Comparação de desempenho Linux × Windows NT

José Eugênio de Assis Gonçalves

DISSERTAÇÃO APRESENTADA
NO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO GRAU
DE
MESTRE EM COMPUTAÇÃO

Área de Concentração: **Computação**
Orientador: **Prof. Dr. Marco Dimas Gubitoso**

durante a realização deste trabalho o autor teve auxílio financeiro do CNPq
- São Paulo, Maio de 2001-

Comparação de Desempenho Linux × Windows NT

Este exemplar corresponde à redação
final da dissertação devidamente corrigida
e defendida por José Eugênio de Assis Gonçalves
e aprovada pela Comissão julgadora
São Paulo, maio de 2001

Banca Examinadora:

- Prof. Dr. Marco Dimas Gubitoso (Presidente) IME-USP
- Prof. Dr. Edson Midorikawa POLI-USP
- Prof. Dra. Liria Matsumoto Sato POLI-USP

À minha esposa Regina, a minha filha Fernanda e
aos meus pais Vicente e Maria José.

Gratidão

"A vida não deve terminar como terminam as horas do dia, agonizando em um entardecer. A vida tem que ampliar seus horizontes; fazer longas as horas da existência para que o espírito, incorporado na matéria, experimente a grandiosidade de sua criação. Para isso tem que renovar-se no passado e no futuro. No passado, reproduzindo constantemente na tela mental todas as passagens vividas com maior intensidade; no futuro, pensando no que ainda resta por fazer, naquilo que se pensou fazer, e, sobretudo, no que se quer ser neste futuro. E quanto mais gratidão o homem experimente pelo passado, quanto mais gratidão guarde pelas horas felizes vividas nele, assim como pelas de luta ou de dor, que sempre são instrutivas, tanto mais abrirá sua vida a novas e maiores perspectivas de realização." (Raumsol)

Quero expressar o quanto sou grato a todos que participaram destas tantas horas que me fizeram chegar até aqui. Foram muitas horas felizes, algumas de dor e também muita luta. Ao meu orientador Gubi, hoje um grande amigo, por tudo que realizamos juntos, por me acompanhar desde o início deste mestrado. Inicialmente auxiliando na escolha das disciplinas e depois me orientando neste trabalho. Aos colegas do curso do IME, obrigado pelo seu apoio no dia-a-dia deste curso, especialmente ao Jorge Euler, à Maria do Carmo, à Vera Nagamuta e à Maria Clara. Ao pessoal do Grupo de Usuários Linux do IME, que disponibilizou o laboratório para os testes de rede. Recordo com gratidão, também, de muitos amigos que participaram da minha caminhada em outras etapas, não menos importantes. No curso de engenharia, quando tive a felicidade de encontrar amigos como o Anísio Rogério Braga, com seu entusiasmo infindável na busca pelo saber. Obrigado Anísio por tantos momentos de convívio feliz, estudando, brincando, fazendo projetos. Recordo também dos professores Hans e Evandro da UFMG, que me orientaram em projetos de iniciação científica. Este período foi, sem dúvida, um grande incentivo para que eu prosseguisse nos estudos.

Um agradecimento especial à minha esposa Regina, que sempre me incentivou e estimulou, principalmente naquelas horas mais difíceis. À minha filha Fernanda que tem sido um motivo de alegria e uma fonte de estímulos para a nossa vida. Ao meu pai e à minha mãe que me possibilitaram chegar até aqui, com o exemplo e com a dedicação. A vocês dedico este trabalho que é uma parte da minha vida.

Abstract

This work presents a metodology for comparing operating systems performance in the same hardware. This metodology was validated with two systems tests, namely Windows NT 4.0 and Debian/GNU Linux 2.0. We propose time measurements for a workload consisting of a single program, and observe the wall clock time, wich is the real time perceived by the user.

Two workloads were generated: a machine language version of Linpack and multiprogrammed network clients (TCP/IP and http). By means of Linpack, we checked the limits of each system regarding the number of simultaneous processes and other effects of intense CPU use. For networked systems the tests indicated a path to be followed, without leading to a more adequate evaluation given the limited resources in the available hardware.

Our evaluation method demonstrated some interesting characteristics of these operating systems and was proven itself to be an interesting alternative for comparing different operating systems in the same hardware. Other works which make use of this technique, for instance exploring multiprocessed machines, maybe of a great utility in comparing the performance of operating systems.This work presents a metodology for comparing operating systems performance in the same hardware. This metodology was validated with two systems tests, namely Windows NT 4.0 and Debian/GNU Linux 2.0. We propose time measurements for a workload consisting of a single program, and observe the wall clock time, wich is the real time perceived by the user.

Two workloads were generated: a machine language version of Linpack and multiprogrammed network clients (TCP/IP and http). By means of Linpack, we checked the limits of each system regarding the number of simultaneous processes and other effects of intense CPU use. For networked systems the tests indicated a path to be followed, without leading to a more adequate evaluation given the limited resources in the available hardware.

Our evaluation method demonstrated some interesting characteristics of these operating systems and was proven itself to be an interesting alternative for comparing different operating systems in the same hardware. Other works which make use of this technique, for instance exploring multiprocessed machines, maybe of a great utility in comparing the performance of operating systems.

Resumo

Este trabalho apresenta uma metodologia para a comparação de características de desempenho de sistemas operacionais em um mesmo hardware. Esta metodologia foi validada com a realização de testes em dois sistemas operacionais, que foram o Windows NT 4.0 e o Debian/GNU Linux 2.0. A proposta que apresentamos consiste na realização de medidas de tempo para a execução de um programa de carga único, tomadas pelo tempo de parede (*Wall clock time*), que é o tempo real visto pelo usuário para a execução de uma determinada tarefa.

Foram gerados dois tipos de programas de carga: uma versão em linguagem de máquina do Linpack e clientes de rede (TCP/IP e http) multiprogramados. Através do Linpack, verificamos os limites de cada sistema com relação ao número de processos simultâneos, ao escalonamento dos processos, e ainda outros efeitos do uso intenso da CPU. No caso de sistemas em rede os testes indicaram um caminho a ser seguido, não permitindo uma avaliação mais adequada devido à limitação dos recursos do *hardware* que utilizamos.

Nosso método de avaliação demonstrou algumas características interessantes dos sistemas operacionais em questão, e revelou-se uma alternativa interressante no caso de análise de sistemas diferentes em um mesmo hardware. Outros trabalhos que façam o uso desta técnica, explorando máquinas multi-processadas, por exemplo, podem ser de grande utilidade na comparação de desempenho de sistemas operacionais.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O desempenho é um fator fundamental em sistemas de computação e as exigências na capacidade de processamento são cada dia maiores. Uma forma de suprir esta necessidade é investir na aquisição de máquinas mais velozes e caras. Em certos casos, entretanto, um ajuste em alguns componentes do sistema pode ser suficiente ou até superar o resultado desejado.

Analisar a influência de cada componente no desempenho, identificando os gargalos, ou seja, onde estão os pontos no sistema que causam lentidão permite que os ajustes sejam identificados corretamente. Esta análise na maioria das vezes é tarefa complexa. Quando temos que, além disto, comparar duas ou mais combinações de hardware e software, o problema é ainda maior. Neste caso, muitas vezes não existem ferramentas que sejam capazes de analisar as variações que se pretendem medir.

O poder de processamento dos computadores cresce a cada nova versão da CPU ou de uma nova modificação na arquitetura da máquina. Tais modificações implicam na maioria das vezes em necessidades de modificações no software, principalmente naquele que é a interface entre o hardware e os demais aplicativos: o sistema operacional. Por outro lado, os desenvolvedores são tentados a incluir novas características e facilidades em seus aplicativos com o objetivo de explorar as capacidades das máquinas modernas. Isto é observado também no desenvolvimento de sistemas operacionais; que têm um número crescente de componentes a cada nova versão. Muitos destes componentes, segundo Tanenbaum [3], não fazem parte do sistema operacional; que ele define como "a porção do software que executa em modo kernel ou modo supervisor, com o objetivo de proteger o hardware da ação direta do usuário final da máquina". Analisando por este ponto de vista, muitos programas distribuídos juntamente com o sistema operacional não são componentes do mesmo, mas programas que utilizam-se dos serviços do sistema.

A evolução dos sistemas de computação está intimamente ligada com ino-

vações no hardware e no sistema operacional. Muitas técnicas de otimização utilizadas nos computadores modernos representam uma combinação de características destes dois componentes. Em alguns casos tanto o hardware como o sistema operacional são desenvolvidos por um único fabricante de uma forma integrada. Neste caso podem existir opções de sistemas operacionais a serem utilizados, mas normalmente, ao se optar por esta arquitetura, acaba-se por utilizar aquele que o fabricante do hardware desenvolve. Temos, por outro lado, arquiteturas de hardware para as quais existem várias opções de sistemas operacionais, sendo que o fabricante do hardware não possui uma opção própria. No caso da arquitetura *Intel*, entretanto, os sistemas da Microsoft (predominantemente o Windows 98/Me) costumam ser incluídos pelos fabricantes ao vender uma máquina; a Microsoft possui acordos com a maioria dos fabricantes fornecendo o software a um preço menor. Para a maioria dos usuários utilizar o sistema que vem incluído no equipamento adquirido é a escolha natural, para outros esta decisão não é tão simples. Uma série de fatores influenciam nesta escolha como os requisitos de desempenho ou a necessidade de utilização de um aplicativo específico. Estudos de desempenho de sistemas operacionais nesta plataforma são importantes para estes casos.

O presente trabalho procura avaliar o desempenho de dois sistemas operacionais para computadores baseados na Plataforma *Intel*. O objetivo é avaliar estes sistemas operacionais e sua influência no desempenho final de um aplicativo ou conjunto de aplicativos. Foram analisados o Microsoft Windows NT 4.0 e o Linux com o kernel 2.0 da distribuição Debian 2.0. A análise foi feita através de cargas sintéticas sem a utilização de monitores disponíveis em cada um dos sistemas, desta forma não foi preciso utilizar métodos peculiares à cada sistema. Além disto esta abordagem permite que se possa analisar através da mesma metodologia outros sistemas operacionais. Foram feitas medidas de desempenho para a sobrecarga de processador, número de processos simultâneos e ainda para cargas de rede (Internet).

Medir o desempenho não é um procedimento trivial e quando se trata dos computadores modernos isto tem se tornado cada vez mais complexo. Quando tentamos fazer estas medidas utilizando sistemas operacionais diferentes as dificuldades são ainda maiores. No nosso estudo deparamos com vários problemas ligados a utilização de ambientes heterogêneos.

Um destes problemas está ligado a como os programas funcionam em cada ambiente. Um mesmo programa fonte escrito em C e compilado em ambos sistemas, na mesma máquina, produzirá códigos bastante diferentes. Esta diferença é devida a várias características particulares de cada sistema operacional e dos compiladores utilizados. Este é um problema difícil de ser detectado uma vez que as diferenças não aparecem no programa fonte. O

desempenho do programa será diferente não apenas em função do sistema operacional mas de como o programa utiliza as bibliotecas do sistema ou os recursos do *hardware*. O melhor aproveitamento da máquina pode ser obtido, por exemplo, com a utilização de chamadas diretas à CPU ou de recursos específicos do hardware ou do sistema operacional.

1.1 Objetivos

O objetivo deste trabalho é uma comparação de características de desempenho dos sistemas operacionais Windows NT 4.0 e Linux do ponto de vista do usuário. As medidas de tempo serão tomadas pelo tempo de parede (*Wall clock time*), ou seja, o tempo real visto pelo usuário para a execução de uma determinada tarefa. Para cada variável analisada foi feito o possível para mantê-la o mais próximo da realidade de modo que se possa obter uma análise imparcial e justa dos resultados.

Estamos interressados em analisar o comportamento dos sistemas com multi-programação e em sistemas de rede. Nossa intenção é explorar uma metodologia, que utilizaremos para realizar algumas medições, comparando os dois sistemas. Pretendemos observar os efeitos da utilização da multi-programação, através da utilização de um programa de carga que faça uso intenso de CPU. Neste caso desejamos verificar o limite no número de processos simultâneos, a distribuição no tempo da execução de cada proceso pelo escalonador ou mesmo outros efeitos ligados à multi-programação. No caso de sistemas em rede utilizaremos uma configuração cliente-servidor, com a geração de uma carga sintética nos clientes; buscaremos, neste caso, determinar os limites de atendimento dos servidores.

1.2 Estruturação do texto

O texto está estruturado da seguinte maneira: No capítulo 2 apresentamos as características de cada sistema. Apresentaremos então, no capítulo 3, uma breve introdução teórica das técnicas de avaliação de desempenho. Em seguida no capítulo 4 serão mostrados alguns trabalhos relacionados ao nosso tema. A forma como foram realizadas as medidas e os recursos utilizados na obtenção dos resultados serão o assunto do capítulo 5. E finalmente, no capítulo 6 teremos os resultados das medições realizadas a partir das técnicas discutidas no capítulo 5.

Capítulo 2

Visão geral dos Sistemas

Neste capítulo serão descritas algumas características dos sistemas operacionais que serão avaliados. As informações apresentadas consistem de dados obtidos de [2] e [12], acrescidas com o que nós conhecemos de cada sistema. As características dos sistemas, conforme comentamos no capítulo 1, fazem parte do que deve ser ponderado no momento da escolha do mesmo. Algumas das particularidades que trataremos aqui não se configuram como um diferencial de um sistema em particular mas, servem para melhorar o entendimento da configuração interna dos mesmos.

2.1 Windows NT

O Windows NT é um sistema micro-kernel com multitarefa preemptiva e múltiplas interfaces de programação de aplicativos (API's), como a Win32, MS-DOS, 16-bit Windows, POSIX e OS/2, podendo ser executado em diversas plataformas de hardware.

O núcleo do Windows NT é chamado de *NT Executive*. O *NT Executive* abrange uma série de componentes que executam em modo kernel e implementam as funções básicas do sistema operacional como o gerenciamento de memória virtual, gerenciamento de objetos (recursos), E/S e sistema de arquivos e comunicação entre processos (IPC). A estrutura básica do NT é mostrada na figura 2.1, adaptada de [2].

Arquitetura do NT

Como um micro-kernel o NT é altamente modular com cada função do sistema sendo manipulada por um componente do sistema operacional, qualquer função somente pode ser invocada através da interface padrão fornecida pelo componente que a implementa.

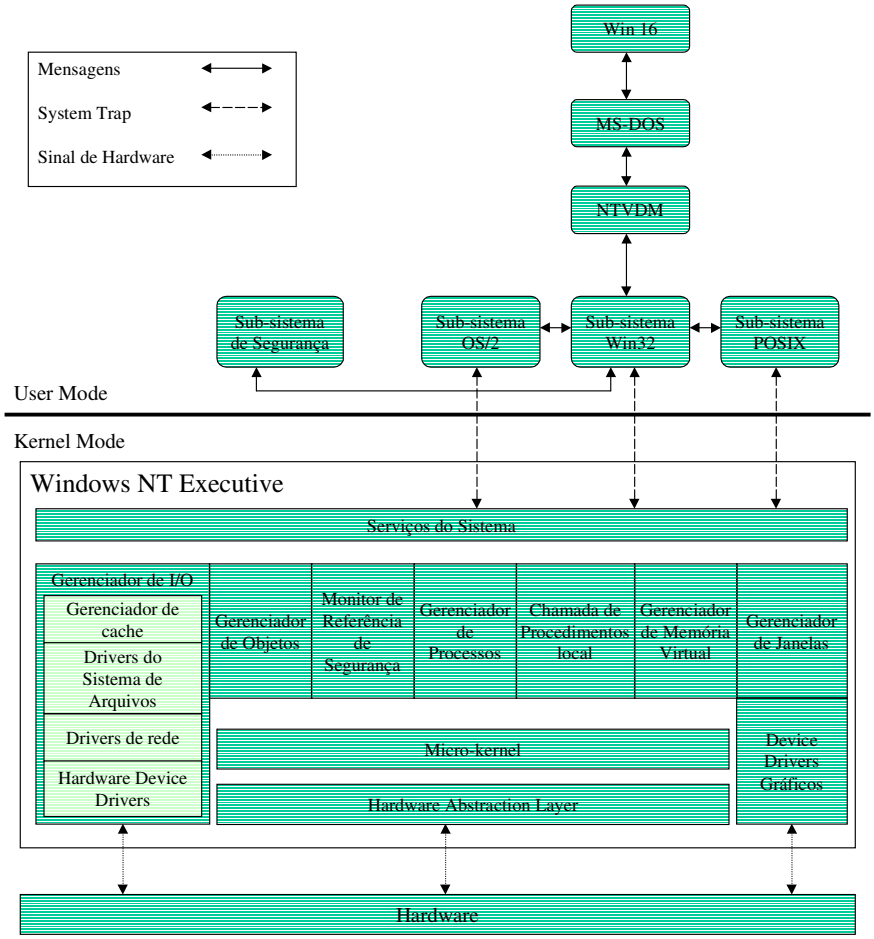

Figura 2.1: Arquitetura do Windows NT 4.0

O *NT Executive*, entretanto, não é um micro-kernel puro [5], [14] e [3], muitas funções que não compõem o micro-kernel rodam no modo kernel, ou seja, fazem parte do *NT Executive*. A razão para isto é o desempenho. Os

desenvolvedores do NT optaram por esta solução para evitar várias trocas de contexto, trocas de modo e a utilização de buffers de memória extra na execução destas funções que foram, assim, incorporadas ao kernel.

