma corporação sem fins lucrativos que buscava promover softwares gratuitos eliminando restrições à cópia, formulando assim a licença *GPL* (*General Public License*).

No final da década de 80, um estudante finlandês chamado *Linus Torvalds* inicia um processo pessoal de aprimoramento do *Kernel* do *Minix*, um sistema operacional do tipo *Unix* escrito por *Andrew Tannenbaum*, chamando esta vertente de *Linux* como abreviação de *Linus's Minix*.

Depois de um certo tempo de trabalho, *Linus* envia uma mensagem para o grupo de discussão do *Minix*, na qual afirma estar trabalhando em uma versão livre de um sistema operacional similar ao *minix* para computadores AT-386, tendo finalmente alcançado o estágio de utilização, e estava disposto a colocar o código-fonte disponível para ampla distribuição.

Como a *FSF* já tinha obtido ou escrito vários componentes importantes do sistema operacional *GNU*, com exceção de um *kernel*⁶, foi uma questão de tempo até que em 5 de outubro de 1991, *Linus Torvalds* anunciasse a primeira versão oficial do *Linux*. No ano seguinte, o *Linux* se integra a *GNU* com o objetivo de produzir um sistema operacional completo.

Desde então, muitos programadores e usuários espalhados pelo globo terrestre têm seguido os ideais de *Richard Stallman* e *Linus Torvalds*, e contribuído para o desenvolvimento do *Linux*.

⁶ *Kernel* é o núcleo do sistema operacional.

III CLUSTERS E PROCESSAMENTO PARALELO

Cluster é um termo amplamente usado, significando uma série de computadores independentes combinados em um sistema unificado de *hardware*, *software* e rede (figura 3.1). Mesmo ao nível fundamental, quando dois ou mais computadores são utilizados juntamente no intuito de resolverem um problema, já considera-se como sendo um *cluster*. Eles podem ser utilizados para duas principais funções, Alta Disponibilidade (*HA – High Availability*) ou Alta Performance (*HPC – High Performance Computing*) para fornecer um poder computacional maior do que aquele fornecido por um simples computador (BROWN, 2006).



Figura 3.1 - Cluster de videogames PlayStation3® da Unicamp.
Fonte: (TILIO, 2008).

Os *Clusters* de Classe I são construídos quase que inteiramente utilizando tecnologia padrão e de fácil acesso, como interfaces SCSI⁷ ou IDE⁸ e placas de rede *Gigabit Ethernet* (Figura 3.2). Eles serão então mais baratos que os *Clusters* de Classe II (figura 3.3), que podem utilizar *hardware* altamente especializado com o objetivo de alcançar alto desempenho (BROWN, 2006).

⁷ SCSI: Small Computer System Interface (Interface para dispositivos de armazenamento).

⁸ IDE: Intelligent Drive Electronics.



Figura 3.2 - Cluster WIGLAF.



Figura 3.3 - Cluster Simulador do Espaço da NASA.

A utilização mais comum dos *Clusters* se dá tanto em aplicações técnicas, como servidores de internet para áudio e jogos quanto simulações: biotecnologia, *petro-clusters*, simulações de mercado financeiro, previsão do tempo, entre outras (BROWN, 2006).

Uma aplicação interessante é a de tolerância à falhas, com dois ou mais PC's ligados entre si: O primeiro executa a tarefa enquanto o segundo tem a função de monitorá-lo constantemente e manter seus dados atualizados em relação ao primeiro. Se o primeiro computador sair do ar por alguma razão, o segundo assume imediatamente suas funções. A tolerância à falhas é muito utilizada em servidores *web* ou servidores de banco de dados em *Intranets* (MORIMOTO, 2003).

Uma segunda aplicação seria o balanceamento de carga, também muito usada em servidores *web*, onde o *Cluster* é formado por pelo menos três PC's, sendo que o primeiro é o mestre, que se encarrega de distribuir as tarefas, como por exemplo requisições de serviços de conexão. Um detalhe interessante neste tipo de configuração é que ao invés de se utilizar um poderoso servidor para fazer o papel de mestre, pode-se utilizar vários microcomputadores comuns para realizarem a mesma tarefa, o que o torna bastante atrativo economicamente.

Uma terceira aplicação e objetivo principal deste trabalho é o processamento paralelo, onde as estrelas principais são os *Clusters Beowulf*. Sua arquitetura baseia-se em uma rede de microcomputadores conectados entre si por dispositivos centralizadores, como *switches* e placas de rede *Gigabit Ethernet*. Têm como grande vantagem o fato de suportarem processadores de famílias e arquiteturas diferentes, ou seja, o funcionamento ou não de um nó não atrapalha a funcionalidade do *Cluster*. Este fato é de extrema importância considerando redes com centenas de processadores. Outro ponto a ser notado é que computadores de

poderio menos elevado podem ser perfeitamente clientes da rede, desde que o servidor “saiba com quem está lidando”. Isso é conseguido através das chamadas às bibliotecas de troca de mensagens – PVM e MPI – adicionadas ao cabeçalho de cada programa nos PC's hospedeiros. Dessa forma o servidor sabe exatamente a arquitetura de cada um de seus clientes, e não requisitará uma tarefa acima de sua capacidade (PVM..., 2008).

Ainda no ramo das vantagens, há de se considerar aquela que pode ser determinante em questões de projeto: a arquitetura *Beowulf* suporta mais de um mestre, da mesma forma que os escravos da rede.

Clusters Beowulf do mundo todo têm contribuído para a solução de diversos problemas de natureza prática, principalmente em aplicações científicas que demandam grande volume de cálculo e processamento de alto desempenho, realização de simulações numéricas avançadas, problemas de escoamento de fluidos, determinação de propriedades físicas dos materiais, otimização, inteligência artificial, previsão do tempo e até em efeitos visuais na indústria cinematográfica. Recentemente, ficaram famosos na mídia e atingiram o status de estrelas principais pela renderização das cenas de filmes como *StarWars*⁹, *Final Fantasy*¹⁰ e muitas outras produções de *Hollywood*.

O princípio de funcionamento de um *Cluster Beowulf* é bastante simples: o servidor (*front-end*) coordena a divisão das tarefas entre os clientes (*back-end*) por intermédio de bibliotecas de troca de mensagens, como a *MPI*¹¹ e *PVM*¹² instaladas previamente e de maneira diferenciada nos PC's mestre e escravos. Somente elas propiciam à este tipo de arquitetura alcançar a computação paralela propriamente dita.

Em *Clusters* desse tipo, é possível processar quantidades imensas de dados, mas apenas ao utilizar aplicativos escritos com suporte à arquitetura, pois em cada programa precisam estar incluídos os cabeçalhos específicos de chamadas às bibliotecas de troca de mensagens, necessárias para a comunicação e reconhecimento das arquiteturas presentes no sistema (MORIMOTO, 2003).

Os programas escritos para a arquitetura *Beowulf* são elaborados utilizando linguagens de programação como C/C++ e FORTRAN (DEITEL; DEITEL, 2001).

⁹ *Star Wars* - Episódio2: O ataque dos Clones. Lucas Arts[®], 2002.

¹⁰ *Final Fantasy: The Spirits Within*, 2001.

¹¹ *MPI: Message Passing Interface*.

¹² *PVM: parallel virtual machine*: Será abordada com detalhes no próximo capítulo.

IV MPI E PVM

4.1 MPI - Message Passing Interface

A *MPI (Message Passing Interface)* é uma biblioteca com funções para troca de mensagens, responsável pela comunicação e sincronização de processos em um *cluster* paralelo. Dessa forma, os processos de um programa paralelo podem ser escritos em uma linguagem de programação seqüencial, tal como C, C++ ou Fortran. O objetivo principal da *MPI* é disponibilizar uma interface que seja largamente utilizada no desenvolvimento de programas que utilizem troca de mensagens, garantindo sua portabilidade em qualquer arquitetura, mas sem a intenção de fornecer uma infraestrutura completa de software para a computação distribuída. Isso a torna recomendável para o desenvolvimento de programas paralelos de alta performance em *MPP's*¹³, ou seja, supercomputadores, que possuem milhares de processadores paralelos em apenas uma máquina (GEIST et al., 1994).

4.2 PVM - Parallel Virtual Machine

O projeto *PVM*¹⁴ teve início no final da década de 80 no *Oak Ridge National Laboratory* (em *Massachussets*, EUA), onde o protótipo fora construído por *Vaidy Sunderman* e *Al Geist*. Primeiramente, esta versão foi utilizada apenas pelo próprio laboratório, sendo que a segunda versão, posteriormente desenvolvida pela universidade do *Tennessee*, foi liberada para distribuição livre em 1991 e começou a ser utilizada em muitas aplicações científicas. Desde então, outras versões vêm surgindo e sendo disponibilizadas em domínio público para livre utilização (PVM..., 2008).

O sistema *PVM* descrito (Figura 4.1), a exemplo da *MPI*, também se utiliza do modelo de troca de mensagens e permite aos programadores explorarem a computação distribuída para uma grande variedade de sistemas computacionais, inclusive *MPP's* (supercomputadores). O conceito chave no *PVM* é fazer com que uma coleção de computadores apresente o comportamento de uma única e poderosa máquina virtual, daí a derivação de seu nome (GEIST et al., 1994).

¹³ *MPP*: Sigla em Inglês para processadores paralelos em massa.

¹⁴ *PVM*: sigla em inglês para Máquina Paralela Virtual.

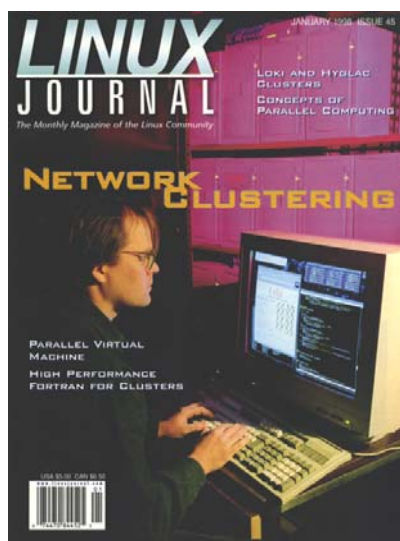


Figura 4.1 - *PVM* na capa da Revista *Linux Journal*.

4.3 Princípio de funcionamento

O *PVM* é um conjunto integrado de software e bibliotecas que possibilita uma computação concorrente e heterogênea para computadores de várias arquiteturas conectados entre si. Algumas de suas funcionalidades são baseadas em (GEIST et al., 1994):

- **Instalação permitida para usuário simples (*host*)¹⁵:** tanto máquinas com apenas um processador até computadores paralelos (com várias unidades de processamento) podem ser escolhidos pelo *host* sem que o mesmo tenha privilégios de super-usuário (*root*). As máquinas podem ser adicionadas ou excluídas durante o processo.
- **Acesso de hardware translúcido:** as aplicações podem “enxergar” o *hardware* no ambiente de trabalho de forma independente da arquitetura, como um conjunto único de elementos necessários ao processamento ou podem escolher explorar a capacidade de máquinas específicas, designando certas tarefas para os computadores mais apropriados ao caso.
- **Computação baseada no processo:** a unidade de paralelismo no *PVM* é a tarefa, ou seja, uma seqüência de controle onde se equilibram comunicação e computação. Em particular, várias tarefas podem ser executadas em um único processador (*pipelining*).
- **Modelo explícito de troca de mensagens:** coleção de tarefas computacionais, sendo cada uma responsável por uma parte da aplicação, porém trabalhando de maneira

¹⁵ *host* = convidado. Usuário com restrições de utilização de um sistema operacional.

cooperativa. O tamanho máximo de uma mensagem é limitado pela quantidade de memória disponível.

- **Suporte à heterogeneidade:** o sistema *PVM* suporta heterogeneidade em termos de máquinas, redes e aplicações. Com relação à troca de mensagens, o *PVM* permite que elas contenham mais de um tipo de dados para que possam ser trocadas entre as máquinas com diferentes sistemas de representação.
- **Suporte a multiprocessadores:** os usuários do sistema nativo *PVM* (de domínio público) têm a vantagem de poderem se comunicar com outros sistemas *PVM* otimizados, fornecidos muitas vezes pelos fabricantes de processadores.

O sistema é composto por duas partes principais: A primeira parte é o *daemon*, chamado de *pvmd3* ou *pvmd*, que reside em todos os computadores formadores da máquina virtual. Um *daemon* pode ser comparado à um programa que roda em segundo plano em um computador pessoal sem que o usuário se dê conta disto, como por exemplo, um aplicativo que gerencia o envio e recebimento de correio eletrônico. O *pvmd3* foi projetado para que um usuário que possua um *login* válido seja capaz de instalá-lo na máquina. Quando o usuário deseja executar uma aplicação, basta criar primeiramente a máquina virtual executando o *PVM*. A aplicação pode então ser chamada a partir de um *prompt Unix* (ou *Linux*) mesmo como *host*. Múltiplos usuários podem configurar várias máquinas virtuais, assim como apenas um usuário pode executar várias aplicações *PVM* simultaneamente (GEIST et al., 1994).

A segunda parte do sistema é a biblioteca com as rotinas de interfaceamento. Ela contém um repertório completo de funções necessárias para a cooperação entre as tarefas de uma aplicação. Essas rotinas, passíveis de chamadas do usuário, vão desde funções para troca de mensagens, coordenação de processos e tarefas, até modificações na máquina virtual.

O modelo de computação *PVM* é baseado no princípio de que a aplicação consiste em várias tarefas. Cada uma é responsável por uma parte do trabalho da aplicação. Às vezes, a aplicação é “paralelizada” por suas funções, isto é, cada tarefa executa uma função diferente, como por exemplo, entrada, saída, montagem do problema, solução e exibição do resultado. Esse processo é geralmente chamado de paralelismo funcional.

O método mais comum de “paralelizar” uma aplicação é chamado paralelismo de dados. Neste método todas as tarefas são as mesmas, mas cada uma resolve e entende apenas uma

pequena parte dos dados do problema. Ele muitas vezes é referido como um modelo de computação *SPMD* (*single-problem multiple-data*, ou problema-único múltiplos-dados).

O *PVM* suporta tanto um único método quanto uma mistura dos dois, dependendo da aplicação, evidenciando a grande heterogeneidade das plataformas de computação suportadas por ele.

Dentre as linguagens de programação suportadas, as mais utilizadas são escritas em *C*, *C++* e *FORTRAN*, mesmo com aplicações orientadas à objetos, utilizando as convenções padrões das respectivas linguagens, inclusive para sistemas baseados em Unix (WELSH; KAUFMAN, 1995).

Programas escritos em *C* e *C++* acessam a biblioteca de funções do *PVM* fazendo a ligação com os arquivos *libpvm3.a* e *libgpvm3.a*, presentes na distribuição padrão do software PVM (GEIST et al., 1994).

Programas em Fortran são implementados mais como subrotinas do que funções, como no caso do *C*. Isto faz-se necessário pela razão de alguns compiladores de arquiteturas suportadas não possuírem interfaces de chamadas de Fortran para *C*. Além do mais, os tipos de funções e bibliotecas utilizadas devem referenciar os tipos trabalhados: *pvm* para *C* e *pvmf* para Fortran. As interfaces Fortran estão contidas em uma biblioteca diferente, a *libpvmf.a*, nos mesmos moldes da anterior.

Todas as tarefas são identificadas por um número inteiro, o *task identifier* (*tid*). Mensagens são enviadas e recebidas através de *tids*. Já que as *tids* devem ser únicas para toda a máquina virtual, elas são sustentadas pelo *pvm* local (*daemon* do nó local) e não podem ser definidas pelo usuário. Embora o *PVM* codifique informações dentro de cada *tid*, é de se esperar que o usuário as veja como identificadores inteiros opacos. O *PVM* contém várias rotinas que retornam um valor *tid* e assim a aplicação do usuário pode identificar outras tarefas no sistema (figura 4.2).


```

main()
{
    int cc, tid, msgtag;
    char buf[100];
    printf("i'm t%x\n", pvm_mytid());
    cc = pvm_spawn("hello_other", (char**)0, 0, "", 1, &tid);
    if (cc == 1) {
        msgtag = 1;
        pvm_recv(tid, msgtag);
        pvm_upkstr(buf);
        printf("from t%x: %s\n", tid, buf);
    } else
        printf("can't start hello_other\n");
    pvm_exit();
}

```

Figura 4.2 - Programa *hello.c*
 Fonte: (GEIST et al., 1994).

O programa, após invocado manualmente (em um terminal Linux), começa imprimindo o número da tarefa (*tid*), obtido da função *pvm_mytid()*. Dispara a cópia de outro programa chamado de *hello_other* (localizado em um nó “escravo”) usando a função *pvm_spawn()*. Se for bem sucedida, faz com que o programa execute o bloqueio de recebimento pela função *pvm_recv*. Após o recebimento da mensagem, o programa imprime a mensagem de enviado por sua contraparte, assim como o número da tarefa disparada; o *buffer* é descompactado com a função *pvm_upkstr*. Finalmente, a função *pvm_exit* dissocia (separa) o programa do sistema PVM.

Na figura 4.3 é listada a programação para o “escravo”. Sua primeira função é obter a identificação da tarefa (*tid*) do “mestre” chamando a função *pvm_parent()*. Obtido seu *hostname*, ele o transmite para o mestre utilizando a chamada de árvore *pvm_initsend* para a transmissão do buffer. A função *pvm_pkstr* coloca uma *string* no *buffer* que será transmitido de uma maneira robusta e independente da arquitetura. Isso é feito por intermédio da função *pvm_send* que o envia ao processo de destino especificado pela *ptid*. Desta forma, a mensagem é enviada “apelidada” com uma *tag* de valor 1.

```
#include "pvm3.h"
main()
{
    int ptid, msgtag;
    char buf[100];
    ptid = pvm_parent();
    strcpy(buf, "hello, world from ");
    gethostname(buf + strlen(buf), 64);
    msgtag = 1;
    pvm_initsend(PvmDataDefault);
    pvm_pkstr(buf);
    pvm_send(ptid, msgtag);
    pvm_exit();
}
```

Figura 4.3 - Programa *hello_other.c*
Fonte: (GEIST et al., 1994).

V AS DISTRIBUIÇÕES GNU/LINUX

O *Linux* é uma espécie de cópia sistema operacional *Unix* para computadores pessoais e que suporta ambientes *multi-tasking* (multitarefa), multi-usuário, interface gráfica *X Window*, redes *TCP-IP* e muito mais. O lançamento do *Linux*, como alternativa ao até então sistema operacional dominante da *Microsoft*, causou furor no mundo inteiro principalmente pelo fato de possuir o código-fonte aberto, possibilitando sua alteração por qualquer um que se habilitasse. Começava a era do software livre e por ironia, o *Linux* teve sucesso em rejuvenescer um dos mais antigos sistemas operacionais até então em uso, o *Unix* (WELSH; KAUFMAN, 1995).

Em termos técnicos, o *Linux* é apenas o núcleo do sistema operacional, responsável pelos serviços básicos de escalonamento de processos, memória virtual, tratamento de exceções, gerenciamento de arquivos, dispositivos entrada/saída etc. No entanto, a maior parte das pessoas utiliza o termo “*Linux*” referindo-se ao sistema completo – núcleo junto das aplicações, interfaces gráficas, jogos e outros. Estas aplicações foram desenvolvidas em sua maior parte graças ao projeto GNU, daí a razão de muitos desenvolvedores denominarem o sistema completo como “GNU Linux” (WELSH; KAUFMAN, 1995).

