Capítulo 33

Construção de máquinas virtuais

A construção de máquinas virtuais é bem mais complexa que possa parecer à primeira vista. Caso os conjuntos de instruções (ISA) do sistema real e do sistema virtual sejam diferentes, é necessário usar as instruções da máquina real para simular as instruções da máquina virtual. Além disso, é necessário mapear os recursos de hardware virtuais (periféricos oferecidos ao sistema convidado) sobre os recursos existentes na máquina real (os periféricos reais). Por fim, pode ser necessário mapear as chamadas de sistema emitidas pelas aplicações do sistema convidado em chamadas equivalentes no sistema real, quando os sistemas operacionais virtual e real forem distintos.

Este capítulo aborda inicialmente o conceito formal de virtualização, para em seguida discutir as principais técnicas usadas na construção de máquinas virtuais.

33.1 Definição formal

Em 1974, os pesquisadores americanos Gerald Popek (UCLA) e Robert Goldberg (Harvard) definiram uma máquina virtual da seguinte forma [Popek and Goldberg, 1974]:

> *Uma máquina virtual é vista como uma duplicata eficiente e isolada de uma máquina real. Essa abstração é construída por um "monitor de máquina virtual" (VMM - Virtual Machine Monitor).*

Para funcionar de forma correta e eficiente, o monitor ou hipervisor deve atender a alguns requisitos básicos, também definidos pos Popek e Goldberg:

Equivalência: um hipervisor de prover um ambiente de execução quase idêntico ao da máquina real original. Todo programa executando em uma máquina virtual deve se comportar da mesma forma que o faria em uma máquina real; exceções podem resultar somente de diferenças nos recursos disponíveis (memória, disco, etc.), dependências de temporização e a existência dos dispositivos de entrada/saída necessários à aplicação.

Controle de recursos: o hipervisor deve possuir o controle completo dos recursos da máquina real: nenhum programa executando na máquina virtual deve possuir acesso a recursos que não tenham sido explicitamente alocados a ele pelo hipervisor, que deve intermediar todos os acessos. Além disso, a qualquer instante o hipervisor pode retirar recursos previamente alocados à máquina virtual.

Eficiência: grande parte das instruções do processador virtual (o processador provido pelo hipervisor) deve ser executada diretamente pelo processador da máquina real, sem intervenção do hipervisor. As instruções da máquina virtual que não puderem ser executadas pelo processador real devem ser interpretadas pelo hipervisor e traduzidas em ações equivalentes no processador real. Instruções simples, que não afetem outras máquinas virtuais ou aplicações, podem ser executadas diretamente no processador real.

Além dessas três propriedades básicas, as propriedades derivadas a seguir são frequentemente associadas a hipervisores [Rosenblum, 2004]:

Isolamento: aplicações dentro de uma máquina virtual não podem interagir diretamente (a) com outras máquinas virtuais, (b) com o hipervisor, ou (c) com o sistema real hospedeiro. Todas as interações entre entidades dentro de uma máquina virtual e o mundo exterior devem ser mediadas pelo hipervisor.

Introspecção: o hipervisor tem acesso e controle sobre todas as informações do estado interno da máquina virtual, como registradores do processador, conteúdo de memória, eventos etc.

Recursividade: alguns hipervisores exibem também esta propriedade: deve ser possível executar um hipervisor dentro de uma máquina virtual, produzindo um novo nível de máquinas virtuais. Neste caso, a máquina real é normalmente denominada *máquina de nível 0*.