A seguir serão descritos os principais módulos do Windows NT que podemos ver na figura 2.1.

Hardware Abstraction Layer O NT foi projetado de forma que fosse um sistema operacional facilmente portável. Para isto existe uma camada de software entre o *NT Executive* e o hardware. Esta camada é o *Hardware Abstraction Layer* (HAL), ela é responsável por mapear comandos e respostas de hardware genéricos nos seus correspondentes específicos para cada arquitetura.

O HAL torna o barramento, o controlador de acesso direto à memória (DMA), o controlador de interrupções, os relógios do sistema e os módulos de memória de cada máquina específica parecerem os mesmos para o kernel. Existem códigos específicos para uma determinada plataforma fora do HAL basicamente no nível inferior do kernel e pequenas partes no gerenciador de memória virtual. Os *device drivers* também contêm códigos específicos para um determinado hardware podendo, entretanto, utilizar funções do kernel ou do HAL para evitar utilizar código dependente do processador ou da plataforma.

Micro-kernel O micro-kernel contém os componentes fundamentais e mais utilizados do sistema. O kernel gerencia o escalonamento de threads, troca de contexto entre processos, manipulação de exceções e interrupções e sincronização entre multiprocessadores. Ao contrário do restante do NT Executive e dos processos de nível de usuário, o micro-kernel não executa em threads. Assim, ele é a única parte do sistema operacional que não é preemptiva ou paginável.

Serviços de Sistema Fornecem a interface para os softwares em modo usuário.

Serviços executivos Incluem uma variedade de módulos para funções específicas do sistema. Todos estes serviços devem conectar-se ao Hardware através do HAL, com exceção do Gerenciador de E/S e os módulos de janela e gráficos. A seguir temos uma descrição dos serviços executivos:

Gerenciador de E/S Processa requisições de uma fila de requisições de E/S em ordem de prioridade.

Gerenciador de Objetos Garante regras uniformes para controlar a segurança de objetos. Cria handlers para objetos, que consistem de informações de controle de acesso e um ponteiro para o objeto.

Monitor de referência de segurança Responsável por garantir validações de acesso e regras de geração de auditoria. O modelo orientado a objetos do NT permite uma visão consistente e uniforme da segurança de cada componente do NT executive, utilizando as mesmas rotinas para validação e auditoria de todos os objetos protegidos, incluindo arquivos, processos, espaços de endereçamento e dispositivos de E/S.

Gerenciador de Processos Cria e elimina objetos e controla processos e threads.

Chamada local de procedimentos Garante um relacionamento cliente-servidor entre aplicações e sub-sistemas do NT executive de forma similar a um sistema de chamada remota de procedimentos (RPC)

Gerenciador de memória virtual Mapeia endereços virtuais no espaço de endereçamento dos processos em páginas físicas da memória.

Módulos de Janelas e Gráficos Cria a interface orientada a janelas e gerencia os dispositivos gráficos.

Sub-sistemas de ambiente do NT

O NT é estruturado para dar suporte a aplicações escritas para o próprio Windows NT, para o Windows 95/98 e para outros sistemas. Isto é possível devido à utilização de sub-sistemas de ambiente protegidos. Os sub-sistemas protegidos são as partes do NT que interagem com o usuário final. Cada sub-sistema é um processo separado e o NT executive protege seus espaços de endereçamento dos outros sub-sistemas e aplicações. Um sub-sistema protegido fornece uma interface gráfica ou de linha de comando que define como o sistema irá interagir com o usuário. Além disto, cada sub-sistema fornece a API do ambiente operacional particular. Os sub-sistemas que o NT possui são:

- MS-DOS Máquina Virtual DOS NT (NTVDM)
- Win16 NTVDM
- sub-sistema OS/2

- sub-sistema POSIX
- sub-sistema Win32

O sub-sistema mais importante é o Win32. Esta é a API padrão, e é implementada tanto no Windows 95/98 como no Windows NT. Algumas características da API Win32 no NT, entretanto, são diferentes ou inexistem na sua versão para o Windows 95/98. Conforme vemos na figura 2.1 o sub-sistema Win32 ocupa uma posição central e todos os outros sub-sistemas trocam mensagens com o NT executive através dele. O Win32 é o ambiente nativo do Windows NT, as outras API's são mapeadas em chamadas a API Win32 para executarem no NT.

Características do NT

Multitarefa O Windows NT trabalha com multitarefa preemptiva. Todos os processos rodam de forma totalmente independente um dos outros. Nenhum processo precisa se preocupar em deixar o processador disponível para outros processos.

Mono-usuário O Windows NT não é um sistema multi-usuário. Apenas um usuário pode utilizar a máquina a cada instante.

Multiprocessamento O Windows NT executa em arquiteturas com multiprocessadores. O que significa que o sistema operacional pode distribuir várias aplicações entre vários processadores.

Independência de Arquitetura É um sistema multi-plataforma. Existem versões para Intel, MIPS, Digital Alpha e PowerPC.

Carga de programas por demanda Somente as partes do programa necessárias para a execução são carregadas na memória.

Paginação O Windows NT utiliza a paginação de memória com páginas de 4k.

***Cache* de disco dinâmico** O tamanho do *cache* de disco é ajustado dinamicamente de acordo com a utilização da memória. Se em um determinado momento a memória fica toda ocupada o tamanho do *cache* de disco é reduzido, voltando a aumentar quando a demanda por memória diminuir.

Bibliotecas compartilhadas Bibliotecas padrão são utilizadas por mais de um processo ao mesmo tempo. Assim estas bibliotecas somente são carregadas uma vez. Além disto, como estas bibliotecas são carregadas somente quando o processo está executando, elas são conhecidas como *Dynamic Link Libraries*.

Suporte ao POSIX O Windows NT adere ao padrão POSIX 1003.1 que é um padrão utilizado atualmente por todos os sistemas operacionais relativamente sofisticados.

Vários formatos para arquivos executáveis Além dos programas escritos para a API Win32, o Windows NT possui uma máquina virtual para executar alguns programas do MS-DOS e da API Win16. É capaz ainda de executar programas feitos para o OS/2.

Modo protegido de Acesso a memória O Windows NT utiliza os mecanismos de proteção de memória do processador para evitar que um processo utilize a memória de outro processo ou do kernel.

Suporte a teclados e fontes internacionais No Windows NT uma grande faixa de teclados internacionais e conjuntos de caracteres pode ser utilizado.

Sistemas de Arquivos diferentes O Windows NT trabalha com 3 sistemas de arquivo: MS-DOS (FAT16), OS/2(HPFS) e o NTFS, sistema de arquivos nativo do NT, que aceita nomes de arquivos de até 255 caracteres e possui características de segurança avançadas.

TCP/IP, SLIP e PPP O Suporte aos protocolos SLIP e PPP permite a utilização do TCP/IP em uma linha serial permitindo o acesso discado a Internet.

2.2 Linux

O Linux é um sistema operacional UNIX. Escrito originalmente para a plataforma *Intel* ele hoje já é portado para diversas arquiteturas (veja na tabela 2.1) e é compatível com o POSIX 1003.1. Uma grande parte do kernel do Linux foi escrita pelo estudante finlandês Linus Torvalds, que disponibilizou os fontes sob a licença pública do GNU. Isto significa que qualquer pessoa tem o direito de utilizar, copiar e modificar os programas livremente.

O sistema operacional completo pode ser obtido gratuitamente juntamente com vários programas. O kernel do Linux foi desenvolvido de forma

monolítica. Uma das razões que levaram os desenvolvedores a optar por um kernel monolítico foi que este sistema foi projetado para evoluir aos poucos e o gerenciamento e projeto de um sistema não monolítico seria bem mais complexo. Um outro aspecto se relaciona ao desempenho. O projeto de um sistema micro-kernel força a criação de interfaces bem definidas entre os componentes do sistema o que impede otimizações muito sofisticadas. Além disto as arquiteturas de hardware atuais forçam a utilização de comunicações entre processos no lugar de chamadas de funções. Estas considerações de desempenho representam muito pouco para o hardware atual mas não tão pouco para um sistema que foi construído para executar de forma razoável em um 386

[12].

Ao contrário do Windows NT, o sistema de janelas utilizado no Linux não é uma parte do kernel, sendo formado por um conjunto de programas rodando como um processo de usuário. Os programas que compõem o sistema de janelas são baseados na arquitetura cliente-servidor com um programa servidor gráfico, como o Xwindow do MIT (Massachusetts Institute of Technology), disponibilizando os recursos do hardware para o programa cliente.

Apesar de existirem versões para outras arquiteturas de hardware a maior parte dos usuários do Linux utiliza a plataforma Intel x86.

2.2.1 Características do Linux

Multitarefa Como o NT o Linux possui um sistema multitarefa preemptiva.

Multi-usuário O Linux permite a vários usuários utilizar o sistema ao mesmo tempo.

Multiprocessamento Desde a versão 2.0 o Linux executa em arquiteturas com multiprocessadores.

Independência de Arquitetura O Linux oferece suporte a várias plataformas de hardware como Amiga, Intel, DEC Alpha, Palm Pilot.

Carga de programas por demanda Somente as partes do programa necessárias para a execução são carregadas na memória. Quando um novo processo é criado usando *fork()*, a memória não é requisitada imediatamente, ao invés disto a memória do processo pai é utilizada por ambos processos. Se o novo processo requisita uma página da memória para escrita, esta página é copiada antes de ser modificada. Este conceito é chamado *copy-on-write*

Paginação O Linux utiliza a paginação de memória com páginas de 4k.

***Cache* de disco dinâmico** O tamanho do *cache* de disco é ajustado dinamicamente como no Windows NT.

Bibliotecas compartilhadas no Linux são chamadas de *shared libraries*.

Suporte ao POSIX e parte do System V e BSD O padrão POSIX define a interface mínima para um sistema operacional UNIX. O Linux adere ao padrão POSIX 1003.1. Também estão implementadas Interfaces adicionais para o System V e o BSD. Assim, no geral um software escrito para UNIX pode ser portado diretamente para o Linux.

Vários formatos para arquivos executáveis O Linux possui emuladores para o MS-DOS e MS-Windows que estão em desenvolvimento. Programas escritos para outros sistemas UNIX que estejam de acordo com o padrão iBCS2 podem também ser executados no Linux.

Modo protegido de Acesso a memória O Linux utiliza os mecanismos de proteção de memória do processador para evitar que um processo utilize a memória de outro processo ou do kernel. Um programa com erros dificilmente consegue derrubar o sistema.

Suporte a teclados e fontes internacionais No Linux uma grande faixa de teclados internacionais e conjuntos de caracteres pode ser utilizado.

Sistemas de Arquivos diferentes O Linux oferece suporte a uma grande variedade de sistemas de arquivo. O mais utilizado é o sistema de arquivos *Ext2*. Este sistema de arquivos aceita nomes de arquivos de até 255 caracteres e tem muitas características que o tornam mais seguro que sistemas de arquivo convencionais UNIX. Outros sistemas de arquivos implementados são o FAT-16 e o VFAT para acesso a partições MS-DOS e Windows 95, o sistema de arquivos ISO para leitura CD-ROM's e o NFS para acesso aos sistemas de arquivos de outros computadores UNIX presentes na rede. Além destes outros sistemas de arquivos são possíveis como o AFF (Amiga), UPS e SysV para acesso a sistemas UNIX de outros fabricantes, HPFS (OS/2). O NTFS (Windows NT) está em desenvolvimento. Existe ainda o sistema de arquivos SAMBA que permite o acesso a sistemas de arquivos exportados de sistemas Windows.

TCP/IP, SLIP e PPP O Suporte aos protocolos SLIP e PPP permite a utilização do TCP/IP em uma linha serial permitindo o acesso discado a Internet.

2.3 Comparação das características do Linux e do Windows NT

A tabela 2.1 mostra uma comparação das características dos sistemas operacionais Linux e Windows NT. Desta tabela verificamos que o Linux fornece suporte a um número maior de sistemas de arquivos inclusive o FAT32 que é usado pela Microsoft no Windows 98. Com relação a arquiteturas de hardware o Linux também tem maiores opções de escolha que o NT. Um outro ponto é que o Linux é um sistema multi-usuário, entretanto, esta característica não é relevante para a nossa análise e em alguns casos práticos não é necessária.

Característica	Windows NT	Linux
Arquitetura de hardware	Intel, MIPS, Alpha, PowerPC	Amiga, Intel, Alpha, Sparc, PowerPC, Palm
Sistema de Arquivos	3: FAT16, HPFS, NTFS	32: Ext2, FAT16, FAT32, AFF, HPFS...
Rede	TCP/IP, SMB, IPX/SPX, Netbeui...	TCP/IP, SMB, IPX/SPX, IPV6...
Multitarefa	Preemptiva	Preemptiva
Multiprocessamento	Sim	Sim
Multi-usuário	Não	Sim
Cache de disco	Dinâmico	Dinâmico
Bibliotecas compartilhadas	Sim	Sim

Tabela 2.1: Comparação de características

Capítulo 3

Avaliação de Desempenho

Como estabelecido em [4], a avaliação de desempenho é uma arte. "Como um trabalho de arte, uma avaliação bem sucedida não pode ser produzida mecanicamente". Um trabalho de análise de desempenho requer um profundo conhecimento do sistema e uma seleção criteriosa da metodologia, da carga e das ferramentas. A necessidade da utilização das técnicas que aqui são descritas é sentida a partir do momento em que se apresenta um problema de dimensionamento de um sistema de computação. Na maioria das vezes, não se sabe qual a melhor configuração a ser especificada, o que se aplica tanto à estudos de acréscimo na capacidade de um sistema existente, quanto na especificação de um novo sistema. As dificuldades começam no estabelecimento da demanda necessária, que normalmente é derivada da intuição do usário, que define as suas necessidades de desempenho genericamente. E comum que as necessidades do usuário sejam expressadas através de frases como: -Este sistema deve ser capaz de atender a 10.000 usuários simultâneos. Assim começa o trabalho do "artista" que precisa, a partir da idéia abstrata, definir o problema real e os meios que se valerá para solucioná-lo dentro do prazo e do orçamento disponível.

3.1 Técnicas de Avaliação de desempenho

As três técnicas de avaliação de desempenho são:

- Modelagem Analítica, que consiste na utilização de métodos matemáticos para a representação do sistema.
- Simulação, através de programas e ferramentas específicas.
- Medição, onde avalia-se um conjunto de características a partir da instrumentação do sistema real.

3.1.1 Selecionando uma Técnica de Avaliação de desempenho

A seleção das técnicas a serem utilizadas depende de uma série de fatores que são explorados a seguir. O estágio em que está o sistema em seu ciclo de vida, por exemplo, é um fator determinante. Para utilizar a medição é necessário que o sistema, ou algo similar já exista, do contrário, somente a simulação e a modelagem poderão ser utilizadas.

Um outro fator importante é o tempo disponível para a avaliação que, na maioria dos casos, é mínimo. A modelagem analítica é a técnica que requer menos tempo, em contrapartida, é a que apresenta menor precisão.

A próxima consideração são as ferramentas. A disponibilidade de profissionais com experiência em modelagem, a existência de linguagens de simulação e pessoal capacitado para utilizá-las, e a possibilidade de utilizar instrumentos de medida, influenciarão na escolha a ser tomada.

O nível de precisão desejado é igualmente importante. A modelagem é a técnica que oferece a menor precisão, pois, normalmente, exige muitas simplificações. A simulação já permite uma precisão maior, no entanto, ainda não muito próxima da realidade. Enfim, a medição oferece a possibilidade de um resultado que se aproxima da situação real, porém, este pode ser prejudicada pela diferença entre o ambiente de testes e o ambiente de produção.