Como o Linux pode ser modificado e distribuído por qualquer um, naturalmente começaram a surgir as tão faladas *distribuições*, ou “*distros*” como são carinhosamente chamadas em alguns círculos. As mais famosas são, com certeza, as americanas *RedHat* e *Debian*, que se tornaram populares em todo mundo devido à facilidade de manuseio, configuração e por disponibilizarem uma infinidade de pacotes (*softwares*) nos repositórios *on-line* de programas. Aliás, os programas são instalados e executados no Linux de uma forma muito mais eficiente, segura e elegante do que em outros sistemas operacionais: o usuário não precisa se preocupar em qual diretório o programa deve ser instalado ou muito menos se irá contrair um vírus ao fazê-lo, pois os pacotes *Linux* vêm pré-compilados, diferentemente dos executáveis “.exe” do *Windows*. Além do mais, no Linux não é necessário ficar horas procurando por “esse” ou “aquele” programa, pois as *distros* vêm com a maioria dos programas essenciais para o trabalho, como editores de texto, planilhas, utilitários para gravação de CDs, tocadores de música e DVD, entre outros. Na pior hipótese, caso o usuário não encontre o programa que deseja, basta digitar um comando simples em um terminal, que o Linux se encarrega automaticamente de procurá-lo nos repositórios *on-line*, fazendo o seu *download* e instalação sem intervenção externa.

Além do poder do *Linux* em transformar qualquer computador pessoal em uma estação de trabalho com o poder do *Unix*, ele também é utilizado para controlar grandes servidores, devido às suas características de flexibilidade e estabilidade para lidar com grandes *arrays* de disco e sistemas multiprocessados, com aplicações que vão desde servidores *web* a bases de dados corporativas (BUENO, 2002).

A utilização do Linux como sistema operacional em um ambiente de *Cluster* se encaixa muito bem, ainda mais para fins acadêmicos. Segundo os administradores do *Avalon* – um dos maiores *Clusters Beowulf* em operação, situado no Laboratório Nacional de *Los Alamos, EUA* – a utilização do Linux não se deve ao fato de ser gratuito e sim porque ele tem o código-fonte aberto, desempenho de rede superior, e está sendo continuamente desenvolvido de maneira aberta e acessível (WARREN, 2002).

Aparentemente não há restrições quanto à distribuição a ser utilizada, pois teoricamente quase todas possuem as ferramentas necessárias e suficientes para a montagem de um *Cluster Beowulf*. Entretanto, vários outros fatores podem influenciar em tal escolha, como facilidade de manuseio, configuração, compatibilidade de hardware e até familiaridade do projetista com a distribuição em questão, visto que podem haver diferenças significativas entre elas dependendo da aplicação, apesar de todas carregarem o nome *Linux*. Tal assunto será abordado com mais detalhes no capítulo de instalação e configuração do sistema operacional.

VI HARDWARE

6.1 Hardware e equipamento básico sugerido

O hardware utilizado é ferramenta chave para o funcionamento do *Cluster Beowulf* (Figura 6.1). Ao optar-se por máquinas acessíveis e de baixo custo, deve-se atentar para sua utilização ótima para usufruir ao máximo os recursos disponíveis.

No presente caso, conta-se inicialmente com 2 computadores de 64 Bits e processadores de núcleo duplo – *Intel Core 2 Duo*® e *AMD X2* – e de núcleo quádruplo – *Intel Core2 Quad*® – comportando um total de 8 processadores trabalhando paralelamente. Um *Cluster* com essa configuração já possibilita a construção de vários programas complexos e a adição de outros nós na rede pode ser feita de maneira simples e rápida, desde que a máquina virtual esteja corretamente instalada e configurada.

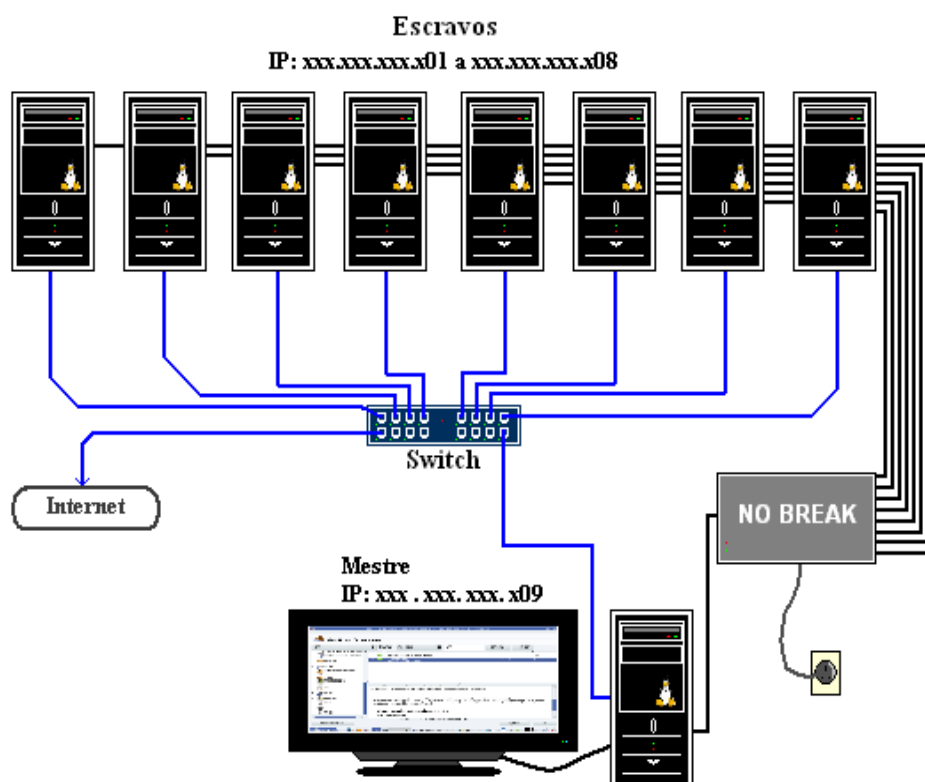
- 2 processadores *AMD X2*® de 64 Bits e 4MB de Memória *Cache L2*;
- 2 processadores *Intel Core2 Duo*® de 64 Bits e 4MB de Memória *Cache L2*;
- 4 processadores *Intel Core2 Quad*® de 64 Bits com 8MB de Memória *Cache L2*;
- 3 Placas-mãe *ASUS*®;
- 3 HDs(memória física) com capacidade de 240GBytes cada;
- 3 placas de vídeo *GeForce 2400+*;
- 3 placas de rede *at11 Gigabit ethernet*, com taxa de transferência 1Gbps(1GBit/s) da *Attansic*®;
- 1 *Switch D-link*® com 16 portas e velocidade de 1Gbps;
- 1 Monitor, teclado e *mouse*;
- 4 cabos de rede *ethernet 1000Base-T*;
- 2 *No-breaks*;
- Equipamento para refrigeração(ar-condicionado ou ventiladores);
- Mesas e/ou suportes para a acomodação das estações de trabalho;

Figura 6.1 - Equipamento sugerido para montagem do *Cluster Beowulf*.

6.2 Instalação física

Não há grandes dificuldades ao se instalar a parte física do *Cluster* – computadores, switch e cabos – o que geralmente pode ser feito de maneira rápida e intuitiva (Figura 6.2):

- Aos computadores, basta conectá-los aos *no-breaks* e eles à rede de alimentação, atentando-se para a tensão de trabalho, 127V ou 220V.
- Para ligação física da rede, basta conectar uma extremidade dos cabos de rede em suas placas ethernet e as outras nas diversas portas do *switch*;
- Se deseja-se acesso à internet, basta conectar um cabo do *switch* a um ponto de acesso, e após configurada a rede, todas as máquinas terão acesso.



Figura

6.2 - Esquema físico de Instalação de um *Cluster Beowulf*.

Para efeito de padronização, será priorizada a programação e exibição de resultados do processamento em apenas um computador, que desempenhará ao mesmo tempo papel de mestre e um dos escravos¹⁶ do *Cluster Beowulf*. Os nós clientes da rede, após configurados e

¹⁶ Isto é possível pois o mestre fica ocioso a maior parte do tempo. Tal assunto será abordado com mais detalhes no capítulo de configuração e programação da máquina virtual.

instalados não necessitarão portanto de teclado, *mouse* ou monitores para o funcionamento do sistema. Serão configurados para que funcionem sem tais dispositivos de entrada e saída, o que pode acarretar em grande economia de implantação do projeto, já que em uma rede com vários clientes, sua contagem teria um peso financeiro considerável.

Para fazer com que o computador funcione sem o teclado basta acessar a BIOS¹⁷ de cada computador separadamente, geralmente pressionando-se a tecla “*delete*” presente no teclado no momento da inicialização(*boot*¹⁸). Ao proceder-se dessa maneira, uma tela básica de configuração da BIOS, o *setup* da máquina, é aberto. Dentre as opções do menu, deve-se procurar a opção que gera um erro na falta do teclado e desabilitá-la, utilizando as teclas “+”(mais) ou “-”(menos) do teclado.

O principal problema deste passo é que cada modelo de placa-mãe – e conseqüentemente cada BIOS – possui um software do *SETUP* diferente, com diferentes nomes e opções. Algumas até reconhecem automaticamente a falta de um dispositivo de E/S e prosseguem o *boot* normalmente, ou então alertam ao usuário o fato e requisitam se ele deseja prosseguir ou não com a etapa de *boot*. Se o usuário tiver alguma dúvida sobre como fazê-lo, basta consultar o manual da placa-mãe em questão, para obter as informações necessárias.

A remoção de monitores e *mouses* contempla uma tarefa bem mais fácil e rápida, já que o sistema não acusa erro caso não estejam presentes, bastando apenas desconectá-los de suas portas. No entanto, é importante lembrar que estes passos somente devem ser realizados após a instalação do sistema operacional, configuração da rede, instalação e configuração completa da máquina virtual.

¹⁷ BIOS: memória somente de leitura ou ROM (não-volátil).

¹⁸ Boot: processo de inicialização de um computador, onde programas essenciais para o funcionamento do hardware e sistema operacional são carregados da memória ROM para a RAM.

VII INSTALAÇÃO DO SISTEMA OPERACIONAL

7.1 Como obter os arquivos de instalação?

Chegou o momento da instalação da distribuição *Linux* e o que para alguns foi um pesadelo no passado, hoje consiste em uma tarefa muito simples. A maioria das distribuições disponíveis atualmente possuem gerenciadores de instalação automatizados, onde são feitas algumas perguntas ao usuário e a instalação ocorre sem problemas de uma maneira até mais fácil que os sistemas operacionais mais populares, como o *Windows*[®] da *Microsoft*.

A primeira coisa a ser feita é a obtenção dos CD's ou DVD's de instalação. Os arquivos de imagem (ISO) para a gravação podem ser obtidos fazendo-se o *download* nos *sites* da distribuição, sítios de busca como o *Google*[®] ou então nos repositórios *online* de programas, que possuem várias distribuições disponíveis. O *Linux* é desenvolvido abertamente e de maneira global e em consequência disto, as empresas e organizações responsáveis pelo lançamento das distribuições lançam uma nova versão a cada seis meses em média. É aconselhável utilizar-se um *Live-DVD* ou *Live-CD*¹⁹ por permitem teste e visualização dos recursos da distribuição antes da instalação definitiva.

Na FIG. 7.1 são listadas algumas das mais populares distribuições disponíveis, com seus respectivos sítios para *download*:

RedHat Linux(distribuição americana, comercial)	http://www.redhat.com
Mandriva Linux(brasileira, versões <i>free</i> ou comercial)	http://www.mandriva.com
Fedora (versão <i>free</i> , disponibilizada pela RedHat)	http://fedoraproject.org
SuSE (americana, baseada na RedHat)	http://www.novell.com/linux
Kurumin (distribuição brasileira, baseada no Debian)	http://www.guiadohardware.net/kurumin
Debian (Distribuição americana, free)	http://www.debian.org
Ubuntu (distribuição sul africana totalmente livre, baseada no Debian)	http://www.canonical.org

¹⁹ Live-CD/DVD: Executa o sistema operacional diretamente da mídia removível.

Slackware (muito parecida com o *UNIX* e disponibilizada por uma única pessoa, *Patrick Volkerding*)

<http://www.slackware.com/>

Figura 7.1 - Distribuições GNU/Linux mais populares.

7.2 Testando a distribuição

Para o usuário iniciante, é aconselhável reservar um HD (*Hard disk*) para a instalação do *Linux*, evitando-se o perigo de danificar eventuais arquivos contidos no disco rígido. Por outro lado, caso o usuário prefira um “*dual boot*”, ou seja, dois ou mais sistemas operacionais na mesma máquina – como *Linux+Windows* – recomenda-se instalar o *Linux* por último, pois ao detectar a presença de outro sistema operacional, o *Linux* requisitará a instalação de um gerenciador de *boot* (*GRUB*), o qual criará um menu de opções no momento da inicialização do computador, aonde o usuário poderá carregar o sistema operacional de sua escolha.

A próxima etapa é a preparação dos microcomputadores para a instalação do sistema operacional, na qual a maneira mais fácil de fazê-lo é proceder o *boot* diretamente do CD ou *DVD-ROM*. Para isso basta configurar a opção “*boot sequence*”(Ordem de *boot*) no *Setup*²⁰ com o valor “*CD/DVD-ROM, C , A*”. Em computadores mais antigos, deve-se acessar também a seção “*PnP/PCI Setup*” e configurar a opção “*PnP OS*”(geralmente a primeira opção) com o valor “No”. Isto obriga a BIOS a detectar e configurar os endereços a serem utilizados por todos os periféricos *Plug-and-play* e entregar o trabalho semi-pronto para o sistema operacional, evitando-se a ocorrência de problemas na detecção de periféricos.

Feito isto, basta ligar o computador, colocando o *Live-CD* da distribuição e aguardar o *boot*. Em quase todas as distribuições atuais, como o *Debian*, *Ubuntu*, *Mandriva*, *Slackware*, *Red Hat*, *Fedora* e tantas outras, o processo de instalação é basicamente o mesmo: “dar” o *boot* pelo CD, particionar o HD, escolher os pacotes que serão instalados, configurar o vídeo e a rede, definir a senha de *root* (superusuário) e configurar o gerenciador de *boot*.

Para evitar problemas, é altamente recomendável testar as distribuições antes de instalá-las, pois existe a possibilidade de ocorrerem problemas na detecção do hardware, principalmente ao utilizar-se máquinas com hardware muito específico, de circulação não muito comum ou com pouco suporte, tais como placas de rede, vídeo e som. Têm-se então duas alternativas para resolver o problema:

- Testar, por meio dos *live-CDs* de instalação das distribuições, se todos os dispositivos de hardware são detectados e configurados automaticamente. Recomendável para a utilização da distribuição sem complicações.

²⁰ É possível acessar o *SETUP* da *BIOS* pressionando-se repetidamente a tecla “delete” ao ligar-se o computador (no momento da contagem de memória).

- Encontrar nos repositórios *on-line* de programas, os *drivers*²¹ de configuração para os dispositivos em questão; o que pode ser uma tarefa bastante trabalhosa, já que os *drivers* devem ser específicos para a distribuição e *kernel Linux* utilizados.

Segundo este contexto, as seguintes distribuições foram testadas:

- *Mandriva Linux Free 2008 for i586(32 Bits)*;
- *Mandriva Linux Free 2008 for X86_64 (64 Bits)*;
- *Fedora Core 8 e 9 for X86_64 (64 Bits)*;
- *Ubuntu e Kubuntu Linux 7.10 for X86_64 (versões GNOME²² e KDE²³)*;

Após o teste com várias distribuições, aquela que mostrou maior compatibilidade e estabilidade com o hardware disponível (mencionado no item 7.1) foi a distribuição *Mandriva Linux 2008 free* – KDE, baseada na americana *RedHat* e com versão do *kernel Linux* 2.6.22. Contudo, a decisão final sobre qual distribuição utilizar é especial em cada situação e depende de vários fatores além daqueles já mencionados, e não é regra obrigatória a utilização de uma em particular.

7.3 Enfim, instalando o Linux!

Após a verificação da distribuição através dos *live-CDs* chegou o momento da instalação. Caso tenha-se utilizado um *live-CD*, na própria área de trabalho do ambiente *KDE* haverá um ícone “Iniciar Instalação”, onde basta ao usuário clicá-lo e seguir as instruções. Se o usuário possuir apenas o *CD* ou *DVD* de instalação, basta proceder o *boot* a partir do disco de instalação e aguardar instruções. Na maioria das vezes, será apenas preciso clicar em “avancar” e “avançar”; procedimento análogo à instalação de um software qualquer em um sistema *Windows*. O instalador solicitará ao usuário informações como Idioma de instalação, *layout* do teclado, hora etc. Mesmo o particionamento do disco é bastante simples no *Mandriva*.

Dentre as perguntas feitas ao usuário, uma é modo de instalação. A primeira opção (padrão) é aconselhável caso se deseje instalar o sistema sem muitas perguntas. No entanto, recomenda-se o modo “*Expert*”, pois permite ter um melhor controle da instalação. Ainda sim e durante

²¹ Arquivos com informações de configuração de hardware: Similares aos *Device Drivers* do *Windows*.

²² GNOME – ambiente gráfico *Xwindow* similar ao *MAC-OS* da *Apple*.

²³ KDE – *K desktop enviroment* – ambiente gráfico *Xwindow* similar ao *Windows*.

todo o processo haverá um assistente tira-dúvidas caso o usuário não esteja familiarizado. Na escolha do idioma de instalação e *layout* do teclado, apenas deve-se ter o cuidado de escolher a opção *ABNT-2* caso o teclado seja em português (possua a tecla cedilha “ç”), ou Internacional caso contrário.

Em seguida, será a vez de ajustar o nível de segurança do sistema. Escolha o nível padrão, pois no capítulo de configuração do *Remote Shell(RSH)*, será necessário desabilitar o *firewall* do Linux. Contudo, estas opções podem ser alteradas mais tarde pelo *Mandriva Control Center (MCC)*.

Chega-se agora à parte mais crítica da instalação: o particionamento do disco. Felizmente, o *Mandriva* possui uma ferramenta bastante amigável para facilitar esta tarefa, o *DiskDrake* (Figura 7.2).