Essas propriedades básicas caracterizam um hipervisor ideal, que nem sempre pode ser construído sobre as plataformas de hardware existentes. A possibilidade de construção de um hipervisor em uma determinada plataforma é definida através do seguinte teorema, definido por Popek e Goldberg [Popek and Goldberg, 1974]:

> *Para qualquer computador convencional de terceira geração, um hipervisor pode ser construído se o conjunto de instruções sensíveis daquele computador for um subconjunto de seu conjunto de instruções privilegiadas.*

Para compreender melhor as implicações desse teorema, é necessário definir claramente os seguintes conceitos:

- *Computador convencional de terceira geração*: qualquer sistema de computação convencional seguindo a arquitetura de Von Neumann, que suporte memória virtual e dois modos de operação do processador: modo usuário e modo privilegiado.
- *Instruções sensíveis*: são aquelas que podem consultar ou alterar o status do processador, ou seja, os registradores que armazenam o status atual da execução na máquina real;
- *Instruções privilegiadas*: são acessíveis somente por meio de códigos executando em nível privilegiado (código de núcleo). Caso um código não privilegiado tente executar uma instrução privilegiada, uma exceção (interrupção) deve ser gerada, ativando uma rotina de tratamento previamente especificada pelo núcleo do sistema real.

De acordo com esse teorema, toda instrução sensível deve ser também privilegiada. Assim, quando uma instrução sensível for executada por um programa não privilegiado (um núcleo convidado ou uma aplicação convidada), provocará a ocorrência de uma interrupção. Essa interrupção pode ser usada para ativar uma *rotina de interpretação* dentro do hipervisor, que irá simular o efeito da instrução sensível (ou seja, interpretá-la), de acordo com o contexto onde sua execução foi solicitada (máquina virtual ou hipervisor). Obviamente, quanto maior o número de instruções sensíveis, maior o volume de interpretação de código a realizar, e menor o desempenho da máquina virtual.

No caso de processadores que não atendam as restrições de Popek/Goldberg, podem existir instruções sensíveis que executem sem gerar interrupções, o que impede o hipervisor de interceptá-las e interpretá-las. Uma solução possível para esse problema é a *tradução dinâmica* das instruções sensíveis presentes nos programas de usuário: ao carregar um programa na memória, o hipervisor analisa seu código e substitui essas instruções sensíveis por chamadas a rotinas que as interpretam dentro do hipervisor. Isso implica em um tempo maior para o lançamento de programas, mas torna possível a virtualização. Outra técnica possível para resolver o problema é a *paravirtualização*, que se baseia em reescrever parte do sistema convidado para não usar essas instruções sensíveis. Ambas as técnicas são discutidas neste capítulo.

33.2 Suporte de hardware

Na época em que Popek e Goldberg definiram seu principal teorema, o hardware dos mainframes IBM suportava parcialmente as condições impostas pelo mesmo. Esses sistemas dispunham de uma funcionalidade chamada *execução direta*, que permitia a uma máquina virtual acessar nativamente o hardware para a execução de instruções. Esse mecanismo permitia que aqueles sistemas obtivessem, com a utilização de máquinas virtuais, desempenho similar ao de sistemas convencionais equivalentes [Goldberg, 1973; Popek and Goldberg, 1974; Goldberg and Mager, 1979].

O suporte de hardware para a construção de hipervisores eficientes está presente em sistemas de grande porte, como os mainframes, mas permaneceu deficiente nos microprocessadores de mercado por muito tempo. Por exemplo, a família de processadores *Intel Pentium IV* (e anteriores) possuía 17 instruções sensíveis que podiam ser executadas em modo usuário sem gerar exceções, violando o teorema de Goldberg (Seção 33.1) e dificultando a criação de máquinas virtuais [Robin and Irvine, 2000]. Alguns exemplos dessas instruções "problemáticas" são:

- `SGDT/SLDT`: permitem ler o registrador que indica a posição e tamanho das tabelas de segmentos global/local do processo ativo.
- `SMSW`: permite ler o registrador de controle 0, que contém informações de status interno do processador.
- `PUSHF/POPF`: empilha/desempilha o valor do registrador `EFLAGS`, que também contém informações de status interno do processador.