Se o objetivo do estudo de desempenho é um melhor ajuste de parâmetros e não a comparação de diferentes alternativas, a modelagem é mais adequada. Isto porque tais parâmetros podem ser bem definidos assim como o inter-relacionamento entre os mesmos.

O custo do trabalho de análise de desempenho é um outro ponto a ser considerado. A medição exige equipamento, instrumentos de medida e tempo. Ela é a mais cara das três técnicas. A facilidade na mudança das configurações a um custo menor do que a medição, justificam a escolha da simulação como alternativa. Finalmente, no item custo, temos a modelagem analítica que é a opção mais barata, exigindo apenas o tempo do analista.

Um outro fator é a aceitação dos resultados pelos usuários, que é maior quando se usa a medição. É mais fácil convencer aos usuários de posse de um resultado de uma medição realizada em um sistema real. Estes, em geral, não aceitam as técnicas de simulação e de modelagem, pois não entendem como foram obtidos os resultados.

Em alguns casos é recomendada a utilização de duas ou mais técnicas simultaneamente. Assim pode-se validar o resultado obtido por uma técnica com o resultado de uma segunda.

3.2 Métricas de Desempenho

Em um estudo de desempenho, temos que definir quais serão as variáveis a serem avaliadas. Uma maneira de definir o conjunto de variáveis em questão é estabelecer uma lista dos serviços oferecidos pelo sistema. Para cada serviço podemos então definir quais são as saídas possíveis. Estas saídas podem ser classificadas em três categorias: o sistema atendeu ao serviço corretamente, o sistema atendeu o serviço incorretamente ou o sistema não atendeu ao serviço.

Se o sistema atende a um serviço corretamente, seu desempenho é medido pelo tempo gasto para atendê-lo, a taxa em que este foi atendido e pelo total de recursos consumido pelo mesmo. Estas três métricas, tempo-taxa-recursos para um desempenho satisfatório são também chamadas das métricas de resposta, produtividade e utilização. Um exemplo para a métrica de resposta é o tempo de atendimento de uma requisição de rede, ou seja, o tempo entre a solicitação de um pacote e a sua chegada. A produtividade de um servidor de rede é medida pela seu *throughput*, o número de pacotes enviados por unidade de tempo. A utilização indica a porcentagem de tempo de uso dos recursos do sistema para uma determinada carga. O recurso com a maior porcentagem de utilização é denominado o "gargalo", a otimização do desempenho deste recurso oferece o melhor custo-benefício. Determinar estes gargalos é uma parte importante do trabalho de análise de desempenho.

Para os casos em que o sistema não responde corretamente, temos as situações de erro. Nestas situações é importante classificar os erros e determinar a probabilidade da ocorrência dos mesmos.

Quando o sistema não executa uma solicitação, dizemos que o mesmo "caiu", que está fora do ar ou indisponível. Novamente é importante a classificação das quedas do sistema, determinando a probabilidade destas. Atualmente é comum que sistemas sejam vendidos alegando a disponibilidade 24x7, ou seja, 24 horas por dia, 7 dias por semana. Isto é uma preocupação crescente no mercado, principalmente quando se trata de servidores de missão crítica.

As métricas associadas com os três resultados, atendimento correto, atendimento com erros e indisponibilidade, são também chamadas de velocidade, confiabilidade e disponibilidade. Para cada serviço oferecido pelo sistema estas três métricas são desdobradas em diversos números, que crescem de acordo com a complexidade do sistema e da quantidade de serviços oferecidos.

Outro aspecto importante com relação às metricas de desempenho está ligado à utilização do sistema por mais de um usuário. É importante que se considere que algumas métricas podem ser vistas sob o ponto de vista de

um usuário, enquanto que outras dizem respeito ao sistema como um todo. Assim a utilização dos recursos, por exemplo, não pode, na maioria dos casos ser isolada para um único usuário. Por outro lado, o tempo de resposta de um aplicativo pode ser medido para cada usuário. Existem casos em que a otimização pode melhorar o tempo de resposta total, prejudicando porém o tempo de resposta individual.

Estabelecido um conjunto de métricas de um sistema, devemos escolher um sub-conjunto das mesmas, tomando o cuidado de selecionar métricas que representem de forma completa o serviço que desejamos avaliar, contemplando todas as possíveis respostas.

3.3 Utilização de Benchmarks

Como foi disposto anteriormente a utilização da medição é a técnica de análise que é mais aceita pelo usuário final. Neste caso a carga de trabalho real é submetida ao sistema e os resultados são medidos. O problema com a utilização da carga de trabalho real é a dificuldade que existe em gerar esta carga fora do ambiente de produção.

Uma alternativa largamente utilizada é a utilização de benchmarks para comparar o desempenho de diferentes sistemas. Os benchmarks são programas especialmente construídos para representar uma carga bem definida. Esta focalização da carga que propicia os benchmarks e a facilidade com que estes podem ser repetidos tornou a sua utilização largamente difundida [13].

Os resultados dos benchmarks, entretanto, podem levar a uma má interpretação dos resultados pelo usuário. Isto se deve ao fato de que este não tem conhecimento do que representa este benchmark com relação ao sistema real, ou seja, o que está sendo medido ou testado. Se, por outro lado, conhecemos bem o benchmark, entendendo os resultados que podemos obter do mesmo, este pode ser utilizado como uma importante ferramenta para avaliar o desempenho de um sistema. É importante ressaltar, porém, que os benchmarks são bons para a comparação de sistemas, mas normalmente são focalizados em um aspecto específico e não permitem uma boa definição da capacidade de um sistema real.

Podemos definir uma hierarquia para os programas de benchmark existentes, cada nível nesta hierarquia é caracterizado pela complexidade dos programas e pela aproximação dos mesmos com os programas reais. No primeiro nível estão os benchmarks que realizam operações básicas, como adição e multiplicação, o Dhrystone e o Whetstone são exemplos desta categoria. O primeiro é um programa voltado para a medida da velocidade de execução de operações de números inteiros, o segundo para operações de ponto flutu-

ante. Estes programas não resolvem nenhum problema real e sua utilidade para a compreensão de sistemas práticos é limitada. No segundo nível estão os chamados "benchmarks de brinquedo", estes são pequenos programas que implementam quebra-cabeças clássicos, como as torres de Hanoi. Novamente, estes programas não se assemelham a cargas reais. Os chamados *kernels*, são o terceiro nível, e constituem porções de código extraído de programas reais. Estas porções de código representam a essência de um cálculo. O Linpack é um bom exemplo deste tipo de benchmark. Normalmente, este tipo de programa é uma excelente carga para a CPU, mas não representam o desempenho final percebido pelo usuário. No último nível da hierarquia, estão os programas que correspondem à programas utilizados pelos usuários na solução de problemas reais.Componentes do compilador GNU C, utilitários para compactar e descompactar arquivos, e transações bancárias de débito e crédito (TPC) são exemplos destes programas .

Capítulo 4

Trabalhos Relacionados

Os trabalhos de análise de desempenho de diferentes sistemas operacionais em um mesmo hardware eram, até pouco tempo, inexpressivos. O crescimento da participação de novos sistemas operacionais, como o Linux, na plataforma INTEL fez com que aparecessem muitos artigos confrontando este sistema e o Windows NT. A maioria destes artigos especulava muito, defendendo um ou outro sistema sem, no entanto, apresentar resultados concretos. Uma outra parte faz a análise utilizando ferramentas de *benchmark*,

apesar disto, nem todos fazem as medidas com a mesma configuração de *hardware*, além de outros problemas que serão discutidos neste capítulo.

Na maioria dos trabalhos deste tipo verifica-se a utilização de *benchmarks* dirigidos, visando determinar a eficiência da arquitetura em um determinado serviço ou utilização. Nesta mesma linha de abordagem está a análise do comportamento dos sistemas operacionais frente a um ou mais aplicativos; neste caso porém a carga utilizada são os próprios aplicativos. Ao explorar mais de uma arquitetura utilizando aplicativos reais é necessário utilizar aplicativos que sejam comuns a todas arquiteturas. Neste caso surgem, porém, vários problemas. A utilização de uma biblioteca específica pelo aplicativo, por exemplo, pode trazer uma sobrecarga adicional ou um ganho de desempenho. Esta diferença pode ser provocada quando a referida biblioteca utiliza recursos de hardware ou software que melhoram o desempenho. Desta forma, aplicativos iguais sobre o ponto de vista do usuário, podem apresentar uma disparidade no desempenho significativa. Ao considerar aplicativos funcionalmente idênticos para mais de um sistema operacional, não se tem idéia de como estes foram construídos, ou seja, da estrutura interna do programa. Assim um aplicativo pode ser mais eficiente em um determinado sistema devido à forma como foi escrito e não devido às características do sistema operacional.

Além das difereças nos aplicativos, ao analisar a influência dos sistemas

operacionais no desempenho devemos levar em consideração ainda as características destes sistemas. Se considerarmos o fato da interface gráfica fazer parte do sistema no Windows NT, podemos ter tanto casos em que isto é um ponto positivo para o desempenho, quanto casos em que isto leva a uma sobrecarga do sistema, penalizando o desempenho. Muitas vezes a construção interna dos sistemas não é bem documentada o que pode dificultar a análise em alguns casos.

4.1 Medição de desempenho de sistemas operacionais para computadores pessoais

Este estudo apresentado em [1] faz uma análise comparativa de três sistemas operacionais que executam na arquitetura de computadores pessoais que utiliza o Pentium da *Intel*. Os sistemas operacionais analisados foram: Windows NT, Windows for Workgroups e NetBSD (uma versão do unix anterior ao Linux). A idéia do trabalho foi realizar a comparação a partir de medidas do custo de operações primitivas da máquina. Estas operações fazem parte de um sub- conjunto dos contadores existentes na CPU Pentium da Intel.

Os autores construíram programas para capturar os valores destes contadores. De posse das medidas dos contadores foi apresentada uma análise do desempenho. Esta análise foi feita para *microbenchmarks* e cargas de aplicações reais.

4.1.1 *Microbenchmarks*

Dentre as operações que foram avaliadas com os *microbenchmarks* temos:

- Latência mínima para realizar chamadas de sistema.

  No caso do Windows foram utilizadas duas chamadas de sistema, uma no modo de 32 bits e outra no modo de 16 bits. Neste teste foi verificado que o NetBSD e o Windows para chamadas de 32 bits apresentaram um custo bem menor. O Windows NT e o Windows no modo de 16 bits têm um custo maior devido à necessidade do chaveamento entre os vários componentes destes sistemas operacionais

- Latência para carregar e executar um programa trivial.

  Foi verificado que o NetBSD teve melhor desempenho para a carga e execução do programa de teste. O estudo conclui que parte da sobrecarga no caso do Windows NT é devida a arquitetura do sistema. Na criação de um processo o NT executive é acionado pelo subsistema

Win32. Como o NT Executive é utilizado por outros subsistemas a interface teve que ser mais genérica e simples, isto levou a uma maior complexidade na implementação. No caso da criação de processos, o subsistema Win32 precisa utilizar duas funções do NT Executive: uma para criar o espaço de endereçamento e outra para criar a tarefa que executará neste espaço de endereçamento.

Apesar da diferença o estudo ressalta que este aspecto não tem grande importância no desempenho geral. Isto é devido ao fato de que os aplicativos típicos do Windows e do Windows NT executam por um longo tempo após iniciados, com as funcionalidades sendo utilizada com chamadas de sub-rotinas ou bibliotecas e não de programas separados.

- Tempo para acesso a memória.

  No teste de acesso a memória foi medido o número de ciclos necessário para ler um vetor de 512 bytes até 1MB. Este teste refletiu a atuação dos caches de memória: 8KB interno do processador e 256KB da placa mãe. Os tempos de acesso apresentam um aumento significativo tanto quanto o tamanho do vetor ultrapassa o tamanho do cache de 8KB, quanto do cache de 256KB.

  As medidas mostram que o comportamento do Windows NT e do NetBSD foi muito semelhante, enquanto o Windows for workgroups teve um desempenho bem menor, devido à limitação no tamanho do segmento deste sistema (64KB). O Windows precisa carregar o registrador de segmento para cada acesso, por não poder determinar quando este acesso irá cruzar a fronteira do segmento atual. Esta necessidade do Windows aumentou o número de ciclos por referência de dois para nove.

- Performace do sistema de arquivos.

  Para o Windows NT foram avaliados os sistemas de arquivo FAT e NTFS, no Windows foi testado o sistema FAT nas versões de 16 e 32 bits e para o NetBSD o FFS. Foram avaliadas apenas algumas operações como o acesso a arquivos e o tempo para a criação de arquivos. No acesso a arquivos quando estes são menores que o cache foi verificado que o NT/FAT necessita de duas vezes mais instruções que o NetBSD, além disto a sobrecarga do NT ao cache e ao TLB é maior. Com isto o NT/FAT ficou 2,5 vezes mais lento neste experimento. Ao utilizar o NTFS o NT ficou ainda mais lento, necessitando 35% mais instruções e gastando um tempo 30% maior do que o acesso via FAT.

Para arquivos maiores que requerem acesso a disco, o Windows NT teve uma latência maior que a dos outros sistemas. Tanto para o NTFS quanto para o FAT o NT foi cerca de duas vezes mais lento.

Na criação de arquivos o NetBSD com FFS foi o mais lento. A diferença, neste caso, se deve a que o sistema de arquivos FFS faz a escrita diretamente ao disco de modo síncrono, enquanto no FAT e no NTFS as operações são armazenadas em memória.

- Operações gráficas.

  O teste consistiu da análise de uma operação gráfica específica porém mostrou diferenças importantes entre o modo como cada sistema trabalha no modo gráfico. O que este *benchmark* faz é apenas o preenchimento da tela com informações de um bit. O preenchimento se repete 760 vezes. O NetBSD precisou de dez vezes mais referências à memória e cem vezes mais instruções. Esta diferença se deve ao fato de que o Windows e o Windows NT copiam cada bitmap apenas uma vez, da aplicação para a tela, numa operação de cópia direta. O X11, gerenciador de janelas do NetBSD, faz a cópia indiretamente, o que gera um número maior de acessos a memória para cada bit a ser transportado para a tela. O menor número de instruções do Windows e do Windows NT com relação ao NetBSD está ligado ao fato do X11 utilizar um código que não faz uso de instruções dependentes da máquina, que poderiam fazer cópias mais rápidas para a memória de vídeo.

- *throughput* de rede.

  Neste teste foi avaliado o tempo para o envio de 256 KB através de uma conexão Ethernet dedicada. Verificando os contadores foi constatado um comportamento similar nos três sistemas. A única diferença significativa foi um maior número de faltas no TLB e no Cache para o Windows NT o que reforça resultados de outros testes que indicam que o NT tem uma pior localidade de instrução que os outros sistemas. Apesar desta diferença o *throughput* foi semelhante em todos sistemas em torno de 1 MB/s, neste caso, o gargalo foi a própria velocidade da rede e não os sistemas operacionais.

4.1.2 Carga de Aplicações

Nos testes com aplicativos reais foram analisados os seguintes programas:

- Wish - um interpretador de comandos para a linguagem TCL com suporte a interfaces gráficas através da ferramenta Tk;

- Ghostscript - visualizador de arquivos Postscript de domínio público;
- Servidor Web - dois servidores distintos foram utilizados o NCSA Http daemon versão 1.3R para o NetBSD e para o Windows NT um servidor baseado no código do NCSA Mosaic.

4.1.3 Wish

Com este programa foi possível comparar o custo de uma interface gráfica com o usuário em cada sistema. Os arquivos de dados utilizados foram pequenos evitando-se acessos ao disco. O tempo de execução para o NT foi de 14,94 segundos e para o NetBSD foi de 7,05 segundos.

Observando a total de cargas do registrador de segmento para o NT e o NetBSD, durante a execução do Wish, temos 1.400.228 e 170.827, respectivamente. Estes números indicam que o NT necessita de um maior número de mudanças nos domínios de proteção. O NT teve ainda um número de faltas no TLB muito maior, comprovando os problemas de localidade de memória do Windows NT.

4.1.4 Ghostscript

Um aspecto que deve ser levado em conta nos resultados com o Ghostscript é a maneira como ele utiliza o modo gráfico em cada sistema. No NetBSD o desenho na tela é feito através de chamadas à funções do XWindows, enquanto que no NT e no Windows a imagem e gerada como um mapa de bits e transferida para a tela de uma só vez.