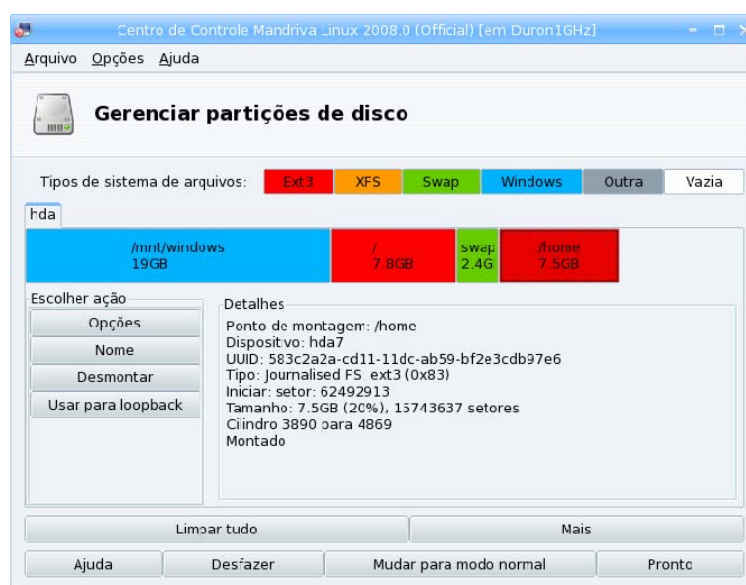


Figura 7.2 - *DiskDrake*, o gerenciador de partições de disco do *Mandriva*.

Pode-se deixar o utilitário redimensionar uma partição *Windows* já existente, usando o espaço livre para instalar o *Linux*, utilizar uma partição *Linux* previamente criada (“Usar partição existente”), usar o espaço não particionado do disco (caso tenha algum) ou simplesmente apagar tudo que estiver gravado e partir para uma instalação limpa. Se o usuário pretende redimensionar uma partição *Windows* pré-existente, existem dois cuidados necessários para que tudo ocorra bem. Primeiro, certificar-se de que exista espaço em disco suficiente. Segundo, deve-se desfragmentar o disco através do *Windows* antes de iniciar a instalação. Ao escolher-se a opção “apagar todo o disco” o programa vai simplesmente limpar a tabela de

partição do *HD* e dividi-lo em duas partições: uma menor, montada no diretório raiz(/), usada para os arquivos do sistema e outra maior, montada no diretório */home*, onde ficam guardados os arquivos dos usuários.

Por último, as duas opções automáticas se adequam bem para o usuário iniciante, porém sem muitas alternativas e contêm apenas os sistemas de arquivos suportados pelo Linux: *Ext3*, *Journalised FS*, *Swap*, *FAT*, *NTFS(Windows)* além de “Outra” (caso o sistema de arquivos não seja reconhecido) e “Vazia” (espaço não particionado), como já mostrado na FIG. 7.2.

7.4 Contas de usuário

Ao final do processo de instalação, será pedido ao usuário que crie uma conta, com nome completo, *login* e senha. Além disso será pedido que forneça uma senha de *root* (administrador). Isto acontece porque, mesmo que o usuário em questão seja o administrador do sistema, o *Mandriva* – por medida de segurança – criará uma conta de usuário simples para sua operação normal. Esta ação do sistema operacional é muito importante para diminuir uma prática ruim por parte dos usuários: a de executarem tudo como superusuários (*root*), o que é de alto risco e potencialmente prejudicial; pois sem restrições, pode-se danificar o sistema apenas digitando-se um comando errado no *prompt*, ou apagando um arquivo de sistema por engano. É recomendável que o usuário utilize a conta simples para as tarefas comuns, como acesso à *internet*, digitação de documentos etc, fazendo uso da conta de *root* apenas quando for necessário alterar a configuração do sistema ou instalar novos programas. A boa notícia é que como usuário normal também é possível ter acesso a todas as ferramentas de configuração, bastando apenas fornecer a senha de *root*. Outro detalhe é que o *root* não aparece na tela de *login*. Caso o usuário queira usá-lo, será necessário digitar “*su*” em um terminal e fornecer a senha de *root* (Figura 7.3). Atentar para o símbolo “\$” que indica usuário simples e “#” que indica superusuário (*root*).



Figura 7.3 - Alternando para superusuário com o comando “su”.


Caso queira-se criar uma outra conta de usuário, basta acessar o *MCC – Mandriva Control Center* (digitando-se *mcc* em um terminal ou clicando-se no ícone correspondente), ir em *Sistema/Gerenciar usuários do Sistema* (Figura 7.4).



Figura 7.4 - Gerenciando contas de usuário.

No capítulo de configuração da máquina virtual, um outro usuário aparecerá em cena: o *PVM*, que necessitará acessar as máquinas da rede sem a intervenção externa.

7.5 Rede

Após a instalação física dos componentes da rede como mostrado na seção 6.2, parte-se agora para a configuração da rede propriamente dita. Ela pode ser facilmente feita no ambiente *KDE* de qualquer distribuição *GNU/Linux* e no *Mandriva* não é diferente. Para isto, o usuário deve dirigir-se ao centro de controle (*Mandriva Control Center* - *MCC*): basta digitar “*mcc*” em um terminal ou no ícone correspondente: . Em seguida, deve-se abrir a aba “Redes e Internet” e clicar em “Configurar uma nova interface de rede”.

A configuração é bem simples, pois os endereços *IP* podem ser configurados automaticamente (*DHCP*) ou o usuário pode escolher configurá-los de acordo com seu desejo (no formato xxx.xxx.xxx.xxx). O mais importante neste momento é escolher adequadamente os nomes dos computadores presentes na rede de forma a facilitar a identificação posterior, pois estes nomes serão utilizados ao longo de toda vida útil do *Cluster*. Serão aqui utilizados nomes associados ao Laboratório de Sistemas Térmicos, Metrologia e Instrumentação da Escola de Minas - UFOP, como “*labsisterXX*” e domínio “*em.ufop.br*”.

Na Fig. 7.5 até a Fig. 7.9 é mostrada uma sequência de ação para configuração da rede.

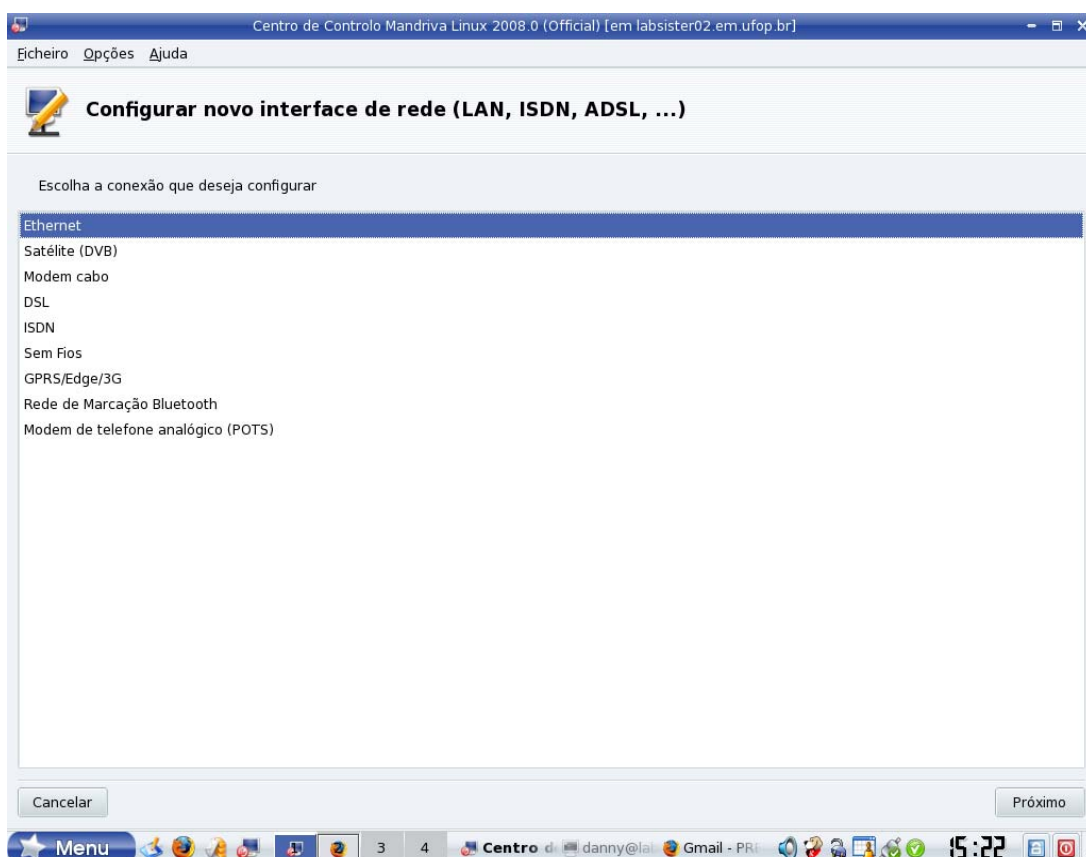
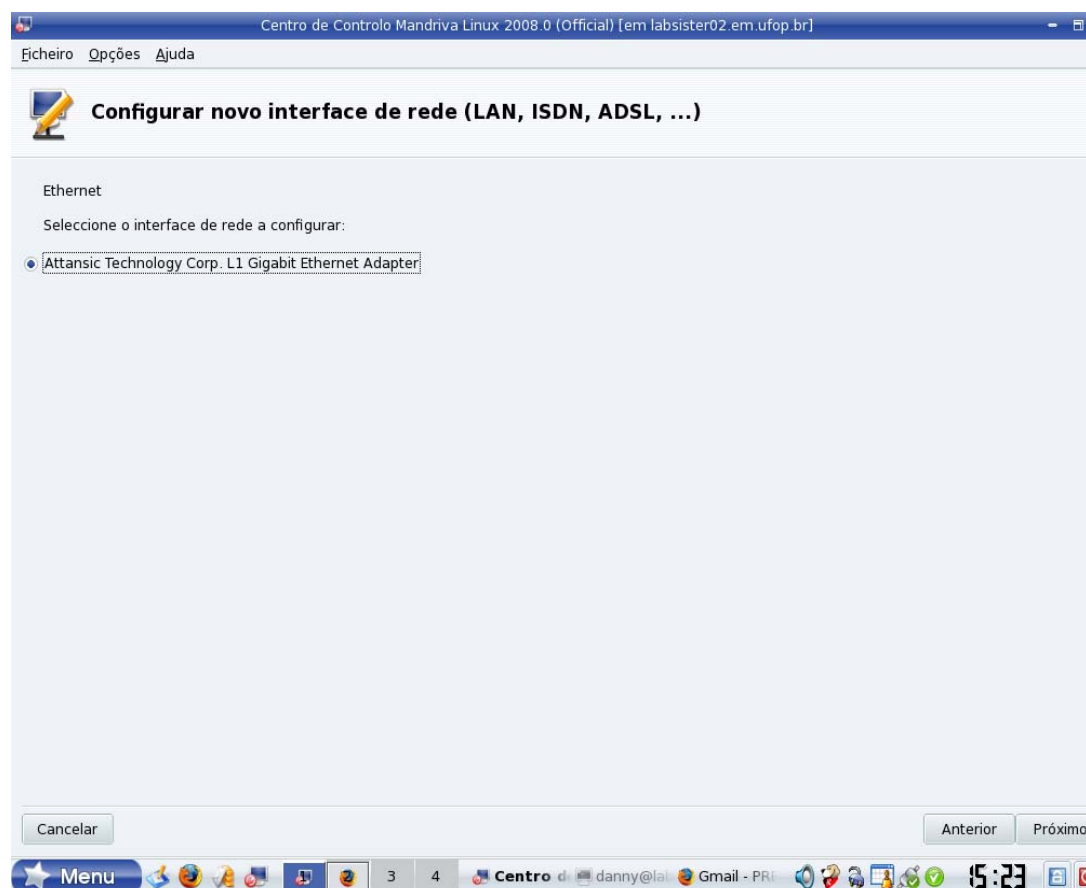
Figura 7.5 - Escolhendo a conexão *Ethernet*.

Figura 7.6 - Selecionando interface de rede.

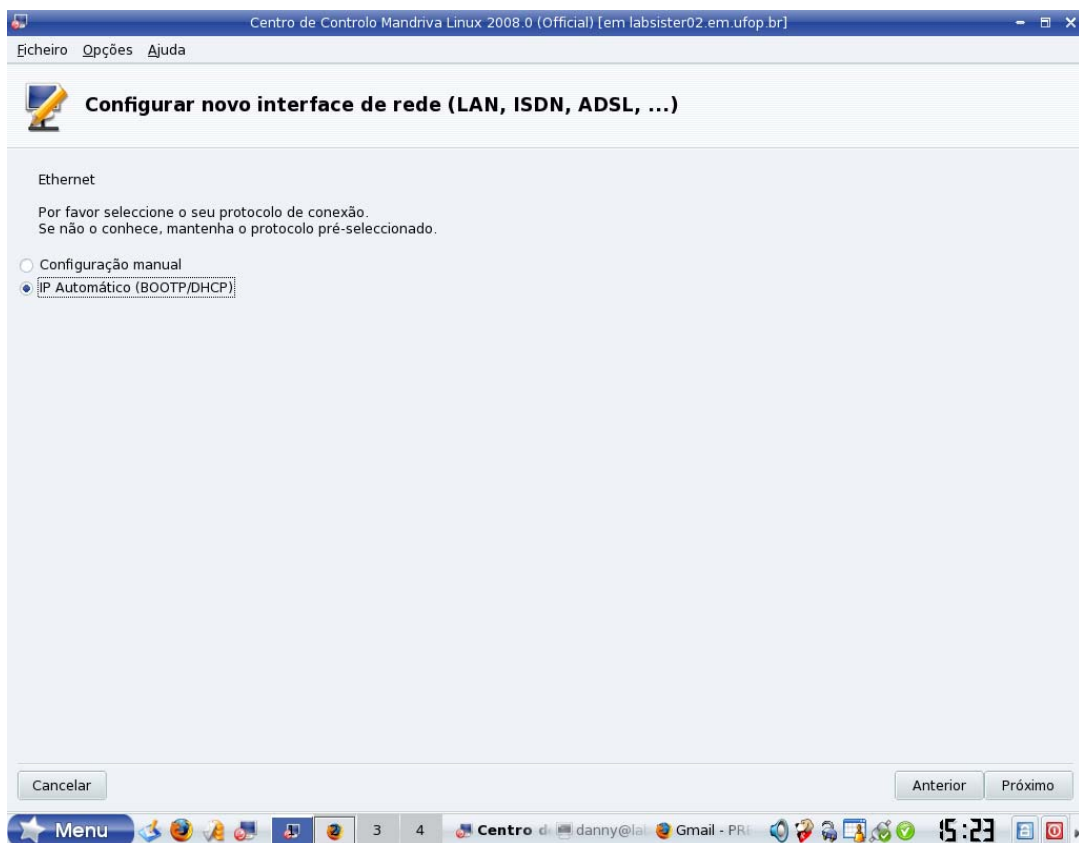


Figura 7.7 - Definindo a opção de número IP.

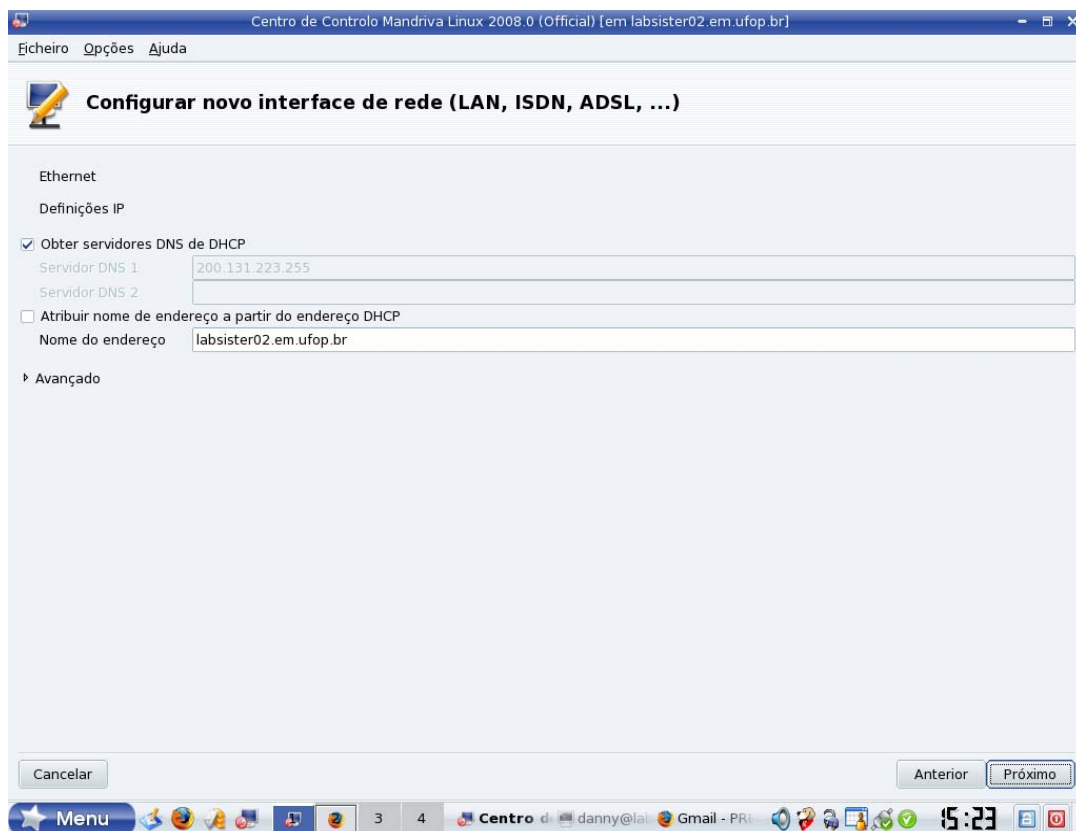


Figura 7.8 - Definindo nome e domínio.

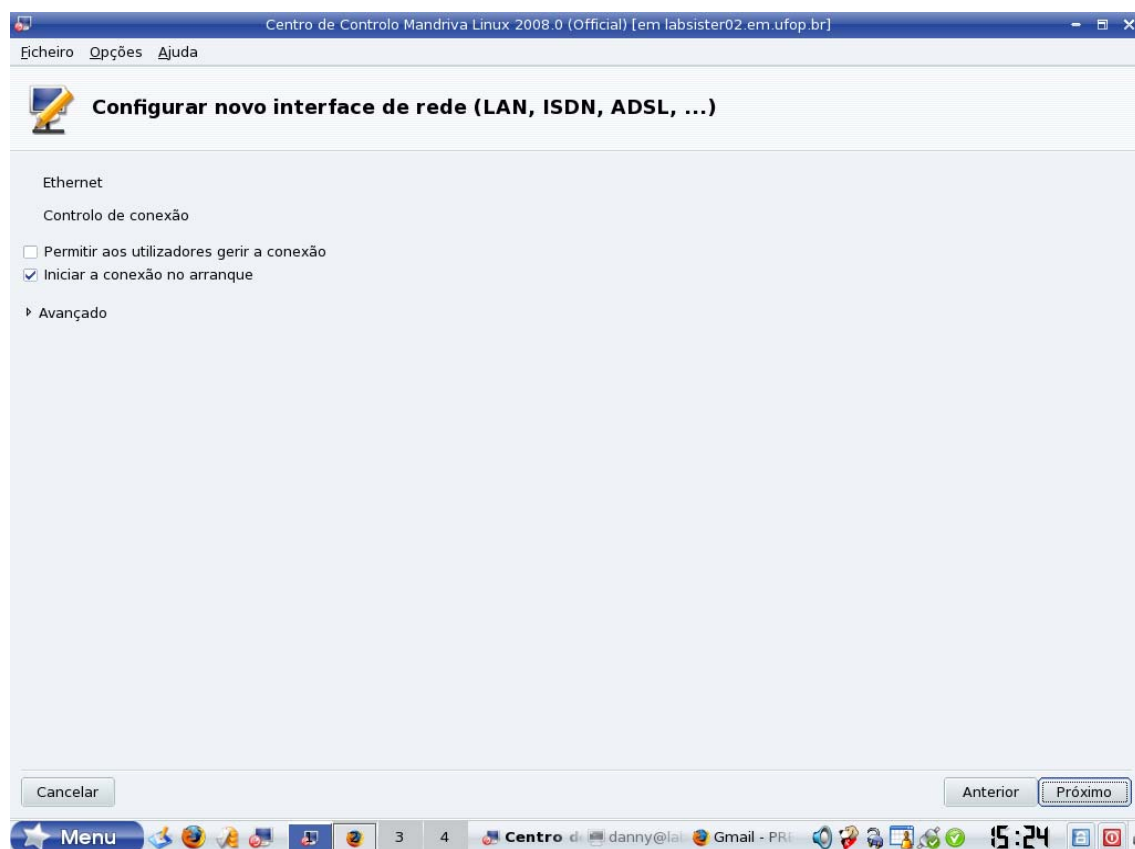


Figura 7.9 - Definindo configurações de inicialização.

