Máquinas virtuais em ambientes seguros

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Arthur Bispo de Castro e aprovada pela Banca Examinadora.

Campinas, 10 de fevereiro de 2006.

Prof. Dr. Paulo Lício de Geus (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Júlia Milani Rodrigues – CRB8a / 2116

Castro, Arthur Bispo de
C279m Máquinas virtuais em ambientes seguros / Arthur Bispo de Castro -- Campinas, [S.P. :s.n.], 2006.

Orientador : Paulo Lício de Geus
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Redes de computação. 2. Sistemas operacionais (Computadores). 3. Sistemas de segurança. I. De Geus, Paulo Lício. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Virtual machines in secure environments.

Palavras-chave em inglês (Keywords): 1. Computer networks. 2. Operating systems (Computers). 3. Security systems.

Área de concentração: Segurança de Redes

Titulação: Mestre em Ciência da Computação

Banca examinadora: Prof. Dr. Fabricio Sérgio de Paula (UEMS)
Prof. Dr. Rodolfo Jardim de Azevedo (IC-UNICAMP)
Prof. Dr. Ricardo Dahab (IC-UNICAMP)
Prof. Dr. Paulo Lício de Geus (IC-UNICAMP)

Data da defesa: 10/02/2006

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 10 de fevereiro de 2006, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Fabrício Sérgio de Paula
UEMS.

Prof. Dr. Rodolfo Jardim de Azevedo
IC / UNICAMP.

Prof. Dr. Paulo Lício de Geus
IC / UNICAMP.

Instituto de Computação
Universidade Estadual de Campinas

Máquinas virtuais em ambientes seguros

Arthur Bispo de Castro

Fevereiro de 2006

Banca Examinadora:

- Prof. Dr. Paulo Lício de Geus (Orientador)
- Prof. Dr. Fabrício Sérgio de Paula
  Universidade Estadual de Mato Grosso do Sul – UEMS
- Prof. Dr. Rodolfo Jardim de Azevedo
  Instituto de Computação – UNICAMP
- Prof. Dr. Ricardo Dahab
  Instituto de Computação – UNICAMP

Resumo

Desde o início da computação a idéia de máquinas virtuais vem sendo aplicada para estender o multiprocessamento, multi-programação e multi-acesso, tornando os sistemas multi-ambiente. O contínuo aumento no poder de processamento dos computadores fez com que máquinas muito rápidas estivessem ao alcance de qualquer usuário, surgindo PCs com processamento, espaço em disco e memória suficiente para comportar mais de um sistema compartilhando o mesmo hardware.

Basicamente, o objetivo das máquinas virtuais é produzir um sistema mais simples, que permita que no mesmo hardware sejam executados vários sistemas operacionais. Sua implementação se resume a um programa gerenciador chamado VMM (*Virtual Machine Monitor*), que cria um ambiente que simula o acesso direto ao hardware. Este gerenciador é classificado em vários tipos, apresentados neste trabalho.

Graças às suas funcionalidades e variedades, o uso das máquinas virtuais pode ser estendido a fim de promover maior segurança e desempenho em redes, apresentando diversas vantagens e garantindo um ambiente mais confiável. São aplicáveis às estações de trabalho, à detecção e aprendizado de novos ataques a sistemas operacionais e aplicações, além de focar o aumento de segurança de servidores. Devido à disponibilidade de um laboratório de segurança de redes, pode-se colocar uma implementação de máquinas virtuais em sistemas reais de produção: servidor web, ftp, dns, e-mail, etc., e seus resultados confirmam o êxito desta utilização.

Abstract

Since the beginning of computing, the concept of virtual machines has been associated with extending multi-processing, multi-programming and multi-access to create multi-environment systems. The continual increase in computer processing power has resulted in the common user having access to personal computers whose processing speed, disk space and memory are sufficient to support more than one operating system, whilst sharing the same hardware.

In simple terms, the goal of a virtual machine is to produce a simple system that permits that various operating systems can be executed on the same hardware. It is implemented via a managing program called VMM (Virtual Machine Monitor) that creates an environment that simulates direct access to the hardware. This managing program can be implemented in diverse ways, as will be discussed in this work.

Due to its functionality and flexibility, virtual machines can be employed to provide greater security and performance in networked environments, offering various advantages and guaranteeing a trustworthy environment. In this work, virtual machines have been employed in workstations to detect and learn new attacks against operating systems and applications, as well as to increase the security of the servers. Due to availability of a network security laboratory the ideas proposed have been implemented in real production systems: web server, ftp, dns, e-mail, etc., and the results confirm the effectiveness of the utilizations.

Agradecimentos

Amigos da república: Cláudio, Daniel, Eric e Felipe, que passaram esse último ano comigo e que me aguentaram e compartilharam excelentes momentos. Esses figuras são incríveis... eh noiz!

Amigos de Bauru, em especial à Aline e Fábio Santos que me deram força e acreditaram em mim e ao Fábio Micheloto um grande amigo, o mais antigo deles, uma pessoa que me apoiou mutíssimo em momentos difíceis (e nos fáceis também!).

Aretha e Diogo minha irmã que me ajudou um monte nesse final de mestrado e ao meu amigo, com quem tive grandes momentos e que me ajudou um monte na pesquisa sobre segurança de sistemas operacionais.

Celmar e Cássia com quem tive um enorme contato no começo do mestrado e por quem tenho um grande carinho.

Colegas do instituto, em especial ao Luciano, com quem tive maior contato, me auxiliando em muitos momentos.

Colegas do LAS: Andre, Bruno, Edmar, Eduardo, Helen, João Paulo, Martim, Ulisses, . . . , em especial à Cleymone que pegou no meu pé, que me ensinou muito e que desenvolveu o MV6 comigo.

Deus, por tudo.

Família, a minha mãe Dona Beth, a minha outra irmã Rafhaella e ao meu pai Nesiano, que, por causa deles, eu sou o que sou hoje. Também agradeço a minha avó (a 'Baiana') e aos meus tios, primos, etc., em especial a tia Neliane e Orlando.

Funcionários do instituto, em especial à Camila, ao Carlão, ao Flávio e ao Sinval, pelo suporte, ajuda e pelas conversas, também aos vários professores, eles que deram a base para eu chegar neste trabalho.

Joice que me deu muita força no início da escrita deste trabalho.

Michelle essa pessoa incrível, que apareceu no momento mais improvável e difícil da minha vida, e que esteve firme ao meu lado, me ajudando mais do que eu poderia sonhar um dia.

Prof. Paulo Lício que me acolheu no começo de 2000, dando suporte e ajuda, me guiando por todo esse caminho até aqui.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 Objetivos

Este trabalho foi desenvolvido com o intuito maior de apresentar o conceito e o uso de máquinas virtuais, e seus objetivos foram organizados da forma apresentada a seguir.

Na primeira parte, faz-se necessário detalhar o conceito de máquina virtual, bem como seu funcionamento, suas características principais e seus tipos. É uma seção introdutória, que dará base para os futuros capítulos. Ela também traz um pequeno histórico das origens das máquinas virtuais.

A segunda parte se aprofunda na descrição dos tipos de máquinas virtuais, apresentando exemplos de cada tipo, suas vantagens e desvantagens e principais características.

A partir destas duas partes explicativas, pode-se descrever como as máquinas virtuais podem ser utilizadas quanto à solução de problemas de segurança em redes, mostrando como essas aplicações são feitas. Procura-se enfatizar, nesta fase, os detalhes que tornam as máquinas virtuais um ambiente confiável e seguro, assim como compreender o funcionamento e a interação entre os sistemas virtuais e reais. Expende-se também o programa que implementa a máquina virtual, visando analisar o impacto e os benefícios que o uso dessa estrutura pode causar.

Por fim, são descritos dois estudos de caso com implementação de máquinas virtuais em utilização real, observando as implicações sofridas principalmente quanto ao desempenho e à segurança.

1.2 Motivação

Durante a evolução no desenvolvimento da tecnologia na produção de máquinas mais rápidas, as possibilidades de uso e aplicações destas também cresceram muito. Atividade-

des que antigamente eram impossíveis de se fazer ou que estavam restritas aos supercomputadores, hoje podem ser feitas em praticamente qualquer ambiente. O desenvolvimento de softwares e tecnologias associadas também seguiram essa tendência, aumentando em tamanho e complexidade. O enorme crescimento do poder de processamento dos computadores pessoais proporcionou uma expansão em seu uso, e assim, tanto pequenas empresas quanto usuários domésticos têm a possibilidade se valer de suas vantagens. Em geral, qualquer empresa tem alguma maneira de fazer acesso a Internet, usando ou oferecendo seus serviços e querendo que a sua integridade e privacidade sejam respeitadas.

O cenário atual é de diferentes arquiteturas de hardware e cada uma dessas arquiteturas possuindo vários sistemas operacionais e seus respectivos softwares de aplicação.

O que muitos usuários e desenvolvedores desejam é uma maneira segura de, a partir do seu sistema operacional principal, executar aplicações deste sistema, de outros e até de outras arquiteturas de hardware. Isso pode ser obtido através de:

Emulação de hardware: todas as instruções geradas são traduzidas para chamadas e/ou instruções da arquitetura/sistema corrente. Cria-se assim um programa que simula algum hardware específico. Perde-se em desempenho, porém ganha-se em portabilidade.

Emulação da API: as aplicações usam uma interface (API - *Application Programming Interface*) de chamadas de sistema para se comunicar com o sistema operacional e realizar suas operações. Nessa abordagem, então, em vez de um tradutor de instruções, tem-se um tradutor de chamadas de sistema. Desta forma, consegue-se a execução de programas de sistemas operacionais distintos no sistema operacional corrente, conseguindo um desempenho muito próximo da execução canônica do programa, e às vezes até mais rápido, devido à possível maior eficiência das chamadas de sistema do hospedeiro. Como somente as chamadas de sistema são traduzidas, essa técnica não permite execução de programas de outras arquiteturas de hardware.

Virtualização: o núcleo da idéia está em permitir que mais de um sistema operacional seja executado sobre o mesmo hardware. Isso não é possível nativamente, pela exclusividade que alguns periféricos e até mesmo alguns processadores exigem. O desempenho é muito semelhante ao de um sistema sendo executado diretamente sobre o hardware.

Existem, como será tratado adiante, algumas definições do que seria uma máquina virtual (VM - *Virtual Machine*), porém, todas elas se resumem basicamente ou num emulador de hardware, ou numa virtualização pura ou num misto de ambos os conceitos. Sendo assim, pode-se desde já dizer que, de modo geral, uma máquina virtual é uma abstração, de algum nível, em software de um hardware.

1.3 Organização

Esta dissertação é composta dos seguintes capítulos:

Capítulo 2: é apresentado o nascimento e a evolução das máquinas virtuais, mostrando a base de seu funcionamento e suas principais características;

Capítulo 3: dada as definições das máquinas virtuais apresentadas no capítulo anterior, é feito um *survey*, analisando as máquinas virtuais mais relevantes;

Capítulo 4: são apresentados os pontos em que o uso de máquinas virtuais contribui para a melhoria do nível de segurança, propondo alguns cenários de aplicação;

Capítulo 5: são descritos dois casos de aplicação do uso de máquinas virtuais, um para aumentar a segurança, usando consolidação de servidores, e outro para a migração de protocolo de rede de maneira segura, minimizando os efeitos da migração;

Capítulo 6: é desenvolvida a conclusão do trabalho, as suas contribuições e as indicações para trabalhos futuros.

Capítulo 2

Máquinas virtuais

Neste capítulo será apresentada um pouco da história das máquinas virtuais. O objetivo é mostrar o que estimulou sua criação, os problemas enfretados para tanto e o que levou à expansão do seu uso. Com isso, mostra-se a sua ligação intíma com os sistemas operacionais e com o hardware da máquina. Com relação a este último, discute-se o suporte que deve existir para a implementação da virtualização. Também são mostradas técnicas de máquinas virtuais que fogem do conceito de virtualização pura, sendo, por exemplo, o caso dos emuladores. Por fim, são discutidos as características e os tipos de monitores de máquinas virtuais (VMMs), programas responsáveis pela criação do ambiente de execução dos sistemas nas máquinas virtuais.

2.1 As origens

A idéia de máquinas virtuais não é uma novidade. Desde o início da computação, essa idéia vem sendo utilizada para estender o multiprocessamento, multi-programação e multi-acesso, tornando os sistemas também multi-ambiente [Gol73].

O resumo a seguir é baseado em [Var97], que relata um histórico bastante completo da evolução do conceito de máquinas virtuais.

Por volta dos anos 50, as pesquisas começaram a vislumbrar a possibilidade de tornar os sistemas operacionais em sistemas de tempo compartilhado (*time-sharing*). Para isso, foram feitas pesquisas em várias instituições e no MIT (*Massachusetts Institute of Technology*) foi desenvolvido um dos primeiros e mais importantes sistemas, chamado CTSS (*Compatible Time-Sharing System*), que influenciou muito esse campo de pesquisa. Uma das primeiras versões do CTSS foi desenvolvida para um IBM 709 no ano de 1961.

A partir do CTSS observou-se que existiam problemas que seriam melhor solucionados por hardware do que por software. Assim, o MIT solicitou à IBM modificações pertinentes no processador. Em 1963, com um IBM 7090 modificado, uma versão do CTSS serviu

como modelo de como se fazer um sistema de tempo compartilhado.

O MIT fundou o projeto MAC (o projeto tinha diversos nomes, entre eles: *Machine-Aided Cognition* ou *Multiple-Access Computing* ou *Man And Computer*), pensando em desenvolver sistemas de tempo compartilhado baseados no CTSS, dentre eles o sistema Multics. No início de 1964 a IBM criou o CSC (*Cambridge Scientifc Center*), no mesmo prédio do projeto MAC. O Objetivo da IBM com o CSC era se tornar mais respeitável para o meio acadêmico.

A partir disso, a questão não era mais saber se era possível o desenvolvimento de um sistema desse tipo, mas se esse sistema seria realmente útil. O posicionamento da IBM foi que era necessário um processamento mais rápido e não um sistema de tempo compartilhado. Sendo assim, em 1964 a IBM lançou o System/360 sem as modificações necessárias.

Os responsáveis pelo CSC consideraram inadequada a atitude da IBM e diante dela decidiram fazer as modificações. Um dos primeiros trabalhos da equipe do CSC foi especificar uma máquina que obedecia os pedidos do projeto MAC. Isso forçou a IBM a propor uma modificação do S/360, mas que não agradou aos coordenadores do MAC.

O MIT e a Bell Labs desistiram dos serviços da IBM e, usando outro fornecedor de hardware, desenvolveram o Multics (1969 - largamente usado no MIT) e o sistema UNIX (também em 1969 - uma simplificação feita pela Bell Labs do CTSS e do Multics).

Ainda em 1964, a coordenação do CSC decidiu fazer seu próprio sistema de tempo compartilhado para o System/360. A partir de outros estudos o CSC especificou um novo processador (que viria a ser chamado de S/360 modelo 67) e um novo sistema operacional chamado TSS (*Time-Share System*).

A IBM mandou o desenvolvimento do TSS para uma outra equipe, deixando a coordenação do MAC em dúvida quanto ao futuro do sistema e, devido a isso, iniciou-se a montagem de outro projeto para desenvolvimento de um sistema operacional para o S/360, chamado projeto CP-40. Uma das principais linhas de pesquisa do CP-40 era na área de memória virtual. Naquela época existia pouco conhecimento sobre o seu funcionamento, apesar de a IBM querer implementá-la no TSS.

No final do ano de 1964, os responsáveis pelo projeto CP-40 desenvolveram a idéia de prover um ambiente de memória virtual e também com máquinas virtuais (no início elas eram chamadas de *pseudo-máquinas*). Cada usuário teria uma máquina virtual System/360 completa. Essa idéia era completamente nova e também elegante, apesar desta última característica permanecer momentaneamente despercebida. Além disso, precisou-se de pouca modificação no hardware e pouco software, usando para desenvolvimento do projeto uma versão modificada do System/360 modelo 40.

Além do sistema CP (*Control Program*), que gerencia e isola as máquinas virtuais, os responsáveis pelo desenvolvimento do projeto perceberam que iriam precisar de um

segundo sistema, o CMS (*Console Monitor System*, que também era conhecido como *Cambridge Monitor System*), um sistema operacional mono-usuário que provê processamento em tempo compartilhado.

Dessa forma, há uma divisão entre o gerenciamento do sistema e o suporte ao usuário. Outro ponto de destaque é a performance e a simplicidade do sistema, visto ser impossível mensurar, a princípio, qual a sobrecarga que essas modificações poderiam causar. A implementação do CP e do CMS começou em meados de 1965, entrando em produção no início de 1967.

Concomitantemente com a implementação do CP, em 1965, a IBM anunciou o System/360 modelo 67 e o sistema TSS. Porém o TSS tinha sérios problemas de desempenho e estabilidade. Em 1966 o CSC começou a converter o CP e o CMS para rodar no modelo 67. Como eles não tinham um System/360 modelo 67, tiveram que modificar o modelo 40 para simular um modelo 67. E em 1967 eles tinham o sistema CP-67 pronto para testar num modelo 67 real. E o sistema foi muito bem sucedido, tornando-se um concorrente do TSS e influenciando o desenvolvimento dos sistemas baseados em tempo compartilhado.

Em fevereiro de 1968 dezoito máquinas 360/67 estavam instaladas, a maioria delas estava rodando TSS (ou tentando rodar). Mesmo assim a IBM resolveu anunciar o encerramento do TSS. No meio desse mesmo ano a primeira versão do CP-67 se tornou um programa Tipo III da IBM[1]. Devido a protestos, em abril de 1969, a IBM voltou atrás e cancelou o encerramento do TSS, porém muitos sistemas já haviam mudado de TSS para CP/CMS. A versão 3 do CP-67 foi a primeira versão com suporte Classe A, no final de 1970.

No final da era do CP-67 muito se havia aprendido sobre máquinas virtuais: a versão 3.1 do CP-67 executava sessenta CMSs num System/360 modelo 67.

Em julho de 1970 a IBM lança o System/370, ainda sem as modificações necessárias, desanimando os clientes do TSS e do CP/CMS. Em maio de 1971 o TSS é definitivamente encerrado e dois novos sistemas seriam desenvolvidos: OS/VS1 (SVS - *Single Virtual Storage*) e OS/VS2 (MVS - *Multiple Virtual Storage*). Além disso, a IBM alegou que não era necessário o uso de máquinas virtuais por seus clientes.

Apesar disso, o desenvolvimento de uma versão do CP e do CMS para o S/370 foi feito. No desenvolvimento de uma versão do CP-67 para o System/370, foi criada uma versão do CP-67 que provia máquinas virtuais System/370 em um System/360 modelo 67; este se torna extremamente importante para os desenvolvedores do MVS, já que eles não tinham acesso a máquinas para teste. O desenvolvimento do CP-370 ocorreu numa máquina virtual de terceiro nível[2], às vezes de quarto ou quinto níveis.

[1] esses programas eram fornecidos pela IBM sem garantia de funcionamento, ficando a cargo dos usuários sua manutenção.

[2] executar uma máquina virtual dentro da outra.

No final de 1971 um S/370 real foi enviado para o CSC. A primeira versão completa do VM/370 CP se tornou funcional em fevereiro de 1972. Em agosto desse mesmo ano, a IBM anuncia dois novos computadores (370/158 e 370/168), as modificações necessárias para virtualização em todos os 370s e quatro novos sistemas operacionais: VM/370, DOS/VS, OS/VS1 e OS/VS2. Uma enfâse especial foi dada à produtividade ganha pela IBM por testar e fazer manutenção de sistemas operacionais em máquinas virtuais CP-67 e VM/370. A Figura 2.1 mostra a estrutura de uma máquina System/370 rodando o VM/370.

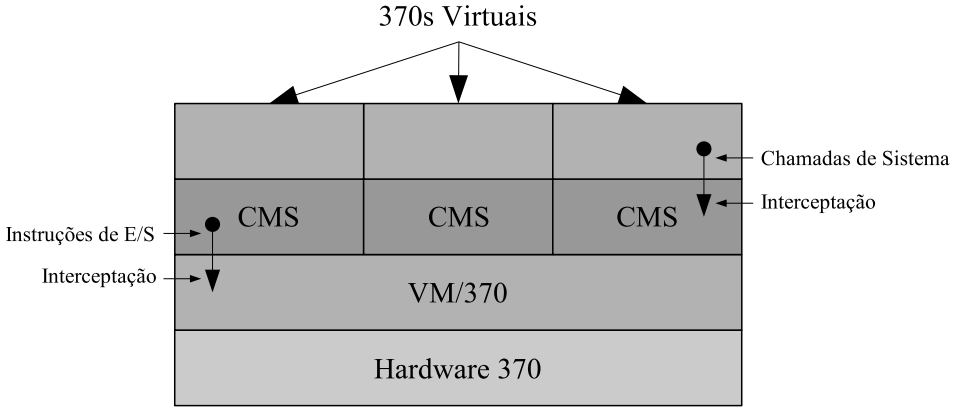

Figura 2.1: Estrutura de um VM/370 com CMS.

Num dos *slides* do anúncio do VM/370:

> Eis um prisma. Considere por um momento o que acontece quando um feixe de luz o atravessa... muitas cores derivam de uma fonte de luz. Permita-me enfatizar esse ponto... muitos de um.
>
> Analogamente, imagine que o feixe de luz é um IBM System/370; o prisma, a *Virtual Machine Facility*/370. As muitas cores produzidas pelo prisma a partir de um feixe de luz são agora muitas 370s virtuais produzidas pelo VM/370 a partir de um 370 real. E cada 370 virtual tem a capacidade de executar seu próprio sistema de programação, tais como OS, DOS ou CMS. Muitos a partir de um... muitas 370s virtuais a partir de um 370 real. E VM/370 faz isso acontecer!

É importante ressaltar que o projeto CP era experimental, a equipe do CSC estava apenas desenvolvendo um sistema para seu próprio uso. Eles estavam experimentando a idéia de um sistema de tempo compartilhado e buscando uma maneira de resolver as limitações dos sistemas anteriores que não tinham suporte para o compartilhamento. Os sistemas de tempo compartilhado também possuem deficiências, mas são bastante flexíveis para atividades comuns. Nessa busca por melhores soluções, a equipe criou

as máquinas virtuais, que se mostraram uma solução muito elegante, estendendo o uso dos computadores; também permitindo que os sistemas existentes pudessem ajudar no desenvolvimento de outros sistemas operacionais, visto que os sistemas da época eram inadequados para tais tarefas. Além disso, as máquinas virtuais puderam ser utilizadas na solução de novos problemas, tais como disponibilidade e segurança de sistemas.