O tempo de execução para cada sistema é apresentado na tabela 4.1. O desempenho do NT foi bem melhor do que o dos outros sistemas. No caso do NetBSD este resultado mostrou que a implementação do modo gráfico deste sistema é menos eficiente, já para o caso do Windows não foram evidenciados os componentes do sistema que penalizaram o desempenho.

	Tempo (seg.)
Windows NT	9,30
Windows	20,32
NetBSD	16,92

Tabela 4.1: Tempo de execução do Ghostscript para visualizar um artigo de 13 páginas, 372KB

4.1.5 Servidor Web

Os servidores utilizados nos testes obtiveram um *throughput* de 20 requisições por segundo para o NetBSD e 12 para o NT. Ao utilizar requisições concorrentes o NT apresentou uma queda no tempo de resposta para um número superior a 15 clientes simultâneos, enquanto que o NetBSD se comportou de forma linear com até 32 clientes simultâneos.

Nas medidas dos contadores do pentium foi verificado que o servidor Web causou grande impacto no sistema de memória que apresentou um número expressivo de faltas no *cache*. No NT as faltas no *cache* de instruções foram maiores do que no NetBSD, apesar do NT necessitar de um número menor de instruções.

4.1.6 Comentários

Consideramos que existem alguns pontos interessantes verificados neste estudo. Ao utilizar os contadores do Pentium ficou evidente o fato de que o NT sobrecarrega mais o sistema, apresentando uma pior localidade de instrução que os outros sistemas. Um problema da abordagem utilizada é de que esta se limita à arquitetura de um processador específico. Além disto, a utilização dos contadores gera informações em um nível muito próximo ao *hardware*, dificultando a interpretação dos resultados.

Ao analisar o desempenho da execução de aplicativos reais podemos ver como é complicada esta análise para sistemas operacionais diferentes. Nos testes com o GhostView, por exemplo, a maneira como cada sistema utiliza os recursos gráficos fez com que o NT tivesse um melhor desempenho. No caso do servidor Web, temos um exemplo de um aplicativo que pode ser muito mais eficiente do que os que foram avaliados, com as versões que são utilizadas atualmente, as quais são melhor ajustadas para os sistemas operacionais e hardware.

4.2 Servidores Web

Com relação a análise de desempenho de servidores Web diversos trabalhos tem sido feitos, pois com o crescimento da Internet a necessidade de servidores rápidos é cada vez maior. Um problema que surge está ligado ao universo de possibilidades no desenvolvimento de aplicativos para a Web que aumenta a cada dia. Os servidores Web que inicialmente funcionavam basicamente como simples servidores de arquivos respondendo às requisições com o retorno do conteúdo de um conjunto de arquivos, passaram também a gerar as suas respostas a partir da execução ou interpretação de um programa. Assim

podemos classificar as requisições de acordo com a forma como o servidor as trata em:

- requisições estáticas:

  A resposta do servidor se baseia em arquivos previamente existentes no servidor

- requisições dinâmicas:

  A resposta do servidor é gerada no momento da requisição a partir de um processamento no servidor. Este processamento pode ser o resultado da execução de um programa externo ou pode ser feito pelo próprio servidor Web.

Em alguns casos não é possível uma comparação entre os resultados apresentados. Em [18] temos por exemplo a apresentação do que se chamou servidores de aplicação, onde analisa-se a solução de comércio eletrônico de 8 fornecedores de software. Naquele estudo cada fornecedor construiu o aplicativo independentemente, desta forma a comparação entre cada solução depende não somente dos servidores e hardware envolvidos como também de como cada aplicativo foi escrito. Foram analisados os produtos: WebObjects 4.0 da Apple Computer, Sapphire/Web 5.1 da Bluestone Software, Hahtsite 4.0 da Haht Software, Windows NT Enterprise Server 4.0 da Microsoft, Apptivity 3.0 da Progress Software, Sybase EAS (Enterprise Application Server) 3.0 da Sybase, e os produtos da aliança Sun/Netscape NetDynamics 5.0 e Netscape Application Server 2.1. Um outro problema daquele trabalho foi que a Microsoft fez os testes em um hardware diferente do demais, assim a comparação entre os resultados não é razoável pois um melhor desempenho pode estar ligado à um *hardware* mais eficiente.

Outro aspecto é que um conjunto muito grande de aplicativos está envolvido para produzir o site. Isto dificulta a avaliação do peso de cada componente no desempenho final.

Outras medidas de desempenho de servidores Web podem ser vistas em [24]. Naquele estudo foram comparados os servidores Novell Netware Web Server, Microsoft Internet Information Server 1.0 e Netscape Communications Server 1.12 for Windows NT. Foi apresentado um estudo de requisições estáticas e dinâmicas aos servidores que utilizaram o mesmo hardware. O servidor da Novell utilizou um sistema operacional da própria Novell, enquanto que os demais utilizaram o Windows NT 3.51. As medidas apresentadas mostraram um desempenho 16% melhor do servidor Novell para requisições estáticas. No caso de medidas de requisições dinâmicas o servidor Novell ainda foi apresentado como mais eficiente, entretanto, esta eficiência não foi

visível nos gráficos e tabelas. O estudo apresentado por [24] utilizou para as suas medidas o programa Webstone 1.1.

Ainda com relação a desempenho de servidores web, podemos citar um trabalho feito pela Sm@rt Reseller [16]. O título do artigo apresenta o Linux como a escolha para servidor web. Foram analisados o servidor apache no Linux e o IIS 4.0 no Windows NT. No Linux utilizou-se 3 distribuições diferentes: RedHat 5.2, Caldera 1.3 e SuSE 5.3. Cada uma das distribuições apresentou um desempenho diferente, mas todas as três distribuições obtiveram desempenho melhor que o IIS com a SuSE ficando 16% mais rápida e a Caldera 50% mais rápida, entretanto, não foram detalhadas as configurações de cada distribuição. O teste foi feito através de um benchmark o WebBench.

Nesta mesma página existe um link para outro artigo que apresenta resultados do Linux utilizando o SAMBA, que é um software que permite a servidores Linux funcionarem como servidores de arquivos em um ambiente de rede Windows NT. Foi mostrado neste caso que o Linux é mais rápido que o NT como servidor de arquivos com uma carga acima de 12 clientes. Novamente analisaram as três distribuições de Linux citadas anteriormente. Verificaram que com 32 clientes o Red Hat foi 250% mais rápido que o NT. Para o teste foi utilizado o benchmark NetBench 5.01. Nos dois testes citados eles utilizaram o mesmo hardware. Um ponto que poderia ser melhor explorado naquele estudo é relativo à análise das diferenças em cada distribuição do Linux que poderiam significar diferenças de desempenho entre elas.

Em meados de 1999 o site da Microsoft [23] divulgou resultados comparativos entre o NT e o Linux procurando mostrar a superioridade do NT; para isto, foram apresentados alguns resultados de benchmarks da indústria com gráficos comparativos entre o Linux e o Windows NT. Dos dados apresentados no site da Microsoft são referenciados testes realizados pela PC Magazine [20], pela PC Week [19] e pela Mindcraft

[21]. A página da Microsoft não apresenta detalhes dos testes que foram realizados.

Nos experimentos realizados pela Mindcraft o Windows NT foi 2,5 vezes mais rápido que o Linux como um servidor de arquivos e 3,7 vezes mais rápido como servidor Web. Nos testes a Mindcraft utilizou o Windows NT Server 4.0 e o Red Hat Linux 5.2 com o kernel do Linux 2.2.2. O *hardware* utilizado foi um Dell PowerEdge 6300/400 server. No Linux, foi utilizado o Samba 2.0.3 como o servidor de arquivos SMB e o Apache 1.3.4 como o servidor Web. Para o NT foi utilizado o servidor de arquivos SMB que faz parte do NT e o Internet Information Server 4.0 da Microsoft como servidor Web.

4.2.1 Servidor de Arquivos

Nos experimentos da Mindcraft para servidores de arquivos foi utilizado o *benchmark* NetBench. O NetBench controla o processo de leitura e escrita em arquivos do servidor a partir de um conjunto de máquinas. Um maior número de máquinas (clientes) implica em uma maior carga para o servidor. Os resultados para um número de clientes entre 1 e 16, foram melhores para o sistema Linux/Samba que foi 26% mais rápido. Entretanto, quando o número de clientes aumentou o Linux atingiu o pico de desempenho com 48 clientes. O NT, por outro lado, continuou a crescer até 112 clientes. Após atingir o pico ambos sistemas apresentaram uma ligeira queda no desempenho com o acréscimo de clientes, a partir deste ponto o desempenho se estabiliza ficando constante. Devemos observar um cliente Netbench representa uma carga maior que a de um único usuário.

4.2.2 Servidor Web

Para testar os servidores web a Mindcraft utilizou o Webbench. Os testes chegaram a um máximo de 3.771 requisições por segundo com o Windows NT Server 4.0/IIS 4 e a 1.000 requisições por segundo com o Linux/Apache 1.3.4. Além disto o artigo ressalta que, a partir da forma das curvas obtidas com o Windows NT 4.0/IIS 4, este sistema provavelmente não atingiu o pico máximo de desempenho. Afirmando ainda que este pico máximo foi atingido para o Linux/Apache 1.3.4.

4.2.3 Open Benchmark

O primeiro *benchmark* da Mindcraft recebeu muitas críticas da comunidade Linux a respeito de como foram feitas as configurações. Em resposta à estas críticas a Mindcraft fez um segundo *benchmark* [22] utilizando o Windows NT Server 4.0 e o Red Hat Linux 5.2 desta vez com o kernel 2.2.6 do Linux. Eles pediram auxílio no ajuste dos sistemas Linux para os desenvolvedores do Linux, Samba, e Apache. Antes do término deste segundo *benchmark*, frente a novas críticas, a Mindcraft decidiu suspender a publicação do segundo *benchmark* e propôs um *benchmark* aberto.

Nestes novos testes, além da repetição dos testes anteriores, os engenheiros da Red Hat e da Microsoft fizeram pessoalmente os ajustes nos softwares. Os resultados dos *benchmark's* foram semelhantes aos resultados previamente obtidos, com as variações dentro da margem de erro esperada.

4.3 Unix x Windows NT

À parte dos estudos citados anteriormente, temos também alguns textos onde a comparação entre sistemas operacionais é uma questão que se confunde com a preferência do usuário. Um artigo que foi divulgado na Internet de John Kirch [15] é um exemplo; seu título: "Microsoft Windows NT Server 4.0 versus UNIX". O autor tem forte tendência ao Unix e apresenta diversos aspectos em que justifica esta tendência, seu artigo foi publicado na Internet em fevereiro de 1998 e cadastrado em diversos sites de busca, alcançando uma grande popularidade. Na época da sua divulgação chegou à 17000 hits em um único dia. Após o sucesso, Kirch criou uma organização, ou melhor, um site: o www.unix-vs-nt.org. Neste site podemos encontrar o artigo, traduzido em várias línguas, e outras informações sobre o assunto.

Kirch apresenta vários argumentos pró Unix, questionando por exemplo a confiabilidade do NT. No entanto, nos limitaremos a comentar o que o autor escreveu relacionado ao desempenho. O autor começa argumentando que o poder de processamento é muito mais uma função do hardware do que do sistema operacional. Em seguida ele reforça este argumento ao dizer que seria ridículo comparar um IBM SP2, uma Sun Enterprise 10000, ou uma Sun Enterprise 450 com qualquer máquina que a Compaq ou a Dell produz. manifestando que não seria justo confrontar o Windows NT com o Unix em hardwares diferentes.

O autor ressalta ainda que uma análise de desempenho baseada em *benchmarks* deixa a desejar, porque não existem muitas alternativas e normalmente os *benchmarks* existentes estão direcionados a áreas específicas como desempenho de servidores web.

Kirch escreve também sobre o fato dos sistemas UNIX possuírem um *kernel* que pode ser customizado para conter somente os programas necessários de acordo com a utilização da máquina. Assim o Linux ou o FreeBSD podem ser mais eficientes que o NT, pois são sistemas operacionais que requerem menos recursos. Ele argumenta que o UNIX não necessita de uma interface gráfica com o usuário para funcionar como o NT; acrescentando que gráficos requerem grande quantidade de memória e espaço em disco. Nesta mesma linha de argumentação, faz críticas também à utilização de arquivos de som pelo NT.

Ainda com relação ao item desempenho, Kirch apresenta o projeto Beowulf da NASA. Este projeto utiliza o Linux em um conjunto de máquinas ligadas em rede. Ele ressalta os excelentes resultados alcançados com o Beowulf a um preço modesto. O Beowulf é um sistema que utiliza uma arquitetura de processamento paralela composta por um conjunto de microcomputadores comerciais executando o Linux. Uma das maquinas, o nó

servidor, distribui as tarefas de processamento para todas as outras maquinas. Um sistema Beowulf com 24 nós custava na época US$57.000, enquanto os supercomputadores comerciais custavam entre US$10 milhões e US$30 milhões. Este sistema Beowulf de 24 nós alcança um throughput de 2.4 gigabytes por segundo, o que significa que um disco rígido de 200 GB pode ser lido em apenas 20 segundos. Atualmente qualquer pessoa pode comprar um CD-ROM Beowulf distribuído em um pacote da Red Hat por US$29. Um sistema Beowulf, o Avalon, da universidade de Los Alamos, em novembro de 1998, figurava na posição 113 da lista dos 500 computadores mais rápidos do mundo

(http://www.top500.org).

Capítulo 5

Metodologia

5.1 Introdução

Nesta seção procuraremos mostrar um pouco da nossa trajetória nos diversos experimentos realizados durante este trabalho. Serão descritos os experimentos realizados porém, deixaremos os resultados para serem apresentados no capítulo 6. Relacionaremos aqui também, as características dos computadores utilizados em cada experimento.

Ao iniciar o estudo dividimos a nossa tarefa em duas fases: análise em uma máquina isolada e análise do comportamento em rede. Na primeira fase nosso objetivo será avaliar a capacidade com a execução de vários programas simultâneos. Neste caso a idéia foi medir o número máximo de processos possíveis e o custo das operações ligadas ao multi-processamento, especialmente a troca de contexto. Na segunda fase utilizamos uma rede TCP/IP de 10 MB/s e construímos programas específicos; para testes da comunicação e para gerar cargas à servidores Web.

Os programas desenvolvidos e as ferramentas de desenvolvimento serão descritos mostrando um pouco dos recursos e ferramentas de programação utilizados. Um outro ponto importante tratado aqui é o histórico de cada experimento, onde apontamos os problemas encontrados e as decisões que foram tomadas para resolvê-los. As listagens dos programas estão disponíveis no apêndice, ou através do site http://www.ime.usp.br/~eugenio.

5.2 Sobrecarga de Processos

Para examinar o desempenho do sistema com processos que exigem muita CPU foi utilizado um benchmark sintético, o Linpack. O Linpack é um conjunto de rotinas para álgebra linear [13] e [17]. Este benchmark se carac-

teriza por um grande número de operações aritméticas de ponto flutuante, utilizadas para resolver um sistema de equações lineares. As principais rotinas utilizadas são a DGEFA e DGESL. A primeira faz o fatoramento de uma matriz através da eliminação gaussiana e, a segunda resolve o sistema a partir da matriz calculada. Todos os experimentos foram feitos com um matriz de ordem 100.

O mesmo benchmark foi executado nos dois sistemas para comparar os resultados. Para esta comparação foram feitas várias medições a partir da versão do Linpack para a Linguagem C que pode ser obtido em www.netlib.org.

Inicialmente foi gerado um executável a partir do código fonte em C do linpack. Este executável foi compilado nos dois ambientes com o mesmo compilador. O compilador utilizado foi o GCC. No Windows NT utilizamos uma versão para DOS do GCC o DJGPP, no Linux o GCC que faz parte da distribuição Debian.

Observou-se que havia muita diferença nos valores da execução do benchmark para os dois sistemas na mesma máquina. Os valores obtidos para um único programa executando eram 6 % maiores para o NT, entretanto, para 10 programas simultâneos obtínhamos um aumento no tempo médio do NT de quase 50 %. Verificamos assim que as medidas de tempo que eram utilizadas não estavam coerentes.

Concluiu-se que o problema estava relacionado com o fato do GCC gerar um programa que executa na máquina virtual DOS no NT (NTDVM). A execução do programa em uma máquina virtual DOS estava levando a uma sobrecarga considerável. No caso da execução simultânea de várias instâncias, este problema se acentuava pois cada programa necessitava de uma máquina virtual para a sua execução. Um outro aspecto deste problema era o de não estarmos utilizando um programa com código específico para o NT.