VIII A MÁQUINA VIRTUAL

8.1 Os pacotes *RPM*

Aproxima-se o momento de montagem da máquina virtual. É ela que proporcionará o processamento paralelo de todos os processadores conectados à rede do *Cluster*. E além do software *PVM* – *Parallel Virtual Machine*, responsável pela distribuição do processamento, também é necessária a utilização de ferramentas que possibilitem a comunicação e acesso remoto às máquinas (software *Remote Shell* - *rsh*), assim como a confecção de programas paralelos (em C/C++ ou outra) no ambiente de desenvolvimento *kdevelop*.

No atual estágio de desenvolvimento do *GNU/Linux*, há uma infinidade de pacotes (*softwares*) específicos para cada distribuição e disponíveis nos repositórios *on-line*. Os pacotes em *Linux* têm uma conotação diferente àquela dos populares executáveis (*.exe*) em sistemas *windows*, pois vêm pré-compilados, tornando-se executáveis apenas após a compilação. Para versões baseadas na *RedHat*, como o *Mandriva*, existem os pacotes *RPM*, que são auto-instaláveis e muito úteis no momento da instalação de programas. Eles necessitam apenas ser executados (clicando-se 2 vezes sobre o ícone correspondente ou digitando-se `#urpmi nome_do_pacote` em um terminal) para que sejam instalados automaticamente.

8.2 Instalando os pacotes *rsh-client*, *rsh-server*

O pacote *RPM rsh-client* – *Remote Shell Client* – contém um conjunto de programas que permite aos usuários executar comandos em máquinas remotas, executar *login* remoto, copiar ou trocar arquivos entre máquinas (*rsh*, *rlogin* e *rcp*). Todos esses três comandos usam o estilo de autenticação do tipo *rhosts* (*remote host* ou convidado remoto). Este pacote *rsh* é necessário para habilitar o acesso remoto e será primordial para o funcionamento da máquina virtual, já que o *PVM* utiliza o modelo de troca de mensagens para coordenar o processamento e necessita acessar todas as máquinas com privilégios de superusuário. É como se o *PVM* fosse um “usuário virtual”.

Os pacotes *RPM* (como o *rsh-client*) podem ser facilmente obtidos através dos repositórios *online* (como o *rpmfind.org* ou sites de busca como o *google*[®]) e sua instalação se dará executando-se o arquivo de extensão “*.rpm*”, clicando-se duas vezes sobre seu ícone correspondente e selecionando a opção “instalar”.

Detalhes do pacote RPM *rsh-client* para *Mandriva 2008 free* 64Bits:

Nome do pacote: *rsh-0.17-18mdv2008.0.x86_64.rpm*
 Versão: *0.17-18mdv2008.0*
 Versão atualmente instalada: *0.17-18mdv2008.0*
 Arquitetura: *x86_64*
 Tamanho: 64 KB

O pacote *rsh-server* (*Remote Shell Server*) é instalado da mesma forma que o *rsh-client*. No entanto, será necessária sua configuração, pois por segurança, o *rsh* é configurando de forma a somente permitir o *login* remoto caso o usuário forneça o nome de usuário (*ID*) e senha; e isto impossibilitaria a utilização do *PVM*, pois ele deve promover um tráfego intenso de dados e informações entre os nós da rede, peça chave do processamento paralelo.

A configuração do *rsh* se dará alterando-se o conteúdo de alguns arquivos específicos, possibilitando o *Remote Shell* (*rsh*), *Remote Login* (*rlogin*) e *Remote Copy* (*rcp*), o que será mostrado mais adiante. O pacote RPM (*rsh-server*) contém os servidores necessários para todos os serviços citados. Ele também contém um servidor de *rexec*, um método alternativo para executar comandos remotos. Os servidores são executados pelo *software xinetd* e os arquivos de configuração estão nos diretórios */etc/xinetd.d* e *pam.d*. Como já dito, todos os servidores são desabilitados por padrão.

Detalhamento do pacote RPM *rsh-server* para *Mandriva 2008 free* 64Bits:

Nome do pacote: *rsh-server-0.17-18mdv2008.0.x86_64.rpm*
 Versão: *0.17-18mdv2008.0*
 Versão atualmente instalada: *0.17-18mdv2008.0*
 Arquitetura: *x86_64*
 Tamanho: 55 KB

Caso seja solicitado pelo sistema operacional, deve-se ainda instalar o pacote *xinetd*:

Detalhamento do pacote RPM *xinetd* para *Mandriva 2008 free* 64Bits:

Nome do pacote: *xinetd-2.3.14-6mdv2008.0.x86_64.rpm*
 Versão: *2.3.14-6mdv2008.0*
 Versão atualmente instalada: *2.3.14-6mdv2008.0*
 Arquitetura: *x86_64*
 Tamanho: 122,5 KB

8.3 Editando os arquivos de configuração do RSH

Após a instalação dos pacotes mencionados, é possível proceder-se a configuração do *Remote Shell*, sendo que editor de textos “*vi*” pode ser usado para esta operação. O “*vi*” é executado

no terminal (menu/ferramentas/konsole), sendo altamente recomendável, pois ao alterar-se um arquivo de texto no ambiente *KDE* pode-se resultar em alguns problemas na execução do *RSH* e conseqüentemente na máquina virtual.

Os arquivos de configuração apresentados são os mesmos em qualquer distribuição *GNU/Linux*, no entanto a sua localização difere entre elas. Basta que o usuário localize os arquivos corretamente e os edite em seus respectivos diretórios. Os seguintes passos devem ser seguidos para a edição de tais arquivos:

Executar-se o terminal (menu/ferramentas/konsole), lembrando que os arquivos mostrados devem ser editados com privilégios de superusuário (figura 7.3):

1- Editar o arquivo *rexec* em */etc/pam.d/*

`$ su` (será pedida a senha de superusuário)

`# cd /etc/pam.d/` (abre o diretório */etc/pam.d/*)

`# ls` (lista os arquivos presentes)

Se o arquivo *rexec* aparecer na lista, digitar:

`# vi rexec`

Quando o editor *vi* iniciar, aperte a tecla “a” do teclado para ativar o modo de edição. Altere o arquivos de forma que suas linhas fiquem como mostrado:

<i>auth</i>	<i>sufficient</i>	<i>pam_nologin.so</i>
<i>auth</i>	<i>sufficient</i>	<i>pam_securetty.so</i>
<i>auth</i>	<i>sufficient</i>	<i>pam_env.so</i>
<i>auth</i>	<i>sufficient</i>	<i>pam_rhosts_auth.so</i>
<i>auth</i>	<i>include</i>	<i>system-auth</i>
<i>account</i>	<i>include</i>	<i>system-auth</i>
<i>session</i>	<i>include</i>	<i>system-auth</i>

Ao terminar, pressione a tecla “*Esc*” para sair do modo de edição, em seguida a tecla “:”, o que fará o editor “*vi*” deslocar o cursor até o final do arquivo e finalmente as teclas “*wq*” e ENTER, que significam “*write*”(escrever), “*quit*”(sair).

Observação: para arquivos que necessitem privilégios de usuário simples, deve-se digitar “:wq!” para encerrar o programa e forçar o editor a salvar o conteúdo.

Da mesma forma, localizar o arquivo *rlogin* (no diretório */etc/pam.d/*), seguir até seu diretório (como usuário *root*), usar o editor “*vi*” e alterar o arquivo da maneira como segue:

```
$ cd /etc/pam.d/      (vai ao diretório especificado)
$ ls                  (lista conteúdo)
$ su                  (altera para modo root, superusuário)
# vi rlogin           (abre o arquivo rlogin no editor vi)
```

Pressione “a” para modo de edição e alterar as linhas do arquivo conforme mostrado:

```
##%PAM-1.0
auth          sufficient    pam_rhosts_auth.so
auth          sufficient    pam_securetty.so
auth          sufficient    pam_nologin.so
auth          include       system-auth
account       include       system-auth
password      include       system-auth
session       include       system-auth
```

Novamente, digitar “:wq” para sair e salvar.

De maneira análoga alterar o arquivo *rsh* no diretório *pam.d* (pasta */etc/pam.d/rsh*):

```
auth          sufficient    pam_nologin.so
auth          sufficient    pam_securetty.so
auth          sufficient    pam_env.so
auth          sufficient    pam_rhosts_auth.so
account       include       system-auth
session       include       system-auth
```

Em sequência, o arquivo *rlogin* no diretório *xinetd.d* (pasta */etc/xinetd.d/rlogin*):

```
# default: off
# description: rlogind is the server for the rlogin(1) program. The server \
#   provides a remote login facility with authentication based on \
#   privileged port numbers from trusted hosts.
service login
{
    socket_type      = stream
    wait             = no
    user             = root
    log_on_success   += USERID
    log_on_failure   += USERID
    server           = /usr/sbin/in.rlogind
    disable          = no
}
```

Por último, alterar o arquivo *rsh* no diretório *xinetd.d* (pasta */etc/xinetd.d/rsh*):

```

#default: off
#description: The rshd server is the server for the rcmd(3) routine and, \
# consequently, for the rsh(1) program. The server provides \
# remote execution facilities with authentication based on \
# privileged port numbers from trusted hosts.
service shell
{
    socket_type = stream
    wait        = no
    user        = root
    log_on_success += USERID
    log_on_failure += USERID
    server      = /usr/sbin/in.rshd
    disable     = no
}

```

8.4 Instalando o pacote PVM - Parallel Virtual Machine

O software PVM habilitará uma coleção heterogênea de computadores ser utilizada como um coerente e flexível recurso de computação concorrente. Os pacotes *RPM* são instalados de maneira análoga aos anteriores. Agora, dois pacotes deverão se instalados, o software *pvm* e o pacote *pvm-devel* que contêm as bibliotecas de funções do PVM *libpvm3.a*, *libfpvm3.a*, *libgpvm.a* e *libfpvm.a*, necessárias ao funcionamento dos programas paralelos, assim como os arquivos de cabeçalho com as funções, o *pvm3.h*:

Detalhes do pacote RPM *pvm*:

Nome do pacote: *pvm-3.4.5-6mdv2008.0.x86_64.rpm*
 Versão: *3.4.5-6mdv2008.0*
 Versão instalada: *3.4.5-6mdv2008.0*
 Arquitetura: *x86_64*
 Tamanho: 795 KB

Detalhes do pacote RPM *pvm-devel*:

Nome do pacote: *pvm-devel-3.4.5-24.x86_64.rpm*
 Versão: *3.4.5-24*
 Versão atualmente instalada: *3.4.5-24*
 Arquitetura: *x86_64*
 Tamanho: 856 KB

8.5 Criando o arquivo *.rhosts*

O arquivo *.rhosts* será responsável por mostrar ao servidor de acesso remoto (*rsh*) presente em determinada máquina, quais dos nós clientes da rede têm permissão para acessá-la via *rsh*.

Isto pode ser realizado criando-se um arquivo de nome “.rhosts”(da forma como escrito, inclusive o ponto) no diretório *home* de cada usuário (/home/nome_do_usuario). Novamente, recomenda-se utilizar o editor “vi”. Abra o konsole e dirija-se ao diretório *home* do usuário como mostrado:

```
$ cd /home/danny      (/home/nome_do_usuario)
```

```
$ vi .rhosts          (cria o arquivo .rhosts)
```

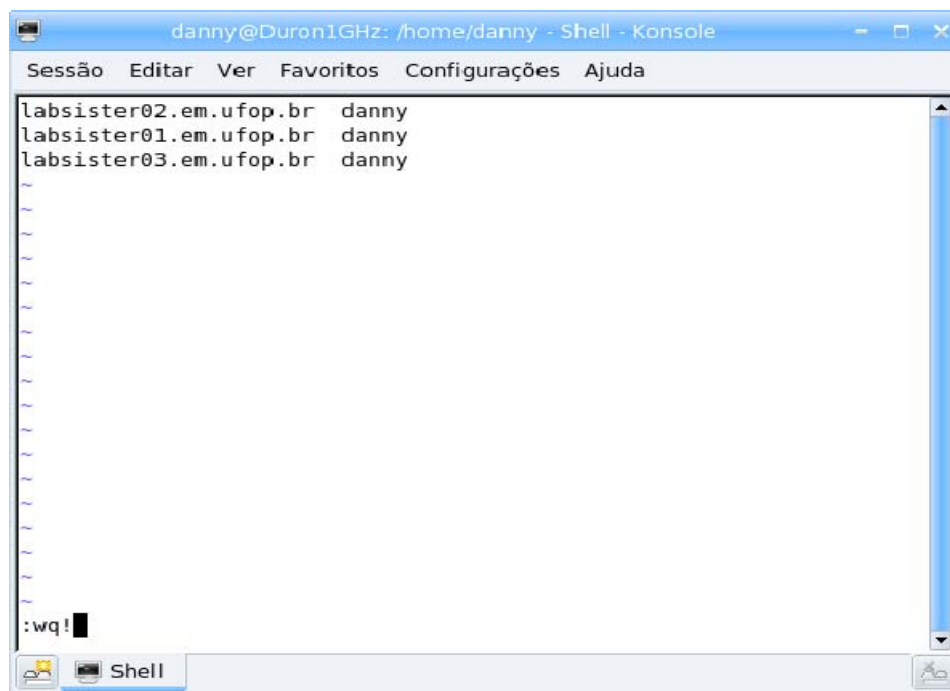
Como pode-se observar pelo símbolo “\$” do *prompt*, esta operação deve ser efetuada sem privilégios de superusuário (simbolizado por “#”). Quando o editor *vi* iniciar, pressionar a tecla “a” do teclado para ativar o modo de edição. Entre com o nome completo, incluindo apelido e domínio das máquinas da rede já configurados anteriormente (Figura 8.1).

Ao terminar, pressionar as teclas “Esc”, para sair do modo de edição e a tecla “:”, o que o fará deslocar-se até o final do arquivo. Em seguida, as teclas “wq!”(write, quit e o sinal “!” para forçar o editor a salvar o conteúdo). Para arquivos que necessitem privilégios de superusuário, apenas digite “:wq” para encerrar o programa e salvar o conteúdo (Figura 8.1).

Para verificar o arquivo criado (ou qualquer outro), basta entrar com o comando:

```
$ more nome_do_arquivo
```

Exemplo: \$ more .rhosts



Para conferir se os arquivos foram editados como usuário ou superusuário (*root*), prosseguir com o comando:

```
$ ls -all
```

O konsole exibirá algo do tipo:

```
-rw-r--r-- 1 danny danny    88 2008-04-11 13:46 .rhosts
```

```
-rw-r--r-- 1 danny danny   237 2008-04-08 16:44 .bashrc
```

Caso algum arquivo tenha sido criado como superusuário, o nome *root* aparecerá em lugar do usuário.

8.6 Configurando o arquivo *.bashrc*

Após instalado o pacote PVM, já é possível criar a máquina virtual, mas sempre que se desejasse fazê-lo, o usuário teria que acessar o diretório onde está instalado o *PVM* e executar o comando “*pvm*” para tal. Para automatizar este processo, basta alterar o arquivo *.bashrc* como usuário simples, ou seja, sem privilégios de *root*. Enfatiza-se a necessidade de proceder-se desta maneira, caso contrário o PVM não conseguirá efetuar o *login* remoto nos clientes da rede.

Dentro da pasta *home/nome_do_usuario*, editar o arquivo *.bashrc* digitando no terminal:

```
$ vi .bashrc    (abre o arquivo .bashrc no editor “vi” como usuário comum)
```

Ao abrir o arquivo, adicionar as seguintes linhas, pressionando-se “a” para modo de edição:

```
PVM_ROOT=/usr/share/pvm3
export PVM_ROOT
PVM_RSH=/usr/bin/rsh
export PVM_RSH
PVM_ARCH="LINUX64"
export PVM_ARCH
```

O conteúdo final do arquivo apresentará o formato:

```
# .bashrc
# User specific aliases and functions
# Source global definitions
if [ -f /etc/bashrc ];
then
    . /etc/bashrc
fi
PVM_ROOT=/usr/share/pvm3
```

```
export PVM_ROOT
PVM_RSH=/usr/bin/rsh
export PVM_RSH
PVM_ARCH="LINUX64"
export PVM_ARCH
```

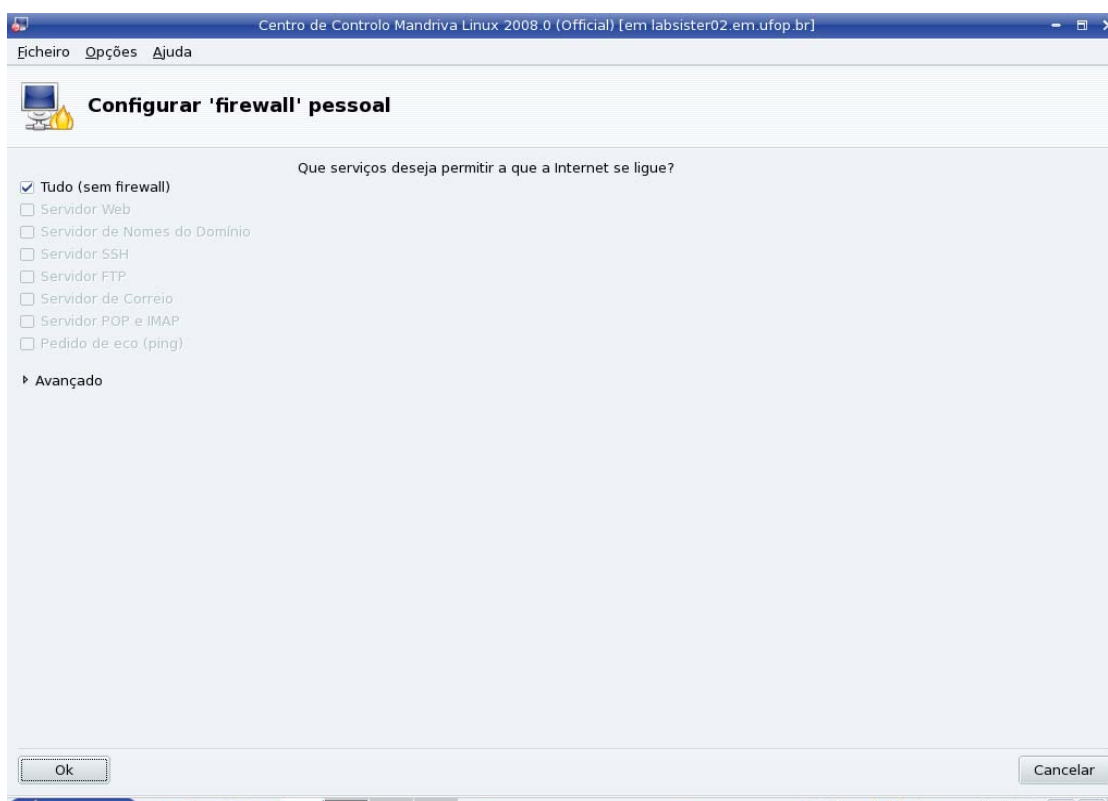
Observação: se a distribuição LINUX instalada for de 32 bits, a penúltima linha fica:

```
PVM_ARCH="LINUX"
```

Para finalizar, digitar “*wq!*” para salvar o conteúdo. Para checar se o conteúdo do arquivo foi alterado, digitar o comando “*more .bashrc*” no mesmo diretório.