Depois desse período, por volta de 1980 até o final da década de 1990, pouco de novo aconteceu. As máquinas virtuais ficaram restritas aos super computadores e aos *mainframes* (basicamente aos sistemas da IBM). A pesquisa e o desenvolvimento girava em torno do desempenho, para minimizar a sobrecarga que seu uso introduz. Na tecnologia, ou seja, na forma de como as máquinas virtuais eram executadas, pouco mudou.

O contínuo aumento no poder de processamento dos computadores causou a mudança neste cenário. Máquinas muito rápidas estão ao alcance de qualquer empresa e até no mundo doméstico. Houve uma espécie de migração de *computação* dos super computadores e *mainframes* para os PCs (e *desktops*). Os PCs começaram a ter processamento, espaço em disco e memória suficientes para suportar mais de um sistema. E da mesma forma que num *mainframe*, um PC poderia comportar vários sistemas, compartilhando o mesmo hardware.

2.2 Conceitos

Com o passar do tempo, foram nascendo novas idéias e funções, o que fez com que a definição de o que é uma máquina virtual fosse se adaptando e/ou se associando a outros tipos de definições. Algo muito semelhante aconteceu com os sistemas operacionais (de certa forma ligados às máquinas virtuais), que começaram como sistemas muito simples, monoprogramados, dando acesso básico ao hardware e que podia conter alguma biblioteca para auxiliar a programação. Com o tempo, eles foram agregando funções e/ou funcionalidades, deixando pouco clara a fronteira de o que é, e o que não é, sistema operacional [SGG01]. Um sistema operacional não é mais necessariamente somente o *kernel*, podendo agregar um conjunto mais amplo de ferramentas e utilitários. Um exemplo bem claro disso está no processo em que a empresa Microsoft sofreu por causa da inclusão do navegador Internet Explorer ao sistema[3].

O conceito inicial de máquinas virtuais está relacionado a sua origem, na qual o objetivo era criar uma espécie de sistema operacional de sistemas operacionais, ou seja, simplificar o processo de fazer a multi-programação, produzindo um sistema mais simples, que permitisse que no mesmo hardware fossem executados vários sistemas operacionais. O sistema responsável pela criação do ambiente era o VM (sucessor do CP-67) e o sistema

[3] `http://en.wikipedia.org/wiki/United_States_v._Microsoft`

operacional CMS, que é mono-usuário, formando a dupla VM/CMS [Ros04a].

O problema encontra-se em como enganar um sistema operacional, que julga estar sozinho e que foi feito com essa premissa, gerenciando os recursos da máquina para *simular* essa característica; além de tudo, impedindo que um sistema operacional afete o funcionamento dos outros sistemas que estarão executando em paralelo. Ou seja, a criação de vários ambientes isolados de execução de sistemas operacionais, que podem ou não ser iguais ao hardware base.

Ademais, as máquinas virtuais precisam reproduzir de alguma forma uma arquitetura de hardware, mantendo o ambiente verossímil (entrada e saída, temporizadores, dispositivos, etc.) e ter um desempenho satisfatório [Gol73].

2.2.1 Sistemas operacionais

Como visto, o principal uso das máquinas virtuais é a execução de sistemas operacionais. Deste modo, as máquinas virtuais desenvolvidas para executar sistemas operacionais serão o objeto deste estudo. Uma das maneiras de aprofundar o entendimento sobre como as máquinas virtuais funcionam está em entender como funciona um sistema operacional.

Um sistema operacional é um gerenciador de recursos do hardware, provendo um ambiente de execução para programas. Internamente os sistemas operacionais diferem muito, fruto das escolhas de cada desenvolvedor e dos objetivos do sistema [SGG01]. Porém, praticamente todos eles, dada sua complexidade e tamanho, são formados por pedaços bem definidos, com funções e características semelhantes, se comparados uns com os outros [TW97]. Pode-se citar, como exemplo, o gerenciador de processos ou o gerenciador de memória.

A interface entre o sistema operacional e os programas de usuário é definida como um conjunto de *instruções estendidas*, que o sistema operacional prove. Essas instruções são tradicionalmente conhecidas como *chamadas de sistema* (*system calls*), e são elas que indicam o que um sistema operacional é capaz de fazer. As chamadas de sistema disponíveis variam de sistema para sistema, apesar de conceitualmente elas serem similares [TW97]. As chamadas de sistema podem ser agrupadas em cinco categorias: controle de processos, gerenciamento de arquivos, gerenciamento de dispositivos, manutenção da informação e comunicações [SGG01]. As chamadas de sistemas fazem parte da API de alguns sistemas operacionais.

Como dito, o sistema inteiro pode ser dividido em camadas [TW97], como pode ser visto na Figura 2.2, com o hardware na camada mais baixa. Interagindo com essa camada existe uma parte do sistema operacional que usa instruções de hardware para se comunicar com a camada inferior e provê um acesso de mais alto nível para a próxima camada, do próprio sistema operacional. Essa lógica serve para todas as eventuais cama-

das intermediárias do sistema operacional finalizando na API que, por fim, os programas usam para ter acesso aos recursos.

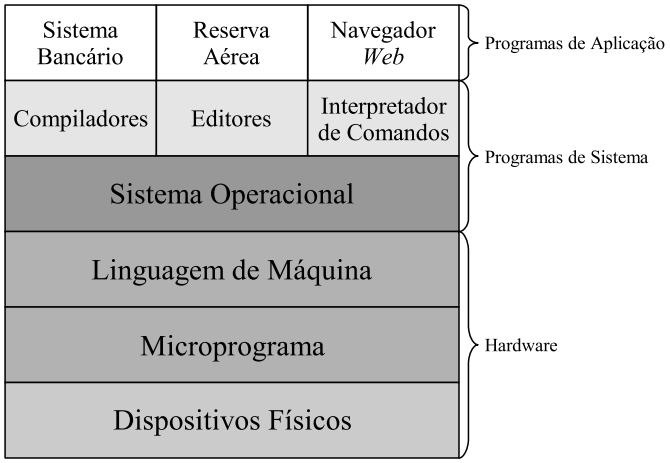

Figura 2.2: Um sistema operacional é formado por camadas: hardware, programas de sistema e programas de aplicação.

2.2.2 Vantagens e funções

De acordo com [SGG01] existem duas vantagens principais no uso de máquinas virtuais. Primeiramente, elas podem ser usadas para proteger os recursos do sistema, fornecendo um nível robusto de segurança. Em segundo lugar, permitem que o desenvolvimento de sistemas possa ser feito sem atrapalhar as operações normais do sistema. Cada máquina virtual é completamente isolada das demais e, sendo assim, não existem problemas de segurança, com os recursos do sistema completamente protegidos [SGG01].

O primeiro tipo de máquina virtual, na Seção 2.1, se baseia na virtualização pura. O segundo provável tipo de máquina virtual é relacionado a um dos seus primeiros usos, o desenvolvimento de sistemas. A diferença está em que o hardware a ser provido ao ambiente virtual passa a não ser necessariamente o mesmo do hardware base, ou seja, tem-se a criação de sistemas para uma outra arquitetura de hardware. Esse tipo de máquina virtual é mais comumente chamada de *emulador*.

Um emulador copia exatamente o comportamento de determinado sistema, nesse caso específico, um hardware [Law99, RI00]. O atrativo em usar esse tipo de máquina virtual está na facilidade em testar o software em desenvolvimento e da não necessidade de ter o hardware final pronto (ou de fazer modificações no projeto do hardware sem precisar de um sistema real).

Esse tipo de máquina virtual substitui o hardware. No entanto, além do hardware, é possível substituir outras camadas do sistema e criar programas que também podem ser

chamados de máquinas virtuais. Por exemplo, pode-se substituir a camada de aplicação, criando um tipo de máquina virtual bem específico. Isso se faz definindo uma certa descrição de uma máquina abstrata e uma linguagem de alto nível. Desta forma, qualquer programa descrito nessa linguagem pode ser compilado e executado (obedecendo a descrição da máquina) [Ros04b]. O caso mais famoso atualmente é da Máquina Virtual Java - JVM (*Java Virtual Machine*).

Na mesma lógica das substituições de camadas e aproveitando uma característica que tem sido muito valorizada, a portabilidade, pode-se substituir a camada dependente do hardware de um sistema operacional, de tal forma que ela permita que este sistema seja executado sobre um outro sistema. Ou seja, em vez da primeira camada do sistema operacional ser feita para fazer chamadas de instruções de hardware, ela é modificada para fazer chamadas da API de algum sistema operacional. Dessa forma, é possível executar um sistema operacional como qualquer outra aplicação. Essa técnica é usada pela máquina virtual UML (*User-Mode Linux*).

Além de modificar camadas e/ou usar técnicas diferentes, também pode-se usar alguma mistura delas. O VMware Workstation é um programa (dispõe de versões para Windows, Linux e BSDs) que usa uma mistura desses tipos de máquinas virtuais descritas. Essa mistura é feita no intuito de evitar modificações no sistema operacional a ser executado, o que faz com que qualquer sistema possa ser instalado nele.

O VMware provê um ambiente semelhante a um hardware, com emulação de periféricos própria, o que de certa forma é uma substituição da camada de hardware, assim como nos primeiros sistemas de máquinas virtuais (VM/CMS). Muitas instruções são executadas diretamente e tenta-se maximizar processamento dessa forma. Com as instruções que não podem ser executadas diretamente é feito um processo chamado recompilação dinâmica (uma técnica também usada em emuladores).

JIT (*Just-In-Time Compiler*), também conhecido como tradução dinâmica, é uma técnica muito utilizada em dois tipos de ambientes de execução: compilação em *bytecode* e compilação dinâmica. Essas técnicas são usadas em algumas linguagens de programação, tais como *Common Lisp*, *Perl* e Java (esta última discutida na Seção 3.5.1).

O uso de tradução dinâmica evita que seja sempre necessário ter o código fonte dos programas que fazem uso direto de algum recurso de hardware, ou até mesmo que modificações tenham que ser repetidas diversas vezes para as novas versões desses programas.

Como visto, alguns tipos de máquinas virtuais nasceram da necessidade de solucionar um problema, porém seus usos não ficaram restritos a esses casos. Dada sua versatilidade, com o passar do tempo, seu uso se expandiu e tornou-se uma ferramenta muito importante.

2.3 Suporte em hardware à virtualização

Quando uma máquina virtual está sendo executada, o sistema operacional considera que tem acesso irrestrito ao hardware da máquina real, o que se fosse verdade poderia causar sérios danos aos sistemas envolvidos [Ros04a]. Dessa forma, o sistema da máquina virtual tem que impedir que determinadas instruções sejam executadas, mascarando sua execução. Para isso, são usados os mecanismos de proteção de acesso ao hardware.

Os primeiros computadores eram essencialmente mono-usuários, com apenas um processo sendo executado. Com a difusão do uso de computadores, os sistemas operacionais começaram a permitir que os recursos fossem divididos entre vários processos. Porém isso trazia um problema: se o programa em execução tivesse algum tipo de erro, ele podia afetar o funcionamento de algum outro programa ou até mesmo o sistema operacional [SGG01].

Muitos dos erros dos programas são detectados pelo hardware, que gera uma interrupção e passam o controle de volta ao sistema operacional. Dessa forma, o sistema consegue forçar o término do programa em questão [SGG01].

Para contornar essa situação o hardware precisa de pelo menos dois modos de operação [SGG01]: modo usuário e modo supervisor. Dessa forma o boot do sistema é feito em modo supervisor, carregando o sistema operacional e este inicia os programas em modo usuário. Esse duplo modo de operação protege o sistema operacional e os programas de algum eventual erro, separando as instruções que podem causar algum dano, as chamadas instruções privilegiadas, permitindo sua execução somente no modo supervisor.

Se algum programa inadvertidamente tenta executar alguma instrução privilegiada o hardware não permite. Este gera uma exceção passando o controle para o sistema operacional que executa uma rotina de tratamento. Para permitir a execução dessas instruções privilegiadas de maneira segura foram criadas as chamadas de sistema; elas provêm meios de interação entre os programas (executados em modo usuário) e o sistema operacional (executados em modo supervisor) [SGG01].

Como exemplo de instruções de acesso privilegiado pode-se citar o acesso a entrada e saída de dados do sistema, a proteção dos espaços de memória e a proteção ao processador (via temporizador de execução).

Para as máquinas virtuais existem algumas instruções, não necessariamente instruções privilegiadas, que poderiam causar inconsistência no funcionamento do sistema: as chamadas *instruções sensíveis*. Estas instruções tentam [RI00]: (1) mudar ou referenciar o modo da VM ou o seu estado; (2) ler ou modificar registros sensíveis (os de tempo e os de interrupção) e/ou posições de memória; (3) referenciar o sistema de proteção de armazenamento, sistemas de memória ou sistema de realocação de endereços, que incluem instruções que permitem à VM acessar qualquer posição que não faz parte de sua

memória virtual; e (4) todas as instruções de entrada e saída. As instruções sensíveis não podem ser executadas pelo sistema operacional dentro de uma máquina virtual, sendo esta responsável por mascarar sua execução.

2.3.1 Arquitetura x86

A arquitetura IA-32 (ou x86-32) tem quatro níveis de privilégios, também chamados de *anéis de proteção*. Na Figura 2.3, esses níveis são numerados de 0 (mais privilegiado) a 3 (menos privilegiado) [Cor05d]. O nível 0 é usado pelo *kernel* e por outras partes críticas do sistema operacional, os níveis 1 e 2 por software menos críticos, culminando no nível 3 que é reservado aos programas de aplicação. Porém, de acordo com [BDF+03], os níveis 1 e 2 não são usados pelos sistemas operacionais mais conhecidos, desde o OS/2.

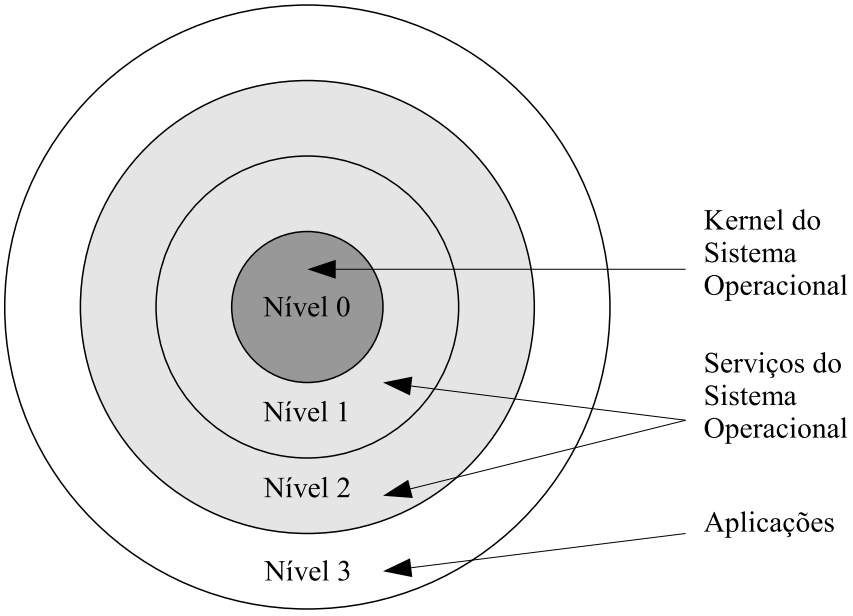

Figura 2.3: Anéis de proteção.

Código de módulos num nível de privilégio mais baixo só pode fazer acesso a código de um nível mais alto por uma interface bem controlada e protegida chamada *gate*. Qualquer tentativa de acesso a um nível mais privilegiado que não seja através do *gate* de proteção e que não tenha permissão de acesso suficiente, gera uma exceção de proteção-geral (*general-protection exception* - #GP) [Cor05d].

Existe um preceito para que uma arquitetura seja completamente virtualizável [PG74]: as instruções sensíveis devem ser um subconjunto das instruções privilegiadas. Ou seja, como o sistema virtual será executado em modo usuário, sempre que ele tentar executar alguma instrução privilegiada, o hardware irá gerar uma interrupção e passará o controle

para o monitor de máquinas virtuais (explicado logo abaixo, na Seção 2.4), que irá tratá-la adequadamente.

Porém, de acordo com [RI00, Cor05a], na arquitetura IA-32 existem dezessete instruções que são sensíveis mas não privilegiadas, o que torna a arquitetura x86 não virtualizável.

Entretanto, esse fato não impossibilita a execução de máquinas virtuais, apenas deixa a tarefa mais complexa, fazendo com que a máquina virtual tenha que rastrear a execução de instruções proibidas.

A AMD e a Intel estão com projetos para tornar a arquitetura virtualizável. Os projetos foram batizados, respectivamente, de *Pacifica* (SVM - *Security and Virtual Machine*) [pac05] e *Vanderpool* (VMX - *Virtual Machine Extensions*) [van05], ambos com lançamento para o ano de 2006.

AMD Pacifica

Para melhor utilização dos recursos do hardware, os *mainframes* e sistemas baseados nos processadores RISC já vem usando há muito tempo a virtualização. Contudo, a utilização de servidores x86 vem aumentando e várias empresas vêm explorando a virtualização desses servidores para obter uma melhor produtividade [Inc05b].

Com o apoio de empresas que têm grande interesse na tecnologia de virtualização, incluindo EMC Corp. (VMware), Hewlett-Packard Co., IBM Corp., Microsoft Corp., Novell Inc., Red Hat Inc., SunMicrosystems Inc., VERITAS Software Corp. e XenSource Inc., a AMD modificou o processador da linha AMD Opteron (64-bit e 32-bit) para suportar virtualização [Inc05b].

O SVM é um conjunto de extensões que dá suporte em hardware à virtualização e à segurança. A arquitetura de máquinas virtuais da AMD provê [Inc05a]: (1) mecanismos para uma troca de contexto entre o VMM (*Virtual Machine Monitor*) e o sistema virtual; (2) habilidade de interceptar determinadas instruções ou eventos do sistema virtual; (3) proteção ao acesso à DMA (*Direct Memory Access*); (4) assistência para o tratador de interrupções e suporte a interrupções virtuais; e (5) marcas no TLB (*Translation Lookaside Buffer*) para diminuir a sobrecarga da virtualização.

Foi criado um novo modo no processador, chamado modo *convidado*, em que o comportamento de algumas instruções é modificado para facilitar a virtualização. Os sistemas virtuais têm acesso garantido a dispositivos de E/S (Entrada e Saída), pois um mecanismo no hardware previne que um dispositivo pertencente a um sistema virtual seja acessado por outro sistema virtual (ou pelo próprio VMM). O VMM é mapeado num espaço de endereçamento diferente dos sistema virtuais e para diminuir a sobrecarga na troca de contexto, um identificador no TLB, chamado ASID (*Address Space Identifier*), permite a distinção entre os diferentes espaços de endereçamento. O VMM pode pedir que in-

terrupções físicas façam com que o sistema virtual saia, permitindo que o VMM trate a interrupção. O VMM pode adicionar interrupções virtuais no sistema virtual. Os sistemas virtuais podem compartilhar o APIC (*Advanced Programmable Interrupt Controller*), enquanto esperam que outros sistemas virtuais (maliciosos ou defeituosos) deixem interrupções de alta prioridade sem respostas (e, assim, excluindo as interrupções dos outros sistemas virtuais). As instruções podem ser reexecutadas após a sua interceptação. Uma instrução e um sistema de suporte associado (TPM[4] - *Trusted Platform Module* [tpm05]) permite a verificação de software confiável. E, por fim, um sistema de limpeza automática da memória, que apaga seu conteúdo no *reset* [Inc05a].

Intel Vanderpool

Uma organização típica de TI (Tecnologia da Informação) aloca 70 a 80 por cento de seu orçamento para gerenciar sistemas e aplicações. Embora o poder de processamento esteja se tornando cada vez mais barato, espaço, energia, instalação, integração e administração não estão. Estas empresas costumavam manter somente uma aplicação por servidor e, atualmente, o desempenho dos servidores aumentou. A virtualização ajuda essas organizações a aproveitar esse poder extra, consolidando múltiplas aplicações e sistemas operacionais em uma única plataforma, otimizando a utilização do servidor, reduzindo gerenciamento, energia e problemas com temperatura [Cor05a].

A tecnologia de virtualização da Intel é só o primeiro passo de uma série de inovações que aumentarão o suporte para avançadas soluções em virtualização. Por enquanto, o suporte a virtualização será incluído nos processadores Intel Itanium 2 e Xeon 64-bit, na linha de servidores; futuramente ela será estendida para os *desktops* e *laptops* [Cor05a].

Para processadores IA-32, a Intel criou uma tecnologia chamada VT-x [Cor05b], que constitui um conjunto de extensões que dá suporte em hardware para sua virtualização, chamado VMX. Esse suporte é dado por uma nova operação, chamada *operação VMX*, e esta tem dois modos: operação root VMX e operação não root VMX. Desta forma, o VMM será executado em modo root VMX (com o comportamento muito semelhante à operação fora do modo VMX) e o sistema virtual em modo não root VMX (com acesso restrito e modificado, para facilitar a virtualização). O modo VMX impõe essas restrições mesmo com o software sendo executado no nível de privilégio 0, o que simplifica o desenvolvimento do VMM.

A operação não root VMX é controlada por uma estrutura chamada VMCS (*Virtual Machine Control Structure*); o VMM pode usar um diferente VMCS para cada máquina virtual e para cada processador virtual associado. Operação VMX restringe as seguintes operações do processador: (1) os valores que podem ser lidos em alguns registradores;

[4]um microcontrolador incluído na placa mãe do computador que guarda chaves, senhas e certificados digitais, buscando implementar uma computação segura.

(2) interrupções A20M são bloqueadas; e (3) o sinal INIT é bloqueado em operação root VMX, mas pode ser executado em operação não root VMX [Cor05b].