Para criar um programa que fizesse um bom uso das características do NT este programa deveria ser gerado de acordo com a API Win32. Devido a este problema optou-se por mudar a estratégia e compilar o benchmark com compiladores diferentes. Desta forma seria possível gerar o benchmark para a API Win32.

Foram testados os seguintes compiladores C++ no NT: IBM Visual Age, Borland C++ e Visual C++. Os resultados obtidos foram muito diferentes, cada programa executável gerado tinha um desempenho melhor ou pior de acordo com o compilador utilizado. Estas diferenças no tempo de execução estão ligadas ao modo como o programa é gerado, com otimizações e particularidades inerentes ao compilador utilizado. Estes resultados não poderiam então ser comparados com os resultados obtidos no Linux.

A partir destes problemas chegamos a um ponto em que não seria possível

gerar o programa de teste através de compiladores diferentes nas duas plataformas. Foi necessário optar-se por uma solução que mantivesse o código do benchmark idêntico nas duas plataformas. Desta forma, a análise seria baseada em um benchmark único para os dois sistemas operacionais.

A solução para que o código fosse o mesmo nas duas plataformas era obter uma versão deste programa em linguagem de máquina. Nosso desafio foi, então, gerar o código de máquina a partir do programa fonte em C do Linpack. Entretanto, converter o código do Linpack para linguagem de máquina não é tão simples. Começamos a pensar então em uma solução alternativa.

Resolvemos, enfim, o problema de criar a versão em código de máquina do Linpack utilizando o próprio código original em C e os recursos do compilador gcc. A idéia foi, partindo do código compilado do programa em C, separar as instruções relacionadas aos cálculos do algoritmo eliminado rotinas de medidas de tempo e impressão de valores. Com este novo código foi gerado então um programa em cada ambiente. Entretanto para a gerar o módulo do linpack em linguagem de máquina foi necessário operar ainda outras modificações no programa fonte em C:

- Eliminar a dependência de quaisquer chamadas a rotinas de bibliotecas. Desta forma o programa estaria totalmente contido dentro do seu próprio código, não necessitando fazer chamadas a bibliotecas externas.

- retirar os trechos do programa relacionados com medidas de tempo, esta alteração está ligada à anterior.

- eliminar a declaração de variáveis estáticas. As variáveis estáticas do C são variáveis que mantém o seu valor entre chamadas de função, no nosso caso, tais variáveis seriam localizadas em posições de memória fora da pilha de execução da função.

- nomear a função main para linp. Nosso programa após compilado geraria um módulo objeto com uma função diferente da função main, que, para o compilador C, representa o ponto de entrada de um programa. Assim o compilador geraria uma saída sem as rotinas necessárias para o caso da função main.

Compilou-se, então, o programa com o gcc para Linux gerando-se um arquivo objeto contendo a função linp() que representa o programa correspondente ao benchmark desejado. Através do comando *strip* eliminou-se as informações do arquivo objeto que não seriam necessárias. Em seguida, a partir de uma

verificação deste arquivo, foi possível identificar o ponto de entrada da função linp(). Assim foi gerada a versão em linguagem de máquina do benchmark.

Este código do Linpack em linguagem de máquina foi transformado em um vetor de bytes que poderia ser então incluído em um programa C. Atribuindo este vetor a um ponteiro para uma função, Linp(), seria então possível executar o mesmo conjunto de comandos nos dois sistemas operacionais. O próximo passo foi gerar dois programas, para o Linux e para o Windows NT, com a chamada a esta função. Um esboço do código para a chamada da função Linp() é apresentado na figura 5.1.

```
#include "linp.inc"
void main()
{
    double t1,t2;
    void (*Linp)();
    Linp = (void (*)()) linpcode;
    t1 =  dtime();
    Linp();
    t2 = dtime();
    printf("Inicio %f Tempo %f\n", t1,t2-t1);
}
```

Figura 5.1: Programa que inclui o código compilado do Linpack

Compilando com o GCC no Linux um programa escrito com a chamada à função Linp() funcionou na primeira tentativa. No caso do Windows NT isto não ocorreu: tanto com o Visual C++ quanto com o IBM Visual Age, quando a função tentava reservar espaço na pilha era gerada uma exceção. Após algumas tentativas depurando o código para o Visual C++ verificamos que este chamava antes das chamadas de função uma rotina, _chkstk, que alocava o espaço da pilha aos poucos. Foi incluído, então, antes da chamada da função Linp() a chamada à função _chkstk, alocando-se o espaço necessário na pilha e um comando em linguagem de máquina para voltar a pilha para a posição inicial. A figura 5.2 mostra como ficaria o código da figura 5.1 com a alteração descrita. Após a inclusão deste código a versão para Windows NT passou a funcionar. Com isto conseguiu-se gerar dois programas que executavam o mesmo código de benchmark nos dois sistemas.

Ao medir o tempo de execução do programa Linp() este foi praticamente o mesmo tanto na versão para o NT quanto para o Linux. Foram realizados, então, testes executando várias instâncias destes programas simultaneamente onde mediu-se o tempo de início e de fim da execução de cada instância. A

```
#include "linp.inc"
void _chkstk();
void main()
{
    double t1,t2;
    void (*Linp)();
    Linp = (void (*)()) linpcode;
    f
    __asm {
      mov  eax,0x4eef0
      call _chkstk
      add  esp,0x4eef0
    };

    t1 =  dtime();
    Linp();
    t2 = dtime();
    printf("Inicio %f Tempo %f\n", t1,t2-t1);
}
```

Figura 5.2: Programa com o Linpack chamando chkstk

partir destes tempos calculamos o tempo total que cada sistema operacional gastou para executar todas as instâncias, obtido da diferença entre menor tempo inicial e o maior tempo final. Os experimentos foram feitos em um computador com processador Intel Pentium-MMX 200 Mhz, com 32 MB de memória. Na medição dos tempos utilizou-se funções diferentes no Windows NT e no Linux. No NT foi utilizada a função *GetTickCount* que faz parte da API Win32 e no Linux utilizou-se a função *gettimeofday* do C. A diferença destas funções não representa um problema porque é desprezível em relação ao tempo total de execução da função Linp(). O número total de processos utilizado foi de 100, 200 e 300.

5.3 Rede TCP/IP

Para analisar o desempenho dos sistemas em redes TCP/IP foram desenvolvidos programas simples numa configuração cliente-servidor. Nestes programas utilizou-se as rotinas básicas da rede TCP/IP, ou seja, as chamadas de

soquetes [8] do TCP/IP dos dois sistemas operacionais.

Para cada sistema operacional foram desenvolvidos dois programas, um cliente e um servidor. O servidor funciona como um eco do cliente, em outras palavras, ao receber uma requisição do cliente o servidor retorna esta mesma requisição ao cliente. O tamanho da requisição utilizada nos testes foi de 64 bytes.

Para as medidas o cliente e o servidor foram executados no mesmo computador utilizado nos testes com o Linpack.

No Linux os programas foram compilados com a versão do gcc que acompanha a distribuição do Debian. No Windows NT foram desenvolvidas duas versões dos programas utilizando-se o MS-Visual C++ 5.0. A primeira versão foi desenvolvida com as mesmas funções de manipulação de soquetes existentes no Linux, já a segunda foi baseada em classes existentes no Visual C++ 5.0 que fazem parte do MFC ("Microsoft Foundations Classes").

5.4 Servidores Web

Na análise de servidores Web utilizamos um programa cliente HTTP que gera requisições de uma página específica para o servidor e aguarda a resposta. O tempo entre a requisição e a resposta do servidor é então calculado pelo programa cliente. Este tempo equivale assim ao tempo gasto do início da requisição até a chegada da resposta completa, que é o tempo que o usuário realmente "enxerga".

O programa cliente foi configurado para fazer requisições sucessivas, ou seja, após cada pedido completado é feito um novo pedido. Desta forma não está sendo levado em conta o tempo entre requisições (*think time*) [6], com o programa cliente fazendo o máximo de requisições possíveis para um dado intervalo de tempo. Com esta configuração cada instância do programa cliente se comporta gerando uma carga maior do que um usuário real é capaz de gerar. Não foram utilizados monitores disponíveis nos sistemas operacionais ou nos servidores Web utilizados. Esta abordagem permite que se possa analisar através desta mesma metodologia outros sistemas servidores WEB, além disto, a sobrecarga que seria gerada com estes monitores foi evitada. Os programas clientes Web foram escritos de forma que fosse possível rodar várias instâncias dos mesmos com o objetivo de aumentar o número de requisições por segundo que chegam ao servidor Web.

5.4.1 Ambiente de Testes

Foram utilizados para os testes dois computadores conectados entre si por um único cabo e sem conexão com nenhum outro computador. A configuração destes computadores é apresentada na tabela 5.1.

	Memória	Processador	Disco Rígido	placa de rede
Computador 1	32 M	Pentium MMX 200	2,1 Gb	NE2000
Computador 2	32 M	Pentium 200	2,1 Gb	NE2000

Tabela 5.1: Configuração dos equipamentos utilizados

No Computador 1, utilizado como servidor Web, foram instalados o Debian Linux 2.0 e o Apache 1.3 (que acompanha esta distribuição), o Windows NT 4.0 com *Service Pack* 3 e o IIS 4.0. Uma versão beta do Apache 1.3 para Windows NT também foi instalada no computador

1. Todos os softwares não tiveram nenhuma configuração especial. No computador 2 que foi usado como cliente, utilizou-se o sistema operacional Debian Linux 2.0.

Capítulo 6

Resultados

6.1 Introdução

Ao planejar a avaliação das características de desempenho, frente à dificuldade de uma análise detalhada, optamos por concentrar nossos esforços na verificação de situações específicas. Estas situações permitiriam verificar aspectos do desempenho ligados à funções importantes dos sistemas operacionais modernos.

Como já relatamos anteriormente dividimos o trabalho em duas fases: testes em uma máquina isolada e testes de máquinas em rede. Na primeira fase foi utilizado o Linpack, um *benchmark* muito conhecido e utilizado, ligeiramente

modificado. Com este *benchmark* realizamos os testes de sobrecarga de processos. Na segunda fase escrevemos programas que utilizaram o TCP/IP. Medimos então, os tempos de resposta para as requisições via rede. Nas medidas de rede foram feitos também testes com servidores Web.

6.2 Sobrecarga de Processos

Os testes a seguir representam aspectos do comportamento dos sistemas com sobre-carga de CPU. O objetivo foi verificar como o desempenho de um programa seria afetado com a execução simultânea de um conjunto de instâncias deste programa. O programa se caracterizava em um benchmark sintético que faz uso intenso de CPU.

Além da verificação do desempenho na execução, estávamos interessados também em como os processos seriam escalonados ao longo do tempo. Com relação ao tempo de execução, o esperado na situação ideal, é que o tempo total de execução deve ser igual ao tempo de um processo multiplicado pelo

número de processos que forem executados simultaneamente. Neste caso assumindo o uso de 100% da capacidade de processamento da máquina.

O benchmark escolhido para este teste foi um programa gerado a partir do Linpack, Linp(). Este programa foi especialmente construído para possuir o mesmo código de máquina nos dois sistemas. Maiores detalhes sobre como foi gerado este programa podem ser vistos no capítulo 5.

Nos testes realizados foram iniciadas entre 100 a 300 instâncias do Linpack. Para iniciar a execução foi utilizado um programa que disparava as rotinas em seqüência.

Ao executar vários processos os resultados obtidos mostraram uma diferença mínima entre os tempos totais de execução. A princípio parecia que não encontraríamos diferenças significativas no comportamento dos dois sistemas. Entretanto analisando os dados de execução individual de cada processo verificou- se uma diferença ligada ao modo como é feito o escalonamento dos processos em cada sistema.

Foi observado que o escalonamento dos processos no Windows NT faz com que os processos colocados para executar não iniciem ao mesmo tempo. O que ocorre é que um conjunto dos processos é iniciado, enquanto que os próximos processos são iniciados com um atraso variável.

No Linux o escalonador se comporta de forma a iniciar todos os processos com uma diferença mínima de tempo, ou seja, quase que simultaneamente.

6.2.1 100 processos simultâneos

Com 100 instâncias do linpack observou-se que o tempo total de execução ficou em torno de 15.000 segundos no NT e no Linux, com uma diferença muito pequena para cada teste realizado. Podemos observar, entretanto, um comportamento diferente dos dois sistemas com relação a distribuição dos processos dentro deste intervalo de tempo. A figura 6.1 mostra o comportamento do NT quando disparamos 100 instâncias do linpack e a 6.2 o comportamento do Linux.

Verifica-se que o NT começa a executar em torno de 50 processos e depois de um tempo vai aumentando o número de processos. O Linux começa todos os 100 processos simultaneamente. Isto leva a um tempo médio de execução de cada processo menor no NT que no Linux, não afetando o tempo total.

Podemos concluir, ainda, destes resultados que o custo das trocas de contexto não foi significativo pois o tempo total dividido pelo número de processos ficou próximo de 150 segundos nos dois sistemas, o que é equivalente ao tempo médio de um processo.

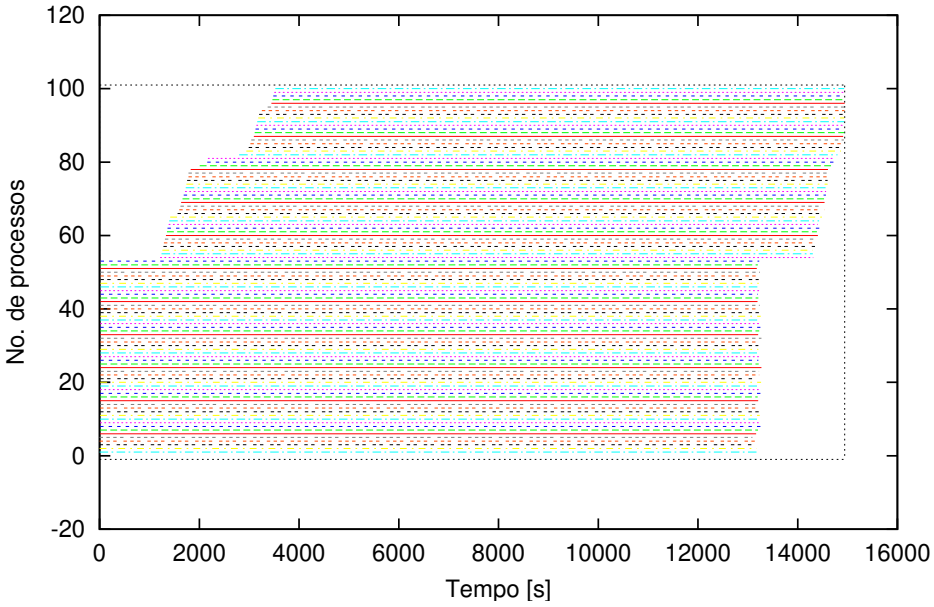

Figura 6.1: 100 Processos no NT

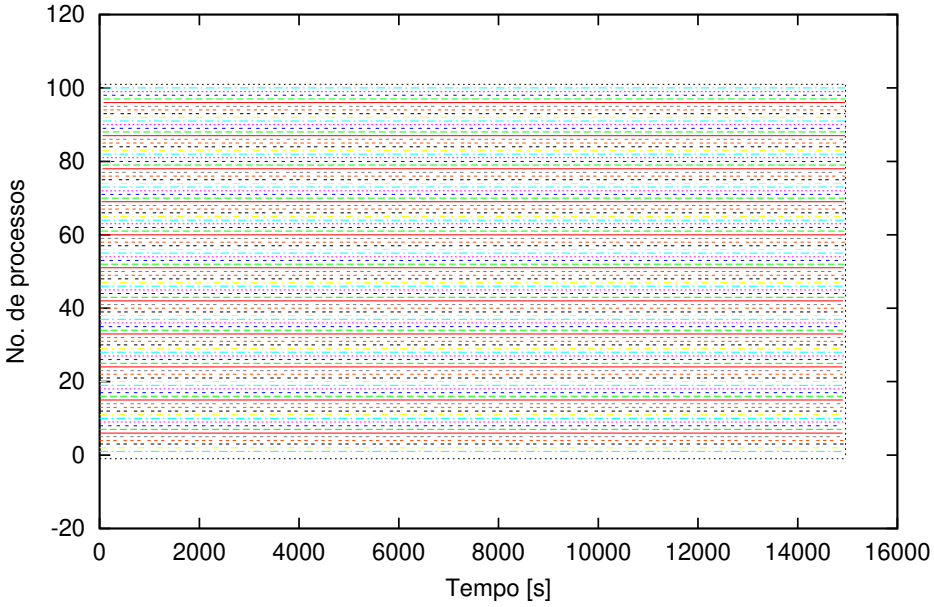

Figura 6.2: 100 Processos no Linux

6.2.2 200 processos simultâneos

Ao tentar iniciar 200 processos simultâneos no NT observamos que o mesmo atingia um limite conseguindo executar até 172 processos. O tempo total

para rodar os 172 processos no NT foi de 25.714 segundos, que dividindo pelo número de processos resulta em 149,5 segundos por processo. O Linux executou os 200 processos com um tempo total de 30.031 segundos, o que representa 150,2 segundos por processo. Novamente, o tempo por processo é bem próximo do tempo de um único processo. A figura 6.3 mostra que o NT continuou apresentando um comportamento diferente, neste caso, o NT começa executando 92 processos com um certo deslocamento no tempo de início de cada um e, após terminar estes processos, começa a executar os outros, novamente com um deslocamento no tempo de início de cada processo.