8.7 Desabilitando o Firewall pessoal

Para que o software *pvm* possa acessar as máquinas sem nenhuma barreira, deve-se também desabilitar o *firewall* do *Mandriva Linux*. Para isto dirija-se ao centro de controle – *mcc/Segurança/Redes e Internet/Configurar seu firewall Pessoal* – e marcar a opção “Tudo (sem *firewall*)” (Figura 8.2).



Ao desabilitar o *firewall* para a utilização do *rsh*, o sistema fica vulnerável à ataques externos. Para evitá-los pode-se proceder de duas maneiras: caso tenha-se acesso à internet, desconectar o ponto de acesso ao utilizar a máquina virtual, ou então utilizar um outro computador ligado à rede para fazer tal papel. Ele pode inclusive ser de poderio menos elevado, já que terá a única finalidade de ser o *firewall* da rede.

8.8 Adicionando membros ao grupo PVM

Após realizadas as etapas de instalação e configuração dos arquivos para a montagem da máquina virtual, os pacotes PVM instalados já terão neste momento criado um grupo de acesso, assim como os usuários configurados previamente durante a instalação do sistema operacional. Isto é explicado pelo fato de o *pvm* agir como um “usuário virtual”, pois necessitará acessar todas as máquinas da rede e gerenciar a comunicação entre elas. No entanto, os usuários autorizados a utilizarem o *Cluster Beowulf* devem adicionar seus respectivos nomes ao grupo PVM existente. Basta acessar o Centro de Controle do *Mandriva(mcc)/Sistema/Gerenciar usuários do Sistema*, localizar o grupo “pvm” (Figura 8.3) e clicar no botão “editar” (Figura 8.4). Feito isso, abrir-se-á uma janela de edição na qual o usuário pode adicionar seu nome ao grupo (Figura 8.5).

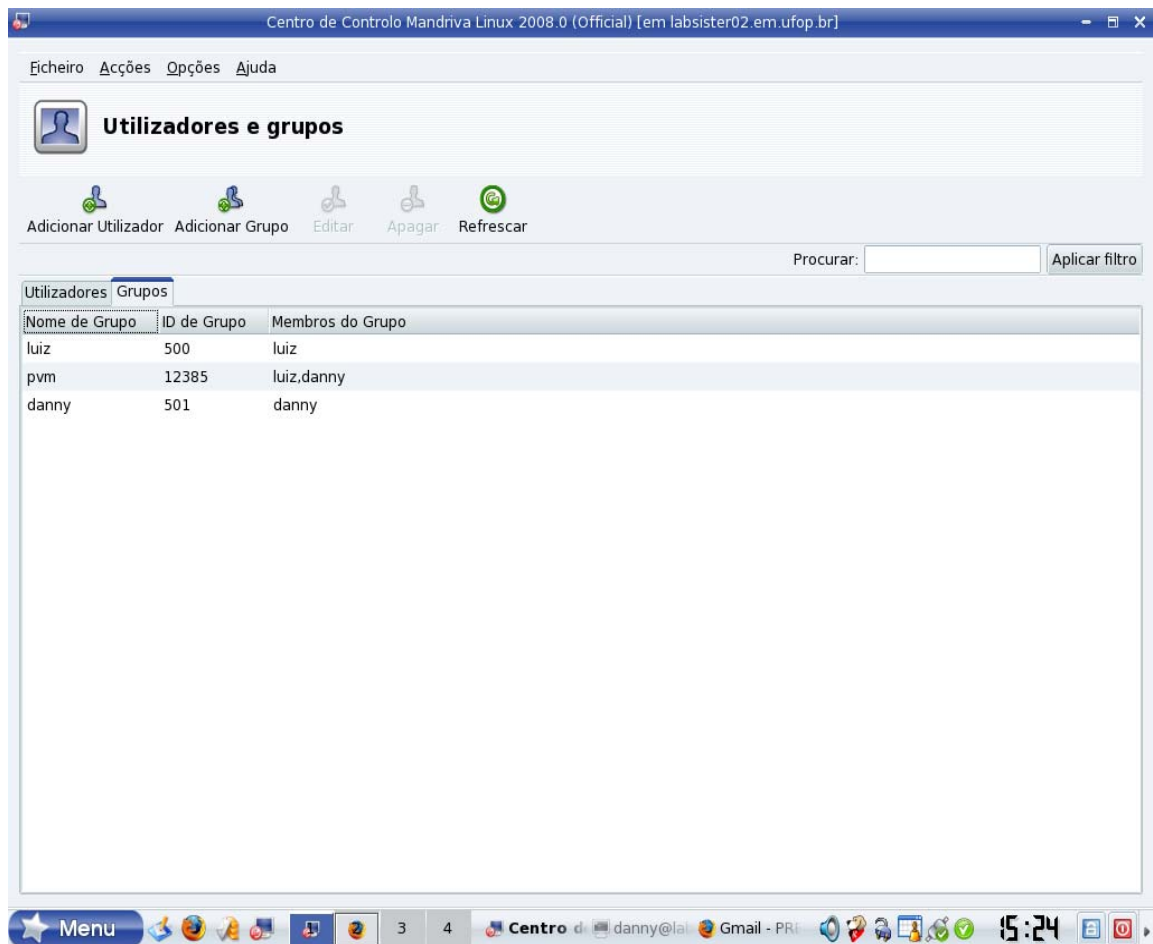


Figura 8.3 - Gerenciando usuários do sistema.

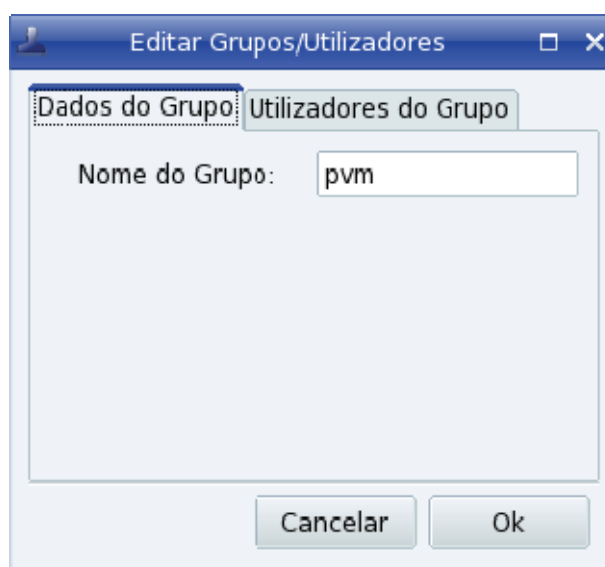


Figura 8.4 - Editando grupo pvm.

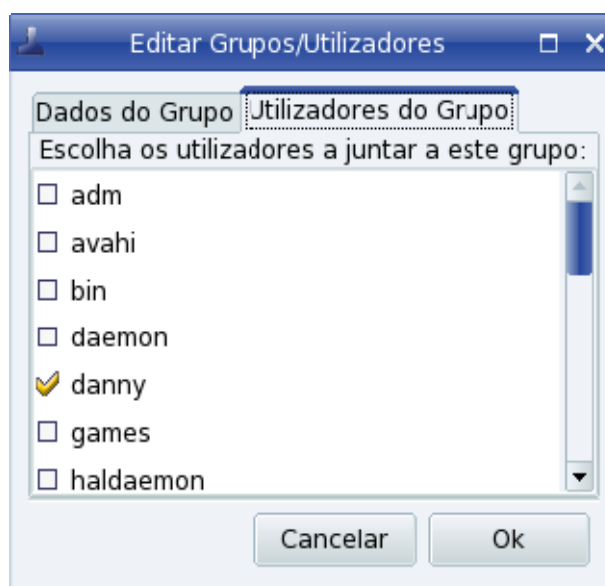
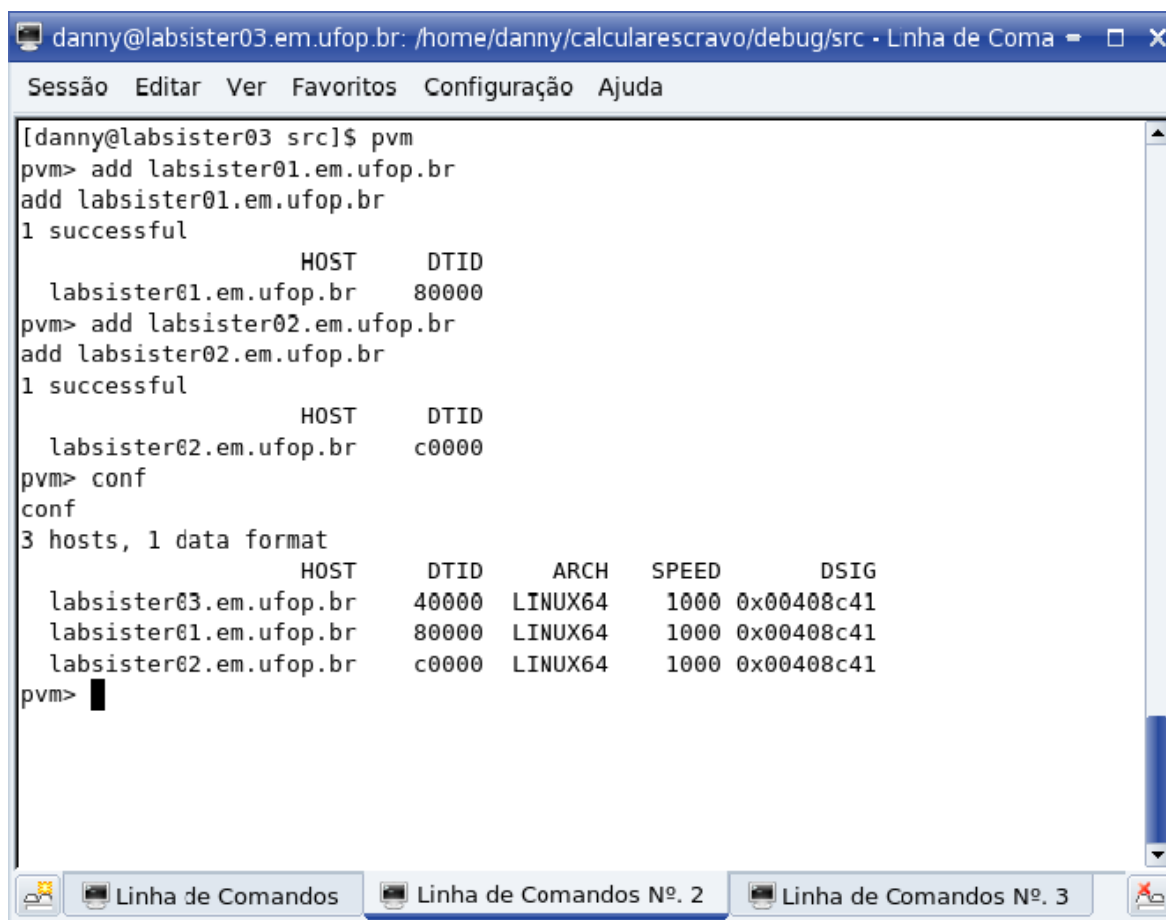


Figura 8.5 - Adicionando usuários ao grupo pvm.

8.9 Criando a máquina Virtual

Após realizadas as etapas que culminaram na configuração do PVM é chegado o momento de “construir” a máquina virtual propriamente dita e que consiste na tarefa mais simples do processo: basta ao usuário executar a aplicação *pvm* (digitando-se “*pvm*” no terminal). A partir daí o *daemon*²⁴, chamado de *pvmd3* ou *pvmd*, que reside em todos os computadores formadores da máquina virtual entrará em ação. Cabe ao usuário agora informar ao *PVM* quais máquinas integram a rede, com o comando *add nome_da_máquina* no *prompt* do *pvm* (Figura 8.6).

²⁴ Um *daemon* pode ser comparado a um programa que roda em segundo plano em um computador pessoal sem que o usuário se dê conta disto.



The screenshot shows a terminal window titled "danny@labsister03.em.ufop.br: /home/danny/calcularescravo/debug/src - Linha de Coma". The window has a menu bar with "Sessão", "Editar", "Ver", "Favoritos", "Configuração", and "Ajuda". The terminal content shows the following sequence of commands and output:

```
[danny@labsister03 src]$ pvm
pvm> add labsister01.em.ufop.br
add labsister01.em.ufop.br
1 successful
      HOST      DTID
labsister01.em.ufop.br  80000
pvm> add labsister02.em.ufop.br
add labsister02.em.ufop.br
1 successful
      HOST      DTID
labsister02.em.ufop.br  c0000
pvm> conf
conf
3 hosts, 1 data format
      HOST      DTID      ARCH      SPEED      DSIG
labsister03.em.ufop.br  40000  LINUX64      1000  0x00408c41
labsister01.em.ufop.br  80000  LINUX64      1000  0x00408c41
labsister02.em.ufop.br  c0000  LINUX64      1000  0x00408c41
pvm> █
```

The terminal window has a taskbar at the bottom with three tabs: "Linha de Comandos", "Linha de Comandos Nº. 2", and "Linha de Comandos Nº. 3".

Figura 8.6 - Criando a Máquina Virtual

Pode-se observar que o *host* no qual a máquina estiver sendo criada não precisará ser adicionado na máquina virtual, pois o PVM já o reconhecerá como primeiro nó da rede.

Na etapa de configuração da máquina virtual, também é possível utilizar pacotes no estilo *SSH*, que são mais seguros, ao invés do *RSH*. No entanto, a codificação (criptografia) dos dados necessária para a transmissão de informações atrapalharia o desempenho do sistema como um todo. A desvantagem porém é a

A máquina virtual está pronta para a execução de um programa paralelo! Para isto, deve-se sair do console *pvm* sem portanto encerrá-lo (digitando-se *quit*) e executar qualquer programa paralelo. Para encerrar o *pvm* basta digitar *halt* no *prompt* do PVM (ANEXO B). A programação e execução de programas paralelos será vista com maiores detalhes no próximo capítulo.

XIX O PROGRAMA “CONVERSA”

9.1 O ambiente *kdevelop*

Para a programação paralela necessita-se primeiramente de um ambiente de programação apropriado. Em distribuições *Linux* com interface *KDE*, há o ambiente de desenvolvimento *kdevelop*. Ele dá suporte à várias linguagens, tais como *C/C++*, *Java*, *FORTRAN*, *Perl* etc. Em caso de no momento da instalação do sistema operacional, o usuário não tenha escolhido instalar os pacotes de desenvolvimento, basta ir ao centro de controle/Gerenciar pacotes/Desenvolvimento, marcar todos os pacotes relacionados à linguagem de sua escolha, desde os compiladores gcc (no caso de C++), até o ambiente kdevelop (figura 9.1); e clicar no botão aplicar. Os pacotes devem referenciar a distribuição Linux utilizada (*i586* para 32Bits e *X86_64* para 64Bits).

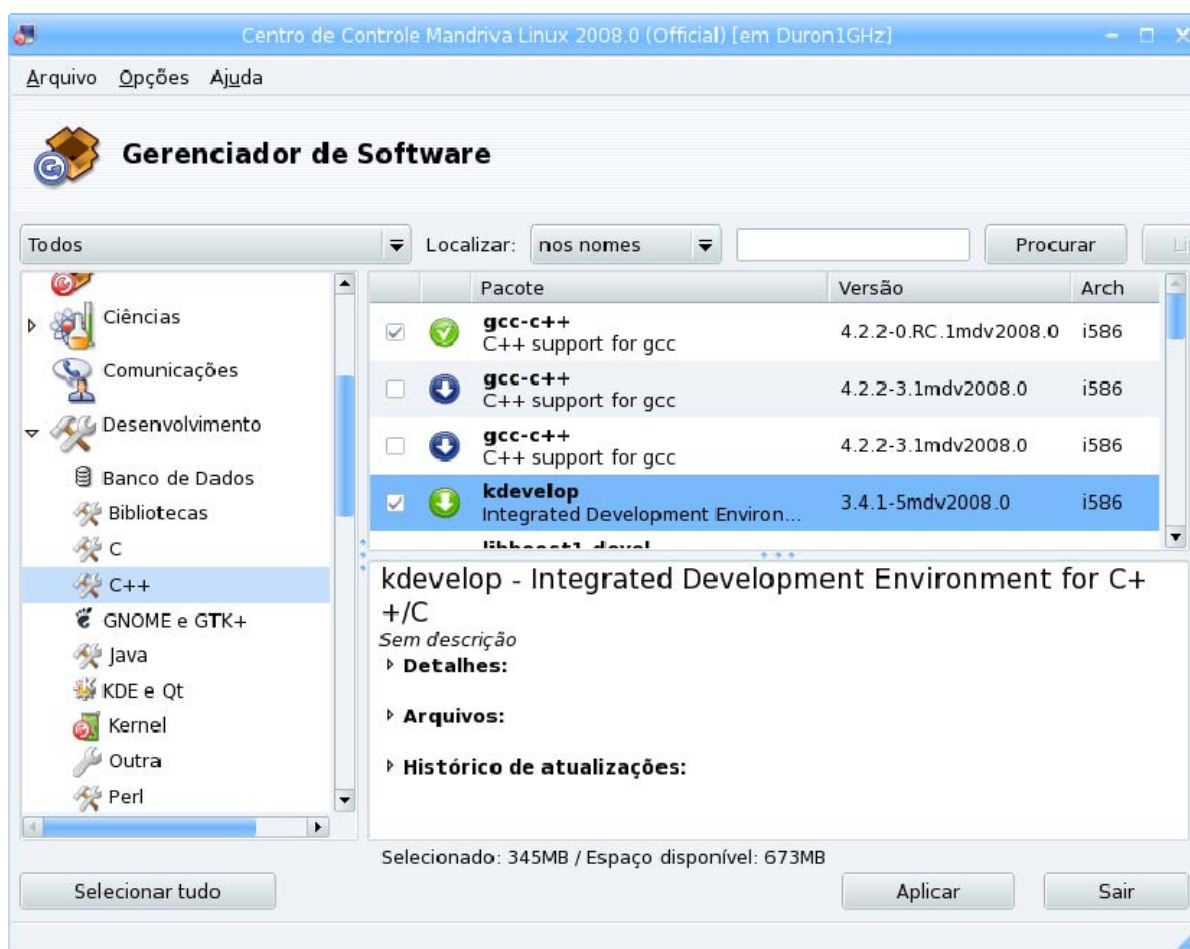


Figura 9.1 - Instalando compiladores e ambiente de desenvolvimento *kdevelop*.