Da mesma forma que a Intel modificou a IA-32, ela também adicionou as extensões nos processadores Itanium, porém com o nome VT-i [Cor05c]. Na arquitetura de virtualização do Itanium, o VMM cria processadores virtuais para apresentar ao sistema virtual. Não existem limites na arquitetura para o número de máquinas e processadores virtuais que podem ser criados pelo VMM.

Um VPD (*Virtual Processor Descriptor*) é um controle de informações de cada processador virtual que representa uma abstração de seus recursos. O VPS (*Virtual Processor State*) é uma estrutura de dados em memória que representa o estado arquitetônico do processador virtual. E os interceptores PAL formam uma interface que transfere o controle para a VMM nos eventos virtualizados [Cor05c].

2.4 Monitor de máquinas virtuais

Até este ponto, foi dada uma idéia de como são implementadas as máquinas virtuais, que basicamente se resumem a um programa gerenciador chamado VMM (*Virtual Machine Monitor*), também conhecido como *hypervisor*.

O VMM é um software que cria um ambiente que simula o acesso direto ao hardware. Ele pode prover um ambiente diferente em relação ao hardware base fornecendo, por exemplo, quantidades diferentes de memória e de espaço em disco; além de permitir que várias máquinas virtuais sejam executadas concorrentemente (que foi a idéia original) [PG74, KDC03].

Os VMMs podem ser divididos em diferentes classes, diferenciando-se pela forma de execução e pelo tratamento dado às instruções. A quantidade de instruções processadas por software ou diretamente no processador é que determina se a máquina é uma CSIM (*Complete Software Interpreter Machine*), uma HVM (*Hybrid Virtual Machine Monitor*), um VMM simples ou uma máquina real [PG74, KDC03, RI00].

Numa máquina real toda instrução é executada diretamente. No outro oposto, no CSIM, como o nome diz, todas as instruções passam por alguma tradução por software antes de ir para execução no processador, sendo o caso dos emuladores. É usada uma técnica chamada tradução dinâmica (um tipo de JIT, Seção 2.2.2), que consiste em traduzir e guardar blocos de código, sendo mais eficiente que a tradução simultânea de instrução por instrução. Uma outra técnica possível é a chamada recompilação dinâmica (outro tipo de JIT), que consiste em procurar trechos de código que são executados várias vezes, traduzindo-os uma única vez, reutilizando-os o máximo possível. Bochs (Seção 3.1.1) e Qemu (Seção 3.1.3) são máquinas virtuais CSIM.

Para uma máquina ser classificada como VMM simples, um subconjunto majoritário

das instruções deve ser executado diretamente, sem precisar de tratamento. Dessa forma, essa classe tende a ter um melhor desempenho se comparada com as outras. Como a definição dessa classe não especifica o seu modo de operação, ela pode ser dividida em dois tipos: tipo I—que é executada diretamente sobre o hardware da máquina— e tipo II—que é executada sobre um sistema operacional.

Uma máquina é dita da classe HVM se todas as instruções privilegiadas são executadas via tratamento por software. Da mesma forma que o VMM simples cabe aqui também a divisão de HVM do tipo I e HVM do tipo II. Essa classe de máquina é usada quando o processador não suporta VMMs do tipo I e do tipo II.

Os VMMs do tipo I (Figura 2.4) são praticamente um sistema operacional completo, responsáveis pelo gerenciamento de recursos, além da gerência das máquinas virtuais. A maior dificuldade na implementação desse tipo de máquina virtual está em conseguir prover suporte a todos os periféricos da máquina real. O VM/370 [Cre81], VMware ESX Server (Seção 3.3.6) e Xen (Seção 3.4.2) são exemplos de máquinas virtuais desse tipo.

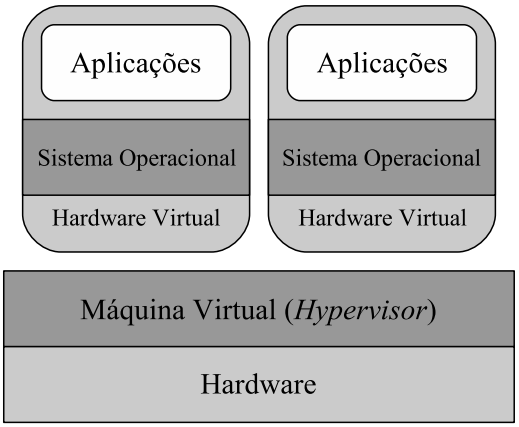

Figura 2.4: Estrutura básica de um sistema do tipo I.

Os VMMs do tipo II, Figura 2.5, são aplicações de um sistema operacional, usando todo o gerenciamento do sistema, tendo que somente prover a gerência das máquinas virtuais. Esse tipo de máquina virtual depende do sistema operacional em que ela vai ser executada. Qualquer falha nesse sistema operacional pode comprometer a VMM. O User-Mode Linux (Seção 3.3.5), VMware Workstation (Seção 3.3.6) e VirtualPC (Seção 3.3.4) são VMMs do tipo II.

Alguns autores classificam máquinas virtuais encontradas na plataforma IA-32 como HVM, devido a arquitetura não suportar virtualização nativamente. O que obriga o desenvolvimento de técnicas para permitir a execução de máquinas virtuais, fugindo da definição de VMM simples.

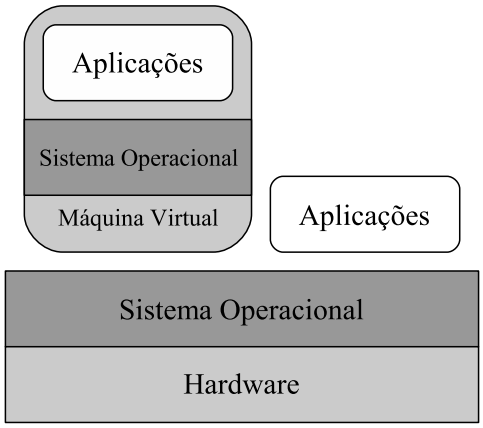

Figura 2.5: Estrutura básica de um sistema do tipo II.

2.4.1 Paravirtualização

Em busca de um melhor desempenho na execução de máquinas virtuais na arquitetura x86 (que, como dito, nativamente não suporta virtualização), com enfâse numa solução diferente das usadas até então, foi proposta a chamada *paravirtualização*. O objetivo é prover um ambiente um pouco diferente para o sistema operacional a ser instalado, com um conjunto ligeiramente diferente de instruções, para diminuir a carga de tratamento de instruções sensíveis [WCSG04, BDF+03].

O maior problema nessa solução é a obrigatoriedade de modificações no sistema operacional para que ele funcione nesse tipo de máquina virtual, porém ela tem um VMM mais simples e um desempenho melhor que as demais técnicas usadas. As máquinas virtuais Denali (Seção 3.4.1) e Xen (Seção 3.4.2) usam essa técnica.

2.5 Conclusão

Neste capítulo foi apresentada a origem das máquinas virtuais e sua evolução, passando por muitas dificuldades até culminar no VM/370, a primeira máquina virtual, que até hoje é amplamente usada em *mainframes*. Foi apresentado como os sistemas operacionais podem ser modificados e/ou como devem ser tratados para que eles possam executar concorrentemente num mesmo hardware. Foram discutidas as características que o hardware deve ter para suportar virtualização e que a arquitetura x86 não as contempla, tornando a implementação de máquinas virtuais mais difícil e prejudicando seu desempenho. Porém, como visto, os dois principais fabricantes de processadores têm projetos de extensão da arquitetura para dar suporte a virtualização. Foram apresentados os VMMs, seu funcionamento e sua classificação em: Emuladores, VMM tipo I, VMM tipo II, Paravirtualização, Virtualização de linguagem de alto nível e Virtualização no nível do sistema operacional.

Cada um desses tipos apresenta diversos exemplos, propostos nesse trabalho, servindo de base para o Capítulo 3.

Capítulo 3

Survey de máquinas virtuais

O objetivo deste capítulo é apresentar um *survey* de máquinas virtuais, para tanto, as máquinas foram classificadas por sua característica mais marcante (dado que algumas delas se encaixam em mais de uma categoria). Foi seguida a classificação proposta nas Seções 2.2.2 e 2.4:

Emuladores - virtualização completa por software;

VMM do tipo I - VMM que é executado diretamente no hardware;

VMM do tipo II - VMM que é executado sobre um sistema operacional;

Paravirtualização - VMM que provê um hardware diferente do da máquina real;

Virtualização de linguagem de alto nível - um tipo diferente de máquina virtual, para execução de programas;

Virtualização no nível do sistema operacional - segmenta o sistema criando vários ambientes de execução de programas.

Como os sistemas descritos são complexos o suficiente para servir por si só para um estudo extenso, não é objetivo deste trabalho entrar em minúcias da implementação. Desta forma, são apresentadas as características mais marcantes de cada implementação, explicitando os pontos mais interessantes da abordagem adotada e as técnicas usadas para tal. Contudo, a existência de alguns sistemas comerciais, cujas informações são escassas, fez com que uma maior uniformidade de tratamento fosse impossibilitada.

3.1 Emuladores

Permitem que programas ou sistemas feitos para executar em uma determinada plataforma sejam executados em outras, como apresentado na Seção 2.4. Os tipos apresentados serão: Bochs, PearPC e QEMU.

3.1.1 Bochs

Bochs [boc05] é um emulador x86 (processador, dispositivos de hardware e memória) de código livre e altamente portável, que emula 386, 486, Pentium, Pentium Pro ou AMD64 (experimental) e pode incluir instruções MMX, SSE, SSE2 e 3DNow!. No início, Bochs era um produto comercial, porém em 2000 a Mandriva Linux (antigo Mandrake Linux) o comprou e abriu seu código sob licença GPL (*General Public License*). Como o Bochs emula toda instrução x86 e todos os dispositivos (não tenta-se nenhum tipo de tradução dinâmica ou virtualização), é impossível esperar um desempenho próximo ao da execução nativa, contudo, os sistemas para serem executados no Bochs não precisam de modificação alguma.

Para o Bochs funcionar corretamente ele precisa interagir com o sistema no qual ele vai ser instalado e cada sistema tem uma particularidade, fazendo com que diferentes sistemas forneçam diferentes opções para o sistema a ser instalado/executado sobre o Bochs. Por exemplo, o código usado para interagir com a interface de rede do FreeBSD é diferente do código usado no Linux, por isso, apesar de poder ser executado em virtualmente qualquer plataforma, para cada sistema, determinada funcionalidade pode não estar disponível.

3.1.2 PearPC

PearPC [pea05] é um emulador PowerPC de código livre (sob licença GPL) capaz de executar a maioria dos sistemas operacionais para este hardware. Ele foi desenvolvido para executar em qualquer plataforma, porém somente o compilador GCC[1] é suportado atualmente. Além disso, somente a arquitetura x86 tem suporte a uma versão otimizada.

O PearPC faz uma emulação de um processador PowerPC (G3 sem *AltiVec*). O sistema cliente irá executar cerca de 500 vezes mais lento que a execução nativa. A versão otimizada usa JIT (explicado na Seção 2.2.2), ou seja, traduz as instruções em tempo de execução e as guarda num cache, conseguindo executar um sistema cliente cerca de 15 vezes mais lento que a execução nativa. Apesar de poder ser compilado para executar em praticamente qualquer plataforma, ele só funciona em arquiteturas 32-bit *little-endian*.

[1] *GNU Compiler Collection*.

3.1.3 Qemu

QEMU [qem05] é um emulador de código livre que usa tradução dinâmica para diminuir a sobrecarga causada pela emulação. O sistema tem dois modos de operação: emulação completa do sistema (processador e periféricos) e emulação em modo usuário (somente para Linux, usado para executar processos compilados de outra arquitetura). Existe um módulo (para Linux) acelerador para otimizar a execução de código x86 em máquina x86, chegando a um desempenho próximo à execução nativa, contudo, apesar de seu uso ser livre, o código é fechado (proprietário).

QEMU pode ser executado em sistemas x86, x86_64 e PowerPC e emular sistemas x86, x86_64, ARM, SPARC, PowerPC e MIPS. E, sendo um emulador completo de hardware, o sistema operacional não precisa de modificações para ser executado no QEMU. Também não é necessário modificações e/ou módulos no sistema principal e, por isso, o QEMU pode ter um desempenho inferior se comparado a algum sistema similar que usa essas modificações.

Como dito, QEMU usa tradução dinâmica, um tipo de JIT (explicado na Seção 2.2.2) e como essa técnica é complexa e dependente do processador, utiliza-se algumas estratégias para obter uma boa portabilidade com um bom desempenho.

3.2 VMM Tipo I

Os VMMs do Tipo I são executados diretamente sobre o hardware da máquina. Eles são um sistema operacional ou *kernel* que têm mecanismos para suportar máquinas virtuais. As máquinas virtuais apresentadas serão: Adeos, Perseus e Plex86.

3.2.1 Adeos

Os sistemas operacionais foram desenvolvidos com a premissa de que eles têm total controle sobre a máquina na qual eles são executados, gerenciando o acesso ao hardware e facilitando o desenvolvimento de programas. Os programadores fazem uso de uma abstração (a API), que é diferente entre os vários sistemas operacionais existentes, de acesso aos recursos da máquina.

Porém, essa abordagem tem algumas desvantagens, deixando os programadores presos à API do sistema onde se desenvolve e os usuários ficam limitados aos programas desenvolvidos para o sistema operacional usado.

O que acaba acontecendo nesse cenário é que funcionalidades de determinado sistema operacional poderiam ser úteis para os usuários de outros sistemas; além da impossibilidade de acesso irrestrito ao hardware pelos programadores de sistemas, deixando-os limitados ao acesso provido pelo sistema usado.

O objetivo principal do Adeos [ade05] é prover um ambiente flexível de compartilhamento dos recursos do hardware entre vários sistemas operacionais, obtendo um estado consistente e confiável entre eles e que permita que programadores e administradores tenham total controle sobre o hardware. O Adeos não tenta impor nenhuma restrição de acesso ao hardware, mesmo que isso possa causar algum tipo de erro no gerenciamento por parte dos programadores e administradores [Yag01].

Funcionamento

A base para o funcionamento do Adeos está na criação de diferentes domínios, cada um formando um ambiente, no qual o sistema tem total controle. E se o sistema souber da existência do Adeos, o uso dos recursos não necessariamente precisa ser com acesso exclusivo, pois o sistema pode interagir com o Adeos, permitindo o compartilhamento entre domínios diferentes, como pode ser visto na Figura 3.1. Nenhum domínio tem acesso ou conhecimento sobre os outros domínios, mas todos têm acesso ao Adeos [Yag02].

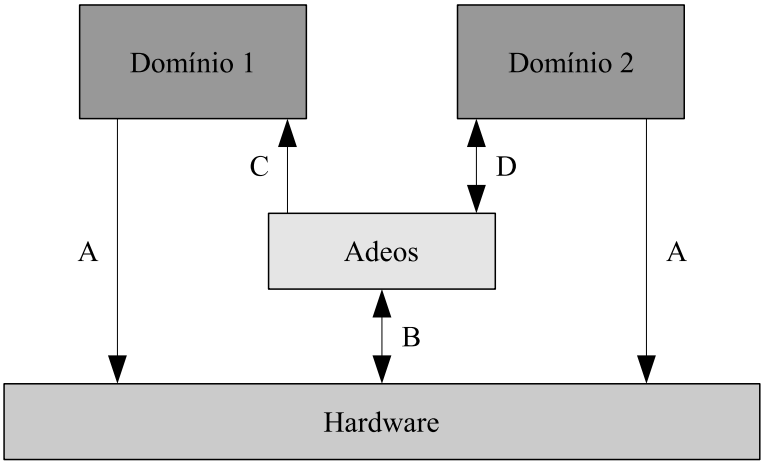

Figura 3.1: Arquitetura do Adeos e as quatro categorias de comuniação: (A) uso normal do hardware; (B) Adeos recebendo o controle do hardware por causa de uma interrupção e as intruções usadas pelo Adeos para controlar o hardware; (C) Adeos notificando o sistema sobre uma interrupção; e (D) que é semelhante à (C), mas o sistema sabe interagir com o Adeos.

Nas mudanças de domínios, o Adeos salva o estado da máquina para poder restaurar e devolver a execução a este domínio. O Adeos tenta não aplicar nenhuma política de uso dos recursos e usa o mínimo necessário para o seu próprio funcionamento. Dada essa proximidade com o hardware do sistema, o Adeos é extremamente dependente de sua arquitetura.

Implementação

Os idealizadores do Adeos fizeram uma modificação no *kernel* do Linux para poder implementar suas idéias. O Adeos foi feito como um módulo do *kernel* e quando ele é lido, faz com que o Adeos assuma o controle do hardware. Para evitar que o *kernel* do Linux afete o funcionamento do Adeos, o *kernel* é movido do nível 0 de acesso ao hardware para o nível 1, diminuindo seus privilégios de acesso, restringindo o uso de algumas instruções. Algo similar pode ser feito com o FreeBSD, usando a mesma plataforma de hardware, obtendo o mesmo resultado final.

3.2.2 Perseus

Perseus [per05] não é exatamente uma máquina virtual, é um projeto mais abrangente de segurança, com uma arquitetura que combina projetos de modernos sistemas operacionais com tecnologias de segurança. Os sistemas operacionais atuais não têm suporte a tecnologias de segurança, não implementando políticas de segurança eficientes, que possam ser mantidas por usuários comuns [PRS+01].

Para os idealizadores do Perseus, modificações em algum sistema operacional comum somente conseguem melhorar a sua segurança, adicionando a ele novas funcionalidades. A abordagem adotada para obter um sistema com um alto grau de segurança é de reescrevê-lo por inteiro, pois a segurança depende da corretude do *kernel* como um todo.

A solução proposta delega ao sistema operacional a responsabilidade da segurança do sistema, diferentemente da abordagem usada atualmente, em que as aplicações são responsáveis pela segurança. Como o objetivo deste capítulo não é discutir aspectos de segurança, que serão contemplados com a minúcia devida no Capítulo 4, aqui o foco volta às funcionalidades e características técnicas dessa abordagem.

Implementação

O sistema é baseado em um *microkernel*, numa plataforma composta por vários subsistemas. Eles fornecem uma interface em que diversos sistemas operacionais podem ser instalados. Uma das preocupações desde o início do projeto era fazer um sistema que pudesse ser instalado em várias arquiteturas de hardware. Sobre esses subsistemas e em paralelo, vários sistemas operacionais podem ser executados, juntamente com aplicações bem específicas relacionadas com segurança (geração de chaves criptográficas, por exemplo).

A Figura 3.2 fornece uma idéia geral dos componentes do Perseus. A linha vermelha divide as aplicações não confiáveis da plataforma segura do sistema. Todos os componentes que estão abaixo desta linha formam a parte crítica, responsável pela segurança do

sistema e somente esses subsistemas têm permissão para acessar diretamente o hardware. Todos os subsistemas, com exceção do *microkernel*, são executados pelo processador em modo usuário.

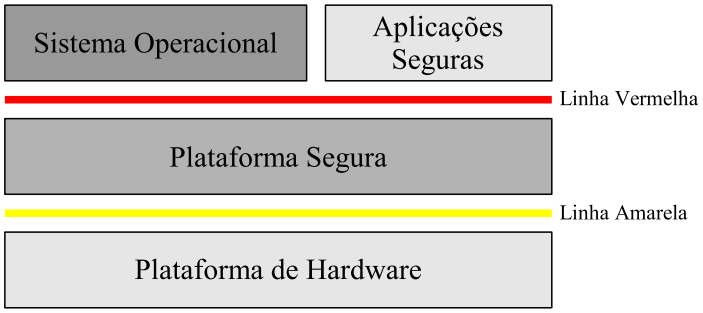

Figura 3.2: Modelo geral da arquitetura do Perseus.

O sistema como um todo é dividido em três camadas: o hardware propriamente dito, o gerenciador de recursos e uma camada de software confiável. Estes dois últimos formam o *kernel* seguro do sistema. O hardware pode ter mecanismos próprios que o deixem mais seguros, como o TPM. A camada de gerenciamento de recursos provê uma interface para uso dos recursos do hardware, como interrupções e memória. A última camada é responsável por garantir o uso dos recursos e garantir o isolamento das aplicações, implementando controle de acesso, entre outras funções.

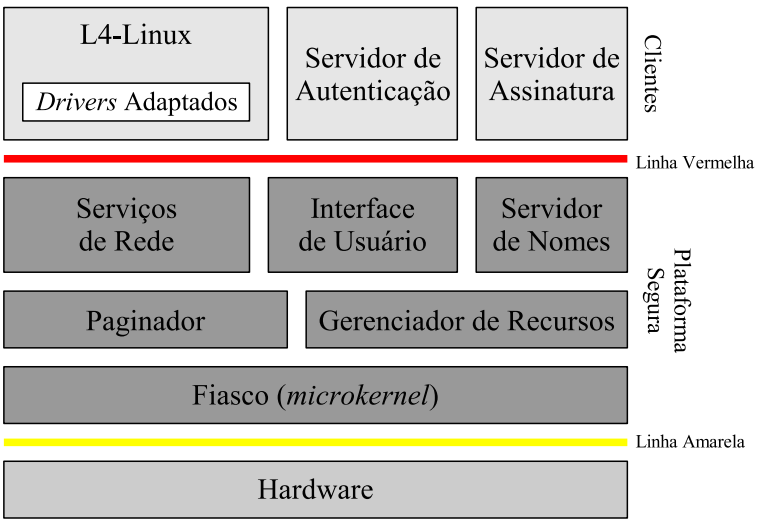

Figura 3.3: Componentes do Perseus.

Essas camadas são formadas por vários subsistemas, específicos para cada tipo de serviço e isolados entre si. Essa abordagem torna o sistema como um todo mais confiável (os subsistemas são menores e mais simples) e portável (dado que todos os subsistemas

podem ser substituídos). Na Figura 3.3 é apresentado o estado atual da implementação do Perseus, baseado no *microkernel* Fiasco [fia05]. Também existe uma abordagem monolítica baseada no sistema sHype da IBM [shy05].

Sobre as camadas, aplicações confiáveis e não confiáveis podem ser executadas em paralelo, incluindo sistemas operacionais inteiros, com o objetivo de manter compatibilidade com as aplicações existentes. Existem duas alternativas para executar sistemas operacionais: a primeira é modificar o *kernel* do sistema para efetuar, em vez de chamadas diretas ao hardware, chamadas da interface do Perseus; a outra consiste em executar o sistema operacional em alguma máquina virtual existente, como o VMware (Seção 3.3.6) e Plex86 (Seção 3.2.3).