Para controlar o acesso aos recursos do sistema e às instruções privilegiadas, os processadores usam a noção de "anéis de proteção" herdada do sistema MULTICS [Corbató and Vyssotsky, 1965]. Os anéis definem níveis de privilégio: um código

executando no nível 0 (anel central) tem acesso completo ao hardware, enquanto um código executando em um nível $i > 0$ (anéis externos) tem menos privilégio. Quanto mais externo o anel onde um código executa, menor o seu nível de privilégio. Os processadores Intel/AMD atuais suportam 4 anéis ou níveis de proteção, mas a quase totalidade dos sistemas operacionais de mercado somente usa os dois anéis extremos: o anel 0 para o núcleo do sistema e o anel 3 para as aplicações dos usuários.

As técnicas iniciais de virtualização para as plataformas Intel/AMD se baseavam na redução de privilégios do sistema operacional convidado: o hipervisor e o sistema operacional hospedeiro executam no nível 0, o sistema operacional convidado executa no nível 1 ou 2 e as aplicações do sistema convidado executam no nível 3. Essas formas de estruturação de sistema são denominadas respectivamente "modelo 0/1/3" e modelo "0/2/3" (Figura 33.1). Todavia, para que a estratégia de redução de privilégio pudessem funcionar, algumas instruções do sistema operacional convidado precisavam ser reescritas dinamicamente no momento da carga do sistema convidado na memória, pois ele foi construído para executar no nível 0.

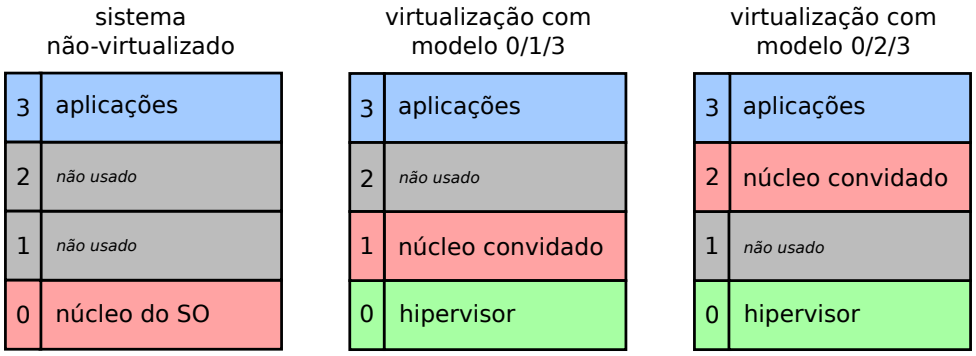

Figura 33.1: Uso dos níveis de proteção em processadores Intel/AMD.

Por volta de 2005, os principais fabricantes de microprocessadores (*Intel* e *AMD*) incorporaram um suporte básico à virtualização em seus processadores, através das tecnologias IVT (*Intel Virtualization Technology*) e AMD-V (*AMD Virtualization*), que são conceitualmente equivalentes [Uhlig et al., 2005]. A ideia central de ambas as tecnologias consiste em definir dois modos possíveis de operação do processador: os modos *root* e *non-root*. O modo *root* equivale ao funcionamento de um processador convencional, e se destina à execução de um hipervisor. Por outro lado, o modo *non-root* se destina à execução de máquinas virtuais. Ambos os modos suportam os quatro níveis de privilégio, o que permite executar os sistemas convidados sem a necessidade de reescrita dinâmica de seu código.

São também definidos dois procedimentos de transição entre modos: *VM entry* (transição *root* → *non-root*) e *VM exit* (transição *non-root* → *root*). Quando operando dentro de uma máquina virtual (ou seja, em modo *non-root*), as instruções sensíveis e as interrupções podem provocar a transição *VM exit*, devolvendo o processador ao hipervisor em modo *root*. As instruções e interrupções que provocam a transição *VM exit* são configuráveis pelo próprio hipervisor.