Se considerarmos o tempo médio de execução o NT teve um tempo menor que o do Linux para cada processo, devido à característica do escalonador do Windows NT de não iniciar todos os processos ao mesmo tempo. Este comportamento pode ser indesejável em algumas situações, onde temos, por exemplo, acesso de diversos usuários e ocorre que uns são prontamente atendidos, enquanto que outros ficam esperando. O Linux se apresentou mais uniforme neste experimento o que pode ser uma importante característica do Linux em relação ao NT.

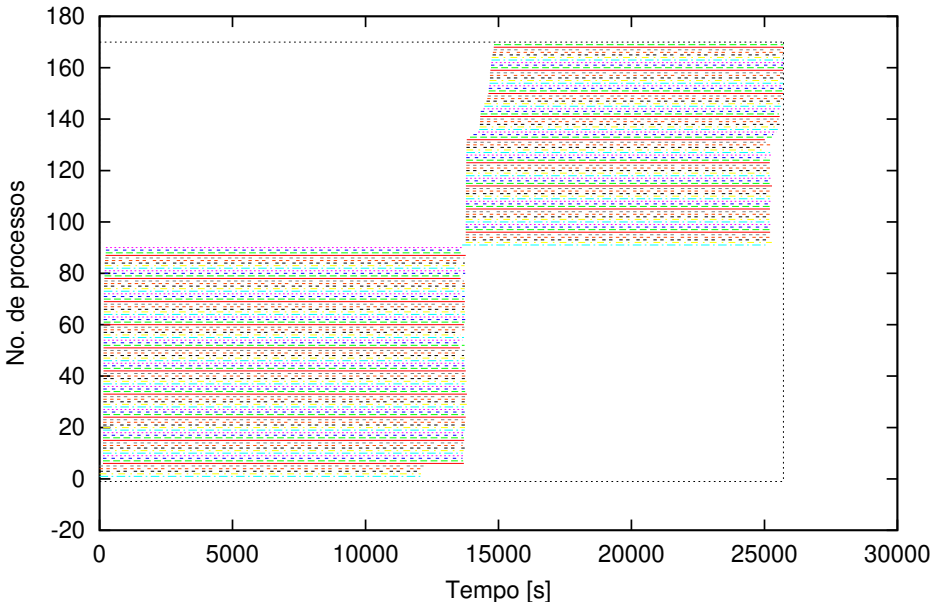

Figura 6.3: Tentativa de 200 Processos no NT

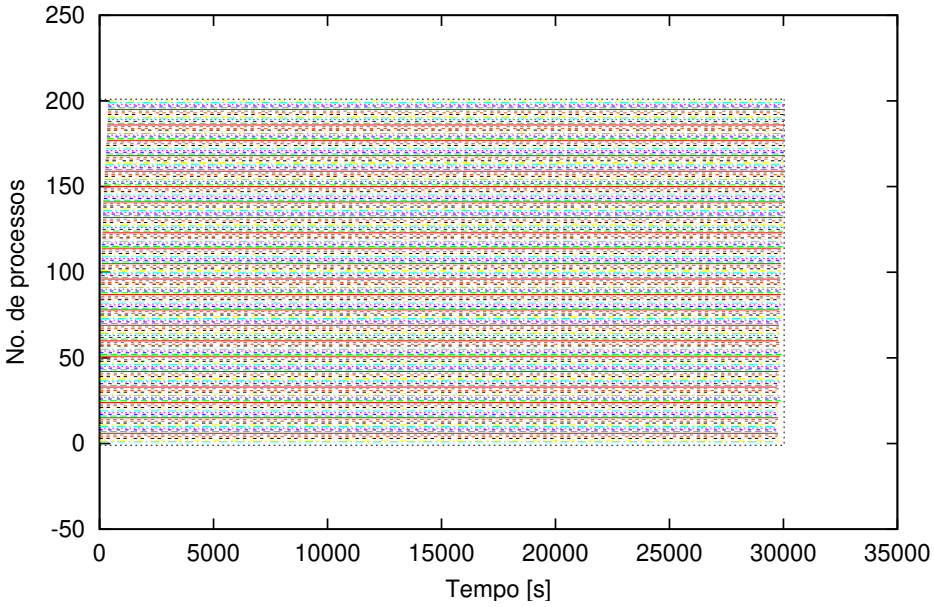

Figura 6.4: 200 Processos no Linux

6.2.3 300 processos simultâneos

Para verificar o limite no número de processos no Linux iniciamos um teste com 300 processos. O Linux conseguiu executar até o fim 243 processos. O sistema conseguiu criar todos os 300 processos mas uma parte deles foi abortada no início.

Verificou-se que a inicialização dos processos, através das chamadas fork() e exec(), eram executadas para todos os 300 processos com sucesso. Mas após esta inicialização alguns processos eram abortados. O que ocorreu foi que na tentativa de alocar espaço na pilha para a execução da função linp não havia espaço disponível. Isto poderia ser evitado se o programa que faz a chamada à função linp() em linguagem de máquina verificasse o espaço disponível antes da chamada. Isto foi verificado com uma alteração no programa que disparava os processos. A partir desta alteração conseguimos identificar que os processos que não continuavam executando recebiam um sinal do sistema operacional após iniciarem. Este sinal era o SIGSEGV que indica violação de segmento.

Observando a figura 6.5 verifica-se um pequeno deslocamento no início da execução de uma parte dos processos no Linux, entretanto, este deslocamento não é comparável ao que ocorre no NT.

O tempo total para os 243 processos foi de 26.930 segundos o que dá um tempo de 110,8 segundos por processo. Neste caso o número de iterações no

linpack era menor e o tempo de um único processo era de 45,05. Com isto o tempo por processo foi bem maior que o esperado.

Esta diferença no tempo médio se deve ao fato de que o sistema teve que utilizar a memória virtual para comportar todos os processos. Desta forma o tempo gasto foi maior devido à necessidade de acesso a disco.

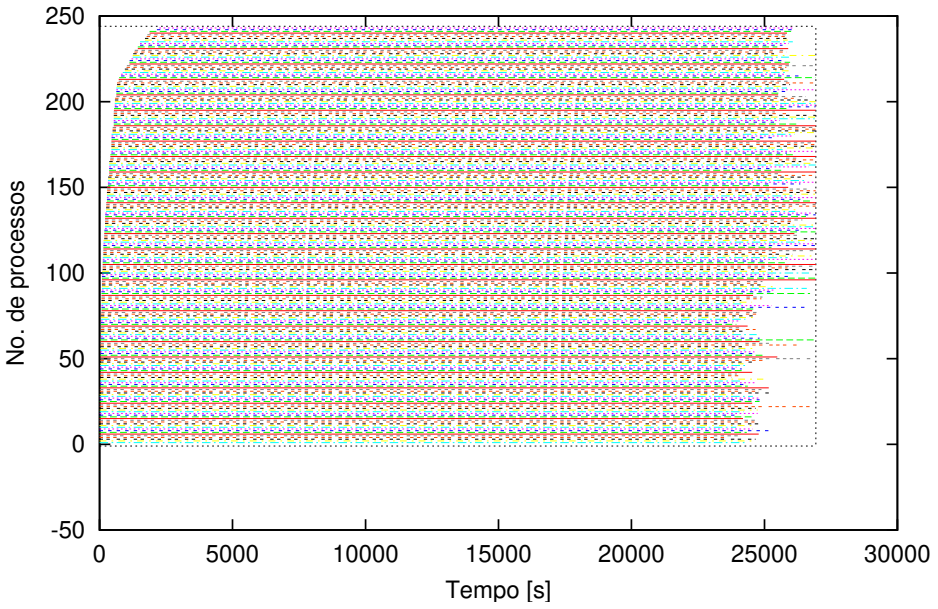

Figura 6.5: Tentativa de 300 Processos no Linux

6.2.4 Escalonamento de processos no Windows NT

Através dos artigos de [11] e [10], obtivemos algumas informações sobre o escalonamento de processos do Windows NT, que podem servir como um indicativo do comportamento observado nas figuras 6.1 e 6.3. O Windows NT é um sistema operacional multi-tarefa preemptivo. Isto significa que o NT permite que vários programas executem simultaneamente, alterando o programa em execução a cada instante com uma frequência suficiente para criar a ilusão de que cada programa é o único programa a executa na máquina.

A maneira como o sistema compartilha a CPU com várias tarefas é complicada, a decisão sobre qual será o próximo programa a iniciar a sua execução deve ser tomada várias vezes por segundo, o mais rápido possível, este é o trabalho do escalonador de processos do NT. A unidade básica do escalonamento no NT é uma *thread*, que é uma uma linha de execução dentro de um processo. O escalonador não faz diferença entre *threads* e processos, ele

examina as prioridades de todas as *threads* prontas para executar em um dado momento para escolher uma.

As prioridades de cada *thread* são definidas por uma numeração de 1 a 31, com os maiores números representando a maior prioridade. A prioridade 0 é utilizada para a *thread* de sistema inativo, que executa quando não existem outras prontas para executar, as de 16 a 31 são reservadas para operações críticas (aplicações de tempo real) e as de 1 a 15 (prioridades dinâmicas) são utilizadas para as demais aplicações. O sistema mantém uma lista de execução para cada prioridade, num total de 31 entradas (a prioridade 0 não faz parte da lista). Estas entradas são percorridas da entrada de maior prioridade para a menor quando o sistema precisa selecionar uma nova *thread* para colocar em execução.

O escalonador limita a execução de uma *thread* em unidades de tempo (quantums), que têm uma duração especifica podendo variar de 120 ms a 20ms. O algoritmo do escalonador é acionado se o quantum da *thread* em execução expira, se esta é bloqueada na espera de um evento ou se outra *thread* fica pronta para executar.

Em algumas situações as *thread's* com prioridades dinâmicas podem ter a sua prioridade aumentada. Este aumento é realizado, por exemplo, para impedir que uma determinada *thread* fique esperando indefinidamente para executar "starvation". No NT o algoritmo que previne este problema procura por *threads* que não foram executadas a mais de 3 segundos, na lista de execução partindo das entradas de menor prioridade. Quando é encontrada uma que se encaixa nesta característica o NT dobra o seu quantum e aumenta sua prioridade para 15, forçando a execução da mesma. Quando esta *thread* termina o seu quantum ou é bloqueada o seu quantum e a sua prioridade retornam ao valor original. Este mecanismo é conhecido como "anti-starvation".

O comportamento apresentado na figura 6.1 pode ser ligado a uma falha no algoritmo "anti-starvation", que fez com que uma *thread* que já executou a mais de 3 segundos ganhe novamente a CPU antes que uma que nunca foi executada. Já no caso da figura 6.3 o problema pode ser ligado à alocação de memória, neste caso as últimas instâncias ficaram bloqueadas aguardando a liberação de memória para a sua execução. Neste caso o mecanismo de "anti-starvation" deve ter sido desligado, o que explica a diferença da primeira parte do gráfico da 6.3 com relação à figura 6.1.

6.2.5 Escalonamento de processos no Windows NT

Através dos artigos de [11] e [10], obtivemos algumas informações sobre o escalonamento de processos do Windows NT, que podem servir como um

indicativo do comportamento observado nas figuras 6.1 e 6.3. O Windows NT é um sistema operacional multi-tarefa preemptivo. Isto significa que o NT permite que vários programas executem simultaneamente, alterando o programa em execução a cada instante com uma frequência suficiente para criar a ilusão de que cada programa é o único programa a executa na máquina.

A maneira como o sistema compartilha a CPU com várias tarefas é complicada, a decisão sobre qual será o próximo programa a iniciar a sua execução deve ser tomada várias vezes por segundo, o mais rápido possível, este é o trabalho do escalonador de processos do NT. A unidade básica do escalonamento no NT é uma *thread*, que é uma uma linha de execução dentro de um processo. O escalonador não faz diferença entre *threads* e processos, ele examina as prioridades de todas as *threads* prontas para executar em um dado momento para escolher uma.

As prioridades de cada *thread* são definidas por uma numeração de 1 a 31, com os maiores números representando a maior prioridade. A prioridade 0 é utilizada para a *thread* de sistema inativo, que executa quando não existem outras prontas para executar, as de 16 a 31 são reservadas para operações críticas (aplicações de tempo real) e as de 1 a 15 (prioridades dinâmicas) são utilizadas para as demais aplicações. O sistema mantém uma lista de execução para cada prioridade, num total de 31 entradas (a prioridade 0 não faz parte da lista). Estas entradas são percorridas da entrada de maior prioridade para a menor quando o sistema precisa selecionar uma nova *thread* para colocar em execução.

O escalonador limita a execução de uma *thread* em unidades de tempo (quantums), que têm uma duração especifica podendo variar de 120 ms a 20ms. O algoritmo do escalonador é acionado se o quantum da *thread* em execução expira, se esta é bloqueada na espera de um evento ou se outra *thread* fica pronta para executar.

Em algumas situações as *thread's* com prioridades dinâmicas podem ter a sua prioridade aumentada. Este aumento é realizado, por exemplo, para impedir que uma determinada *thread* fique esperando indefinidamente para executar "starvation". No NT o algoritmo que previne este problema procura por *threads* que não foram executadas a mais de 3 segundos, na lista de execução partindo das entradas de menor prioridade. Quando é encontrada uma que se encaixa nesta característica o NT dobra o seu quantum e aumenta sua prioridade para 15, forçando a execução da mesma. Quando esta *thread* termina o seu quantum ou é bloqueada o seu quantum e a sua prioridade retornam ao valor original. Este mecanismo é conhecido como "anti-starvation".

O comportamento apresentado na figura 6.1 pode ser ligado a uma falha no algoritmo "anti-starvation", que fez com que uma *thread* que já executou

a mais de 3 segundos ganhe novamente a CPU antes que uma que nunca foi executada. Já no caso da figura 6.3 o problema pode ser ligado à alocação de memória, neste caso as últimas instâncias ficaram bloqueadas aguardando a liberação de memória para a sua execução. Neste caso o mecanismo de "anti-starvation" deve ter sido desligado, o que explica a diferença da primeira parte do gráfico da 6.3 com relação à figura 6.1.

6.3 Rede TCP/IP

Para analisar o desempenho dos sistemas em redes TCP/IP foram desenvolvidos programas simples numa configuração cliente-servidor. Nestes programas utilizou- se as rotinas básicas da rede TCP/IP, ou seja, as chamadas de soquetes do TCP/IP dos dois sistemas operacionais.

Para cada sistema operacional foram desenvolvidos dois programas, um cliente e um servidor. O servidor funcionava como um eco do cliente, em outras palavras, ao receber uma requisição do cliente o servidor retorna esta mesma requisição ao cliente. Estes programas foram testados executando o cliente e o servidor no mesmo computador, que foi o computador 1.

No Linux os programas foram compilados com a versão do gcc que acompanha a distribuição do Debian. No Windows NT foram desenvolvidas duas versões dos programas utilizando-se o MS-Visual C++ 5.0. Na primeira versão usou-se as mesmas funções de manipulação de soquetes existentes no Linux, já na segunda foram utilizadas classes existentes no Visual C++ 5.0 que fazem parte do MFC ("Microsoft Foundations Classes"). Os resultados obtidos são apresentados na tabela 6.1, podemos observar que no Windows NT os programas apresentaram um desempenho inferior ao do Linux. Os programas no Windows NT responderam as requisições TCP/IP em um tempo cerca de duas vezes mais lento no caso da versão sem MFC e de 10 vezes mais lento ao se utilizar as classes do MFC.