Para abrir o ambiente instalado, clicar em Menu/Desenvolvimento/kdevelop C/C++. A tela inicial é similar ao da figura 9.2:

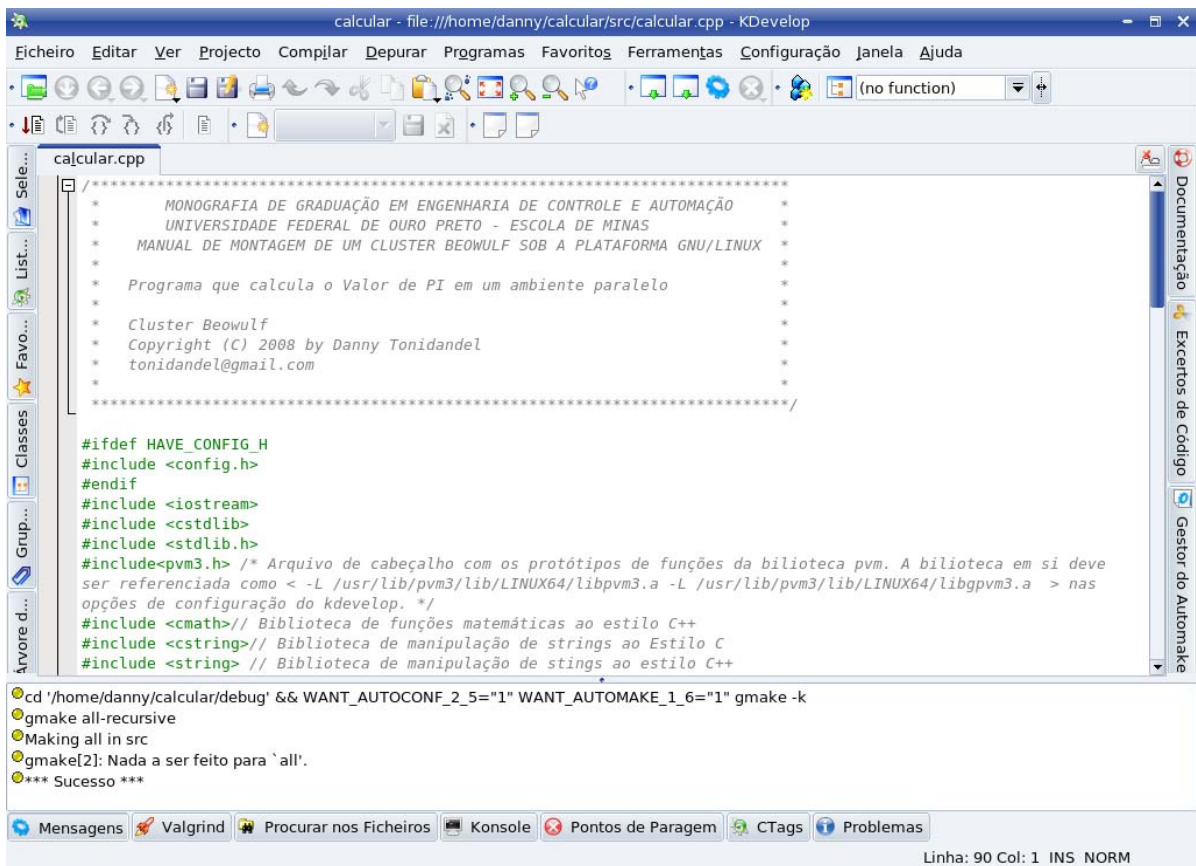


Figura 9.2 - Ambiente *kdevelop*.

O programa “conversa” terá a função de testar a comunicação entre mestre e escravos do *cluster*. Como visto no capítulo 4 (figuras 4.2 e 4.3), em um *Cluster Beowulf* utilizando o modelo *pvm* deve-se ter programas diferenciados para mestre e escravos. Assim deveremos executar duas instâncias do ambiente *kdevelop* na máquina escolhida para a programação. Em uma janela será desenvolvido o programa para o mestre e em outra o programa para o(s) escravo(s).

9.2 Programa mestre “*conversa.cpp*”

Para criar um novo projeto no ambiente *kdevelop* deve-se abrir o menu projeto/novo projeto/C++/programa “olá mundo”; dar um nome apropriado ao programa, como “*conversa*”, avançar as etapas até que o programa principal seja aberto. No lugar daquele criado, escrever o programa para o mestre que comandará o processamento paralelo. É importante lembrar que deve-se referenciar o arquivo de cabeçalho *pvm3.h* que contém os protótipos de funções²⁵ do PVM (Figura 9.3):

```

/******
 *                               Copyright (C) 2008 by Danny Tonidandel                               *
 *                               tonidandel@gmail.com                               *
 *****/
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif
#include <iostream>
#include <cstdlib>
#include <pvm3.h> // arquivo de cabecalho com as funcoes usadas pelo pvmd
using namespace std;

int main(int argc, char *argv[])
{
    int numt, tid;
    char buf[100];
    cout << "*****" << endl;
    cout << "*"          Ola Escravos! Voces estao ai?          "*" << endl;
    cout << "*"          Eu sou BEOWULF!!                    "*" << endl;
    cout << "*****" << endl;
    cout << endl;
    printf( "Minha tarefa é = %x\n", pvm_mytid() );
    cout << endl;

    /* Da função pvm_spawn: O argumento 1 indica o nome do arquivo disparado. O terceiro argumento(flag)
    indica opções de disparo (0 = o PVM decide em quem disparar a tarefa; 1 = o argumento seguinte será o nome
    indicado). Se o terceiro e quarto argumentos forem, 0 e Null respectivamente, o PVM dispara em toda máquina
    virtual; Ex: numt = pvm_spawn ( "escravo1", NULL, 1, "labsister02.em.ufop.br", 1, &tid ); */
    for(int conta=0; conta<3; conta++)
    {
        numt = pvm_spawn ( "escravo", NULL, 0, NULL, 1, &tid ); //Dispara todos os processos da
        rede(programas-escravo)
    :
        numt = pvm_recv ( -1, -1 ); // Recebe mensagens enviadas de qualquer processo
        pvm_buinfo ( numt, ( int* ) 0, ( int* ) 0, &tid );
        pvm_upkstr ( buf ); // Descomapcta mensagem recebida
        printf ("Sim, tarefa disparada = %x. Eu sou o escravo %s\n", tid, buf );
        cout << endl;
    }
    cout << endl << "número de tarefas disparadas = " << numt << endl;

```

Antes de compilar o programa principal, deve-se também referenciar (“linkar”) as bibliotecas *libpvm3.a* e *libgpvm3.a*. Elas são essenciais para a execução do programa paralelo, sem as quais ele não funcionaria. Para isto, deve-se alterar as opções de projeto do *kdevelop*²⁶, dirigindo-se ao menu projeto/opções de configuração, inserir no campo LDFLAGS o comando “-L” (biblioteca, *library*), seguido do diretório onde está a biblioteca em questão entre colchetes:

`< -L caminho_do_arquivo -L caminho_do_arquivo ... >`

Para o *kdevelop*, as bibliotecas referentes à linguagem C/C++ são referenciadas como:

`< -L /usr/lib/pvm3/lib/LINUX64/libpvm3.a -L /usr/lib/pvm3/lib/LINUX64/libgpvm3.a >`

Basta ao usuário agora compilar e construir o projeto (*build*). Vale lembrar que caso o usuário escolha utilizar alguma outra distribuição diferente do *Mandriva Linux*, os diretórios onde estão contidas as bibliotecas podem ser diferentes das mostradas, devendo-se assim adequá-las ao caso.

9.3 Programa escravo “*escravo1.cpp*”

A programação para os escravos ocorre de maneira similar ao mestre (figura 9.5), porém salienta-se que ela deve ser feita em separado, criando-se outro projeto no *kdevelop* (pode ser executada ao mesmo tempo em outra área de trabalho do *KDE*). Após escrito o programa e referenciadas as bibliotecas, deve-se ainda copiar os executáveis referentes aos escravos, geralmente em `</home/nome_do_usuário/diretório_do_escravo/debug/src>`, para as pastas nas quais o pvm irá procurar caso necessário. No caso mostrado, o diretório do executável se encontra em:

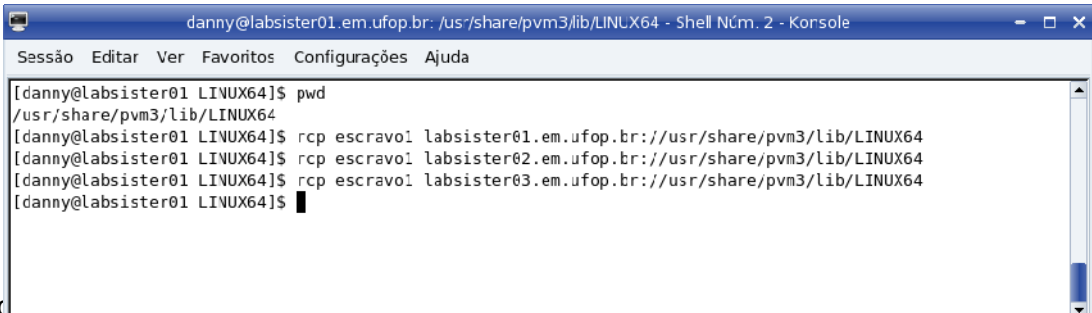
`</home/danny/escravo1/debug/src>`

²⁶ *kdevelop*: ambiente de desenvolvimento do KDE.

E o diretório de destino será:

`</usr/share/pvm3/lib/LINUX64>`

A cópia dos arquivos pode ser realizada manualmente (CTRL+C / CTRL+V) ou via comando remoto (`< rcp nome_do_arquivo caminho_de_destino >`), e o procedimento é mostrado na Fig. 9.4:



```

[danny@labsister01 LINUX64]$ pwd
/usr/share/pvm3/lib/LINUX64
[danny@labsister01 LINUX64]$ rcp escravo1 labsister01.em.ufop.br:/usr/share/pvm3/lib/LINUX64
[danny@labsister01 LINUX64]$ rcp escravo1 labsister02.em.ufop.br:/usr/share/pvm3/lib/LINUX64
[danny@labsister01 LINUX64]$ rcp escravo1 labsister03.em.ufop.br:/usr/share/pvm3/lib/LINUX64
[danny@labsister01 LINUX64]$

```

```

#ifdef __cplusplus
#include <config.h>
#endif
#include <iostream>
#include <cstdlib>
#include <pvm3.h>
using namespace std;

main() // programa principal
{
    int ptid; // variável onde será guardada o tid do mestre
    int msgtag = 0;
    int info = 0;
    char buf[100];

    ptid = pvm_parent(); // Obtem o tid do mestre
    strcpy(buf, "");
    gethostname(buf + strlen(buf), 64);
    pvm_initsend(PvmDataDefault); /* Se prepara para compactar uma nova
mensagem. O argumento inteiro especifica o esquema de codificação da
mensagem.*/
    pvm_pkstr(buf); // Compacta o buffer(do tipo string) para envio da
mensagem
    pvm_send(ptid, 3); // envia a mensagem ao mestre com uma tag de valor
3

    pvm_exit();

    exit(0);
}

```

Figura 9.5 - Programa “*escravo1.cpp*”.

9.4 Máquina Virtual

Após feita a programação do mestre e escravo(s), com os executáveis-escravo corretamente copiados para as pastas do PVM correspondentes (figura 9.4), basta compilar, criar a

aplicação (*build*) e executar o programa. Porém antes de executá-lo é necessário criar a máquina virtual: para isto, deve-se executar a aplicação *pvm*, entrando em execução o *daemon pvmd*, responsável pelo gerenciamento das rotinas de distribuição de processamento.

Uma aplicação paralela em *PVM* é feita executando-se apenas o programa mestre, que se encarregará em “disparar”(com a função *pvm_spawn*) os programas escravos. Pode-se executá-lo a partir do ambiente *kdevelop* ou dirigindo-se até o diretório onde está gravado o executável (Figura 9.6).

```

danny@labsister01.em.ufop.br: /home/danny/conversa/debug/src - Shell - Konsole
Sessão Editar Ver Favoritos Configurações Ajuda

[danny@labsister01 src]$ pwd
/home/danny/conversa/debug/src
[danny@labsister01 src]$ ./conversa
*****
*                                *
*      Ola Escravos! Voces estao ai?      *
*              Eu sou BEOWULF!!              *
*                                *
*****

Minha tarefa é = 40004

Sim, tarefa disparada = 80002. Eu sou o escravo labsister03.em.ufop.br
Sim, tarefa disparada = c0002. Eu sou o escravo labsister02.em.ufop.br
Sim, tarefa disparada = 40005. Eu sou o escravo labsister01.em.ufop.br

[danny@labsister01 src]$

```

Figura 9.6 - Executando o programa “conversa”.

Em suma, o programa “conversa” faz com que o mestre primeiramente se identifique na rede e exiba o número de sua tarefa correspondente (função *pvm_mytid()*). Em seguida, dispara os programas contidos nos escravos da rede pela função *pvm_spawn()* e solicita que também se identifiquem (compacta um *string* com a função *pvm_pkstr()*, envia a mensagem para os escravos com função *pvm_send()* e espera a resposta).

Sempre que houver uma função de envio ou recebimento no mestre (figura 9.3), deverá haver uma função correspondente (e igualmente oposta) nos escravos (figura 9.5) e vice-versa. Antes de enviar qualquer mensagem ao processo de destino, deve-se compactá-la usando uma função coerente ao tipo de dado enviado (*string*, *integer*, *float* etc) (ANEXO A). O mesmo acontece no recebimento, aonde após receber a mensagem, deve-se descompactá-la antes de sua utilização. Atentar para os nomes das variáveis escolhidas, pois estas devem ser as mesmas tanto no mestre quanto nos escravos. Este é o modelo de programação PVM, no qual todo o paralelismo é devido a uma série de funções de troca de mensagens.

X APLICAÇÃO: CÁLCULO DA CONSTANTE π

10.1 O cálculo de π

Um programa bastante utilizado em teste e análise de desempenho na área da supercomputação e computação paralela em *Clusters* é o da integração numérica para o cálculo da constante π (equação 10.1). Seu objetivo principal é observar o comportamento do sistema como um todo, desde a comunicação entre processadores – na qual analisa-se a correção do cálculo mediante a um resultado esperado – até a análise de desempenho, observando-se o tempo de resposta mediante a uma ação requerida.

$$\pi = 4 \int_0^1 \frac{1}{1+x^2} dx \quad (10.1)$$

Segundo o teorema fundamental do cálculo, a integral de uma função contínua qualquer corresponde à área abaixo da curva definida por ela dentro do intervalo desejado (figura 10.1). O valor de π é dado portanto pela área abaixo da curva definida pela função. Como se trata de uma dízima infinita, o intervalo de integração definirá a precisão do resultado, delimitado apenas pelos limites físicos do sistema de computação em questão, como memória, tipos de dados usados na programação, versão do *PVM* etc.

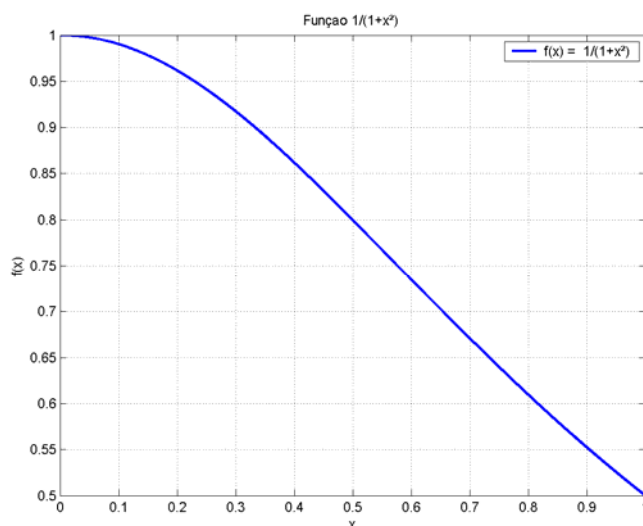


Figura 10.1 - Área de integração para o cálculo de π .

10.2 Programa mestre “*calcular.cpp*”

Na Fig. 10.2 mostra-se o programa “mestre” para o cálculo da constante π . É ele quem faz necessariamente a distribuição do processamento: sua primeira ação é determinar o número de escravos (*hosts*) presentes na rede e obter, mediante ação do usuário, o número de pontos nodais (que gerará os intervalos de integração infinitesimais) desejados para o cálculo. Após isso, faz uma varredura (*loop For*) em todos os *hosts* (inclusive ele mesmo!) no intuito de disparar os processos contidos em cada um.

Para cada processo (programa escravo) disparado, o mestre determina o passo da integração (Δx) e limites de integração de cada um antes de enviar a mensagem. Como há neste caso três escravos de igual capacidade, cada um receberá um terço do limite de integração total (balanceamento de carga), facilitando o trabalho do mestre e minimizando trocas de mensagens desnecessárias.

Após o envio dos dados que serão usados pelos escravos em seus cálculos, o mestre espera, da parte deles, uma resposta positiva com seus respectivos resultados parciais. Somente aí então o mestre (que não tem mais trabalho algum além deste) poderá computar os resultados parciais enviados pelos *hosts* e exibir o resultado final: o valor da constante π . Também é feita uma última comparação com uma constante π previamente definida, na intenção de obter-se o erro de cálculo para aquele número de pontos desejado.