3.2.3 Plex86

Plex86 [ple05] é um projeto que tem por objetivo fazer uma máquina virtual para Linux/x86 que seja muito leve. Por não implementar suporte completo a vários sistemas operacionais, a máquina tende a ser mais simples e com um desempenho otimizado. O Plex86 não usa técnicas complicadas como compilação dinâmica (JIT, Seção 2.2.2) ou monitoramento de execução; seu funcionamento tem como base o uso de um monitor de máquinas virtuais, VMM.

O sistema Linux requer poucas modificações para prepará-lo para ser executado dentro do Plex86. Praticamente todas as instruções de E/S e interação com o processador podem ser removidas de maneira muito simples (uma opção de compilação do *kernel*). Desta forma, todas essas instruções são substituídas por comunicação via HAL (*Hardware Abstraction Layer*[2]) através de *drivers* especiais. Isso torna a implementação da máquina virtual muito simples e eficiente, dado que não usa emulação para efetuar operações de E/S.

3.3 VMM Tipo II

Um VMM do Tipo II é uma aplicação de um sistema operacional, ele usa todo gerenciamento deste, somente provendo os mecanismos para a virtualização. Esta seção apresentará as seguintes máquinas virtuais: coLinux, FAUmachine, Mac-on-Linux, Microsoft Virtual PC, User-mode Linux e VMware. Nas seções das máquinas Microsoft Virtual PC e VMware serão apresentadas máquinas virtuais de outras classificações, mas que são produtos semelhantes das que são de maior interesse deste estudo.

[2]HAL é uma camada de abstração entre o hardware e os programas que são executados pelo computador. Essa camada é responsável por uniformizar o hardware, provendo uma plataforma consistente de execução para os programas [hal05].

3.3.1 coLinux

Cooperative Linux (coLinux) [col05] foi a primeira máquina virtual de código livre a permitir uma execução otimizada do Linux no Windows nativamente [Alo04]. Ele usa um tipo de máquina virtual chamada Máquina Virtual Cooperativa (CVM - *Cooperative Virtual Machine*) que, diferentemente das máquinas apresentadas até então, não é executada com um nível menor de privilégio que o sistema operacional principal [Alo04].

Com a máquina virtual sendo executada com o mesmo acesso pleno ao hardware, ela se torna muito mais simples e com um tempo de desenvolvimento muito menor, além de vantagens no desempenho (não sofre ineficiências com trocas de contexto). Porém, a máquina como um todo perde em estabilidade e segurança: qualquer problema em algum dos sistemas operacionais pode, ocasionalmente, comprometer todos os outros sistemas.

Os *kernels* dos sistemas operacionais são transformados para trabalharem cooperativamente, como corrotinas[3], em que cada *kernel* decide quando devolver o controle para o outro *kernel*. Um dos sistemas operacionais fica responsável pela alocação de memória e pelo hardware físico, provendo uma interface, ou seja, uma abstração de um hardware para os outros sistemas.

Implementação

Foram feitas modificações no *kernel* do Linux, basicamente para dar início ao coLinux e configurar seu funcionamento e a adição de *drivers* virtuais de rede (conet), dispositivos de bloco (cobd) e console (cocon). Um dos objetivos do coLinux era modificar minimamente o *kernel* oficial do Linux, mantendo as alterações localizadas e menores possíveis.

Para o sistema instalado diretamente no hardware, o coLinux aparece como um simples processo, deixando toda a responsabilidade sobre os processos para o sistema dentro da máquina virtual.

3.3.2 FAUmachine

FAUmachine [fau05] é um projeto de uma máquina virtual de código livre que é executada em modo usuário, por isso, inicialmente, o seu nome era UMLinux (de User-Mode Linux, homônimo à máquina virtual UML, explicada na Seção 3.3.5). Um dos objetivos do FAUmachine é prover um ambiente para injeção de falhas e controle de experimentos e testes.

Inicialmente, era objetivo do projeto modificar o *kernel* do Linux somente o suficiente para simular a funcionalidade que o hardware oferece [HBS02]. Isso é feito com a subs-

[3]subrotina é um caso especial de corrotina. Numa corrotina, os componentes trabalham como num time, revezando a execução, com retornos nos pontos em que a execução foi passada para a outra corrotina [Knu68].

tituição das instruções de hardware (instruções *assembly*) por chamadas do sistema operacional hospedeiro. A máquina é executada em modo usuário, sem nenhum módulo ou modificação no sistema em que ela será executada.

Cada processador do sistema virtual é simulado como um processo de usuário no sistema hospedeiro e a memória virtual desses processos é o espaço de endereçamento do sistema virtual. As máquinas de estados de todas as máquinas virtuais e as implementações das funções que substituem as instruções de hardware ficam no *simulator*[4]. Existe também um processo responsável pela interação com o sistema hospedeiro, controlando as resposta do teclado, *mouse* e vídeo.

Porém, o fato de não haver nenhuma modificação no sistema hospedeiro trazia duas desvantagens [HSW04]. A primeira foi a impossibilidade de executar binários do sistema que não tenham código fonte, ou seja, módulos do *kernel* que já venham compilados. A segunda foi a sobrecarga causada pelo método usado para controlar a execução das chamadas de sistema da máquina virtual.

Desta forma, para contornar os problemas citados, o sistema foi modificado para executar de quatro maneiras diferentes: modo tradicional, modo tradicional com JIT, com redirecionador no *kernel* do sistema hospedeiro e com redirecionador no *kernel* do sistema hospedeiro com JIT. De acordo com [fau05], também é possível executar o FAUmachine usando o processador virtual do simulador QEMU (discutida na Seção 3.1.3), porém não foram encontrados detalhes sobre essa abordagem.

Modo tradicional

Como a intenção dessa implementação é executar o sistema inteiro, inclusive o *kernel*, em modo usuário, o maior problema é como fazer com que uma chamada de sistema de uma aplicação já compilada seja tratada pelo *kernel* apropriado (o do sistema da máquina virtual, responsável pela execução daquela aplicação e não o *kernel* da máquina real) [HBS02]. Como os binários executados na máquina virtual e na máquina real são idênticos, toda vez que acontecer uma interceptação para tratamento de uma chamada de sistema, o *kernel* da máquina real será usado, em vez do *kernel* em modo usuário.

A solução está em rastrear a execução do processo usando a chamada de sistema *ptrace*[5].

Entretanto, tratar as chamadas usando *ptrace* é muito custoso: cada chamada de sistema gera quatro chaveamentos de contexto entre o rastreador e o processo que simula o processador e algumas chamadas de sistemas extras para tratar a execução do processo

[4] responsável por virtualizar o hardware, controlando a comunicação entre o sistema virtual e a máquina virtual.

[5] chamada de sistema que permite que um processo controle a execução de outro, usada principalmente para depuração e rastreamento de chamadas de sistema [Lin].

sob observação.

Contudo, como dito, neste método não é possível executar um sistema não modificado (como feito em algumas máquinas virtuais, como o VMware, Seção 3.3.6).

Redirecionador no *kernel* do sistema hospedeiro

No início era objetivo do projeto não fazer modificações no *kernel* do sistema hospedeiro, porém essa abordagem não conseguiu ter um desempenho satisfatório. A solução natural era tentar reduzir a sobrecarga usada pelo método tradicional, tentando tratar o redirecionamento das chamadas de sistema de uma forma mais eficiente.

Para contornar esse problema, os coordenadores do projeto decidiram adicionar uma chamada de sistema no *kernel* do sistema hospedeiro, que implementaria esse tratamento mais eficiente. Essa nova chamada se chama *sys_faumachine_helper()*. As modificações feitas para adicionar essa nova chamada são semelhantes às usadas para implementação da chamada *ptrace* e, por isso, acabam adicionando alguma sobrecarga na execução no sistema hospedeiro.

Esse mecanismo reduz o custo da redireção da chamada de sistema significantemente [HSW04], pois diminui a carga de execução para somente o sinal que executa a nova chamada de sistema.

JIT

Durante a execução, um trecho de código é transformado, usando tradução dinâmica, em código que pode ser executado em modo usuário. Esse novo código é uma mistura do código original e algumas chamadas do *simulator*. Para melhorar a eficiência dessa abordagem, toda instrução traduzida é mantida numa cache.

Desta forma, é possível executar normalmente sistemas Linux sem ter que modificá-los, ao custo de um desempenho inferior.

3.3.3 Mac-on-Linux

Mac-on-Linux [mol05], ou simplesmente MOL, é uma máquina virtual que propõe executar qualquer sistema operacional para PowerPC (MacOS, MacOS X e Linux) em um Linux/ppc. Portanto, MOL é uma máquina virtual para o ambiente PowerPC (não x86), apesar de estar sendo considerada a possibilidade de se adicionar um emulador para x86.

Pouca informação foi encontrada sobre essa máquina, somente a documentação para usuário, apesar de ser um sistema de código aberto, desenvolvido sob GPL.

3.3.4 Microsoft Virtual PC

Virtual PC [vir05] é a máquina virtual da Microsoft, baseada na tecnologia que ela comprou da Connectix, em 2003. Cerca de seis anos antes [Mic03], a Connectix lançava o Virtual PC, um sistema pioneiro para que usuários de *Macintosh* pudessem executar programas feitos para Windows. A Microsoft decidiu comprar uma solução de máquinas virtuais, pois ela achou que seus clientes precisavam de uma opção suportada pela empresa para executar sistemas Windows em máquinas virtuais.

A Microsoft comprou três sistemas: Virtual PC para Windows (rebatizado para Microsoft Virtual PC), Virtual PC para Mac e Virtual Server. O primeiro sistema permite que outros sistemas operacionais sejam executados concorrentemente num PC com Windows. Virtual PC para Mac é uma máquina virtual que faz uma emulação completa de um PC, para que usuários de Mac possam instalar sistemas Windows e suas aplicações. Por fim, o Virtual Server é voltado para sistemas servidores, com a linha Windows voltada para servidor [Dav05, Mic05].

Como o sistema é proprietário e de código fechado, muito pouca informação sobre o seu funcionamento pode ser encontrada. A documentação oferecida pela fabricante do sistema relata somente as funcionalidades e suas aplicações. A única informação encontrada indica que o Virtual PC usa um esquema híbrido de emulação e virtualização de hardware [Hon03].

Porém, de acordo com [Con01], o sistema, ainda sob posse da Connectix, funcionava com tradução de binários, num mecanismo que usa um JIT (semelhante ao explicado na Seção 2.2.2). O sistema Virtual PC para Mac usa essa técnica para traduzir código de x86 para código de PowerPC e, no Virtual PC para Windows, o código x86 é modificado em algumas partes para permitir sua execução.

3.3.5 User-mode Linux

Um dos primeiros usos de máquinas virtuais é no auxílio ao desenvolvimento de sistemas operacionais. Isso pois, pode-se executar o sistema em desenvolvimento sem precisar usar um computador somente para isso e depurar seus erros no próprio ambiente de desenvolvimento. E essa foi a motivação para o desenvolvimento do UML (*User-Mode Linux*) [uml05], uma ferramenta para auxiliar a depuração do *kernel* do Linux. O UML é um porte do *kernel* do Linux para que ele faça chamadas de sistema (do próprio Linux) em vez de executar instruções de hardware [Dik00], desta forma, podendo ser executado em modo usuário. Um resumo disto pode ser visto na Figura 3.4. Todos os dispositivos são abstrações providas pelo sistema hospedeiro, portanto, sendo todos eles virtuais, não existindo nenhuma emulação do código executado no espaço de usuários.

O maior problema dessa abordagem está em como fazer com que as chamadas de

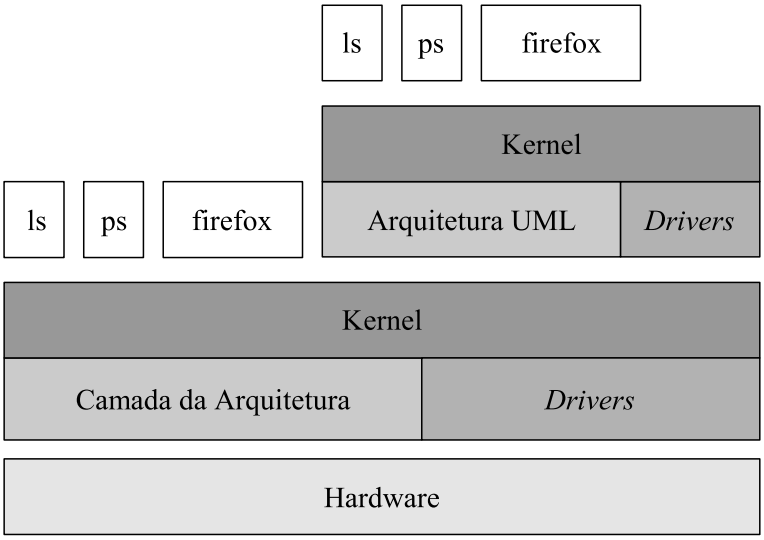

Figura 3.4: Arquitetura do UML.

sistemas do sistema virtual sejam tratadas pelo kernel virtual (um problema semelhante ao encontrado pela máquina virtual FAUmachine, Seção 3.3.2). Para resolver esse problema, uma *thread* especial é usada (*thread* de rastreio), cuja função é assistir à execução dos programas, interferindo toda vez que uma chamada de sistema é realizada [Dik01a]. A chamada de sistema *ptrace* é usada para isso. De forma semelhante, quando um processo recebe um sinal, a *thread* de rastreio a percebe antes do próprio processo. Toda sobrecarga do desempenho está ligada a essa *thread* de rastreio, que realiza vários chaveamentos de contexto para realizar a sua função [Dik01b].

A parte do *kernel* que é independente do hardware não consegue perceber a diferença, com sua execução como se fosse numa máquina normal. Para o sistema hospedeiro, o UML é apenas um conjunto de processos, mas para os processos dentro da máquina virtual, ele é um *kernel* normal.

Modo SKAS

O modo SKAS (*Separate Kernel Address Space*) é uma modificação no *kernel* do sistema hospedeiro para permitir a execução do *kernel* virtual em um espaço de endereçamento diferente. No modo anterior, chamado modo TT (*tracing thread* - *thread* de rastreio), o *kernel* era executado no mesmo espaço de endereçamento dos demais programas, caracterizando um problema de segurança; num sistema executado em hardware o processador não permite que os programas modifiquem a memória usada pelo *kernel*, o mesmo não acontece com o *kernel* UML. Também é feita uma modificação na chamada *ptrace* para eliminar os sinais enviados em cada chamada de sistema do UML, melhorando o desem-

penho do sistema. Para o sistema hospedeiro, em vez de vários processos (no modo TT), o UML aparecerá como quatro processos.

3.3.6 VMware

A plataforma VMware de virtualização [vmw05] é uma linha de produtos que permite a execução de vários tipos de ambientes de máquinas virtuais, ajudando na consolidação de servidores, melhorando o desenvolvimento de programas e aumentando a disponibilidade de sistemas. VMware é uma empresa criada em 1998, cujo objetivo era trazer para o mundo dos computadores a tecnologia de máquinas virtuais existente nos *mainframes*. Voltada para o mundo comercial, ela dispõe dos seguintes programas:

VMware Workstation - programa de virtualização voltada para *desktops*, permitindo a execução de vários sistemas operacionais num mesmo computador;

VMware GSX Server - máquina virtual voltada para a consolidação de servidores; é executada sobre outro sistema operacional, gerenciando um conjunto grande de sistemas virtuais;

VMware ESX Server - sistema executado diretamente sobre o hardware, desenvolvido para consolidação de servidores em ambientes de alto desempenho; executa várias máquinas virtuais com um controle fino na utilização dos recursos do servidor.

Como esses sistemas são produtos comerciais e de código fechado, pouca informação é disponibilizada sobre o seu funcionamento e sobre as técnicas utilizadas para reduzir a sobrecarga inerente ao uso de máquinas virtuais. Porém, foram encontradas algumas informações sobre o VMWare Workstation e o VMware ESX Server, que seguem nas subseções seguintes.

VMware Workstation

A arquitetura x86 não foi feita para suportar virtualização, o que faz com que seja muito difícil desenvolver uma máquina virtual com um bom desempenho. O segredo para contornar esse problema é maximizar o número de instruções que são executadas diretamente, sem tratamento prévio pelo VMware. Também é usada a tradução de binários (um tipo de JIT, explicado na Seção 2.2.2). Esse é um esquema denominado *virtualização transparente*, que permite a execução dos sistemas sem ter que modificá-los. Além dessa técnica, o VMware dispõe de *drivers* especiais para os dispositivos providos pela virtualização, que são feitos para minimizar a sobrecarga causada pela máquina virtual.

O VMware Workstation é instalado no sistema como um programa comum, como pode ser visto na Figura 3.5. A virtualização do processador é feita pelo VMM e quando

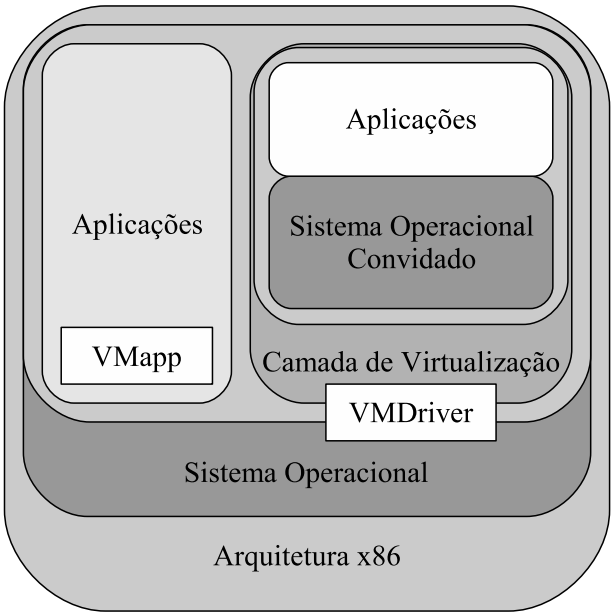

Figura 3.5: Arquitetura do VMware Workstation.

o sistema tenta executar uma instrução de E/S, o VMM chaveia para fora da máquina virtual e passa o controle para uma aplicação que faz parte do VMware, denominada VMApp; o chaveamento é feito por um *driver*, chamado VMDriver. Quando a VMApp assume o controle ela executa a operação como se fosse para ela própria e passa o resultado de volta para o VMM [SVL01].

VMware ESX Server

Um dos elementos básicos para o VMWare ESX Server é o *hypervisor* (correspondendo ao VMM), que é o componente que faz a virtualização, as partições e gerenciamento do hardware. A estrutura básica do sistema pode ser visto na Figura 3.6.

Além do funcionamento padrão, o VMware ESX Server provê uma interface chamada *Hypercall* (*VMware Hypercall Interface*), permitindo que os sistemas operacionais sejam modificados para fazer uso dessa interface. Esse esquema faz parte de outra técnica de virtualização chamada *paravirtualização*.

3.4 Paravirtualização

Paravirtualização é uma técnica de virtualização em que o VMM apresenta às máquinas virtuais uma interface similar ao hardware da máquina real. Esta técnica é usada para obter máquinas virtuais com um melhor desempenho. As máquinas Denali e Xen serão

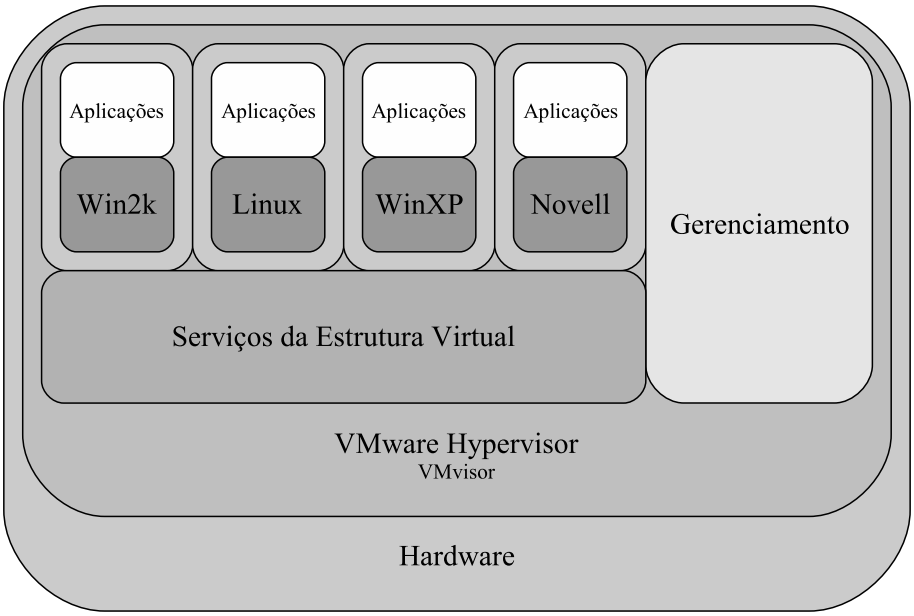

Figura 3.6: Arquitetura do VMware ESX Server.

apresentadas nesta seção.

3.4.1 Denali

O desenvolvimento da tecnologia na área de redes transformou as aplicações de tal forma que agora qualquer empresa tem que oferecer seus serviços pela Internet. Os benefícios dessa infra-estrutura computacional também se estendem a pequenos serviços, porém, manter uma estrutura para um pequeno servidor pode ser custoso e/ou ser um desperdício de poder computacional. Muitos serviços são mantidos em organizações especializadas em prover uma infra-estrutura que comporte vários pequenos servidores. Uma característica desses servidores é que a infra-estrutura não pode confiar nos serviços de seus clientes e que esses servidores não confiam um no outro [WSG02b]; o que força a existência de um mecanismo que isole um servidor do outro e da infra-estrutura, de forma segura e que mantenha um desempenho satisfatório. Se o isolamento entre os servidores puder ser garantido sem introduzir uma sobrecarga proibitiva, um computador pode hospedar um grande número de servidores (centenas ou até milhares) [WSG02b].