Para gerenciar o estado do processador (conteúdo dos registradores), é definida uma *Estrutura de Controle de Máquina Virtual* (VMCS - *Virtual-Machine Control Structure*). Essa estrutura de dados contém duas áreas: uma para os sistemas convidados e outra para o hipervisor. Na transição *VM entry*, o estado do processador é lido a partir da área de sistemas convidados da VMCS. Já uma transição *VM exit* faz com que o estado do

processador seja salvo na área de sistemas convidados e o estado anterior do hipervisor, previamente salvo na VMCS, seja restaurado. A Figura 33.2 traz uma visão geral da arquitetura Intel IVT.

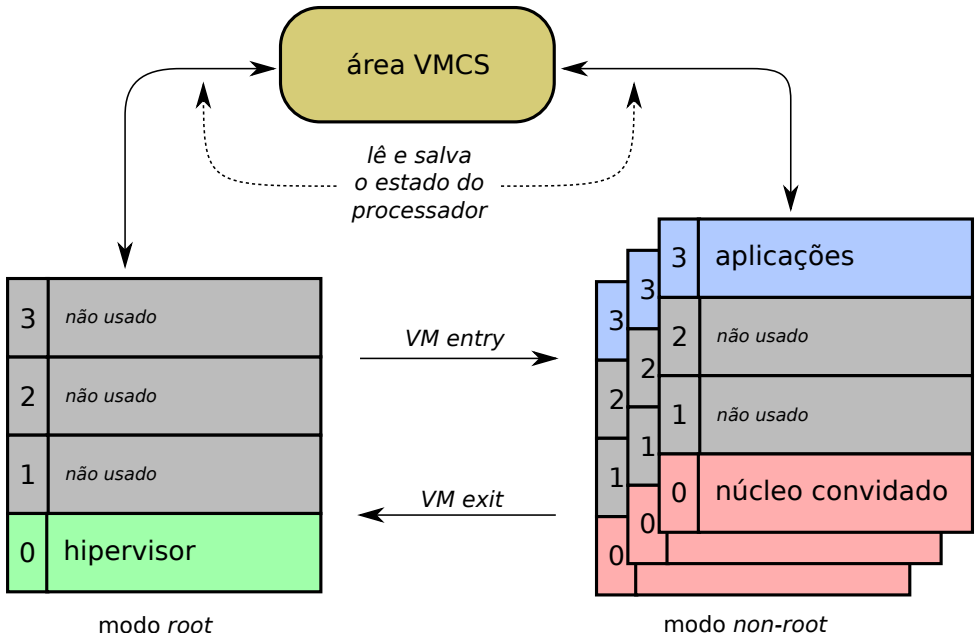

Figura 33.2: Visão geral da arquitetura *Intel IVT*.

Além da Intel e AMD, outros fabricantes de hardware se preocuparam com o suporte à virtualização. Em 2005, a *Sun Microsystems* incorporou suporte nativo à virtualização em seus processadores *UltraSPARC* [Yen, 2007]. Em 2007, a IBM propôs uma especificação de interface de hardware denominada *IBM Power ISA 2.04* [IBM], que respeita os requisitos necessários à virtualização do processador e da gestão de memória.

Conforme apresentado, a virtualização do processador pode obtida por reescrita dinâmica do código executável ou através do suporte nativo em hardware, usando tecnologias como IVT e AMD-V. Por outro lado, a virtualização da memória envolve outros desafios, que exigem modificações significativas dos mecanismos de gestão de memória. Por exemplo, é importante prever o compartilhamento de páginas de código entre máquinas virtuais, para reduzir a quantidade total de memória física necessária a cada máquina virtual. Outros desafios similares surgem na virtualização dos dispositivos de armazenamento e de entrada/saída, alguns deles sendo analisados em [Rosenblum and Garfinkel, 2005].