Dos resultados obtidos foi possível observar que a utilização do MFC gera uma sobrecarga considerável. No caso das chamadas nativas aos soquetes TCP/IP, o melhor desempenho da versão Linux indica que as bibliotecas utilizadas e o compilador do Linux fizeram o programa final ficar mais rápido.

6.4 Servidores WEB

Para testar o desempenho dos servidores HTTP, utilizou-se um programa cliente gerando requisições para o servidor. Inicialmente foram feitos testes com uma única instância do programa cliente rodando no computador 2,

Nº de Requisições	Tempo [s]		
	Linux	Windows NT	Windows NT (com MFC)
5000	0,79	1,75	8,80
12000	1,90	4,24	21,20
25000	4,03	8,98	44,18
50000	8,11	17,59	88,88
100000	15,64	36,00	176,20

Tabela 6.1: Tempos de resposta dos programas cliente-servidor usando soquetes

acessando o servidor do computador 1. Os dois computadores foram ligados em rede através de um cabo isolado evitando, assim, possíveis interferências de outros computadores da rede. O programa cliente requisitava uma página de 1 KB. Foi medido o tempo para um número de requisições entre 1000 e 10000. O resultado obtido é apresentado na tabela 6.2 e na figura 6.6. Neste experimento podemos ver que o número de requisições atendidas por segundo permaneceu constante ao longo do tempo.

Nº de Requisições	Tempo [s]	
	Apache 1.3 Linux	IIS 4.0 Windows NT
1000	342,8	302,0
2000	637,2	420,2
3000	953,2	623,6
4000	1303,6	892,4
5000	1595,8	1038,2
6000	1911,8	1247,0
7000	2227,4	1453,0
8000	2545,4	1660,4
9000	2867,6	1873,2
10000	3189,0	2082,0

Tabela 6.2: Tempo gasto para um número fixo de requisições

Com o objetivo aumentar o número de requisições para o mesmo intervalo de tempo foram executadas várias instâncias do programa cliente (pelo menos 15 vezes). Nesta nova configuração todas as instâncias do programa cliente foram iniciadas e rodaram por 30 segundos sendo medido então o número de vezes que cada cliente conseguiu enviar uma requisição e receber a resposta. Foram feitos testes com páginas de 1 KB, 10 KB e 100 KB. Optou-se por utilizar uma única página para cada teste tendo em conta que

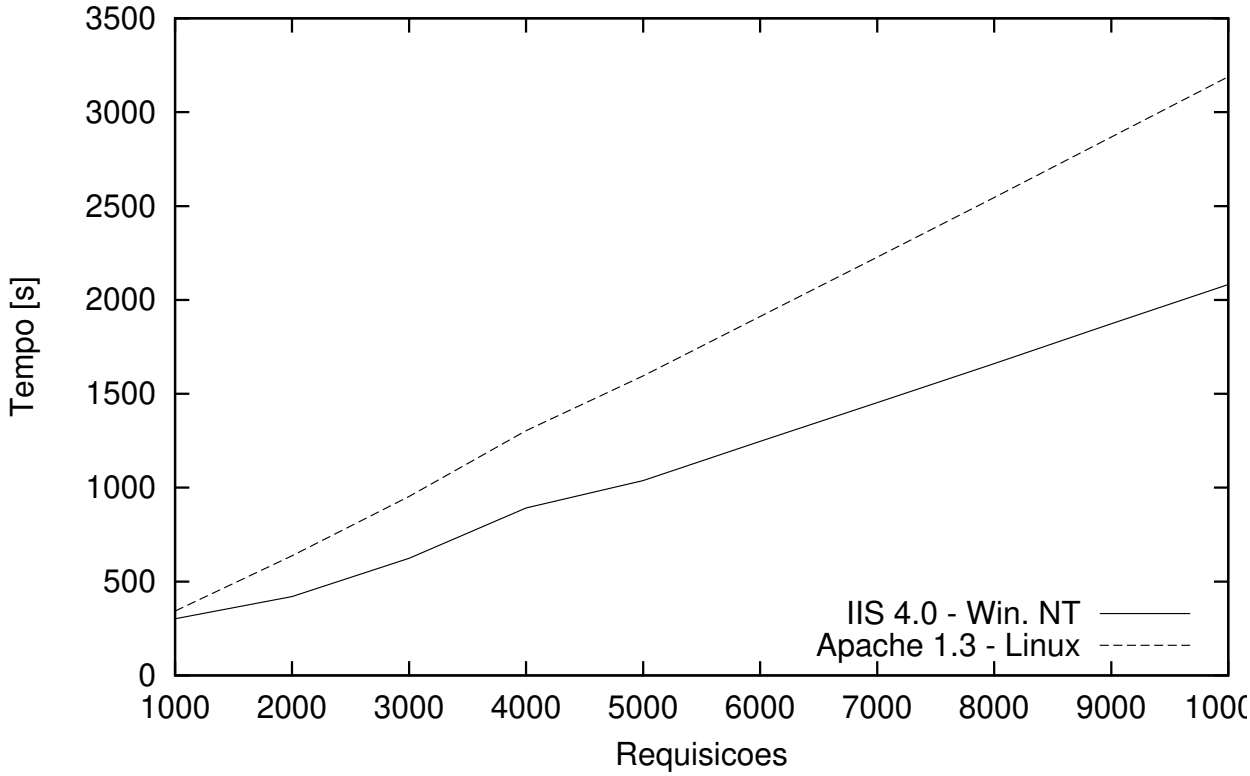

Figura 6.6: Tempo gasto para um número fixo de requisições

esta estará armazenada no cache do servidor que desta forma não utilizará o disco durante os testes para fazer a busca da página. O objetivo foi tentar provocar uma saturação do conjunto cliente-servidor a medida que a taxa de requisições por segundo aumentasse. Os resultados obtidos estão resumidos na tabela 6.3 e nas figuras 6.7 a 6.9. Observou-se uma estabilidade muito grande nos valores medidos e as variações foram desprezíveis.

Analisando os resultados podemos constatar que o aumento no número de clientes simultâneos e o consequente aumento no número de requisições por segundo, fez com que o *throughput* aumentasse. Este aumento, entretanto, exibe um comportamento com uma tendência a saturar quando o número de clientes aumenta. Comparando os dois sistemas verificamos que o Apache 1.3 no Linux (Apache) teve um desempenho pior para páginas até 10 KB que o IIS 4.0 no Windows NT (IIS). Para páginas de 100 KB o desempenho das duas configurações testadas foi semelhante conforme mostra a figura 6.9, com o Apache apresentando um desempenho ligeiramente melhor. A figura 6.7 é a que mostra a maior diferença entre os dois servidores analisados, neste

Servidor	Tamanho da Página	Número de Clientes				
		1	2	4	8	16
Apache Linux	1k	0,086	0,129	0,198	0,199	0,201
IIS Windows NT	1k	0,197	0,273	0,314	0,349	0,356
Apache Linux	10k	0,318	0,592	0,761	0,823	0,858
IIS Windows NT	10k	0,311	0,635	0,806	0,858	0,869
Apache Linux	100k	0,795	1,015	1,076	1,092	1,084
IIS Windows NT	100k	0,711	1,006	1,047	1,058	1,068

Tabela 6.3: Variação do throughput com o número de processos cliente

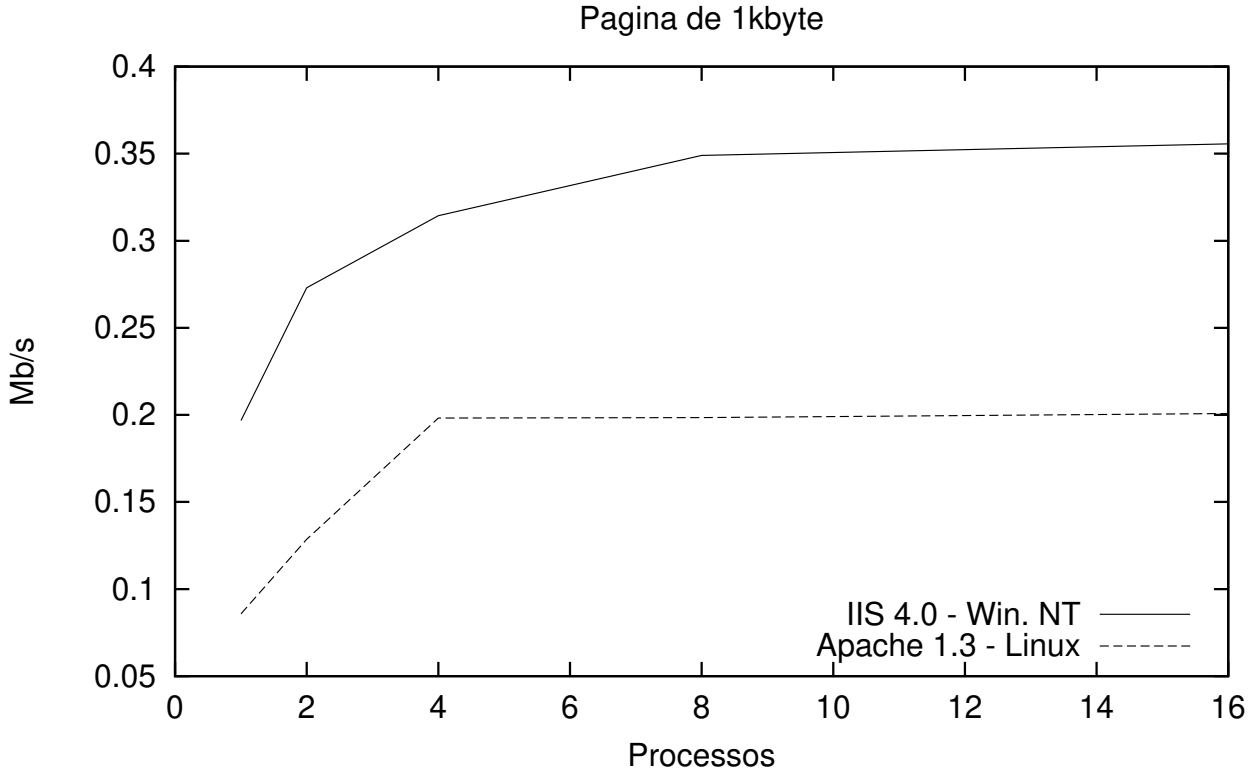

Figura 6.7: Variação do throughput com o número de processos cliente - Página de 1KB

caso, o IIS chegou a um *throughput* em torno de 2 vezes maior que o Apache. Nos testes com páginas de 10 KB (figura 6.8), que é o tamanho médio de 90% das páginas da Web segundo [7] e [9], os dois servidores tiveram praticamente o mesmo desempenho. Neste caso o IIS foi em média 3% mais eficiente que o Apache. Para páginas de 100 KB o Apache foi um pouco melhor que o IIS

apresentando um *throughput* 4% maior.

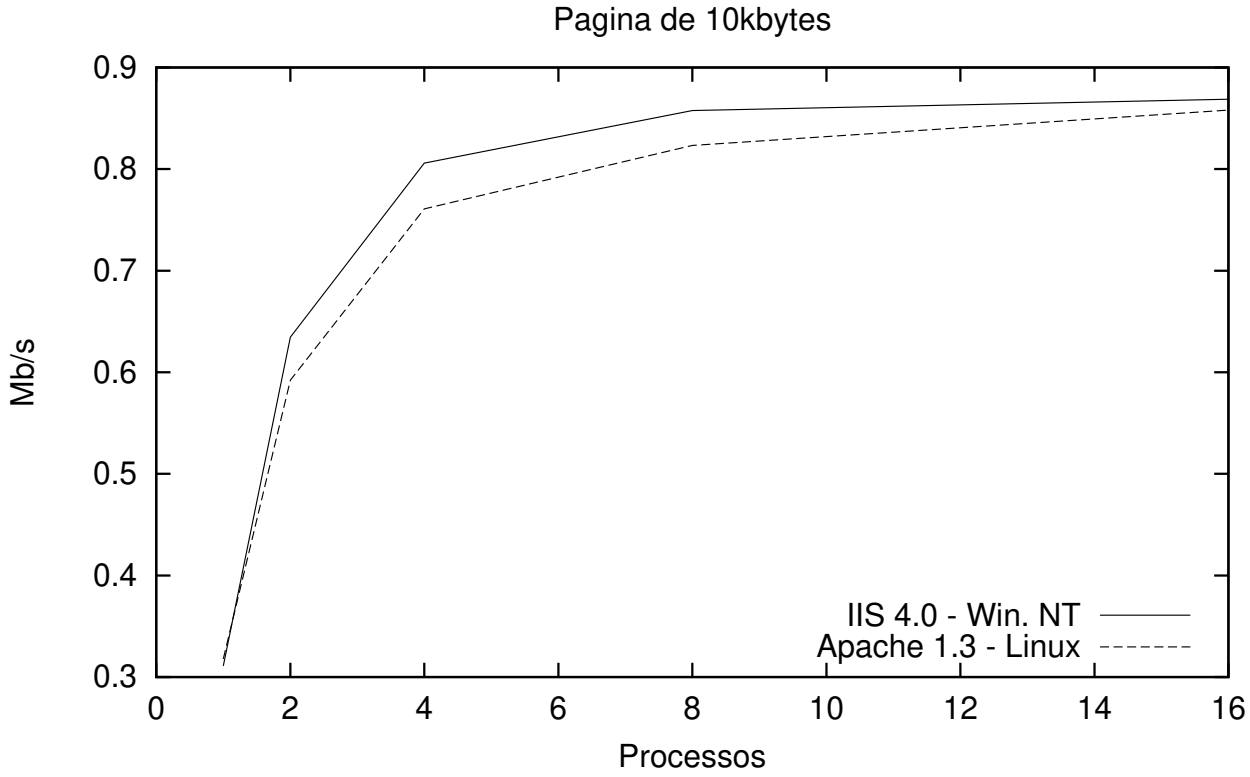

Figura 6.8: Variação do throughput com o número de processos cliente - Página de 10 KB

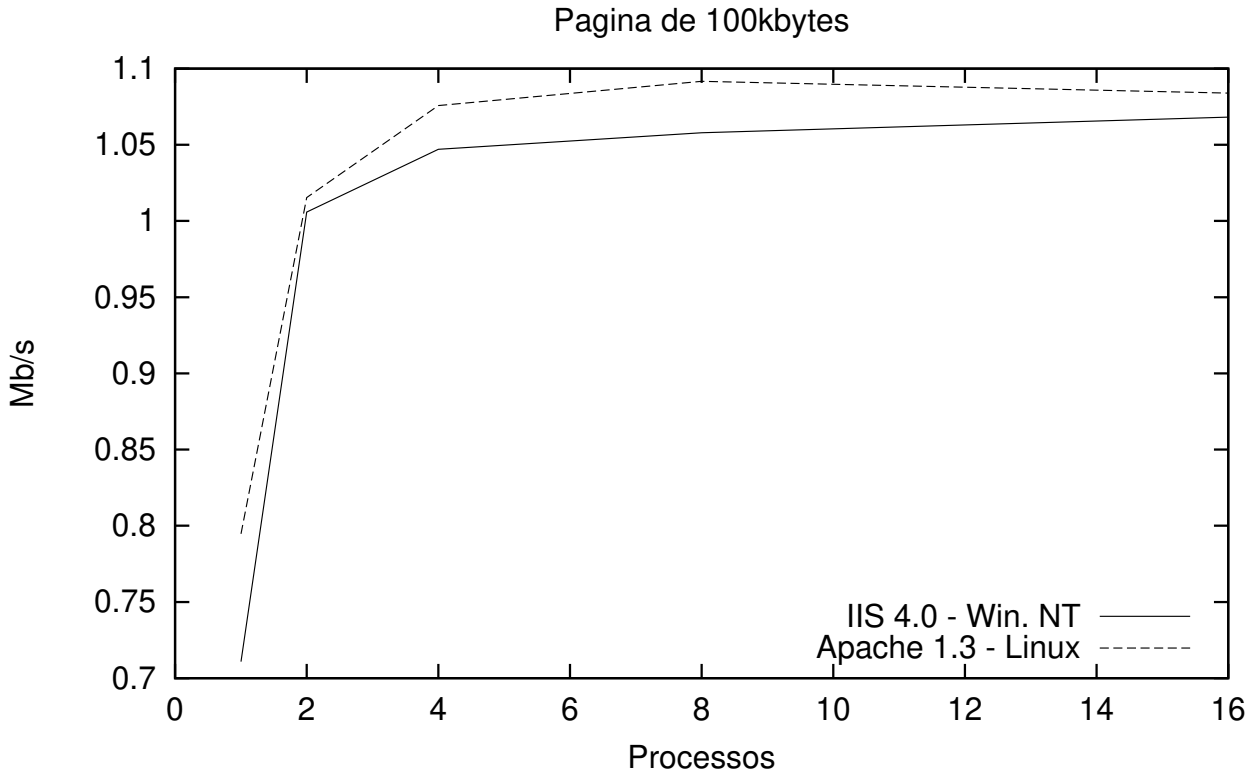

Figura 6.9: Variação do throughput com o número de processos cliente - Página de 100 KB

Capítulo 7

Conclusões

7.1 Dificuldades encontradas

A análise do desempenho de sistemas operacionais torna-se difícil a medida em que cada vez mais tais sistemas oferecem maiores funcionalidades e facilidades aos aplicativos. Os sistemas operacionais aliados às máquinas modernas realizam muitas tarefas que anteriormente deviam ser preocupação dos programadores. Um exemplo disto é o gerenciamento de memória, os processadores fornecem um conjunto de recursos para o controle da memória que podem ser utilizados pelo sistema operacional.