Denali [den05] é um projeto que se propõe a implementar domínios protegidos, permitindo que vários servidores possam ser executados, usando a tecnologia de máquinas virtuais. Esses domínios devem: (1) ser isolados entre si, não permitindo qualquer interação entre programas de domínios diferentes; (2) suportar vários domínios, podendo

executar centenas ou milhares de máquinas virtuais num mesmo computador[6]; e (3) trocar rapidamente entre eles, respondendo em tempo hábil as requisições de todos os serviços, inclusive os pouco usados.

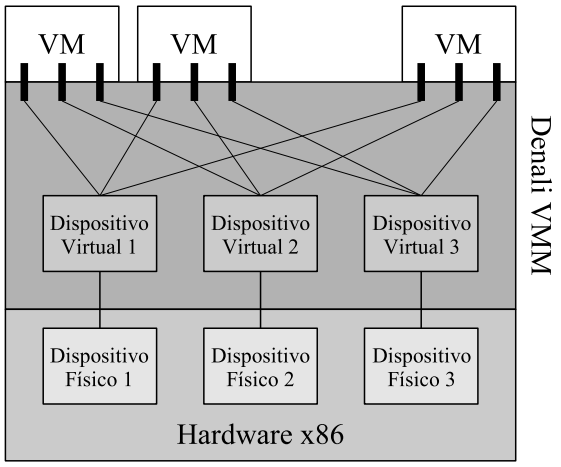

Figura 3.7: Arquitetura do Denali.

Essas ações ocorrem com uma camada de software que virtualiza o hardware, ilustrado na Figura 3.7, realizando um serviço semelhante a um VMM. Isto é, a camada não emula o acesso ao hardware, mas provêm uma arquitetura ligeiramente diferente, focando em simplicidade, escalabilidade e desempenho [WSG02a]. Além disso, algumas instruções foram removidas e outras foram introduzidas, não existindo na arquitetura física[7]. A maior desvantagem é perda de compatibilidade com os sistemas operacionais existentes. Denali tem um número reduzido de dispositivos e todos eles com uma interface muito simplificada, realizando a maioria dos serviços com apenas uma única instrução.

Para fazer uso dessa nova arquitetura provida pelo Denali, foi especialmente desenvolvido um sistema operacional hospedeiro, chamado *Ilwaco*, que contém um porte da pilha TCP/IP do BSD, suporte a *threads* e a um subconjunto da API POSIX [WSG02c].

μDenali

μDenali é uma importante reimplementação do VMM Denali, que procura facilitar alguns tipos de serviços do sistema, tais como migração, detecção de intrusão e *replay logging*[8]. Todos esses serviços dependem do VMM ter a habilidade de observar e intervir nas funcionalidades e no estado do sistema. E pensando nisso, a arquitetura do μDenali foi refeita

[6] em [WSG02c] é demonstrado que Denali suporta mais de 10.000 VMs num único computador.
[7] como a instrução *idle*, que permite à VM ceder o controle do processador.
[8] registro da execução de uma máquina virtual, de tal maneira que este possa ser reexecutado.

(Figura 3.8), para permitir que as máquinas virtuais possam intervir em suas máquinas filhas [WCSG04].

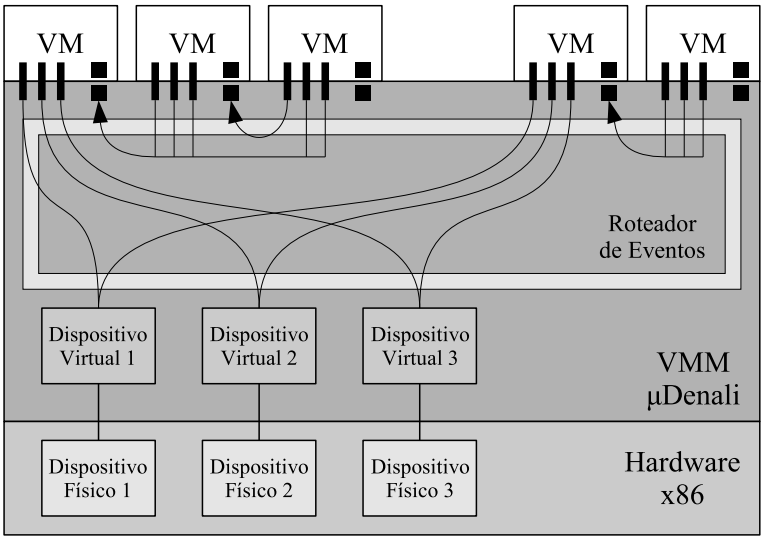

Figura 3.8: Arquitetura do μDenali.

3.4.2 Xen

Computadores modernos têm poder suficiente para executar um grande número de pequenas máquinas virtuais. Essa é a proposta da máquina virtual Xen [xen05], um VMM gerenciador de recursos com alto desempenho. Com poucas modificações num sistema operacional, é possível executar várias instâncias do sistema de maneira isolada e com pouca sobrecarga causada pela virtualização [BDF+03]. Também buscou-se suporte a vários sistemas operacionais, tais como Linux, FreeBSD, NetBSD, Plan9 e Windows XP (que não está disponível por problemas com licença). Para o Xen, a execução de sistemas operacionais deveria seguir a mesma lógica da execução de processos, multiplexando os recursos do sistema na granularidade de um sistema operacional inteiro [BDF+03].

Xen baseia-se em apresentar uma abstração ligeiramente diferente do hardware da máquina, de tal forma que seja mais simples implementar a virtualização. A estrutura usada é apresentada na Figura 3.9, explicitando a máquina virtual que controla o sistema inteiro. O VMM do Xen é denominado *hypervisor* e é executado num nível de privilégio acima dos sistemas convidados. Instruções privilegiadas são paravirtualizadas (*hypercalls*), pois precisam ser validadas e executadas pelo Xen. Para otimizar a execução das chamadas de sistemas (um dos gargalos) elas são registradas num tratador rápido de exceções, sendo validadas somente quando registradas na tabela de exceções do hardware.

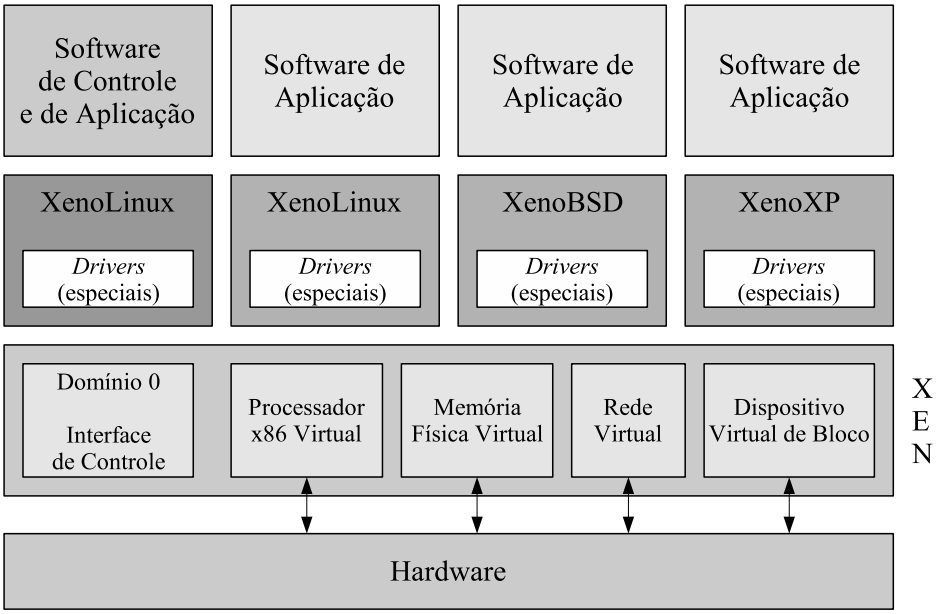

Figura 3.9: Estrutura do Xen.

Os dispositivos são apresentados como abstrações mais simples e o Xen usa memória compartilhada e um *buffer* assíncrono, conseguindo uma comunicação mais rápida. O sistema que é criado no *boot* da máquina tem a permissão para usar a *interface de controle*, responsável pelo gerenciamento do sistema.

As versões mais novas do Xen terão suporte à arquitetura 64-bit, aos novos processadores da Intel e AMD (com suporte a virtualização), a sistemas com mais de um processador e à realocação de máquinas virtuais (transferência de uma máquina virtual em execução de uma máquina para outra).

3.5 Virtualização de linguagens de alto nível

Máquinas virtuais dessa categoria podem ser entendidas como emuladores, porém, o hardware delas não existem, sendo elas produtos de uma especificação. Apesar de seu objetivo não ser executar um sistema operacional, mas executar programas, é descrita a Máquina Virtual Java, dada a importância e particularidade.

3.5.1 Máquina Virtual Java

A JVM (*Java Virtual Machine*) [jav05] é uma máquina abstrata [LY99], sendo ela a base para o funcionamento da linguagem Java. Isso a torna independente de hardware e sistema operacional, com pequeno código compilado e com a habilidade de proteger os usuários

de códigos maliciosos. Assim como qualquer computador real, a JVM tem um conjunto de instruções e manipula áreas de memória quando está executando algum código.

Contudo, a JVM não conhece nada sobre a linguagem de programação Java, conhecendo somente um formato binário, que contém as instruções da JVM (conhecidas como *bytecodes*), chamado arquivo *class*. O livro [LY99] especifica como deve ser implementada a JVM, que se resume a ler um arquivo *class* e executar as operações pertinentes, porém, fica a cargo do desenvolvedor escolher as estratégias usadas em alguns pontos da implementação da JVM, tais como leiaute de mémoria, algoritmo para o *garbage-collection* ou otimizações das instruções que serão executadas pela JVM.

O código no arquivo *class*, por segurança, passa por uma forte verificação de formato e de estrutura antes de ser executado. Além disso, o código executado pode passar por um fino controle e ser executado dentro de uma *sandbox* com diferentes níveis de acesso.

3.6 Virtualização no nível do sistema operacional

Os sistemas desse tipo tem como principal característica o fato de usar um único *kernel* para todos os sistemas. Para isso, o *kernel* é especialmente modificado para ter essa noção de isolamento entre diferentes ambientes de execução. FreeBSD *Jail*, Solaris *Zones* e Linux-VServer serão apresentadas.

3.6.1 FreeBSD *Jail*

O modelo de segurança do sistema UNIX é simples, mas inexpressivo [KW00]. O sistema de controle de acesso é baseado em dois tipos de usuários: um com e outro sem privilégios administrativos. Seria desejável que se pudesse delegar algumas funções administrativas e, ao mesmo tempo, impor políticas na interação e compartilhamento entre os processos.

Muitos sistemas operacionais tentam solucionar problemas de segurança implementando um controle de acesso mais refinado. Porém, essas tentativas sofrem de três limitações [KW00]: o aumento da complexidade administrativa, o que normalmente causa uma falsa sensação de segurança; muita segregação administrativa, causando dificuldades em executar algum código; e a introdução de novas APIs para gerenciamento de segurança, o que pode gerar incompatibilidade com aplicações existentes.

Para solucionar as proposições acima, o sistema FreeBSD provê uma funcionalidade chamada *jail* [jai05]. Jail é um mecanismo que particiona o ambiente do sistema, ao mesmo tempo que mantém a simplicidade do sistema UNIX [Fre]. Ele segue duas filosofias: (1) confinar uma aplicação específica ou (2) criar uma imagem virtual do sistema, executando uma variedade de serviços. Processos dentro da *jail* têm acesso pleno aos arquivos,

processos e serviços de rede que lhe são associados, não enxergando ou tendo acesso fora dessa "partição". Partição é um ambiente de gerenciamento, em que o sistema é dividido.

Quando um processo é colocado num *jail*, ele e seus descendentes estão confinados a esse ambiente, não podendo deixá-lo. Certas restrições são impostas: o espaço de nomes é restrito a uma nova raiz de sistema e só é permitida a escuta de uma porta do endereço IP da *jail*. O acesso a recursos e manipulação de processos é restrito e só é permitido interagir com processos do mesmo *jail*.

Desta forma, obtém-se um nível de acesso mais restrito ao administrador dentro do ambiente *jail* e processos sem nível de acesso privilegiado não vão sentir muita diferença.

As seguintes funcionalidades são proibidas ao usuário *root* dentro da *jail* [KW00]:

- modificar o *kernel* ou ler módulos;
- modificar configurações de rede, interfaces, endereços ou tabelas de roteamento;
- montar ou desmontar sistemas de arquivos;
- criar *device nodes*;
- acesso a *raw sockets*[9] e afins;
- modificar parâmetros de execução do *kernel*;
- alterar *flags* relacionadas ao *securelevel*;
- acesso a recursos de rede não associados à *jail*.

E essas funcionalidades têm acesso limitado:

- sinalizar processos fora da *jail* é proibido;
- modificar o dono e os modos de acesso de qualquer arquivo da *jail*, se for permitido;
- apagar qualquer arquivo da *jail*, se permitido;
- escutar em portas reservadas TCP e UDP, desde que seja no endereço associado à *jail*;
- funções que alteram o espaço de uid/gid são permitidas, obedecendo algumas restrições do sistema de arquivos.

[9] permite envio e recebimento de datagramas sem que utilize os protocolos TCP e UDP.

3.6.2 Linux-VServer

Com o passar do tempo, os computadores se tornaram poderosos o suficiente para executar várias instâncias de um sistema operacional e, para isso, existem vários tipos de máquinas virtuais, cada uma fazendo uma determinada abstração. Cada máquina virtual pode exigir ou não modificações no sistema hospedeiro, que costumeiramente é modificado com o intuito de melhorar o desempenho geral do sistema ou prover alguma funcionalidade extra.

O ideal seria poder usar toda a capacidade de processamento que um computador atual tem, já que é possível suportar mais de dez vezes o que um servidor Linux pode executar [lin05]. Contudo, realizar esta tarefa em um único ambiente, fazendo malabarismos com configurações de diversos serviços, é uma tarefa que pode diminuir a estabilidade e a segurança do sistema.

Devido a esta dificuldade, foi criado o Linux-VServer [lin05], que separa o espaço de usuário em unidades distintas, chamadas VPS (*Virtual Private Servers*), de tal forma que cada VPS seja o mais próximo possível de um servidor real para os processos dentro dele. Linux-VServer permite execução simultânea, em um único *kernel*, sem acesso direto ao hardware e que compartilhe recursos de maneira eficiente.

Muitas das ferramentas necessárias para fazer o Linux-VServer funcionar já estavam no *kernel*, fornecendo características seguras. São elas:

Sistema de *Capability* do Linux: baseado nas *Capabilities* POSIX, é um conjunto de 3 *bitmaps* que o processo usa para provar que pode executar determinada ação num objeto, como, por exemplo, acertar o relógio do sistema;

Limitador de recursos: limita o uso de recursos de um processo, semelhante à *Capability* do Linux, impondo dois tipos de limites, um limite leve (que tenta ser imposto) e outro limite forte (um teto); um exemplo é o tempo de uso do processador;

Atributos de arquivos: para permitir a mudança de certas propriedades dos arquivos, como a permissão de poder fazer somente a operação de anexar num arquivo;

Modificador de raiz do sistema: executa um comando com um outro diretório sendo a raiz do sistema; apesar de não ser muito seguro, ele aumenta o isolamento do processo no sistema de arquivos;

Contudo, algumas modificações precisaram ser feitas no *kernel* para completar o suporte:

Separação de contextos: adicionar ao *kernel* a noção de contextos, proibindo interação entre processos de diferentes contextos;

Separação de rede: limita os processos a um subconjunto de endereços de rede (não usa *devices* virtuais);

Barreira para o *chroot*: tenta contornar os problemas de segurança que a chamada *chroot* apresenta;

Funcionalidades da *Capabilities* POSIX: as funcionalidades ligadas ao sistema de arquivos não foram implementadas, porém elas são importantes para o funcionamento do Linux-VServer; com elas, dá para deixar segura a execução de arquivos com *setuid* ou *setgid*;

Isolamento de recursos: alguns recursos precisam de isolamento extra, até para evitar problemas de segurança ou melhorar a contabilidade (*accounting*); é o caso do número de identificação dos usuários e processos;

XID (identificador de contexto): não é obrigatório, mas é usado para aumentar o nível de isolamento do sistema de arquivos e o isolamento de contexto, adicionando aos arquivos o identificador do contexto dono daquele arquivo.

E, por fim, modificações não obrigatórias que vieram a se tornar bastante úteis com o passar do tempo:

Sinalizador de contexto: para habilitar funcionalidades do Linux-VServer, por exemplo, um sinalizador para pôr todos os processos do contexto na fila de espera (como se fosse uma pausa no processamento daquele contexto);

***Capabilities* de contexto:** semelhante ao sinalizador de contexto (seguindo uma separação mais lógica do que prática), servindo como um ajuste fino às *Capabilities* do Linux; um exemplo é o sinalizador que permite montagem segura;

Contabilidade de contexto: limitação sobre uso dos recursos do contexto como um todo, mas de um ponto de vista mais administrativo, dando uma idéia de quantidade, tratando o sistema como um todo; é o caso do número de pacotes de rede transmitidos e recebidos;

Limitação de contexto: limitação sobre o uso dos recursos do sistema, como consumo de memória ou número de processos, colocando limite mínimo, máximo leve e máximo forte;

Virtualização: ajuste de pontos específicos do sistema, para dar uma idéia maior de individualidade, aumentando o realismo do sistema dentro de um contexto; um exemplo é o ajuste do *uptime* para o contexto;

Modificações no *procfs*: protege as entradas dinâmicas desse sistema virtual, limitando a visibilidade dentro dos contextos, para que eles só vejam os seus próprios processos;

Ajudante do *kernel*: programas do espaço de usuários que precisam ser modificados para fazerem a tarefa coerente ao contexto; é o caso do programa que faz o reinício do sistema (*reboot*), que deve somente fechar o contexto.

Por fim, para a implementação do Linux-VServer foi criada uma chamada de sistema (*sys_vserver()*), que executa várias funções, dependendo do valor de um dos parâmetros. Ela foi feita para ser o mais portável possível, podendo ser executada em várias arquiteturas.

3.6.3 Solaris *Zones*

O sistema operacional Solaris tem um mecanismo, chamado *zones* [zon05], que cria um ambiente isolado para executar aplicações. Processos sendo executados em uma zona têm acesso restrito ao sistema, prevenindo-o de monitorar ou interferir em outras atividades do sistema [Sun]. Solaris *zones* pode ser administrado de maneira muito similar a máquinas separadas.

A consolidação de servidores é uma tecnologia que vem despertando interesse de um número cada vez maior de usuários e seu uso, que já é comum no mundo dos *mainframes*, vem se tornando um diferencial no mundo UNIX [TC04].

Os trabalhos nessa área envolvem a execução de várias instâncias de um sistema operacional numa mesma máquina, o particionamento de hardware e as máquinas virtuais. O particionamento de hardware provê um bom isolamento entre os sistemas, mas é custoso para implementar e normalmente é limitado a sistemas grandes[10]. O uso de máquinas virtuais sofre de uma significativa sobrecarga no desempenho. A principal diferença entre o uso de máquina virtual e a execução de várias instâncias do sistema operacional está em que há somente um *kernel*, que foi melhorado para prover esse isolamento de grupos de processos. A virtualização, nesse caso é do ambiente de execução do sistema operacional e não do hardware.

Zones é uma forma de *sandbox* em que, numa única instância de um sistema operacional, uma ou mais aplicações podem ser instaladas e executadas sem afetar as outras zonas ou o sistema. Além de não necessitar de modificações, o desempenho da execução de aplicações dentro de uma zona é praticamente idêntico ao de sua execução canônica [PT04]. O sistema operacional forma a chamada *zona global* e as demais zonas

[10] pois exige hardware adicional autônomo, que necessita seu próprio mecanismo de comunicação com o usuário: console, teclado, etc.

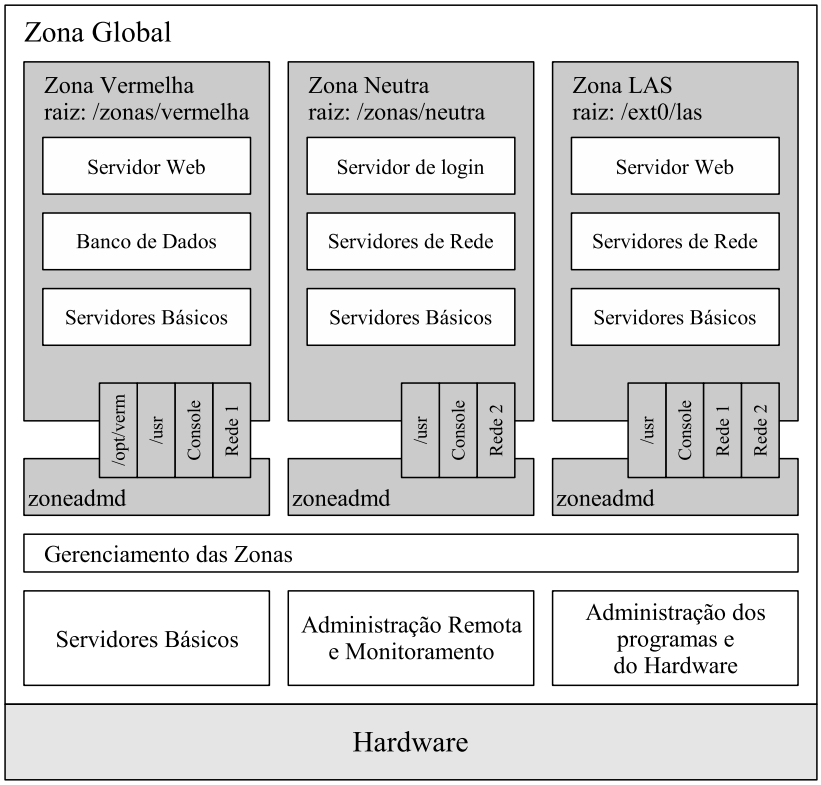

Figura 3.10: Exemplo de uma estrutura com três zonas não globais.

(chamadas de *zonas não globais*) são instâncias dessa zona principal, como pode ser visto na Figura 3.10.