33.3 Níveis de virtualização

A virtualização implica na reescrita de interfaces de sistema, para permitir a interação entre componentes de sistema construídos para plataformas distintas. Como existem várias interfaces entre os componentes, também há várias possibilidades de aplicação da virtualização em um sistema. De acordo com as interfaces em que são aplicadas, os níveis mais usuais de virtualização são [Rosenblum, 2004; Nanda and Chiueh, 2005]:

Virtualização do hardware: toda a interface ISA, que permite o acesso ao hardware, é virtualizada. Isto inclui o conjunto de instruções do processador e as interfaces de acesso aos dispositivos de entrada/saída. A virtualização de hardware (ou virtualização completa) permite executar um sistema operacional e/ou aplicações em uma plataforma totalmente diversa daquela para a qual estes foram desenvolvidos. Esta é a forma de virtualização mais poderosa e flexível, mas também a de menor desempenho, uma vez que o hipervisor tem de traduzir todas as instruções geradas no sistema convidado em instruções do processador real. A máquina virtual Java (JVM, Seção 34.2.6) é um bom exemplo de virtualização do hardware. Máquinas virtuais implementando esta forma de virtualização são geralmente denominados *emuladores de hardware*.

Virtualização da interface de sistema: virtualiza-se a *System ISA*, que corresponde ao conjunto de instruções sensíveis do processador. Esta forma de virtualização é bem mais eficiente que a anterior, pois o hipervisor emula somente as instruções sensíveis do processador virtual, executadas em modo privilegiado pelo sistema operacional convidado. As instruções não sensíveis podem ser executadas diretamente pelo processador real, sem perda de desempenho. Todavia, apenas sistemas convidados desenvolvidos para o mesmo processador podem ser executados usando esta abordagem. Esta é a abordagem clássica de virtualização, presente nos sistemas de grande porte (*mainframes*) e usada nos ambientes de máquinas virtuais VMware, Xen e KVM.

Virtualização de dispositivos de entrada/saída: virtualizam-se os dispositivos físicos que permitem ao sistema interagir com o mundo exterior. Esta técnica implica na construção de dispositivos físicos virtuais, como discos, interfaces de rede e terminais de interação com o usuário, usando os dispositivos físicos subjacentes. A maioria dos ambientes de máquinas virtuais usa a virtualização de dispositivos para oferecer discos rígidos e interfaces de rede virtuais aos sistemas convidados.

Virtualização do sistema operacional: virtualiza-se o conjunto de recursos lógicos oferecidos pelo sistema operacional, como árvores de diretórios, descritores de arquivos, semáforos, canais de IPC e IDS de processos, usuários e grupos. Nesta abordagem, cada máquina virtual pode ser vista como uma instância distinta do mesmo sistema operacional subjacente. Esta é a abordagem comumente conhecida como *servidores virtuais* ou *contêneres*, da qual são bons exemplos os ambientes *FreeBSD Jails*, *Linux VServers* e *Solaris Zones*.

Virtualização de chamadas de sistema: permite oferecer o conjunto de chamadas de sistema de uma sistema operacional *A* usando as chamadas de sistema de um sistema operacional *B*, permitindo assim a execução de aplicações desenvolvidas para um sistema operacional sobre outro sistema. Todavia, como as chamadas de sistema são normalmente invocadas através de funções de bibliotecas, a virtualização de chamadas de sistema pode ser vista como um caso especial de virtualização de bibliotecas.

Virtualização de chamadas de biblioteca: tem objetivos similares ao da virtualização de chamadas de sistema, permitindo executar aplicações em diferentes sistemas operacionais e/ou com bibliotecas diversas daquelas para as quais foram

construídas. O sistema *Wine*, que permite executar aplicações Windows sobre sistemas UNIX, usa essencialmente esta abordagem.

Na prática, esses vários níveis de virtualização podem ser usados para a resolução de problemas específicos em diversas áreas de um sistema de computação. Por exemplo, vários sistemas operacionais oferecem facilidades de virtualização de dispositivos físicos, como por exemplo: interfaces de rede virtuais que permitem associar mais de um endereço de rede ao computador, discos rígidos virtuais criados em áreas livres da memória RAM (os chamados *RAM disks*); árvores de diretórios virtuais criadas para confinar processos críticos para a segurança do sistema (através da chamada de sistema `chroot`), emulação de operações 3D em uma placa gráfica que não as suporta, etc.