Apesar do *Cluster* possuir um total de 8 processadores, foi disparado – para efeito didático – apenas um processo (de mesmo nome: “*calcularescravo*”) em cada computador, totalizando 3 núcleos de processadores em uso.

```

/*****
* MONOGRAFIA DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO
* UNIVERSIDADE FEDERAL DE OURO PRETO - ESCOLA DE MINAS
* MANUAL DE MONTAGEM DE UM CLUSTER BEOWULF SOB A PLATAFORMA GNU/LINUX
*
* Programa que calcula o Valor de PI em um ambiente paralelo - Cluster Beowulf
*
* Copyright (C) 2008 by Danny Tonidandel
*****/
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif
#include <iostream>
#include <cstdlib>
#include <stdlib.h>
#include <cmath> // Biblioteca de funções matemáticas ao estilo C++
#include <cstring> // Biblioteca de manipulação de strings ao Estilo C
#include <string> // Biblioteca de manipulação de strings ao estilo C++

```

```

#include<pvm3.h> /* Arquivo de cabeçalho com os protótipos de funções da biblioteca pvm. Deve ser
referenciada como: < -L /usr/lib/pvm3/lib/LINUX64/libpvm3.a -L/usr/lib/pvm3/lib/LINUX64/libgpvm3.a> nas
opções de configuração do kdevelop. */
using namespace std;
int main(int argc, char *argv[]) // Programa principal
{
    int numt, tid;
    double PI_PARCIAL = 0; double PI_FINAL = 0;
    double l2 = (1.0/3.0); double l1 = 0.0; // limites de integração (total de 0 a 1, 1/3 para cada nodo)
    double PI = dacos(-1); // Constante pi para comparação e calculo do erro
    int N = 0; // fornecerá o número regiões (espaçamento) de integração
    int info2; int num = 0;
    struct pvmhostinfo *hostp; // Struct que obtêm informações sobre os escravos da rede.
    int info, i, nhost, narch;
    info = pvm_config(&nhost, &narch, &hostp);
    cout << "Numero de escravos no cluster = " << nhost << endl << endl << endl;
    cout << "*****" << endl;
    cout << "                Ola Escravos! Eu sou BEOWULF!!                *" << endl;
    cout << "                Calculem o valor de PI!!                *" << endl;
    cout << "*****" << endl; cout << endl << endl;
    printf ( "Minha tarefa é = %x\n", pvm_mytid() ); // Exibe a tid do mestre  cout << endl;
    cout << "Numero de escravos no cluster = " << nhost << endl << endl;
    cout << "Entre com o numero de divisoes na regioao de integracao";
    cin >> num; cout << endl;
    for (i=0; i < nhost; i++) // Varre escravos da rede (o mestre também é um dos escravos!!)
    {
        //----- Dispara o processo indicado na função pvm_spawn -----
        numt = pvm_spawn ( "calcularescravo", NULL, 1, hostp[i].hi_name, 1, &tid ); /*Dispara processo no
escravo(i)*/
        //----- Número de áreas de integração a ser enviado aos escravos -----
        if (strcmp(hostp[i].hi_name, "labsister03.em.ufop.br")== 0 ) N = (int)(0.33*num); //Core2Quad
        if (strcmp(hostp[i].hi_name, "labsister02.em.ufop.br")== 0 ) N = (int)(0.33*num); //AthlonX2
        if (strcmp(hostp[i].hi_name, "labsister01.em.ufop.br")== 0 ) N = (int)(0.34*num); //Core2Duo
        // ----- Envia limites da integral aos escravos -----
        pvm_initsend(PvmDataDefault); // Prepara-se para envio de dados
        pvm_pkdouble(&l1, 1, 1); // Compacta o Limite de integração l1 (do tipo double)
        pvm_pkdouble(&l2, 1, 1); // Compacta o Limite de integração l2 (do tipo double)
        pvm_pkint(&N, 1, 1); // Compacta um inteiro dentro da mensagem
        info2 = pvm_send(tid, 2); // Envia a mensagem ao processo disparado com uma tag=2
        if (info2>=0) cout << "Envio de dados -- ok" << endl; // Está tudo ok? Confirme.
        // ----- Recebe mensagem (dados vindos dos escravos) -----
        numt = pvm_recv ( -1, 3 ); /* Recebe mensagem de qualquer processo com tag==3 */
        // ----- Descompacta mensagem (dados recebidos) -----
        // Todas as funções de compactação e descompactação exigem que o 1o argumento seja um ponteiro
        pvm_upkdouble(&PI_PARCIAL, 1, 1); // descompacta a variável com o resultado parcial
        pvm_upkint(&N, 1, 1); // Descompacta variável "N"(número de pontos nodais)
        // ----- Computa resultados e Exibe o número de pontos calculados pelo escravo[i] -----
        cout << endl << "pontos nodais em " << hostp[i].hi_name << " = " << N;
        PI_FINAL += PI_PARCIAL; // Soma os resultados parciais na variável PI_FINAL;
        cout << endl << "resultado parcial[" << (i+1) << "] = " << PI_PARCIAL << endl;
        cout << "Do escravo: " << hostp[i].hi_name << endl;
    }
    cout << endl;
    // ----- Exibe valor de PI e compara com um valor calculado -----
    printf("\aPI vale aproximadamente %.15f\nCom erro absoluto = %.7f\n", PI_FINAL, fabs(PI_FINAL-PI));
    pvm_exit(); // Encerra o pvm.
    return EXIT_SUCCESS;
}

```

Figura 10.2 - Programa “calcular.cpp”.

10.3 Programa “*calcular_escravo.cpp*”

O programa destinado aos escravos – “*calcular_escravo*” (Figura 10.3) – funciona de maneira similar ao que já foi mostrado até o momento (Figura 9.5). Ele possui, na mesma ordem e de maneira complementar, funções de recebimento e envio de mensagens sincronizadas com o programa mestre. Após disparado, o processo escravo aguarda o recebimento dos dados, faz a integração nos limites especificados e guarda o resultado parcial em uma variável a ser enviada ao processo “pai”.

```

/*****
 * MONOGRAFIA DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO
 * UNIVERSIDADE FEDERAL DE OURO PRETO - ESCOLA DE MINAS
 * MANUAL DE MONTAGEM DE UM CLUSTER BEOWULF SOB A PLATAFORMA GNU/LINUX
 * Programa que calcula o Valor de PI em um ambiente paralelo – Cluster Beowulf
 *****/

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif
#include <iostream>
#include <cstdlib>
#include <stdlib.h>
#include <pvm3.h> // Arquivo de cabeçalho com os protótipos de funções da biblioteca pvm
#include <cmath> // Biblioteca de funções matemáticas ao estilo C++
#include <cstring> // Biblioteca de manipulação de strings ao Estilo C
#include <string> // Biblioteca de manipulação de strings ao estilo C++
#define f(x) (1.0/(1.0 + x*x)) // integrando
using namespace std;

int main(int argc, char *argv[])
{
// ----- Inicialização de variáveis -----
    int ptid; // Variável que guardará a tid do computador mestre
    int info = 0;
    double PI_PARCIAL = 0;
    ptid = pvm_parent(); // Obtem o tid do mestre e guarda em ptid
// ----- Recebe dados vindos do mestre -----
    double l2 = 0.0; double l1 = 0.0; // limites de integração a serem recebidos pelo mestre
    int N = 0; // Numero de pontos
    int num, ltotal, nprocs;
    pvm_recv(ptid, 2); // Recebe a mensagem de tag=2 do
    pvm_upkdouble(&l1, 1, 1); // Descompacta o Limite de integração L1
    pvm_upkdouble(&l2, 1, 1); // Descompacta o Limite de integração L2
    pvm_upkint(&N, 1, 1); // Descompacta inteiro com numero dePontos
// ----- Programa Principal -----
    double pi_local = 0.0;
    double passo = 0.0;
    for (int k=0; k < N; k++)
    {
        pi_local += (double) f((0.5 + k)/(N)); // Calcula a área
    }
    passo = (double) ((l2-l1)/N); // intervalo de integração proporcional a 1/N
    pi_local *= (double) ((4.0)*(passo));
// ----- Envio dos dados para o mestre -----
    PI_PARCIAL = pi_local; // guarda resultado a ser enviado ao processo pai
    pvm_initSend(PvmDataDefault); // prepara-se para envio
    pvm_pkdouble(&PI_PARCIAL, 1, 1); // compacta resultado parcial
    pvm_pkint(&N, 1, 1); // compacta número de pontos nodais feito por ele

```



```

        pvm_send(ptid, 3); /* Envia a mensagem ao processo mestre com uma tag==2 */
// ----- Fim do processo -----
pvm_exit(); // encerra o pvm
return EXIT_SUCCESS;
}

```

Figura 10.3 - Programa “*calcularescravo.cpp*”.

O procedimento para a execução do programa é idêntico ao mostrado para o programa “*conversa*”: compilar o programa mestre e escravos, copiar os executáveis-escravo para a biblioteca pvm (figura 10.4), criar a máquina virtual (seção 8.9), localizar o diretório do programa mestre (Figura 10.5) e finalmente executá-lo digitando *./nome_do_programa* (Figura 10.6).

```

danny@labsister03 ~]$ pwd
/home/danny
[danny@labsister03 ~]$ ls
Área de Trabalho/ calcularescravo/ Download/ Música/ teste/
bin/ conversa/ escravol/ Piserial/ tmp/
calcular/ Documentos/ Imagens/ tempos.ods* Vídeos/
[danny@labsister03 ~]$ cd calcularescravo/debug/src
[danny@labsister03 src]$ ls
calcularescravo* calcularescravo.lo calcularescravo.o Makefile
[danny@labsister03 src]$ rcp calcularescravo labsister03.em.ufop.br:/usr/shar
vm3/lib/LINUX64
[danny@labsister03 src]$ rcp calcularescravo labsister02.em.ufop.br:/usr/shar
vm3/lib/LINUX64
[danny@labsister03 src]$ rcp calcularescravo labsister01.em.ufop.br:/usr/shar
vm3/lib/LINUX64
[danny@labsister03 src]$ █

```

```

[danny@labsister03 ~]$ pwd
/home/danny
[danny@labsister03 ~]$ ls
Área de Trabalho/  calcularescravo/  Download/  Música/  teste/
bin/               conversa/         escravol/  Piserial/  tmp/
calcular/          Documentos/       Imagens/   tempos.ods*  Vídeos/
[danny@labsister03 ~]$ cd calcular/debug/src
[danny@labsister03 src]$ ls
calcular*  calcular.lo  calcular.o  Makefile
[danny@labsister03 src]$ ./calcular

```

Figura 10.5 - Localizando o programa “calcular”.

```

[danny@labsister03 src]$ ./calcular
Numero de escravos no cluster = 3

*****
*                                     *
*      Ola Escravos! Eu sou BEOWULF!!      *
*      Calculem o valor de PI!!             *
*                                     *
*****

Minha tarefa é = 40006

Numero de escravos no cluster = 3

Entre com o número de iterações: 150000000

Iterações em labsister03.em.ufop.br = 49500000
resultado_parcial[1] = 1.047197551
Do escravo: labsister03.em.ufop.br

Iterações em labsister01.em.ufop.br = 51000000
resultado_parcial[2] = 1.047197551
Do escravo: labsister01.em.ufop.br

Iterações em labsister02.em.ufop.br = 49500000
resultado_parcial[3] = 1.047197551
Do escravo: labsister02.em.ufop.br

PI vale aproximadamente 3.141592653589571
Com erro absoluto = 0.000000000000221
[danny@labsister03 src]$

```

Figura 10.6 - Executando o programa “calcular”.

XI ANÁLISE DE DESEMPENHO

11.1 Medida de desempenho com o *benchmark* “*calcular*”

Para melhorar o desempenho de um *software* executado em um determinado *hardware* é necessário conhecer quais fatores do *hardware* em questão afetam o desempenho global do sistema. Além disso, deve-se determinar a importância relativa de cada um. Entre os fatores do *hardware* que têm maior influência no desempenho podemos citar:

- A ação do compilador na geração do código de máquina;
- A execução de instruções (arquitetura do processador);
- Comportamento dinâmico da memória;
- Comportamento dinâmico dos dispositivos de entrada e saída;

Em um *software* paralelo, fatores como o balanceamento de cargas e os modos de operação do sistema de troca de mensagens (*PVM*) são os grandes “gargalos” para o desempenho final do processamento. Cabe ao programador identificá-los, assim como reduzir o número total de trocas de mensagens. A compreensão de como determinar o impacto no desempenho de cada um destes fatores é importante para aumentar a eficiência dos programas projetados.

Existem várias maneiras de se avaliar o desempenho global em sistemas paralelos. Uma delas consiste basicamente em adotar como sistema de melhor desempenho aquele que executa em menor tempo um determinado programa, ou ainda, aquele sistema que executa o maior número de programas em um determinado intervalo de tempo. Dessa forma, estamos considerando dois aspectos diferentes na avaliação de desempenho: o tempo de resposta ou tempo de execução da aplicação e o denominado “*throughput*”, que considera a quantidade de trabalho executado na unidade de tempo.

Como o processador pode trabalhar em vários programas simultaneamente, o desempenho melhora quando o *throughput* é aumentado. Dessa forma, muitas vezes é interessante distinguir entre o tempo total de execução de um programa e o tempo gasto pelo processador num programa em particular.

A aplicação proposta para o cálculo de π pode ser útil na análise de desempenho do *Cluster Beowulf*. Ele se encaixa na categoria dos “*toy benchmarks*”, programas de 10 a 100 linhas de código que geram um resultado previamente conhecido.

Nesta avaliação utilizou-se como ferramenta de auxílio à coleta de dados (de desempenho) um pacote chamado *timer*: um pequeno software que executa o programa desejado quantas vezes se queira, registra os tempos de execução e calcula sua média ao final do processamento. Sua sintaxe é bastante simples:

timer [-n número] [-avg] <nome_do_programa>

Exemplo:

timer -n 100 -avg ./calcular

Neste caso, o *timer* executa cem (-n 100) vezes o programa “*calcular*”, exibindo ao final a média dos tempos obtidos (-avg).

Na análise descrita, o programa “*calcular*” foi executado 100 vezes para cada número de pontos (de 100 até 1×10^8 pontos nodais), computando-se a média dos tempos obtidos. Foram realizadas duas baterias de experimentos: uma para o programa em série (um processador) e outra para o programa paralelo, todas com o auxílio do pacote *timer* (Figura 11.1).

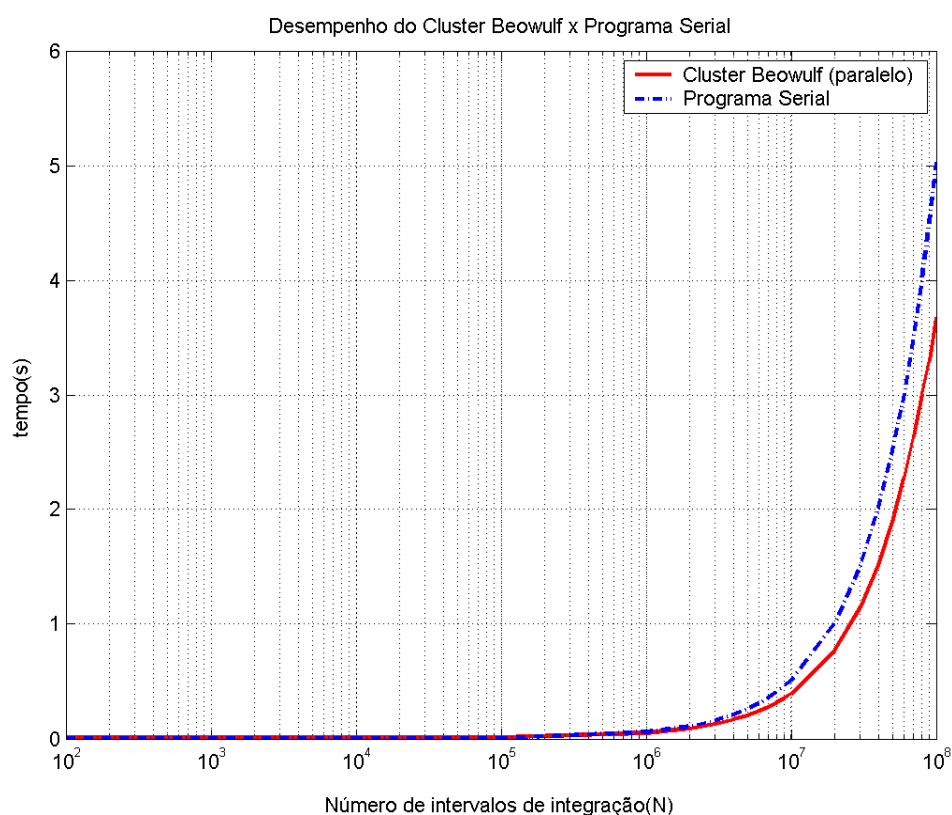


Figura 11.1 - Desempenho do programa paralelo versus programa serial.

Com o aumento do número de pontos calculados, chega um momento no qual ocorre uma inversão no desempenho dos dois sistemas: o programa executado no *Cluster* começa a ser mais rápido que o programa em série (Figura 11.2).

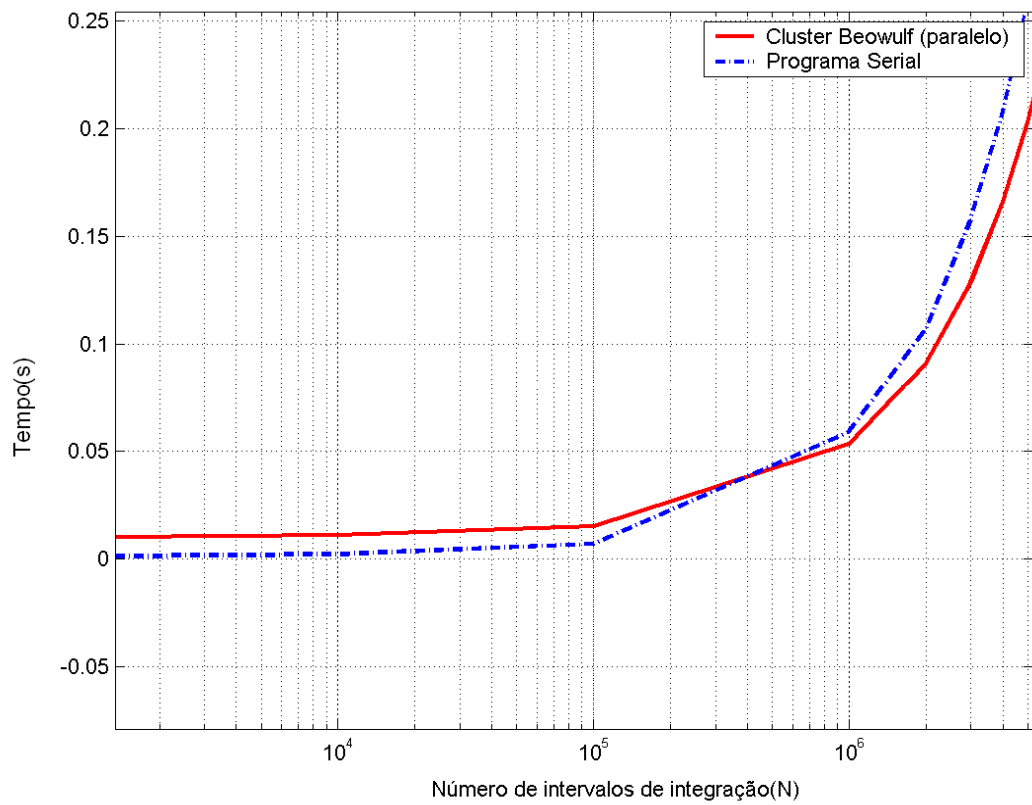


Figura 11.2 - Região na qual o programa paralelo começa a ser mais rápido.

É verdade que um tempo final de processamento da ordem de 1 segundo é muito pequeno. Também é verdade que a análise do processamento paralelo em pouco tempo esconde a real potencialidade do paradigma como um todo. Mas ao observar-se que na maioria das aplicações existentes o processamento pode levar dias e até semanas de execução, já é possível ter uma idéia da eficiência de um *Cluster Beowulf*. Em um programa similar ao da Fig. 11.2, caso a exigência seja tal que obrigue um computador serial a trabalhar durante 2 semanas, o mesmo programa, executado em um *Cluster* similar ao descrito, executaria o processamento em pouco mais de 9 dias! São quase 5 dias de diferença entre eles.