Uma zona não global é executada com um conjunto reduzido de privilégios. Para prevenir compartilhamento entre processos de diferentes zonas, por exemplo, um identificador de zona é associado a cada objeto, baseado na zona em que aquele processo foi criado. Somente a zona global pode observar processos e outros objetos de zonas não globais (porém, o envio de sinais é restrito ao usuário *root* da zona global). Da mesma forma, o acesso à rede é restrito ao uso dos protocolos de transporte *TCP* e *UDP*, não sendo permitido o acesso a *raw sockets*. Cada zona é associada a um ponto específico do sistema de arquivos, transformando-o na raiz do sistema daquela zona. Por fim, o *zones* está integrado ao sistema de gerenciamento de recursos do Solaris.

3.7 Conclusão

Tabela 3.1: Resumo das principais características das máquinas virtuais apresentadas.

Nome	Tipo	Funcionamento	Principal característica
Adeos	VMM Tipo 1	Separação dos ambientes em domínios	Prover um ambiente flexível do hardware
Perseus	VMM Tipo 1	Microkernel, com vários subsistemas	Projeto de segurança
Plex86	VMM Tipo 1	Modificação da camada dependente do hardware	Sistema leve para Linux/x86
VMware ESX Server	VMM Tipo 1	Gerenciamento do hardware a partir de um hypervisor	Executa vários sistemas em paralelo
Denali	VMM Tipo 1	Provêm uma arquitetura ligeiramente diferente do hardware físico	Gerenciamento do hardware com a criação de vários domínios isolados
Xen	VMM Tipo 1	Simplicar a virtualização com uma abstração ligeiramente diferente	Execução de várias instancias do mesmo sistema com poucas modificações
coLinux	VMM Tipo 2	Permitir que vários sistemas rodem cooperativamente	Primeiro sistema de execução otimizada de Linux no Windows
FAUmachine	VMM Tipo 2	Execução em modo usuário	Substituição de instruções do hardware
Mac-on-Linux	VMM Tipo 2	Executar sistemas de PowerPC em Linux/ppc	Máquina virtual para executar MacOS, MacOSX e Linux
Microsoft Virtual PC	VMM Tipo 2	Hibrido entre emulação e virtualização	Executar Windows em máquinas virtuais

Continua na próxima página...

Tabela 3.1 – Continuação

Nome	Tipo	Funcionamento	Principal característica
User-mode Linux	VMM Tipo 2	Executar chamadas de sistema Linux ao invés de instruções de hardware	Fácil execução de sistemas em desenvolvimento
VMware Workstation	VMM Tipo 2	Virtualização transparente	Não precisa de modificações no sistema convidado
Bochs	Emulador	Emula toda instrução x86 e todos os dispositivos	Emulador altamente portável de x86
PearPC	Emulador	Emula toda instrução PowerPC e todos os dispositivos	Emulador de PowerPC independente de arquitetura
QEMU	Emulador	Baseado em tradução dinâmica	Emulador e virtualizador genérico
Máquina Virtual Java	Linguagem de alto nível	Conjunto de instruções de uma máquina abstrata	Linguagem independente de hardware e sistema operacional
FreeBSD Jail	Sistema operacional	Particionando o ambiente do sistema	Melhorar o controle de acesso mantendo o sistema simples
Linux-VServer	Sistema operacional	Separação do espaço de usuário em várias unidades distintas	Execução simultânea de vários serviços em um mesmo kernel
Solaris Zones	Sistema operacional	Ambiente isolado para executar aplicações	Consolidação de servidores

Foram apresentadas, neste capítulo, as principais máquinas virtuais existentes, dentro da classificação proposta. Um resumo pode ser visto na Tabela 3.1.

Na primeira parte, sobre Emuladores, foram estudadas três implementações: (1) Bochs, um emulador para x86 que não usa nenhum tipo de otimização na sua execução; (2) PearPC, um emulador de PowerPC; e (3) QEMU, um emulador para x86, x86_64, ARM, SPARC, PowePC e MIPS que usa tradução dinâmica para obter um melhor desempenho.

Em seguida, são apresentadas três máquinas virtuais do tipo I: (1) Adeos, que procura fazer um ambiente flexível de gerenciamento do hardware, permitindo uma maior interação por parte dos programadores e administradores; (2) Perseus, um projeto cuja maior preocupação é a segurança, propondo uma arquitetura baseada em *microkernel*; e (3) Plex86, uma máquina virtual feita especialmente para executar sistemas Linux.

Na terceira parte, sobre VMM do tipo II, são apresentadas seis máquinas: (1) coLinux, permite a execução de Linux no Windows, com os dois sistemas com acesso privilegiado, usando o conceito de corrotina; (2) FAUmachine, máquina virtual Linux para injeção de falhas, controle de experimentos e teste; (3) Mac-on-Linux, para executar sistemas operacionais para PowerPC num Linux/ppc; (4) Microsoft Virtual PC, máquina virtual feita para executar sistemas Windows; (5) UML, um porte do *kernel* do Linux para que ele possa ser executado em outro Linux, uma das mais importantes implementações; e (6) VMware, uma linha de produtos para virtualização da arquitetura x86, outra implementação muito importante;

Na quarta seção, que tratou de paravirtualização, são apresentadas duas máquinas: (1) Denali, implementa domínios protegidos, permitindo a execução de centenas, até milhares de sistemas; e (2) Xen, um gerenciador de recursos de alto desempenho, que tenta provocar pouca sobrecarga na virtualização, atualmente ela tem tomado posição de destaque.

Na penúltima parte, foi apresentada a Máquina Virtual Java, que implementa uma máquina abstrata, num processo semelhante a um emulador.

E, por fim, três sistemas que propõem a criação de ambientes de execução de sistemas, todos eles são semelhantes, mas implementados para *kernels* de sistemas diferentes: (1) FreeBSD Jail; (2) Linux-VServer; e (3) Solaris Zones.

Capítulo 4

Segurança com máquinas virtuais

Neste capítulo, será discutida a problemática da segurança nos sistemas operacionais, assim como as aplicações existentes para garantí-la. Atenta-se também para o fato de que essas são ineficazes, sendo necessário, portanto, que novas técnicas sejam utilizadas. Aqui será proposta a idéia do uso das máquinas virtuais, que provêm o isolamento necessário para garantir a segurança requerida.

4.1 Introdução

A principal preocupação de qualquer organização que se utilize de um parque computacional conectado à Internet consiste na segurança das informações que circulam em sua rede local e entre a rede local e a Internet. A obtenção ou destruição dessas informações por terceiros pode causar enormes prejuízos às organizações comprometidas pelo ataque; isso as têm levado a investir cada vez mais em segurança, tanto das informações que trafegam na rede, quanto das informações armazenadas em seus servidores e estações de trabalho [NdG02]. A ânsia de resolver esses e outros problemas trouxe o interesse em sistemas que possam garantir a autenticidade, integridade, privacidade, anonimato e confiabilidade [PRS+01].

Existe à disposição um grande número de soluções para garantir a segurança das informações trafegando pela rede: cifragem, chaves públicas e privadas, assinaturas digitais, autenticação de usuários, tunelamento entre redes locais, entre outros [GS96]. Essas soluções visam proteger o canal de comunicação mas, apesar de necessárias, não são suficientes para proteger as informações transferidas. Este fato se deve à possibilidade de se comprometer um dos computadores envolvidos na comunicação, desprotegendo desse modo não somente as informações em trânsito, mas também as contidas na máquina invadida. Além disso, algumas aplicações, como os navegadores, tratam de dados complexos, como páginas *web*, que vêm de lugares não confiáveis [PRS+01].

Por sua vez, os sistemas operacionais modernos também possuem variados mecanismos que procuram garantir a sua segurança e estes agrupam-se em dois conjuntos principais: autenticação de usuário e controle de acesso [SGG01]. O primeiro consiste em um variado conjunto de técnicas, tanto de software quanto de hardware, com o objetivo de garantir que o usuário utilizando os recursos do computador seja de fato quem ele diz ser. Dentre essas técnicas cita-se senhas, senhas *one-time*, cartões inteligentes, biometria, etc. Mecanismos de controle de acesso, por sua vez, consistem no controle da interação entre o sistema operacional as aplicações em execução e os recursos em uso (arquivos, impressoras, etc.) visando proteger os mesmos contra leitura, escrita e/ou execução por parte de usuários não autorizados. Desta forma, mesmo que um atacante obtenha acesso à conta de algum usuário, estará restrito às informações que o usuário em questão teria acesso.

Infelizmente, toda segurança provida pelos sistemas operacionais modernos é obsoleta, sendo baseada em paradigmas de segurança presentes nos primeiros sistemas operacionais, que tinham por objetivo a simplicidade e não uma forte segurança. Desse modo, boa parte da segurança fica sob a responsabilidade das aplicações e uma vez comprometida uma aplicação, o sistema operacional não dispõe de nenhum mecanismo para garantir que as demais aplicações não possam ser atingidas [PRS+01]. Novos paradigmas de segurança já foram estudados, como os apresentados em [LSM+98], e já existem inclusive sistemas operacionais no meio acadêmico que se enquadram nos novos paradigmas. Entretanto, esses sistemas ainda devem demorar a atingir o mercado, dado o volume de aplicações existentes que só podem ser utilizadas nos sistemas operacionais comerciais.

A maioria dos usuários não são especialistas em administração ou segurança de computadores e nem é esperado que eles sejam. É difícil até para um especialista saber se determinada modificação, atualização ou instalação pode ter efeitos colaterais desastrosos. E, ainda assim, mesmo com o sistema operacional configurado perfeitamente, é possível burlar toda essa segurança se for permitido, por exemplo, para alguma aplicação o acesso direto ao hardware[1] [PRS+01].

Unido a isso, os freqüentes ataques sofridos ao redor do mundo demonstram claramente que o modelo de garantir a segurança a partir das aplicações está sendo insuficiente e que soluções mais eficientes baseadas nos sistemas operacionais típicos de produção precisam ser postas em prática. Neste trabalho serão apresentadas soluções de segurança utilizando máquinas virtuais, que podem ser enquadradas nos novos paradigmas de segurança, utilizando os sistemas operacionais típicos de produção como base de instalação.

[1] se alguma aplicação, por exemplo, tiver acesso ao controlador de disco, ela pode fazer acesso a qualquer região de memória, via DMA, sem o conhecimento do sistema operacional.

4.2 Máquinas virtuais e segurança

Antes de propor soluções de segurança utilizando máquinas virtuais, faz-se necessário compreender qual a falha de segurança dos sistemas operacionais modernos que se pretende cobrir.

Nesses sistemas cada usuário está associado a um domínio, isto é, a um conjunto de relações entre arquivos e permissões, que define quais os tipos de acessos que podem ser realizados pelo usuário; desse modo, restringindo o acesso a arquivos para os quais não tenha autorização. Entretanto, os sistemas operacionais dispõem de um usuário especial, o administrador (ou super-usuário), que não somente possui acesso irrestrito a todos os arquivos e aplicações, mas também é o único com acesso a operações privilegiadas de hardware.

Outra característica presente nos sistemas operacionais é que as aplicações obtêm acesso ao mesmo domínio que o usuário que comandou sua execução. Em decorrência deste fato, caso a aplicação possua vulnerabilidades que possam ser exploradas, o atacante terá acesso ao mesmo domínio que o usuário, comprometendo toda segurança associada a ele. Como existem aplicações que, por questões de implementação, precisam ser executadas no domínio do administrador, o comprometimento destas expõe inteiramente o sistema operacional ao atacante.

Entretanto, depender somente das aplicações para garantir a segurança do sistema operacional é ineficaz em termos de segurança; mesmo uma aplicação relativamente inofensiva pode conter falhas não descobertas por vários anos, as quais poderiam ser exploradas por um atacante que delas tomasse conhecimento [GS96]. Como apresentado em [LSM+98], depender das aplicações para promover a segurança é uma suposição falsa, e os sistemas operacionais modernos não possuem os recursos de segurança necessários para isolar as aplicações que porventura venham a ser comprometidas.

A solução, portanto, é suprir o sistema operacional com algum método que permita restringir as aplicações a um subdomínio, de forma que seu comprometimento não possa se alastrar pelas demais aplicações. É importante frisar que, com isso, não serão eliminadas as vulnerabilidades presentes nas aplicações. Elas ainda poderão ser exploradas por um atacante, porém com implicações bem menores para a segurança do sistema.

Por meio do uso de máquinas virtuais, é possível estabelecer essas restrições, criando o isolamento necessário para garantir a segurança do sistema operacional. Como apresentado nos capítulos anteriores, as chamadas de operações privilegiadas realizadas pelo sistema operacional convidado são convertidas pelo VMM em operações não privilegiadas. Desta forma, a máquina virtual não precisa ser executada a partir do domínio do administrador, isolando as operações privilegiadas.

Adicionalmente, a máquina virtual estará sendo executada em um ambiente mais

restrito do que uma aplicação comum do usuário. A partir de um *shell*, por exemplo, é possível tentar explorar alguma falha presente nas chamadas de sistema, possibilitando a obtenção de privilégios de super-usuário. O mesmo não ocorre com a máquina virtual, pois as chamadas de sistema realizadas a partir desta serão primeiramente interpretadas pela VMM e convertidas em chamadas da API do hospedeiro.

A máquina virtual, portanto, cria uma dupla camada de isolamento entre as aplicações que estarão sendo executadas no sistema operacional convidado e o sistema operacional hospedeiro. Assim, se um atacante obter sucesso em subverter a aplicação e dominar o sistema convidado, terá ainda que comprometer o VMM e, em seguida, obter privilégios de administrador a partir de uma conta de usuário não privilegiado, para enfim atingir a máquina real.

Um ponto importante a se frisar é que não basta simplesmente utilizar uma máquina virtual como sistema de trabalho, pois um sistema operacional executado sobre uma máquina virtual estará sujeito às mesmas vulnerabilidades a que estaria sujeito sobre a máquina real.

Além de não poder atingir o sistema operacional hospedeiro a partir da máquina virtual, outra característica necessária ao uso de máquinas virtuais para isolar operações é o isolamento entre máquinas virtuais. Se, a partir de uma máquina virtual for possível obter acesso direto às demais máquinas virtuais em execução, o objetivo final de isolar aplicações efetivamente não é alcançado. Todavia, os sistemas operacionais já garantem algum isolamento entre aplicações de usuário—especialmente se estes forem distintos—e, por conseguinte, das máquinas virtuais, impedindo a manipulação de informações alheias à aplicação. A única exceção é o usuário administrador, mas como já apresentado, as máquinas virtuais não estarão associadas a esse domínio privilegiado.

4.3 Aplicações de segurança com máquinas virtuais

A garantia de isolamento entre as máquinas virtuais e a máquina real é a base para as soluções propostas. Apesar de não existirem estudos estabelecendo o nível real de isolamento, a inexistência de casos registrados de falhas de isolamento em máquinas virtuais para os exemplos citados, permitem um grau de confiança superior ao apresentado pelo atual cenário de aplicações e sistemas operacionais.

Nesta seção, serão descritas técnicas a serem utilizadas visando a segurança das aplicações. A primeira delas é aplicável às estações de trabalho, a segunda é aplicável à detecção e aprendizado de novos ataques a sistemas operacionais e aplicações e as duas restantes focam o aumento de segurança de servidores.

4.3.1 Estações de trabalho: execução de aplicações inseguras

Existe uma grande preocupação das organizações em proteger os servidores de suas redes, pois estes são os verdadeiros concentradores de informação da empresa. De acordo com [CZ95], quando alguém se conecta à Internet, está colocando três coisas em risco: seus dados (as informações mantidas nos computadores), seus recursos (os próprios computadores) e sua reputação. Entretanto, é importante ressaltar que simplesmente proteger os servidores não é suficiente para proteger a rede como um todo. As estações de trabalho, que de modo geral são mais inseguras, têm acesso a pelo menos uma parte dessas informações, tornando-se alvos potenciais.

Considerando essa insegurança, já existe um variado conjunto de configurações de redes que visam isolar as máquinas internas dos perigos provenientes da Internet, através do uso de *firewalls* (no sentido amplo, como visto em [CZ95, GS96, NdG02]), baseado no pressuposto de que as estações de trabalho serão inatingíveis. Tentativas de ataque às estações serão barradas pelo *firewall*, inclusive algumas das contidas em *e-mails* e páginas *web*, garantindo que apenas um pequeno conjunto de programas necessite ser atualizado constantemente, simplificando o processo administrativo. Entretanto, este modelo de segurança só é efetivo contra vulnerabilidades conhecidas, ficando as máquinas internas vulneráveis, portanto necessitando de segurança adicional.

Não basta, entretanto, instruir os usuários da organização a não abrirem anexos de origem duvidosa. Técnicas de engenharia social e vírus que exploram falhas dos sistemas de *e-mail* estão evoluindo e tornando mais difícil a sua contenção. Somado a isso, os sistemas de *plug-ins* e códigos maliciosos em geral permitem ao atacante executar código próprio na máquina do usuário, o que permite estabelecer controle parcial ou completo sobre a conta do usuário ou sobre a máquina inteira, especialmente nas plataformas Windows. Nos dias de hoje o exemplo mais comum de código de controle externo é o *key logger*, que se especializou em capturar eventos de teclado e pequenos snapshots ao redor do cursor e de maneira seletiva, só o fazendo durante acesso *web* a *sites* selecionados de organizações financeiras, enviando posteriormente os dados coletados por *e-mail* ao atacante, sem que o usuário legítimo perceba qualquer atividade.

Partindo da idéia de isolamento seguro apresentado em [LSM+98] e com a premissa de segurança garantida pelo uso de máquinas virtuais, é proposto o uso de máquinas virtuais em estações de trabalho. A máquina real, neste modelo, seria usada somente para as aplicações referentes à atividade da organização. A aplicação de leitura de *e-mails*, o acesso à Internet e outras operações inseguras devem usar máquinas virtuais diversas, de forma que, uma vez comprometida uma máquina virtual por causa de um *e-mail* ou de uma página maliciosa aberta em um navegador, esse comprometimento fica restrito somente a máquina virtual em questão, enquanto as demais máquinas (especialmente a real) continuam íntegras. Da mesma forma, pode-se executar aplicações que requerem

um cuidado extra, tais como aplicações bancárias, em máquinas virtuais próprias.

4.3.2 *Honeypots* e *honeynets*

Garantida a segurança das estações de trabalho, o próximo tópico de análise aborda o objetivo final dos atacantes: os servidores, cujas informações centralizadas podem ser valiosas para o atacante. De um modo geral, os servidores concentram várias aplicações, muitas das quais críticas à segurança de informações como sistemas de arquivos, servidores de internet, servidores de *e-mail*, entre outros; apresentando assim um número muito maior de vulnerabilidades que podem ser exploradas.

No intuito de analisar as técnicas de ataques, foi desenvolvido o conceito de *Honeypots* [Bar02], máquinas com sistemas operacionais típicos de produção, com as aplicações de interesse de estudo, cuja finalidade é estar aberta para ataques externos, permitindo que os mesmos sejam estudados. Com isso, é possível obter informações a respeito dos métodos, ferramentas e motivações dos invasores. Uma *honeynet* nada mais é do que a extensão do conceito de *honeypot* para o ambiente de rede.

Entretanto, uma grave limitação é que toda vez que um *honeypot* sofre uma invasão bem sucedida, a máquina tem que ser retirada da rede para estudo e, em seguida, reinstalada completamente, antes que possa ser exposta a novos ataques. Neste ponto, torna-se interessante o uso de máquinas virtuais, graças à capacidade de realizar o *backup* do sistema virtual, na máquina real, apenas pela cópia do arquivo que comporta o sistema de arquivos da máquina virtual, correspondendo virtualmente ao seu disco rígido. Separado o arquivo anterior para análise, basta restaurar o sistema operacional original pronto para ser atacado e colocá-lo novamente disponível para invasão.

Adicionalmente, como é possível executar várias máquinas virtuais em uma mesma máquina real, é possível criar toda uma *honeynet* utilizando somente um computador [Pro03].

4.3.3 Servidor consolidado

O uso de *honeypots* para identificar e corrigir problemas de segurança explorados por atacantes é uma técnica reativa, pois somente após o primeiro ataque bem sucedido é que se terá conhecimento das formas com que ele poderá ser evitado. Entretanto, em paralelo vários sistemas reais estarão expostos e podem ser atacados antes dos *honeypots*. Dado que pode demorar até que as correções para eventuais falhas nas aplicações estejam disponíveis, faz-se necessário minimizar o efeito do ataque [GS96].

É fácil perceber que quanto mais aplicações servidoras estiverem sendo executadas em uma mesma máquina, maior o número de falhas potenciais podem ser exploradas e, por conseguinte, menor será a segurança do sistema operacional como um todo. Assim, o

ideal seria disponibilizar uma máquina para cada conjunto mínimo de aplicações correlacionadas, de forma que a possibilidade de invasão seja reduzida. Entretanto, se fossem utilizadas várias máquinas reais para tal o custo seria elevado, tanto em termos de hardware quanto de manutenção administrativa. O baixo custo advindo do uso de máquinas virtuais torna a abordagem bastante atraente.

Para garantir a segurança do servidor faz-se necessária uma análise em duas etapas. A primeira consiste em garantir a segurança das aplicações a serem executadas. Como elas estarão isoladas em máquinas virtuais diferentes, ao ocorrer o comprometimento de uma aplicação, o acesso que poderá ser obtido por meio dela se restringirá às aplicações contidas na máquina virtual, que será, como visto há pouco, um conjunto mínimo. O atacante não poderá, a partir da máquina virtual comprometida, atingir a máquina real ou qualquer uma das outras máquinas virtuais.

A segunda etapa consiste em garantir a segurança do sistema operacional real que está suportando as máquinas virtuais. O comprometimento desse sistema permitiria o acesso irrestrito a todas as máquinas virtuais e toda a segurança analisada na primeira etapa seria inútil. Para evitar possíveis comprometimentos, o sistema operacional real deverá realizar apenas duas tarefas: servir de base para as máquinas virtuais—não sendo aplicações de rede, não podem ser atacadas externamente—e de *gate*[2], conforme proposto em [GS96].

4.3.4 *Firewalls*

Esta aplicação de máquinas virtuais à segurança de redes será focada na ligação entre a rede interna e a Internet. Uma das principais ferramentas de segurança em redes consiste no *firewall* [NdG02, GS96]. Entende-se por *firewall* um conjunto de máquinas que provém segurança e interligam, como único ponto de acesso, várias redes distintas, por escrutínio do tráfego entre as regiões de confinamento, que passa todo por ele. Sua utilização vem se tornando cada vez mais necessária, em função do crescimento do uso da Internet com sua intrínseca insegurança.