33.4 Técnicas de virtualização

A construção de hipervisores implica na definição de algumas estratégias para a virtualização. As estratégias mais utilizadas atualmente são a emulação completa do hardware, a virtualização da interface de sistema, a tradução dinâmica de código e a paravirtualização. Além disso, algumas técnicas complementares são usadas para melhorar o desempenho dos sistemas de máquinas virtuais. Essas técnicas são discutidas nesta seção.

33.4.1 Emulação completa

Nesta abordagem, toda a interface do hardware é virtualizada, incluindo todas as instruções do processador, a memória e os dispositivos periféricos. Isso permite oferecer ao sistema operacional convidado uma interface de hardware distinta daquela fornecida pela máquina real subjacente, caso seja necessário. O custo de virtualização pode ser muito elevado, pois cada instrução executada pelo sistema convidado tem de ser analisada e traduzida em uma ou mais instruções equivalentes no computador real. No entanto, esta abordagem permite executar sistemas operacionais em outras plataformas, distintas daquela para a qual foram projetados, sem modificação.

Exemplos típicos de emulação completa abordagem são os sistemas de máquinas virtuais *QEMU*, que oferece um processador x86 ao sistema convidado, o *MS VirtualPC for MAC*, que permite executar o sistema Windows sobre uma plataforma de hardware *PowerPC*, e o sistema *Hercules*, que emula um computador *IBM System/390* sobre um PC convencional de plataforma Intel.

Um caso especial de emulação completa consiste nos **hipervisores embutidos no hardware** (*codesigned hypervisors*). Um hipervisor embutido é visto como parte integrante do hardware e implementa a interface de sistema (ISA) vista pelos sistemas operacionais e aplicações daquela plataforma. Entretanto, o conjunto de instruções do processador real somente está acessível ao hipervisor, que reside em uma área de memória separada da memória principal e usa técnicas de tradução dinâmica (vide Seção 33.4.3) para tratar as instruções executadas pelos sistemas convidados.

Um exemplo típico desse tipo de sistema é o processador *Transmeta Crusoe/Efficeon*, que aceita instruções no padrão *Intel 32 bits* e internamente as converte em um conjunto de instruções VLIW (*Very Large Instruction Word*). Como o hipervisor desse processador pode ser reprogramado para criar novas instruções ou modificar as

instruções existentes, ele acabou sendo denominado *Code Morphing Software* (Figura 33.3).

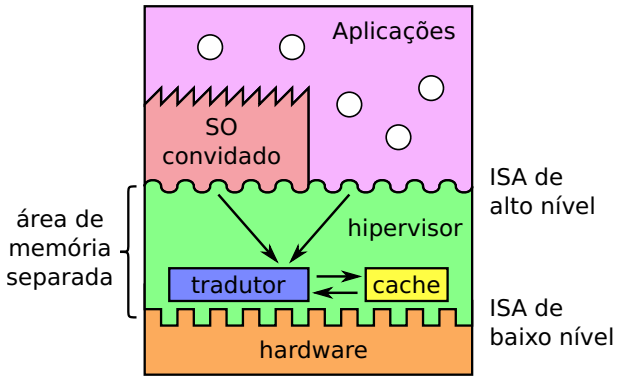

Figura 33.3: Hipervisor embutido no hardware.

33.4.2 Virtualização da interface de sistema

Nesta abordagem, a interface ISA de usuário é mantida, apenas as instruções privilegiadas e os dispositivos (discos, interfaces de rede, etc.) são virtualizados. Dessa forma, o sistema operacional convidado e as aplicações convidadas veem o processador real. Como a quantidade de instruções a virtualizar é reduzida, o desempenho do sistema convidado pode ficar próximo daquele obtido se ele estivesse executando diretamente sobre o hardware real.