Pode-se observar no entanto que o programa descrito apresenta uma baixa eficiência: cerca de 47,7%. Isto se deve ao fato de não haver tráfego de mensagens e processamento paralelo na estrita acepção da palavra: ocorre apenas uma distribuição, por parte do mestre, de tarefas entre os escravos da rede, que as executam de maneira independente. Em um programa paralelo real, ocorre tráfego intenso de dados e cálculo simultâneo, onde muitas das vezes, os diferentes nós da rede necessitam de dados vindos de outros para continuarem o processamento.

Observa-se que para fins didáticos e de testes, a execução do programa para cálculo de π cumpre os requisitos de funcionalidade. Além do mais, é possível estender-se o processamento para até 8 processadores, bastando para isso disparar mais de um processo em cada escravo. Tal ação pode acarretar em um aumento de velocidade final.

É claro que este aumento de velocidade não é ilimitado e tampouco linear: aumentar o número de processadores não significa que o tempo de processamento cairá na mesma proporção. Chegará o momento em que o tempo gasto na divisão dos trabalhos e troca de mensagens tornará sua execução inviável. Este número é diferente em cada aplicação, ou seja, o grau de paralelismo alcançável dependerá de características inerentes ao problema em questão e não é objetivo principal deste trabalho determiná-lo.

11.2 Precisão e convergência

Um fato interessante na análise de desempenho concerne à convergência e precisão de cálculo. Assim como os tempos de cálculo gastos pelos sistemas em série e em paralelo, a precisão do cálculo em comparação à uma constante PI predefinida, foi diferente para os dois sistemas em uma faixa de pontos determinadas (Figura 11.3).

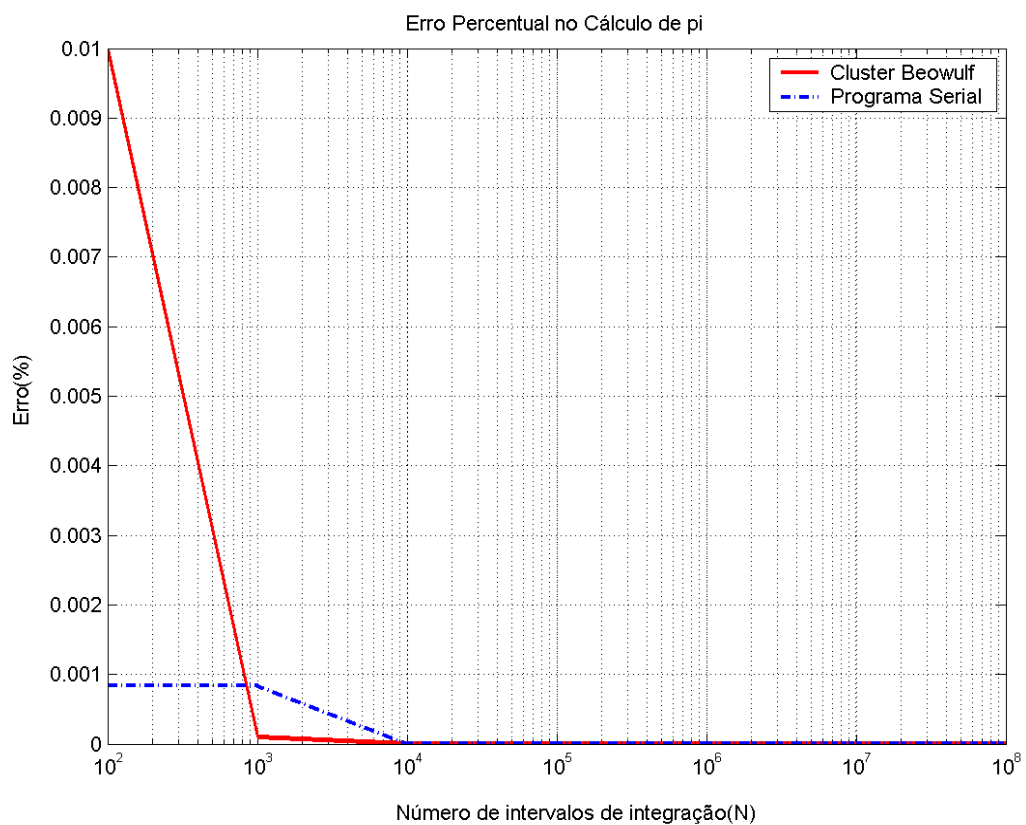


Figura 11.3 - Convergência do erro no cálculo de pi

Nota-se que o programa paralelo converge mais rapidamente para o valor desejado do que o programa em série, o que não esperava-se a princípio, já que o código é o mesmo. E que com um número baixo de pontos nodais, o programa paralelo apresentou um grande erro percentual no cálculo do valor de PI.

No entanto, à medida que as áreas de integração diminuem em tamanho (maior divisão dos espaços), os valores encontrados da constante PI nos dois casos começam a tender para um mesmo valor (Figura 11.4).

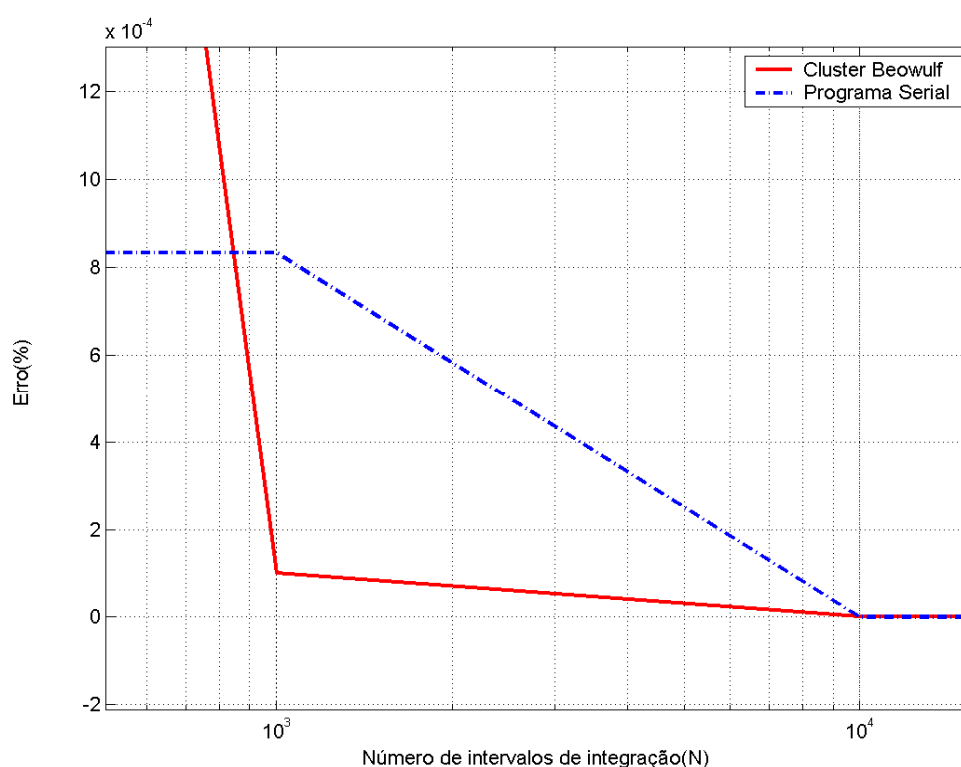


Figura 11.4 - Ampliação da figura 11.3

Um segundo fator constatado, voltado agora aos sistemas de trocas de mensagens como o *PVM* e *MPI*, diz respeito à limitação dos tipos de dados e variáveis suportadas pelo PVM: o tipo de dados utilizado para gravação dos resultados parciais e trânsito ao longo da máquina virtual foi o “*double*” (em C++). A versão do compilador C++ utilizada não suporta variáveis do tipo “*long*”, que possibilitariam alcançar uma maior precisão (mais casas decimais) nos cálculos.

XII CONSIDERAÇÕES FINAIS

O projeto de um Cluster Beowulf envolve diversos desafios. Em se tratando da elaboração de um tutorial, o principal cuidado que se teve foi de fazê-lo o mais didático possível sem torná-lo, no entanto, redundante nas informações apresentadas.

A escolha da distribuição *GNU/Linux* foi um capítulo à parte e que exigiu grande atenção. Foi uma das etapas que demandou maior tempo na fase preliminar, mas se mostrou, no decorrer dos trabalhos, uma escolha satisfatória; cumprindo os requisitos de desempenho e estabilidade esperados de uma plataforma *Linux*.

A montagem física do *Cluster Beowulf* foi por outro lado tarefa bastante simples, ao contrário de sua configuração, com a instalação dos pacotes requeridos, a configuração dos servidores de acesso remoto e do próprio *PVM*. A principal dificuldade enfrentada nesta etapa consistiu em encontrar trabalhos correlatos na literatura, já que trata de um trabalho de natureza técnica.

Após realizadas as etapas de montagem e configuração, ficou evidente que o conceito chave da programação paralela é a divisão de tarefas. Mas qual o limite? Se um operário constrói uma casa em um ano, dois operários realizarão o mesmo trabalho em seis meses. Então pode-se concluir que 365 operários gastariam apenas 1 dia para concluir o trabalho?

Obviamente haverá um limite, pois o trabalho dos operários só seria eficiente se os mesmos estivessem perfeitamente equilibrados e sincronizados. No primeiro caso, todos eles teriam a mesma carga de trabalho, executando 0,27% da construção. Ou então, havendo certa especialização, alguns cuidariam do cimento, encanamento, outros da parte elétrica etc.

Com isso, ao imaginar-se todas as tarefas necessárias para uma construção fica evidente que algumas delas não podem ser “paralelizadas”, ou seja, seria um absurdo admitir 365 operários para assentarem uma porta ou em cima do telhado! A construção não suportaria o peso e acabaria por ruir-se.

Deve existir portanto um limite para o número de operários (processadores) trabalhando em paralelo, e quanto mais este limite seja ultrapassado, tanto pior será o desempenho da construção como um todo, podendo torná-la inviável.

Têm-se então dois grandes problemas: o quanto uma tarefa pode ser “paralelizada” e quantos processadores devem ser alocados. Além disso, o problema central na programação paralela está em escolher e adaptar procedimentos numéricos que sejam eficientes e que se adequem a

este tipo de paradigma computacional, ou seja, sincronizar o trabalho dos processadores para evitar redundância e garantir um balanceamento de carga eficiente de acordo com a capacidade de cada um. Em resumo, um desafio maior que o projeto de sistemas paralelos como o *Cluster Beowulf* é sua utilização ótima.

XIII REFERÊNCIAS BIBLIOGRÁFICAS

GEIST, A. et al. **PVM: Parallel Virtual Machine: a user's Guide and Tutorial for Networked Parallel Computing**. Cambridge: MIT Press, 1994. Disponível em <www.netlib.org/pvm3/book/pvm-book.html>. Acesso em: 14 dez. 2007, 13:03.

PVM man pages. **Commands and Library Calls**. Disponível em <www.csm.ornl.gov/pvm/manpages.html>. Acesso: 21 de jan. 2008, 19:27.

WELSH, M.; KAUFMAN, L. **Running Linux**. 1. ed. United States of America: O'Reilly & Associates, Inc, 1995. ISBN 1-56592-100-3.

WARREN, M. **The Avalon Beowulf Cluster: A Dependable Tool for Scientific Simulation**. Los Alamos National Laboratory, 2002. Session K6.005 - PC Clusters for Computational Science Theory and Practice. Archives of the Bulletin of the American Physical Society. Disponível em <flux.aps.org/meetings/YR00/MAR00/abs/S4050.html#SK6.005>. Acesso em: 04 dez. 2007, 22:48.

BROWN, R. G. **What makes a Cluster Beowulf?** Duke University Physics Department, 2006. Disponível em <www.beowulf.org/overview/howto.html>. Acesso em: 15 Fev. 2008, 11:24.

TURNER, D. **Scientific Applications on Workstation Clusters vs Supercomputers**. Ames Laboratory, 2002. Session K6.004 - PC Clusters for Computational Science Theory and Practice. Archives of the Bulletin of the American Physical Society. Disponível em <flux.aps.org/meetings/YR00/MAR00/abs/S4050.html#SK6.004>. Acesso em: 04 dez. 2007, 22:55.

BUENO, A.D. **Introdução ao Processamento Paralelo e ao Uso de Clusters de Workstations em Sistemas GNU/LINUX**. UFSC, 2002. disponível em: <www.rau-tu.unicamp.br/nou-rau/softwarelivre/document/stats.php>. Acesso em: 27 Jan. 2008, 16:03.

CRUZ, A. O; SILVA, G. P. **Programação Paralela e Distribuída: Um curso Prático**. Rio de Janeiro : UFRJ. progpar.pdf. Disponível em <equipe.nce.ufrj.br/gabriel/progpar/>. Acesso: 07 Jan. 2008, 13:21.

DEITEL, H. M.; DEITEL, P. J. **C++: como programar**. 3. ed. Porto Alegre: Bookman, 2001. ISBN 85-7307-740-9.

SILVA, J. E. da. **História do GNU/Linux: 1965 assim tudo começou!** Fórum da comunidade Viva o Linux, 2006. Disponível em <www.vivaolinux.com.br/artigos/verArtigo.php?codigo=4409>. Acesso em 13 Fev. 2008, 15:01.

ALECRIM, E. **A história do Tux, o pingüim-símbolo do Linux**. Infowester: Revista Eletrônica, 2004. Disponível em <<http://www.infowester.com/tux.php>>. Acesso em: 13 Fev. 2008, 15:06.

MORIMOTO, C. E. **Brincando de Cluster**. Artigo do fórum guia do Hardware, 2003. Disponível em < www.guiadohardware.net/artigos/cluster/>. Acesso: 12 Fev. 2008, 14:13.

TILIO, G. **Cluster de videogames PlayStation3[®] da Unicamp**. Rio de Janeiro : Portal G1, 2008. Disponível em <g1.globo.com/noticias/ciencia/>. Acesso: 20 Fev. 2008, 10:00.

ANEXOS

ANEXO A - DESCRIÇÃO DAS FUNÇÕES DA BIBLIOTECA PVM

A.1 Classificação

Nesta seção é mostrada uma breve listagem das subrotinas, comandos e funções da biblioteca PVM, contidas em *libpvm3.a* (C/C++) e *libfpvm3.a* (Fortran) (GEIST et al., 1994). A descrição completa de todas as funções da biblioteca *PVM*, com sintaxe e exemplos de aplicação pode ser encontrada nas “PVM Man pages”, *site* oficial da *netlib*, de domínio público (PVM..., 2008).

As subrotinas podem ser divididas em 5 classes:

Troca de mensagens: *pvm_bufinfo*, *pvm_freebuf*, *pvm_getrbuf*, *pvm_getsbuf*, *pvm_initsend*, *pvm_mcast*, *pvm_mkbuf*, *pvm_nrecv*, *pvm_pack*, *pvm_prerecv*, *pvm_probe*, *pvm_psend*, *pvm_recv*, *pvm_recvf*, *pvm_send*, *pvm_sendsig*, *pvm_setmwid*, *pvm_setrbuf*, *pvm_setsbuf*, *pvm_trecv*, *pvm_unpack*.

Controle de processos: *pvm_exit*, *pvm_kill*, *pvm_mytid*, *pvm_parent*, *pvm_pstat*, *pvm_spawn*, *pvm_tasks*;

Mensagens de grupos: *pvm_barrier*, *pvm_bcast*, *pvm_gather*, *pvm_getinst*, *pvm_gettid*, *pvm_gsize*, *pvm_joiningroup*, *pvm_lvgroup*, *pvm_reduce*, *pvm_scatter*;

Controle da máquina virtual: *pvm_addhosts*, *pvm_config*, *pvm_delhosts*, *pvm_halt*, *pvm_mstat*, *pvm_reg_host*, *pvm_reg_rm*, *pvm_reg_tasker*, *pvm_start_pvm*;

Miscelânea: *pvm_archcode*, *pvm_catchout*, *pvm_getopt*, *pvm_hostsync*, *pvm_notify*, *pvm_perror*, *pvm_setopt*, *pvm_setmask*, *pvm_tidtohos*;

ANEXO B - COMANDOS DO CONSOLE PVM

B.1 Listagem dos comandos

Nesta seção são descritos os comandos do console *PVM*, representado pelo sinal de prontidão “*pvm>*” após o *daemon pvm* entrar em execução. É a partir do console que se gerencia toda a criação e funcionamento da máquina virtual.

<i>pvm</i>	Inicia o daemon <i>pvm</i> , criando a máquina virtual(comando do console Linux).
<i>help</i>	Apresenta uma lista semelhante a esta. O comando <i>help</i> pode ser seguido de um nome de comando, quando então lista as opções disponíveis para este comando
<i>delete</i>	Remover nós da máquina virtual
<i>add</i>	seguido do um ou mais nomes de <i>hosts</i> , adiciona estes <i>hosts</i> à máquina virtual(cria um nó).
<i>reset</i>	Matar todos os processos da MV exceto a(s) consoles
<i>quit</i>	Sair da console deixando o <i>pvm</i> executando(necessário para executar uma aplicação)
<i>halt</i>	Mata todos os processos PVM, encerrando a console e desmontando a máquina virtual. Todos os <i>daemons</i> encerram sua execução.
<i>conf</i>	lista a configuração da máquina virtual com todos os nós, incluindo o <i>hostname</i> , <i>pvm</i> , <i>task id</i> , tipo de arquitetura e um índice de velocidade relativa.
<i>ps -a</i>	Listar todos os processos executando na Máquina Virtual
<i>alias</i>	Define ou lista os apelidos (alias) dos comandos.
<i>echo</i>	Ecoa os argumentos.
<i>id</i>	Imprime o <i>task id</i> da console.
<i>jobs</i>	Imprime os <i>jobs</i> em execução .
<i>kill</i>	Pode ser utilizado para terminar qualquer processo PVM.
<i>mstat</i>	Mostra o status dos <i>hosts</i> especificados.
<i>pstat</i>	Mostra o “status” de um único processo PVM.
<i>setenv</i>	Mostra ou seta variáveis de ambiente.
<i>sig</i>	Seguido por um número e um TID, envia o sinal identificado pelo número para a tarefa TID.
<i>trace</i>	Seta ou mostra a máscara de trace de eventos.

unalias Desfaz o comando alias.

90

spawn Inicia uma aplicação PVM(dispara um arquivo). Com as opções:
[-count] número de tarefas; padrão é 1.
[-host] dispare no host; o padrão é qualquer.
[-ARCH] dispare nos hosts com arquitetura do tipo ARCH.
[-?] habilita depuração.
[->] redireciona saída padrão para console.
[-> file] redireciona saída padrão para arquivo.
[->> file] redireciona saída padrão para anexar ao arquivo.
[-@] trace job, mostra saída na console.
[-@ file] trace job, saída para arquivo

Figura B.1 - Comandos do console pvm.