Existem diversas topologias de *firewalls*, apresentadas em [NdG02], e elas são compostas por três elementos principais: filtros, DMZ's e máquinas *bastion*. Um exemplo de topologia pode ser visto na Figura 4.1. Os filtros são responsáveis por limitar os tipos de conexões possíveis entre as redes, selecionando os pacotes permitidos segundo regras bem definidas. São usados para criar a rede denominada DMZ (sigla proveniente do termo "zona desmilitarizada", *Demilitarized Zone*). A DMZ tem por objetivo isolar a rede interna e a Internet de conexões diretas, forçando que essas conexões sejam realizadas

[2]máquinas que ficam no perímetro do *firewall*, recebendo conexões externas e tratando-as adequadamente; as máquinas *gate* são especialmente configuradas pensando na sua segurança.

através do *bastion*, uma máquina com segurança reforçada, de forma que somente esta esteja acessível a eventuais ataques externos.

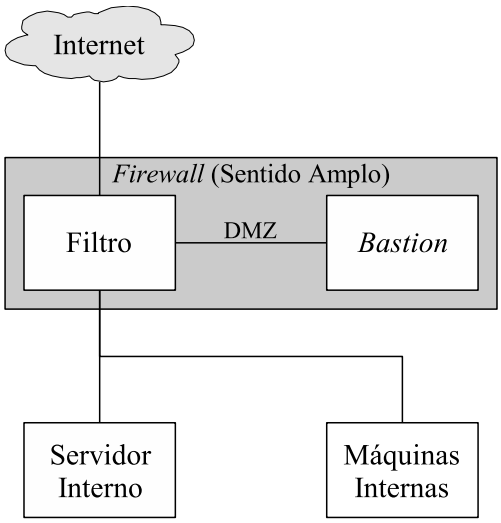

Figura 4.1: Diagrama de uma das topologias de um Firewall.

A montagem de um *firewall* utilizando máquinas virtuais consiste em apenas aplicar essa arquitetura de *firewalls* ao modelo apresentado na Seção 4.3.3, com apenas uma ressalva: uma vez que filtros de pacotes apresentam implementação simples, e portanto são considerados seguros, o papel de filtro pode ser desempenhado pela própria máquina hospedeira, sem incorrer em redução da segurança desta solução.

4.4 Conclusão

Foi discutida neste capítulo a importância de se prover segurança aos sistemas operacionais, garantindo assim a sua utilização por grandes empresas de forma mais confiável. Mostrou-se que a aplicação das máquinas virtuais nesse sentido reduz os custos com segurança, tornando-se uma opção extremamente atraente.

Além disso, foram apresentadas as técnicas a serem utilizadas visando à segurança das aplicações: às estações de trabalho, onde se concentram a maioria das informações de uma empresa; à detecção de novos ataques, utilizando *honeypots* e *honeynets* (esse último aplicado no ambiente de rede), e mostrando como essas aplicações podem ser facilitadas pelo uso das máquinas virtuais; à consolidação de servidores, buscando otimizar a utilização dos recursos de hardware das máquinas, ao mesmo tempo em que serve de ajuda para reduzir os custos de espaço e de infra-estrutura; e, por último, por meio de *firewalls*.

Capítulo 5

Estudo de caso

O objetivo deste capítulo é mostrar dois estudos de caso realizados, um envolvendo consolidação de servidores e outro sobre uma proposta de migração de redes IPv4 para IPv6.

No primeiro caso, a estrutura da rede era baseada em três máquinas: um servidor externo, um servidor interno e um roteador. O objetivo era converter essas máquinas numa única máquina real, hospedando várias máquinas virtuais (uma para cada conjunto correlacionado de serviços). Porém, devido a problemas, os planos tiveram que ser adaptados, fazendo com que a migração não pudesse ser feita para apenas uma máquina.

O segundo caso é o desenvolvimento de uma proposta para migração de redes IPv4 para redes IPv6 utilizando máquinas virtuais, chamado MV6. O MV6 herda muitas das vantagens instrínsecas ao uso de máquinas virtuais, se apresentando como uma solução importante, permitindo que a implantação de redes IPv6 possa ser feita de maneira segura e gradual, além de garantir compatibilidade com programas feitos para IPv4.

5.1 Servidor de alta demanda com UML

As propostas do capítulo anterior foram implementadas em um laboratório de redes[1], cuja topologia consistia de um *bastion*, um servidor interno e um filtro isolando a DMZ da rede interna e da Internet (Figura 5.1). Essa topologia é uma variação da arquitetura *screened subnet*, como descrito em [NdG02]. O objetivo final era substituir o esquema apresentado por uma máquina real fazendo o papel de filtro, de hospedeiro das máquinas virtuais e de servidor de arquivos.

É importante observar que o uso da máquina real como servidor de arquivos não estará contradizendo o pressuposto de segurança apresentado na Seção 4.3.3. Esse serviço adicional fez-se necessário pelo fato de que o acesso a disco através da máquina virtual

[1] segunda metade de 2003 até novembro de 2004.

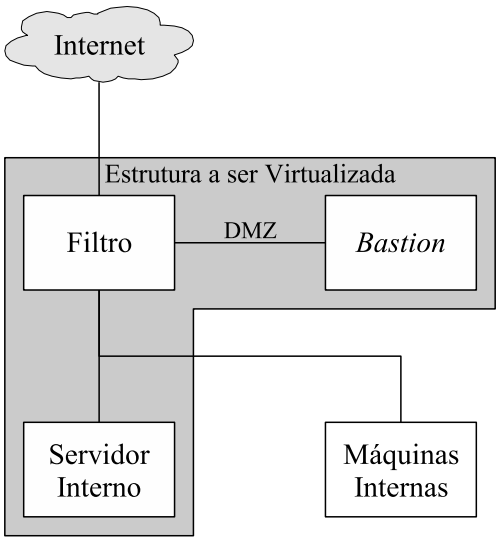

Figura 5.1: Topologia básica da rede antes da inclusão de máquinas virtuais.

é lento e por questões de desempenho decidiu-se utilizar um servidor na máquina real, onde o desempenho seria igual ao da topologia original. Entretanto, através do filtro a ser implantado na máquina real, será possível bloquear qualquer acesso partindo da Internet ao serviço, evitando que o mesmo seja explorado como ponte de entrada para invadir o sistema hospedeiro.

As máquinas da rede original utilizavam Linux como sistema operacional e o uso da máquina virtual UML tornou-se a escolha natural, visto que na época ela era a opção mais estável e madura.

A adaptação da estrutura para máquina virtual foi dividida em etapas. A primeira etapa consistiria em configurar uma máquina real como filtro e transformar a DMZ em uma rede virtual, simplesmente convertendo a máquina *bastion* em uma máquina virtual, mantendo as configurações. A segunda etapa consistiria em agregar o servidor interno a outra máquina virtual, de maneira análoga ao *bastion*. Somente na terceira etapa os serviços seriam separados em máquinas virtuais distintas, aumentando o nível de isolamento. A divisão, que poderia ter sido realizada em paralelo com as duas primeiras etapas, foi postergada visando determinar eventuais conflitos entre serviços, que poderiam incorrer no uso de uma máquina real para os mesmos.

A primeira etapa da implantação contou com problemas não previsíveis. O módulo responsável pela interface de rede virtual da máquina real deixava todas as interfaces de rede do sistema operacional anfitrião sem funcionamento. O problema, entretanto, não estava ligado à máquina virtual e sim à inclusão do suporte a IPv6 no *kernel* do sistema Linux. Este problema foi resolvido primeiramente com a desativação do suporte a IPv6 e, depois, com uma nova versão do *kernel*, com problema corrigido. Entretanto, existiam

inúmeros problemas na implementação, tanto no UML quanto no *kernel* do Linux, que atrasaram consideravelmente o andamento das experiências.

Após a primeira etapa, surgiu a necessidade de adiantar a etapa de divisão da máquina virtual, pois esta possuía servidores *web/ftp/rsync* com alta demanda, atingindo mais de 500 conexões simultâneas[2]. A lentidão era sensível, devido ao constante acesso a disco, o que forçou o isolamento provisório destes serviços em outra máquina real, ainda utilizando uma máquina virtual para executá-los.

Além da lentidão, esse excesso de atividade causava uma certa instabilidade no sistema da máquina real. Não foi possível isolar o problema e descobrir a sua causa, pois a máquina apresentava a instabilidade em momentos completamente imprevisíveis, podendo passar desde dias até poucas horas em funcionamento normal. Felizmente, com o passar do tempo, com as novas versões do *kernel* do Linux e do programa UML, a máquina se apresentava mais estável. Este fato faz acreditar que o problema era ligado a algum tipo de erro no código ou do UML ou do *kernel* do Linux, sendo este último o mais provável. Nessa época o *kernel* do Linux estava na versão 2.4, mas prestes a sair a versão 2.6 e muitas funcionalidades desta nova versão estavam sendo portadas para a versão antiga. O número de conexões e quantidade de dados transmitidos levam à suspeita de que o módulo responsável pela interface de rede virtual era a causadora da instabilidade (que também se aplicou no problema com o suporte a IPv6).

Neste ponto (final de 2003) o cenário era o seguinte:

imperio (máquina real): roteadora, filtragem de pacotes, servidor de arquivos para os diretórios de usuários e hospedeira da bastion;

bastion (máquina virtual): servidor DNS, servidor de e-mail, cache de web e responsável pela atualização de algumas partes do diretório público;

argonath (máquina real): hospedeira da seraph, servidor de arquivos do diretório público e responsável pela sincronia de algumas partes do diretório público;

seraph (máquina virtual): servidor HTTP, servidor FTP e servidor RSYNC.

No início do ano de 2004, um mês após o lançamento oficial do *kernel* versão 2.6, o UML encontrava-se com as seguintes versões e seus modos de execução:

- *kernel* hospedeiro versão 2.4 e UML com *kernel* 2.4 roda em modo SKAS e em modo TT;
- *kernel* hospedeiro versão 2.6 e UML com *kernel* 2.4 roda somente em modo TT;

[2]servidor espelho (*mirror*) das distribuições: debian, fedora, gentoo, kurumin e redhat; além de eventuais versões de outras distribuições.

- *kernel* hospedeiro versão 2.4 e UML com *kernel* 2.6 roda somente em modo SKAS;
- *kernel* hospedeiro versão 2.6 e UML com *kernel* 2.6 não funciona[3].

Dado este cenário, os sistemas continuaram com a versão 2.4 do *kernel* do Linux e o conjunto foi se tornando cada vez mais estável.

Com o passar do tempo, a utilização do servidor só aumentou, impossibilitando a reunificação, mudando completamente os planos iniciais. A única mudança foi a divisão da máquina servidora de *web/ftp/rsync*: o servidor *web* foi para outra máquina virtual na mesma máquina hospedeira. E o conjunto ficou assim por cerca de seis meses, apresentando problemas pontuais.

A natureza dos erros indicava as fraquezas da utilização dessa implementação de máquina virtual num ambiente como este. Quando o servidor passava por alguma exigência mais forte (normalmente ligado ao lançamento de alguma nova versão de alguma distribuição de Linux que hospedávamos) era comum ele travar de tempos em tempos. Todas as tentativas administrativas nas máquinas só resolviam paliativamente os problemas e quando o servidor voltava, a sobrecarga dele permanecia, o que novamente ocasionava a sua queda. Nestes casos, somente com a atualização de um dos *kernels* para uma versão mais recente, na esperança de obter uma melhor estabilidade, é que se conseguia resolvê-los.

Eis a transcrição de um trecho da mensagem mais comum que o *kernel* da máquina hospedeira mostrava no momento em que ela travava:

```
BUG at panic.c:162!
INVALID OPERAND:0000

Kernel Panic: Attempted to kill the idle task!
In idle taks - not syncing
```

Vale observar que enquanto esta máquina passava por esses problemas, a outra máquina real e sua máquina virtual funcionavam corretamente, não apresentando maiores problemas. Este fato indica que a migração dos serviços de alta demanda foi uma escolha acertada e que eles eram os causadores dos problemas enfrentados.

Esse tipo de uso para máquinas virtuais, em servidor de alta demanda, se mostrou um uso além da capacidade da implementação do UML para a época. Uma carga, que até para alguns servidores hospedados em máquinas reais, causaria algum grau de lentidão, nessa máquina virtual gerava problemas que se propagavam para a máquina real. Máquinas

[3]pois não existe o *patch* do SKAS para o *kernel* hospedeiro, e o UML com *kernel* 2.6 não roda em modo TT.

virtuais, como era de se esperar, sofrem com aplicações que fazem uso de instruções privilegiadas, pois, como visto no Capítulo 2, elas tem que ter seu uso mascarado pela máquina virtual. E este é o caso deste servidor de alta demanda, que requer muita leitura de arquivos, executando muitas instruções de E/S, que são privilegiadas.

Desta forma, pode-se comprovar que o uso de máquinas virtuais foi adequado para o caso de um servidor com as características da máquina *imperio* (filtro e hospedeira de uma máquina virtual de pouca carga) e foi inadequado para um servidor que exige muito acesso a disco, que é o caso da máquina *argonath* (hospedeira de uma máquina virtual cuja principal característica é acesso intenso a disco).

5.2 Migração IPv6 - Projeto MV6

5.2.1 Introdução

O IPv4, versão atual do protocolo IP, não antecipou e, conseqüentemente, não contempla as necessidades atuais da Internet. O problema mais evidente é o crescimento exponencial da Internet e resultante ameaça de exaustão do espaço de endereçamento IPv4. Logo, técnicas como NAT (*Network Address Translation*) e CIDR (*Classless Inter-Domain Routing*) foram desenvolvidas para resolver paliativamente este problema.

Aliado ao problema de endereçamento IP, a necessidade de configuração simplificada, a ausência de mecanismos de segurança mais robustos e a necessidade de suporte melhorado para entrega de dados em tempo-real, estimularam o IETF (*Internet Engineering Task Force*) a desenvolver um conjunto de protocolos e padrões conhecido como IP versão 6 (IPv6). Conseqüentemente, o protocolo IPv6, nos últimos anos, vem adquirindo um grau de importância cada vez maior mundialmente, principalmente em países asiáticos e europeus, cuja falta de endereços IPv4 já é bastante evidente.

Como o conceito de telecomunicações está cada vez mais voltado à convergência das redes e ao "mundo IP" e tendo em vista que o protocolo IPv4 é o principal ator nas infra-estruturas das redes atuais, é bastante coerente que a migração para o protocolo IPv6 seja realizada de uma forma metódica e progressiva.

Em decorrência desta necessidade, o IETF criou o Grupo de Trabalho NGTRANS (*Next Generation Transition*), cujo objetivo era desenvolver ferramentas e mecanismos que permitam que redes e máquinas IPv4 migrem suavemente para IPv6, já que os cabeçalhos dos protocolos IPv4 e IPv6 não são interoperáveis entre si. Durante o período de dois anos, o Grupo NGTRANS criou e disponibilizou inúmeros esboços (*drafts*), que estabeleceram uma miríade de mecanismos para a integração com IPv6, que vão desde um simples tunelamento a complexos mecanismos como Teredo[4] [Hui05]. Entretanto, estes vários

[4]encapsula os pacotes IPv6 através do protocolo UDP numa rede IPv4.

mecanismos têm desvantagens que não os recomendam em cenários onde: (1) aplicações críticas rodam em IPv4 e precisam iniciar experiência com redes IPv6; e (2) redes IPv6 com seus programas e aplicações não portadas para IPv6 que precisam rodar.

5.2.2 Mecanismos para a transição de redes IPv4 para redes IPv6

Os mecanismos de transição das redes IPv4 atuais para as novas redes IPv6 são divididos em três categorias principais: pilha dupla (*dual-stack*), tunelamento e tradução [SC05].

O mecanismo de pilha dupla permite que elementos de rede (máquinas, roteadores, etc.) implementem e executem as pilhas de protocolos IPv4 e IPv6 simultaneamente, criando assim uma rede IPv6 paralela a infra-estrutura IPv4 já existente. Elementos de rede implementando pilha dupla terão dois endereços de rede (um para cada protocolo, para cada interface). A Figura 5.2 ilustra tal mecanismo [GN00].

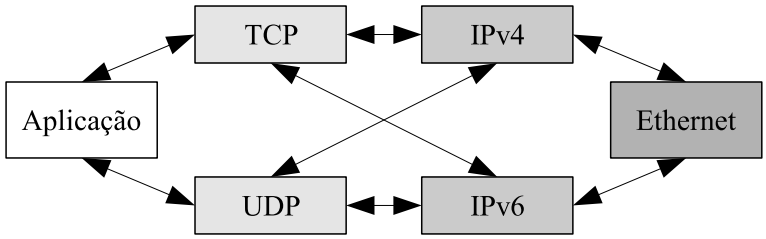

Figura 5.2: Mecanismo de transição pilha dupla.

Pilha dupla é um mecanismo flexível e fácil de usar. Contudo, ele tem as seguintes desvantagens: cada máquina precisa ter as duas pilhas rodando separadamente, o que demanda poder de processamento adicional e memória, assim como tabelas de roteamento para os dois protocolos. Um resolvedor DNS também precisa ser capaz de resolver ambos os tipos de endereços. Geralmente, todas as aplicações rodando na máquina pilha dupla devem ser capazes de determinar se a máquina está se comunicando com uma máquina IPv4 ou IPv6.

O mecanismo de tunelamento permite que duas redes IPv6 distintas, sejam nativas ou baseadas em pilha dupla, se comuniquem utilizando uma rede IPv4 como se fosse um enlace conectando o roteador de saída de uma rede ao roteador de entrada da outra. Existem tunelamentos que tratam a rede IPv4 como um enlace virtual ponto-a-ponto e também aqueles que tratam como um enlace virtual NBMA (*Non Broadcast MultiAccess*). Exemplos deste mecanismo são: tunelamento manual [GN00], tunelamento automático 6to4 [CM01], tunelamento automático ISATAP (*Intra-Site Automatic Tunnel Address Translation*) [TGTT05], dentre outros.

Tunelamento tem as seguintes desvantagens: a carga adicional colocada no roteador, já que cada ponto de entrada e de saída precisa de tempo e processamento para encapsular e desencapsular pacotes; e os pontos de entrada e saída representam pontos únicos de falha. A solução de problemas se torna mais complexa ao entrar em detalhes de *hop count* e MTU (*Maximum Transmission Unit*), assim como problemas de fragmentação. Um exemplo é mostrado na Figura 5.3:

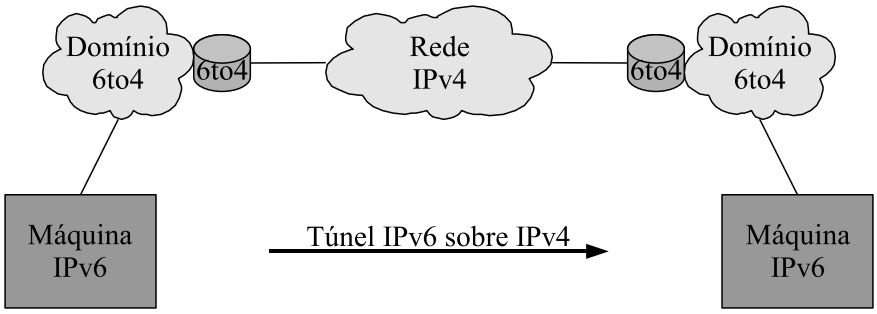

Figura 5.3: Mecanismo de transição tunelamento.

Por fim, o mecanismo de tradução é utilizado para comunicação entre nós IPv6 e IPv4. Exemplos de tal mecanismo são o SIIT (*Stateless IP/ICMP Translation*), que não mantém o estado das conexões criadas entre origem e destino, e o NAT-PT (*Network Address Translation - Protocol Translation*), que mantém os mapeamentos entre endereços IPv4 e IPv6 das conexões. O mecanismo de tradução apresenta como vantagem permitir que máquinas IPv6 se comuniquem diretamente com máquinas IPv4, porém suas desvantagens são: não suporta características avançadas de IPv6 como segurança fim-a-fim, impõe limitações à topologia da rede, pois as respostas de qualquer mensagem enviada pelo roteador de tradução devem retornar para o mesmo roteador de tradução, além do roteador de tradução ser um ponto único de falha. Este mecanismo é recomendável somente quando nenhum outro mecanismo for possível e deve ser visto como uma solução temporária até outra qualquer ser implementada.

5.2.3 Mecanismo de transição proposto

O mecanismo de transição proposto, chamado MV6 [dCdSdG04], é baseado na utilização de máquinas virtuais para a criação de um ambiente onde os dois protocolos IPv4 e IPv6 coexistam de forma independente. A idéia básica é formar duas redes independentes e paralelas, como na Figura 5.4, uma rodando IPv4 e a outra IPv6, sendo que a rede chamada nativa será a rede do sistema operacional hospedeiro e a rede paralela estará na máquina virtual. É pré-requisito obrigatório que estas redes rodem protocolos distintos.

As principais vantagens do uso de máquinas virtuais no novo mecanismo são: eliminar

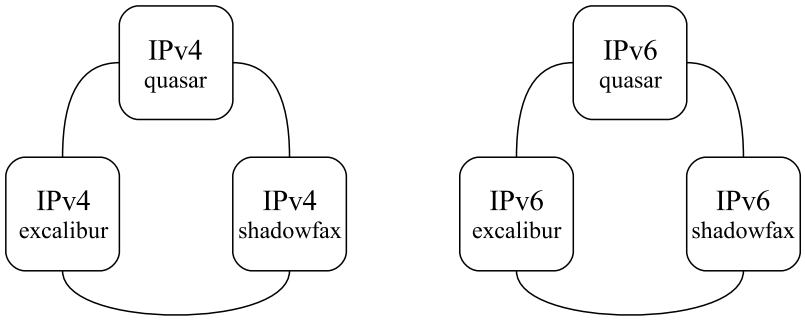

Figura 5.4: Representação da rede IPv4 e IPv6, que coexistem paralela e independentemente.

a necessidade de portar aplicações legadas, testar o novo protocolo em máquinas virtuais antes da implantação, suportar aplicações legadas durante a migração para o novo protocolo, além de minimizar o sofrimento dos usuários durante a transição de IPv4 para IPv6.