Cabe lembrar que a virtualização da interface de sistema só pode ser aplicada diretamente caso o hardware subjacente atenda os requisitos de Goldberg e Popek (cf. Seção 33.1). No caso de processadores que não atendam esses requisitos, podem existir instruções sensíveis que executem sem gerar interrupções, impedindo o hipervisor de interceptá-las. Nesse caso, será necessário o emprego de técnicas complementares, como a *tradução dinâmica* das instruções sensíveis. Obviamente, quanto maior o número de instruções sensíveis, maior o volume de interpretação de código a realizar, e menor o desempenho da máquina virtual. Os processadores mais recentes das famílias Intel e AMD discutidos na Seção 33.2 atendem os requisitos de Goldberg/Popek, e por isso suportam esta técnica de virtualização.

Exemplos de sistemas que implementam esta técnica incluem os ambientes *VMware Workstation*, *VirtualBox*, *MS VirtualPC* e *KVM*.

33.4.3 Tradução dinâmica

Uma técnica frequentemente utilizada na construção de máquinas virtuais é a *tradução dinâmica* (*dynamic translation*) ou recompilação dinâmica (*dynamic recompilation*) de partes do código binário do sistema convidado e suas aplicações. Nesta técnica, o hipervisor analisa, reorganiza e traduz as sequências de instruções emitidas pelo sistema convidado em novas sequências de instruções, à medida em que o código é carregado na memória, ou mesmo durante a execução do sistema convidado.

A tradução binária dinâmica pode ter vários objetivos: (a) adaptar as instruções geradas pelo sistema convidado à interface ISA do sistema real, caso não sejam idênticas; (b) detectar e tratar instruções sensíveis não privilegiadas (que não geram interrupções

ao serem invocadas pelo sistema convidado); ou (c) analisar, reorganizar e otimizar as sequências de instruções geradas pelo sistema convidado, de forma a melhorar o desempenho de sua execução. Neste último caso, os blocos de instruções muito frequentes podem ter suas traduções mantidas em cache, para melhorar ainda mais o desempenho.

A tradução dinâmica é usada em vários tipos de hipervisores. Uma aplicação típica é a construção da máquina virtual Java, onde recebe o nome de JIT – *Just-in-Time Bytecode Compiler*. Outro uso corrente é a construção de hipervisores para plataformas sem suporte adequado à virtualização, como os processadores Intel/AMD 32 bits. Neste caso, o código convidado a ser executado é analisado em busca de instruções sensíveis, que são substituídas por chamadas a rotinas apropriadas dentro do supervisor.

No contexto de virtualização, a tradução dinâmica é composta basicamente dos seguintes passos [Ung and Cifuentes, 2006]:

1. *Desmontagem* (*disassembling*): o fluxo de bytes do código convidado em execução é decomposto em blocos de instruções. Cada bloco é normalmente composto de uma sequência de instruções de tamanho variável, terminando com uma instrução de controle de fluxo de execução;

2. *Geração de código intermediário*: cada bloco de instruções tem sua semântica descrita através de uma representação independente de máquina;

3. *Otimização*: a descrição em alto nível do bloco de instruções é analisada para aplicar eventuais otimizações; como este processo é realizado durante a execução, normalmente somente otimizações com baixo custo computacional são aplicáveis;

4. *Codificação*: o bloco de instruções otimizado é traduzido para instruções da máquina física, que podem ser diferentes das instruções do código original;

5. *Caching*: blocos de instruções com execução muito frequente têm sua tradução armazenada em cache, para evitar ter de traduzi-los e otimizá-los novamente;

6. *Execução*: o bloco de instruções traduzido é finalmente executado nativamente pelo processador da máquina real.

Esse processo pode ser simplificado caso as instruções de máquina do código convidado sejam as mesmas do processador real subjacente, o que torna desnecessário traduzir os blocos de instruções em uma representação independente de máquina.