As dificuldades são ainda maiores quando se pretende verificar o desempenho associado à aplicativos. Neste caso deve-se levar em conta que as facilidades dos sistemas operacionais e do hardware podem não ser bem exploradas por estes. Foi possível constatar isto nos experimentos em que geramos o código binário do linpack com diversos compiladores. Verificamos que cada compilador, a partir de um mesmo programa, gerou um código com diferentes níveis de eficiência. Este problema foi resolvido com a utilização de uma versão do linpack com o mesmo código de máquina, ou seja, independente do sistema operacional. Para isto foi necessário algumas modificações no código do linpack que relatamos no capítulo 5.

7.2 Sobrecarga de processos

O programa linpack modificado foi utilizado para os testes de sobrecarga com vários processos. A idéia foi utilizar o linpack como uma carga constante à CPU, verificando o comportamento dos sistemas com vários processos. Não verificamos diferenças significativas no desempenho devidas ao multi-programação nos dois sistemas. O custo das trocas de contexto, por

exemplo, não foi considerável, pois conseguimos um tempo médio por processo equivalente ao tempo de um único processo em ambos sistemas. Neste sentido o *hardware* do Pentium e os recursos que este oferece podem ter contribuído para tornar estas operações mais eficientes.

As medidas revelaram, no entanto, que o escalonamento dos processos em cada sistema tem comportamentos diferentes. No Windows NT verificamos que a carga de um grupo de programas é tratada de forma que uma parte inicia a sua execução imediatamente enquanto o restante vai sendo iniciado aos poucos. Considerando que uma única instância do nosso programa de carga dura quase 3 minutos, o tempo de atraso para o início no caso de 100 processos foi de mais de uma hora. Nos testes com 200 processos este tempo foi de quase quatro horas neste caso cerca de 100 processos já haviam terminado. No Linux este efeito acontece de forma menos acentuada, com o atraso no início da execução sendo observado porém com um deslocamento no tempo muito menor. Apenas no experimento em que tentou-se 300 processos no Linux é que foi observado um atraso maior que chegou a quase uma hora. O comportamento do NT fez com que o tempo de execução de cada processo fosse menor no NT que no Linux, mas não afetou o tempo total ou o tempo médio por processo. Inicialmente poderíamos pensar que o NT poderia ter um ganho de desempenho devido ao tempo em que este fica tratando um número menor de processos, isto não foi observado e acabou por reforçar nossa conclusão de que em nossos experimentos as trocas de contexto não representaram uma penalidade significativa para o desempenho final.

Outra observação importante dos testes foi relativo ao limite máximo de processos. Este limite ficou em 172 processos no NT e no Linux conseguimos chegar a até 243 processos. O Linux teve que utilizar a memória virtual para comportar todos os 243 processos e os valores de tempo aumentaram consideravelmente.

7.3 Sistemas em rede

Uma das primeiras dificuldades que surgem na avaliação de sistemas em rede é a obtenção de um bom ambiente de testes. Obter uma rede isolada evitando a interferência de outras aplicações nem sempre é possível. Um outro problema é a velocidade da rede que deve ser compatível com os parâmetros dos servidores, ou seja, espera-se que a capacidade da rede supere a dos servidores para que esta não imponha um limite aos mesmos.

7.3.1 Programas de teste TCP/IP

Nos testes de rede utilizando a programação de soquetes verificamos um desempenho maior no Linux. Os programas foram escritos com as funções de manipulação de soquetes padrão para o Linux e para o NT. Foi feito também um programa com a utilização das classes existentes no Visual C++ 5.0 que fazem parte do MFC ("Microsoft Foundations Classes").

A versão do programa com a utilização do MFC apresentou uma diferença de desempenho ficando cerca de 10 vezes mais lenta com relação ao programa feito para o Linux. Esta diferença é relacionada à sobrecarga imposta pela utilização do modelo de classes da MFC. Verificamos que no manual do Visual C++ esta sobrecarga é admitida ("Você deve estar ciente da sobrecarga implicada na utilização de algumas das classes." [25]). No caso das chamadas nativas aos soquetes TCP/IP, a versão Linux continuou a ser mais rápida o que pode ser uma indicação de uma melhor implementação do tratamento dos soquetes no Linux.

7.3.2 Servidores Web

Os testes realizados para os servidores Web indicaram um caminho a ser seguido na avaliação destes servidores. Na configuração avaliada o Windows NT com IIS apresentou um desempenho melhor que o Linux com Apache. Este melhor desempenho foi acentuado para páginas de 1 KB enquanto que, para páginas de 10 KB, a diferença não foi tão significativa ficando em apenas 3%. Os resultados não evidenciaram a saturação total dos servidores indicando a necessidade de um ambiente de hardware que permita uma maior exigência dos servidores.

Com o programa cliente escrito especialmente para a nossa análise foi possível conhecer e entender melhor os problemas ligados às medidas de desempenho de servidores Web. As características destes servidores que a cada dia são capazes de atender a um número maior de clientes simultâneos e a crescente exigência de desempenho dos mesmos fazem com que estes sejam cada vez mais eficientes e rápidos. Neste cenário trabalhos que procurem identificar os gargalos, ou seja os pontos que limitam o desempenho final, são importantes.

7.4 Trabalhos futuros

A utilização de um programa de carga idêntico para avaliar diferentes sistemas operacionais no mesmo hardware foi uma abordagem que revelou características interessantes dos sistemas avaliados. Utilizar esta técnica ex-

plorando outros recursos ligados à utilização do processador por diferentes sistemas operacionais é uma idéia que pode ser útil em outros trabalhos do gênero. Este tipo de medição pode ser interessante na verificação de sistemas com mais de um processador, podendo por exemplo, mostrar a eficiência no aproveitamento desta característica do hardware.

Para a análise de sistemas em rede, especialmente servidores Web, verificamos a necessidade de novas pesquisas sobre o assunto. Os resultados existentes atualmente são baseados em benchmarks comerciais ou patrocinados. Entre estes benchmarks podemos destacar o Webbench 3.0 e o WebStone que são gratuitos e o SPECWeb99 que é um benchmark comercial. Estes benchmarks podem ser um bom ponto de partida mas nem sempre representam uma avaliação adequada para cada sistema particular, especialmente se considerarmos que estes benchmarks são sintéticos, ou seja, avaliam os servidores com base em uma implementação específica. Esta implementação pode ser bem diferente da carga de trabalho real de um determinado site Web.

Apêndice A

Linpack

Este apêndice inclui os programas utilizados nos experimentos utilizando o Linpack. Estão aqui também as instruções necessárias para a geração do executável para o NT e para o Linux. Os códigos fontes dos programas utilizados para a versão modificada do Linpack são apresentados a seguir. Para gerar o executável basta seguir as intruções abaixo:

- Compilação do linpack alterado (função linp()) Compilar o programa linp.c no Linux com o comando:

  ```
  gcc -c linp.c
  ```

  .

- Geração do vetor com o código do linpack

  Executar os seguintes comandos:

  ```
  strip linp.o
  decl <linp.o >linp.inc
  ```

- Geração do programa final

  Linux:

  ```
  gcc gcclinp.c
  ```

  Windows (Visual C):

  ```
  cl vclinp.c
  ```

A.1 Função linp(), programa linp.c

A função linp(), consiste no próprio código fonte do Linpack com algumas alterações, para ilustrar estas alterações apresentamos a seguir um trecho da mesma que inclui as sub-rotinas principais do algoritmo.

```
dgefa(a,lda,n,ipvt,info)
float a[];

int lda,n,ipvt[],*info;

/*
     dgefa factors a double precision matrix by gaussian
     elimination.
     ....
*/
{

  /*     internal variables   */
  float t;
  int idamax(),j,k,kp1,l,nm1;
  double MU=-1;               //---> Está variável foi criada
                              //     para substituir uma refe-
                              //     rência à -ONE  no código
                              //     original

  /*     gaussian elimination with partial pivoting   */
  *info = 0;
  nm1 = n - 1;
  if (nm1 >=  0) {
   for (k = 0; k < nm1; k++) {
     kp1 = k + 1;
     /* find l = pivot index   */
     l = idamax(n-k,&a[lda*k+k],1) + k;
     ipvt[k] = l;
     /* 0.0 pivot implies this column already triangularized */
     if (a[lda*k+l] != 0.0) { //---> Aqui no lugar de 0.0
                             //     o algoritmo original
                              //     utiliza a constante
                              //     ZERO
      /* interchange if necessary */
```

```
      if (l != k) {
        t = a[lda*k+l];
        a[lda*k+l] = a[lda*k+k];
        a[lda*k+k] = t;
      }

      /* compute multipliers */
      t = MU/a[lda*k+k];              //---> MU substitui o -ONE
      dscal(n-(k+1),t,&a[lda*k+k+1],1);
      /* row elimination with column indexing */
      for (j = kp1; j < n; j++) {
        t = a[lda*j+l];
        if (l != k) {
         a[lda*j+l] = a[lda*j+k];
         a[lda*j+k] = t;
        }
        daxpy(n-(k+1),t,&a[lda*k+k+1],1,
            &a[lda*j+k+1],1);
      }
     }
     else {
      *info = k;
     }
   }
  }
  ipvt[n-1] = n-1;
  if (a[lda*(n-1)+(n-1)] == 0.0) //---> Aqui no lugar de
     *info = n-1;                //     0.0 o  algoritmo
                                 //     original utiliza
                                 //     a constante ZERO
}
```

sectionPrograma Decl - para copiar o arquivo linp.o para um vetor (decl.c)

```
#include <stdio.h>
main()

{
  int c;
  int i;
```

```
  for (c = 0; c < 64; c++) getchar();
  printf("char linpcode[] = {\n  ");
  while ((c=getchar()) != EOF) {
printf("'\\x%02x', ", c);
if (i++ == 8) {
  printf("\n  ");
  i=0;
}
  }
  printf("};\n");
}
```

A.2 Programa final para o Linux (gcclinp.c)

```
#include <stdio.h>
#include <signal.h>
#include "linp.inc"

double dtime();

void hsignal(int i) {
   printf("Recebido sinal %d\n",i);
}


void main()
{
    double t1,t2; int i;
    void (*rec)();
    rec = (void (*)()) linpcode;
    for(i=1;i<31;i++) {
       if(i!=9 && i!=19) signal(i,&hsignal);
    }
    t1 =  dtime();
    rec();
    t2 = dtime();
    printf("Inicio %f Tempo %f\n", t1,t2-t1);
```

```
}
```

```
#include <sys/time.h>
struct timeval tnow;
double dtime()
{
 double q;
 gettimeofday(&tnow,NULL);
 q = (double)tnow.tv_sec + (double)tnow.tv_usec * 1.0e-6;
 return q;
}
```

A.3 Programa final para o Windows - Visual C (vclinp.c)

```
#include <stdio.h>
#include <windows.h>
#include <winbase.h>
#include "linp.inc"
void _chkstk();

void main(void)
{
   void (*rec)();
DWORD tick1,tick2;
__asm {
mov  eax,0x4eef0
        call _chkstk
add  esp,0x4eef0

   };
   rec = (void (*)()) linpcode;
tick1=GetTickCount();
   rec();
tick2=GetTickCount();
   printf("Inicio %d Tempo %d\n",tick1,tick2-tick1);
```

```
}
```

Referências Bibliográficas

[1] J.B. Chen, Y. Endo, K. Chan, D. Mazières, A. Dias, M. Seltzer, e M.D. Smith.

The Measured Performance of Personal Computer Operating Systems.

In *ACM Trans. on Computer Systems*, Vol. 14, No. 1,
February 1996 pp 3-40.

[2] H. Custer.

Inside Windows NT

Microsoft Press, Redmond, Washington, 1993

[3] A.S. Tanenbaum.

Modern Operating Systems

Prentice Hall,Inc, 1992

[4] R. Jain.

The Art of Computer Systems Performance Analysis.

John Wiley & Sons, Inc, February 1996.

[5] W. Stallings.

Operating Systems: internals and design principles

Prentice-Hall, Inc, 1998, 3a. ed.

[6] D.A. Menascé, V.A.F. Almeida.

Capacity Planning for Web Performance: metrics, models & methods

Prentice-Hall, Inc, 1998

[7] M. Arlitt, C. Williamson.

Web Server Workload Characterization

Proc. of 1996 SIGMETRICS Conference on Measurement and Modeling of Computer Systems, May 1996.

[8] W.R. Stevens.

Unix Network Programming - Volume 1

Prentice-Hall, Inc, 1997

[9] M. Seltzer, S. Manley.

Web Facts and Fantasy

http://eecs.harvard.edu/ vino/web/sits.97.html em 29/2/1999

[10] M. Russinovich.

Inside the Windows NT Scheduler

http://www.winntmag.com/Articles/Index.cfm?ArticleID=302 julho 1997

[11] D.A. Solomon.

The Windows NT Kernel Architecture

In *IEEE Computer* Vol 31, No. 10, October 1998 pp 55-62.

[12] M. Beck, H. Böhme, M. Dzia dzka, U. Kunitz, R. Magnus, D. Verworkner.

Linux Kernel Internals, 2nd edition

Addison Wesley, 1998

[13] K. Dowd, C. Severance.

High Performance Computing

O'Reilly, 1998

[14] A. Silberschatz, P. B. Galvin.

Operating System Concepts

Addison Wesley, 1994

[15] Microsoft Windows NT Server 4.0 versus UNIX

http://www.unix-vs-nt.org/kirch

[16] S.J. Vaughan, E. Carr.

Linux is the Web Server's Choice

http://www.zdnet.com/sr/stories/issue/0,4537,2196115,00.html

[17] J.S. Dongarra.

Performance of Various Computers Using Standard Linear Equations Software

http://www.netlib.org/benchmark/performance.ps

junho 1999.

[18] T. Dyck.

8 Web App Servers That Deliver

http://www.zdnet.com/pcweek/stories/news/0,4153,409380,00.html

julho 1999.

[19] Tests show more than numbers

http://www.zdnet.com/pcweek/stories/jumps/0,4270,401961,00.html

agosto de 1999.

[20] Benchmark Tests: Web Servers

http://www.zdnet.com/pcmag/stories/reviews/0,6755,402311,00.html

agosto de 1999.

[21] Web and File Server Comparison:

Microsoft Windows NT Server 4.0 and Red Hat Linux 5.2 Upgraded to the Linux 2.2.2 Kernel

http://www.mindcraft.com/whitepapers/nts4rhlinux.html

13 de Abril de 1999.

[22] Open Benchmark:

Windows NT Server 4.0 and Linux

http://www.mindcraft.com/whitepapers/openbench1.html

30 de Junho de 1999.

[23] Benchmarks da indústria mostram que o Windows NT Server 4.0 supera o

Linux em desempenho

www2.uol.com.br/info/infonews/101999/07101999-6.shl

agosto de 1999.

[24] Haynes & Company - Shiloh Consulting.

Performance Benchmarck Tests of Novell, Microsoft, and Netscape Web Servers

Responding to HTML, CGI, and Proprietary API Requests

http://www.tedhaynes.com/haynes1/benchmk/novell/shiloh.htm

fevereiro de 1999. (Não disponível atualmente)

[25] Visual C++ Programmer's Guide