Cenários de aplicação

Dois cenários de aplicação deste mecanismo de transição foram mapeados e serão explicados em detalhes. O primeiro cenário tem o protocolo IPv4 como protocolo nativo da rede, ou seja, o IPv4 é o protocolo utilizado desde a formação da rede e com o qual os usuários estão ambientados. Além disto, nesta rede existem aplicações críticas que rodam em IPv4 e que não podem ser comprometidas por nenhum problema na rede. Entretanto, nesta rede, os administradores querem se familiarizar com os problemas advindos do uso constante de IPv6, ao mesmo tempo em que criam a cultura do novo protocolo em seus usuários, sem contudo comprometer o funcionamento da aplicação crítica IPv4. Objetivando conciliar estes requisitos contrários, é criada, via máquinas virtuais, uma rede IPv6 paralela à rede IPv4, que permite a existência da rede IPv4 e o funcionamento de suas aplicações críticas independentemente da existência e do funcionamento da rede IPv6 (Figura 5.5). Por fim, este cenário é útil quando é necessário ter contato com IPv6 sem interferência no ambiente de trabalho corrente, que se utiliza de IPv4.

O segundo cenário tem o protocolo IPv6 como protocolo nativo da rede, já que esta rede migrou para IPv6. No entanto, algumas aplicações ainda não foram portadas para o novo protocolo, precisando assim do protocolo IPv4 para funcionarem corretamente. Este cenário busca suprir esta necessidade ao criar uma rede paralela IPv4 à rede nativa IPv6, que permite a coexistência entre os dois tipos de rede passivamente. O usuário utiliza o protocolo IPv6 nativamente e quando precisa acessar uma aplicação que ainda não foi portada para IPv6, ele usa a máquina virtual com rede IPv4 para, de forma paralela, acessar a aplicação IPv4 (Figura 5.6).

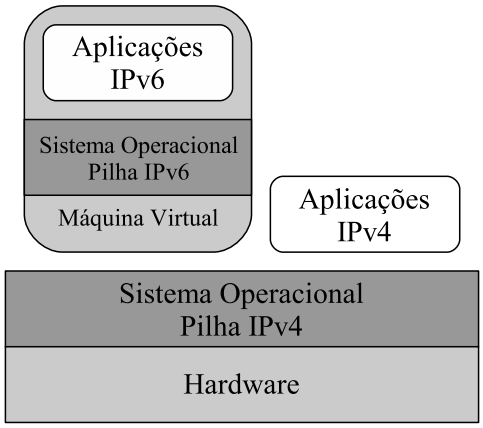

Figura 5.5: Estrutura de uma máquina virtual com pilha IPv6.

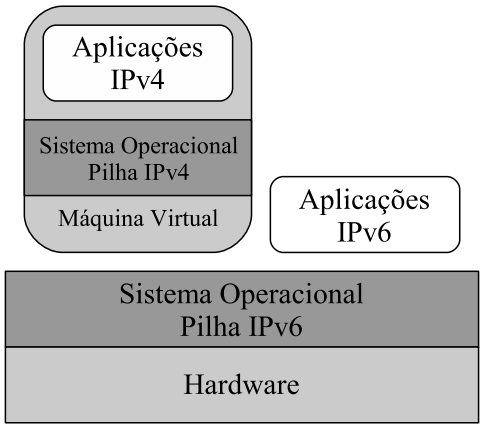

Figura 5.6: Estrutura de uma máquina virtual com pilha IPv4.

5.2.4 Resultados

Estudos realizados em laboratório de redes, que utilizava Linux como sistema operacional e a máquina virtual UML (Seção 3.3.5) para virtualização de servidores, mostraram que a implantação de máquinas virtuais como meio de migração IPv4 - IPv6 pode ser feita de forma segura e simples. Os problemas que aconteceram durante os testes não foram somente de questão técnica, mas também referentes a recente[5] implementação do suporte a IPv6 no *kernel* do sistema Linux, assim como na Seção 5.1.

Os outros tipos de problemas estão relacionados ao aprendizado, normal, exigido por qualquer outro tipo de programa. Especificamente neste caso, em como compilar corretamente um *kernel* do Linux e de que maneira e qual sistema operacional deve dever instalado.

[5]os testes foram realizados no ano de 2004.

Os testes foram realizados em duas etapas, seguindo as duas propostas de utilização do MV6: (1) a rede nativa em IPv4, e a rede de máquinas virtuais em IPv6; e (2) na situação oposta, com a rede nativa em IPv6, e a rede virtual em IPv4. Para verificar seu funcionamento, foram feitos testes com aplicações comuns de uso de rede, tais como: consulta DNS a algum provedor conhecidamente em IPv6, acesso a alguma página web, envio de e-mail, transferência de arquivos, etc. Para todos os casos, tanto em IPv4, quanto em IPv6, as aplicações funcionaram corretamente.

O mecanismo de transição proposto mostra ser uma proposta que se adequará corretamente ao processo metódico de migração da conectividade de rede IPv4 para IPv6, bem como ao funcionamento de aplicações IPv4 que não migrarão para IPv6.

Capítulo 6

Conclusão

A proposta inicial deste trabalho era entender o que são máquinas virtuais e quais as suas principais características, com o foco nas que as tornam uma importante ferramenta para a segurança de sistemas.

Para tanto, primeiramente foi feito um estudo das origens das máquinas virtuais, acompanhando a sua evolução, culminando na sua expansão para o mundo dos PCs. Foi apresentada a problemática que o não suporte nativo no hardware da arquitetura x86 apresenta, assim como as propostas de sua inclusão. Mostrou-se as classificações das máquinas virtuais, e suas principais características.

Fez-se um estudo das principais máquinas virtuais existentes, buscando apresentar a motivação de seu desenvolvimento, que acaba se refletindo nas suas características.

O próximo passo foi explicar a problemática de segurança dos sistemas operacionais atuais, que sobrecarregam as aplicações de responsabilidade de manter a segurança do sistema, além de fornecer um controle não muito flexível de nível de acesso, em que ou o usuário tem acesso total (administrador ou *root*) ou acesso super restrito (conta comum). Dado este cenário, foi proposto o uso de máquinas virtuais como uma solução elegante e de excelente qualidade. Uma das principais premissas de uma máquina virtual está no isolamento que ela impõe aos sistemas, o que resolve o primeiro problema, isolando as aplicações umas das outras, não permitindo que um problema numa máquina virtual afete o funcionamento do sistema como um todo. Uma outra característica das máquinas virtuais é a criação de um ambiente em que o nível de acesso do usuário administrador é mais restrito que o de um usuário comum da máquina real. A máquina virtual não pode fazer chamadas privilegiadas no sistema hospedeiro, além de ter a sua execução monitorada, ou seja, o ambiente de uma máquina virtual é um subdomínio de execução, mas que mantém as características funcionais de um sistema operacional em sua execução canônica.

Foram descritos alguns cenários de utilização em que máquinas virtuais são peças-chave

para a segurança das aplicações, tornando estações de trabalho mais seguras, ajudando na análise e estudo de técnicas de invasões e ataque a servidores, fechando com a consolidação de servidores, otimizando a utilização do poder computacional, sem nenhuma perda em confiabilidade.

Por fim, apresenta-se dois cenários em que foram postos em prática os conhecimentos de até então. O primeiro é um caso de consolidação de servidores, usando um cenário real, em que o servidor é de alta demanda. Durante sua implementação, foram encontrados vários problemas de estabilidade, relacionados a erros no código do *kernel* dos sistemas envolvidos. Também houve problemas de excesso de carga de utilização, explicitando a sobrecarga que a virtualização causa (em especial as instruções de E/S, que são instruções privilegiadas, exigindo tratamento especial pelo VMM). O segundo estudo foi sobre uma proposta de migração de redes IPv4 para IPv6, criando uma rede paralela, permitindo que a rede principal da organização funcione com um dos protocolos e a rede virtual com o outro. Com isso, é possível já ir se familiarizando com a problemática de usar o novo protocolo, sem influenciar ou alterar o funcionamento da rede. E, numa etapa posterior, é possível executar aplicações não portadas para IPv6, numa rede virtual IPv4 (invertendo os protocolos da rede, quando as aplicações e os testes já tiverem sido realizados).

6.1 Trabalhos futuros

A expansão natural desse trabalho está na análise das implementações de máquinas virtuais, buscando meios de burlar o seu isolamento, assim como em [GA03] em que é usado erros no espaçamento de endereços para comprometer a JVM e a máquina virtual .NET.

Da mesma forma, com a recente entrada no mercado do suporte à virtualização em hardware na arquitetura x86, esta pode apresentar problemas que podem ser explorados para comprometer a máquina virtual. Outro ponto de estudo é as modificações e adaptações necessárias aos atuais VMMs para que eles possam fazer uso da virtualização em hardware, uma análise do ganho em desempenho, assim como um comparativo mais completo entre a abordagem proposta pela AMD e Intel.

Outro ponto está em fazer testes comparativos de desempenho do MV6 com outras propostas de migração.

Referências Bibliográficas

[ade05] The Adeos Project. `http://home.gna.org/adeos/`, December 2005. (Data de Acesso).

[Alo04] Dan Aloni. Cooperative Linux. In *Linux Symposium*, volume 1, Ottawa, Ontario, Canada, July 2004.

[Bar02] Ryan C. Barnett. Monitoring VMware Honeypots. Technical report, Honeypots: Monitoring and Forensics, September 2002. `http://honeypots.sourceforge.net/monitoring\_vmware\_honeypots.html`.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *ACM Symposium on Operating Systems Principles*, oct 2003.

[boc05] The Open Source IA-32 Emulation Project. `http://bochs.sourceforge.net/`, December 2005. (Data de Acesso).

[CM01] B. Carpenter and K. Moore. Connection of IPv6 Domains via IPv4 Clouds. Technical Report RFC 3056, The Internet Society, February 2001. `http://www.ietf.org/rfc/rfc3056.txt`.

[col05] Cooperative Linux. `http://www.colinux.org/`, December 2005. (Data de Acesso).

[Con01] Connectix Corp. *The Technology of Virtual Machines*, October 2001.

[Cor05a] Intel Corp. Enhanced Virtualization on Intel Architecture-based Servers. Technical Report 304266-001, Intel Corp., February 2005.

[Cor05b] Intel Corp. Intel Virtualization Technology Specification for the IA-32 Intel Architecture. Technical Report C97063-002, Intel Corp., April 2005.

[Cor05c] Intel Corp. Intel Virtualization Technology Specification for the Intel Itanium Architecture (VT-i). Technical Report 305942-002, Intel Corp., April 2005.

[Cor05d] Intel Corporation. *IA-32 Intel Architecture Software Developer's Manual*, volume 1: Basic Architecture. Intel Corporation, June 2005.

[Cre81] R. J. Creasy. The Origin of the VM/370 Time-Sharing System. *Journal of Research and Development*, 25(5):483, 1981.

[CZ95] D. Brent Chapman and Elizabeth D. Zwicky. *Building Internet Firewalls*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1995.

[Dav05] Megan Davis. Virtual PC vs. Virtual Server: Comparing Features and Uses. Technical report, Microsoft Corp., May 2005. `http://www.microsoft.com/windowsserversystem/virtualserver/techinfo/vsv%svpc.mspx`.

[dCdSdG04] Arthur Bispo de Castro, Cleymone Ribeiro dos Santos, and Paulo Lício de Geus. MV6 - Um Mecanismo de Transição Baseado em Máquinas Virtuais. In *IV Workshop em Segurança de Sistemas Computacionais*, pages 237–248, Gramado, RS, Brasil, January 2004.

[den05] Denali: Lightweight Virtual Machines for Distributed and Networked Systems. `http://denali.cs.washington.edu/`, December 2005. (Data de Acesso).

[Dik00] Jeff Dike. A User-Mode Port of the Linux Kernel. In *4th Annual Linux Showcase and Conference*, pages 63–72, Atlanta, Georgia, USA, October 2000.

[Dik01a] Jeff Dike. User-Mode Linux. In *Ottawa Linux Symposium*, Ottawa, Canada, July 2001.

[Dik01b] Jeff Dike. User-Mode Linux. In *5th Annual Linux Showcase & Conference*, pages 3–14, Oakland, CA, USA, November 2001.

[fau05] FAUmachine Project - Developing an Open Source Virtual Machine. `http://www.faumachine.org/`, December 2005. (Data de Acesso).

[fia05] The Fiasco microkernel. `http://os.inf.tu-dresden.de/fiasco/`, December 2005. (Data de Acesso).

[Fre] FreeBSD 4.11. *Jail (5) - Imprison Process and its Descendants*. (man page).

[GA03] Sudhakar Govindavajhala and Andrew Appel. Using Memory Errors to Attack a Virtual Machine. In *IEEE Symposium on Secutiry and Privacy*, page 154, Washington, DC, USA, 2003. IEEE Computer Society.

[GN00] R. Gilligan and E. Nordmark. Transition Mechanisms for IPv6 Hosts and Routers. Technical Report RFC 2893, The Internet Society, August 2000. `http://www.ietf.org/rfc/rfc2893.txt`.

[Gol73] R. P. Goldberg. Architecture of Virtual Machines. In *AFIPS National Computer Conference*, July 1973.

[GS96] Simson Garfinkel and Gene Spafford. *Practical Unix & Internet Security*. O'Reilly & Associates, Inc., USA, 2 edition, 1996.

[hal05] HAL - Hardware Abstraction Layer. `http://www.freedesktop.org/wiki/Software_2fhal`, December 2005. (Data de Acesso).

[HBS02] Hans-Jörg Höxer, Kerstin Buchacker, and Volkmar Sieh. Implementing a User Mode Linux with Minimal Changes from Original Kernel. In *9th International Linux System Technology Conference*, pages 72–82, Koln, Germany, September 2002.

[Hon03] Jerry Honeycutt. Microsoft Virtual PC 2004 Technical Overview. Technical report, Microsoft Corp., November 2003.

[HSW04] Hans-Jörg Höxer, Volkmar Sieh, and Martin Waitz. Advanced virtualization techniques for FAUmachine. In *11th International Linux System Technology Conference*, Erlangen, Germany, September 2004.

[Hui05] C. Huitema. Teredo: Tunneling IPv6 over UDP through NATs. Technical report, Internet Engineering Task Force (IETF), April 2005. `http://www.ietf.org/internet-drafts/draft-huitema-v6ops-teredo-05.txt`.

[Inc05a] AMD Inc. AMD64 Virtualization Codenamed "Pacifica" Technology - Secure Virtual Machine Architecture Reference Manual. Technical Report 33047, AMD Inc., May 2005.

[Inc05b] AMD Inc. Putting Server Virtualization to Work. Technical Report 32951A, AMD Inc., 2005.

[jai05] Jail - Imprison Process and its Descendants. `http://www.freebsd.org/`, December 2005. (Data de Acesso).

[jav05] Java Technology. `http://java.sun.com/`, December 2005. (Data de Acesso).

[KDC03] Samuel T. King, George W. Dunlap, and Peter M. Chen. Operating System Support for Virtual Machines. In *USENIX Annual Technical Conference*, pages 71–84, San Antonio, TX, USA, June 2003.

[Knu68] Donald E. Knuth. *The Art of Computer Programming*, volume 1: Fundamental Algorithms. Addison-Wesley Publishing Company, 1 edition, 1968.

[KW00] Poul-Henning Kamp and Robert Watson. Jails: Confining the Omnipotent Root. In *2nd International SANE Conference*, Maastricht, The Netherlands, May 2000.

[Law99] Kevin Lawton. Running Multiple Operating Systems Concurrently on an IA32 PC Using Virtualization Techniques. Technical report, Plex86 x86 Virtual Machine Project, nov 1999.

[Lin] Linux 2.6.6. *ptrace (2) - process trace*. (man page).

[lin05] Linux VServer Project. `http://linux-vserver.org/`, December 2005. (Data de Acesso).

[LSM+98] Peter Loscocco, Stephen Smalley, Patrick Muckelbauer, Ruth Taylor, S. Jeff Turner, and John Farrell. The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments. In *21st National Information Systems Security Conference*, pages 303–314, oct 1998.

[LY99] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Addison-Wesley Professional, 2 edition, April 1999.

[Mic03] Microsoft Corp. *Microsoft Virtual PC 2004 Evaluation Guide*, November 2003.

[Mic05] Microsoft Corp. *Virtual Server 2005 R2 Technical Overview*, December 2005.

[mol05] Mac-on-Linux. `http://www.maconlinux.org/`, December 2005. (Data de Acesso).

[NdG02] Emilio Tissato Nakamura and Paulo Lício de Geus. *Segurança de Redes em ambientes cooperativos*. Editora Berkeley, Campinas, SP, BRA, 2002.

[pac05] Server Virtualization Powered by AMD Opteron Processors. `http://enterprise.amd.com/enterprise/serverVirtualization.aspx`, December 2005. (Data de Acesso).

[pea05] PearPC - PowerPC Architecture Emulator. `http://pearpc.sourceforge.net/`, December 2005. (Data de Acesso).

[per05] The PERSEUS Project. `http://www.perseusos.org/`, December 2005. (Data de Acesso).

[PG74] Gerald J. Popek and Robert P. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. In *Communications of the ACM*, jul 1974.

[ple05] The New Plex86 x86 Virtual Machine Project. `http://plex86.sf.net/`, December 2005. (Data de Acesso).

[Pro03] Honeynet Project. Know Your Enemy: Defining Virtual Honeynets. Technical report, Honeynet Project, January 2003. `http://www.honeynet.org/papers/virtual/index.html`.

[PRS+01] Birgit Pfitzmann, James Riordan, Christian Stüble, Michael Waidner, and Arnd Weber. The PERSEUS System Architecture. In Dirk Fox, Marit Köhntopp, and Andreas Pfitzmann, editors, *VIS 2001, Sicherheit in komplexen IT-Infrastrukturen*, pages 1–18. Vieweg Verlag, April 2001.

[PT04] Daniel Price and Andrew Tucker. Solaris Zones: Operating System Support for Consolidating Commercial Workloads. In *18th Large Installation System Administration Conference*, pages 241–254, Atlanta, GA, USA, November 2004.

[qem05] QEMU. `http://fabrice.bellard.free.fr/qemu/`, December 2005. (Data de Acesso).

[RI00] J. Robin and C. Irvine. Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor. In *9th USENIX Security Symposium*, pages 129–144, Denver, CO, USA, August 2000.

[Ros04a] Robert Rose. Survey of System Virtualization Techniques, March 2004.

[Ros04b] Mendel Rosenblum. The Reincarnation of Virtual Machines. ACM Queue, August 2004. `http://acmqueue.com/modules.php?name=Content\&pa=showpage\&pid=168\&pag%e=1`.

[SC05] Christian Schild and Tim Chown. Final IPv4 to IPv6 transition cookbook for end site networks/universities. Technical report, 6NET, June 2005. `http://www.6net.org/publications/deliverables/D2.3.4v2.pdf`.

[SGG01] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. John Wiley & Sons, Inc., New York, NY, USA, 6 edition, June 2001.

[shy05] IBM Research - sHype Secure Hypervisor. `http://www.research.ibm.com/secure_systems_department/projects/hypervis%or/`, December 2005. (Data de Acesso).

[Sun] SunOS 5.10. *Zones (5) - Solaris Application Containers*. (man page).

[SVL01] Jeremy Sugerman, G. Venkitachalam, and B. Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *2001 USENIX Annual Technical Conference*, jun 2001.

[TC04] Andrew Tucker and David Comay. Solaris Zones: Operating System Support for Server Consolidation. In *3rd Virtual Machine Research and Technology Symposium*, March 2004.

[TGTT05] F. Templin, T. Gleeson, M. Talwar, and D. Thaler. Intra-Site Automatic Tunnel Addressing Protocol (ISATAP). Technical Report RFC 4214, The Internet Society, October 2005. `http://www.ietf.org/rfc/rfc4214.txt`.

[tpm05] Trusted Computing Group: TPM. `https://www.trustedcomputinggroup.org/groups/tpm/`, December 2005. (Data de Acesso).

[TW97] Andrew S. Tanenbaum and Albert S. Woodhull. *Operating Systems: Design and Implementation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2 edition, 1997.

[uml05] The User-Mode Linux Kernel Home Page. `http://user-mode-linux.sourceforge.net/`, December 2005. (Data de Acesso).

[van05] Intel Virtualization Technology. `http://www.intel.com/technology/vt`, December 2005. (Data de Acesso).

[Var97] Melinda Varian. VM and the VM Community: Past, Present, and Future. Technical report, Office of Computing and Information Technology - Princeton University, Princeton, NJ, USA, aug 1997.

[vir05] Microsoft Virtual PC. `http://www.microsoft.com/virtualpc/`, December 2005. (Data de Acesso).

[vmw05] VMware, Inc. `http://www.vmware.com/`, December 2005. (Data de Acesso).

[WCSG04] Andrew Whitaker, Richard S. Cox, Marianne Shaw, and Steven D. Gribble. Constructing Services with Interposable Virtual Hardware. In *1st Symposium on Networked Systems Design and Implementation*, pages 169–182, March 2004.

[WSG02a] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Denali: A Scalable Isolation Kernel. In *10th ACM SIGOPS European Workshop*, St. Emilion, France, September 2002.

[WSG02b] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Denali: Lightweight Virtual Machines for Distributed and Networked Applications. In *5th USENIX Symposium on Operating Systems Design and Implementation*, pages 195–209, December 2002.

[WSG02c] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Scale and Performance in the Denali Isolation Kernel. *SIGOPS Operating Systems Review*, 36(SI):195–209, 2002.

[xen05] The Xen Virtual Machine Monitor. `http://www.cl.cam.ac.uk/Research/SRG/netos/xen/`, December 2005. (Data de Acesso).

[Yag01] Karim Yaghmour. Adaptative Domain Environment for Operating Systems. Technical report, Opersys inc., http://www.opersys.com/, 2001.

[Yag02] Karim Yaghmour. A Pratical Approach to Linux Clusters on SMP Hardware. Technical report, Opersys inc., http://www.opersys.com/, 2002.

[zon05] Solaris Zones. `http://www.sun.com/bigadmin/content/zones/`, December 2005. (Data de Acesso).