33.4.4 Paravirtualização

A Seção 33.2 mostrou que as arquiteturas de alguns processadores, como o *Intel x86*, eram difíceis de virtualizar, porque algumas instruções sensíveis não podiam ser interceptadas pelo hipervisor. Essas instruções sensíveis deviam ser então detectadas e interpretadas pelo hipervisor, em tempo de carga do código na memória.

Além das instruções sensíveis em si, outros aspectos da interface software/hardware trazem dificuldades ao desenvolvimento de máquinas virtuais de sistema eficientes. Uma dessas áreas é o mecanismo de entrega e tratamento de interrupções pelo processador, baseado na noção de uma *tabela de interrupções*, que contém uma

função registrada para cada tipo de interrupção a tratar. Outra área da interface software/hardware que pode trazer dificuldades é a gerência de memória, pois o TLB (*Translation Lookaside Buffer*, Seção 14.7.4) dos processadores *x86* é gerenciado diretamente pelo hardware, sem possibilidade de intervenção direta do hipervisor no caso de uma falta de página.

No início dos anos 2000, alguns pesquisadores investigaram a possibilidade de modificar a interface entre o hipervisor e os sistemas operacionais convidados, oferecendo a estes um hardware virtual que é similar, mas não idêntico ao hardware real. Essa abordagem, denominada *paravirtualização*, permite um melhor acoplamento entre os sistemas convidados e o hipervisor, o que leva a um desempenho significativamente melhor das máquinas virtuais.

Modificações na interface de sistema do hardware virtual (*system ISA*) exigem uma adaptação dos sistemas operacionais convidados, para que estes possam executar sobre a plataforma virtual. Em particular, o hipervisor define uma API denominada *chamadas de hipervisor* (*hypercalls*), que cada sistema convidado deve usar para acessar a interface de sistema do hardware virtual. Todavia, a interface de usuário (*user ISA*) do hardware é preservada, permitindo que as aplicações convidadas executem sem necessidade de modificações. A Figura 33.4 ilustra esse conceito.

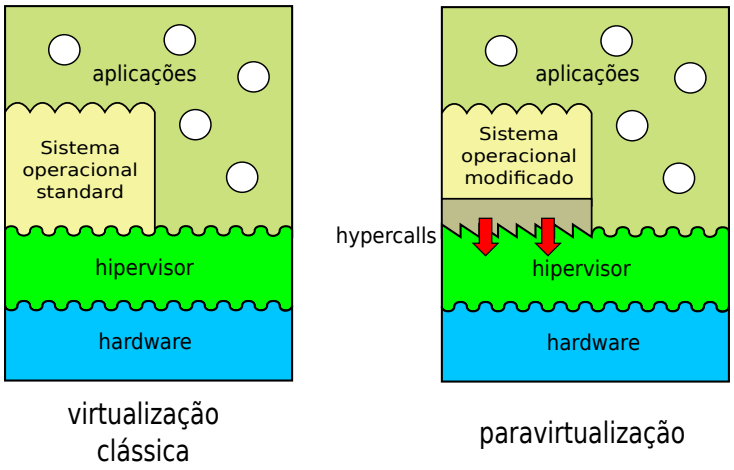

Figura 33.4: Paravirtualização.

Os primeiros ambientes a adotar a paravirtualização foram o *Denali* [Whitaker et al., 2002] e o *Xen* [Barham et al., 2003]. O *Denali* é um ambiente experimental de paravirtualização construído na Universidade de Washington, que pode suportar dezenas de milhares de máquinas virtuais sobre um computador PC convencional. O projeto *Denali* não se preocupa em suportar sistemas operacionais comerciais, sendo voltado à execução maciça de minúsculas máquinas virtuais para serviços de rede. Já o ambiente de máquinas virtuais *Xen* (vide Seção 34.2.2) permite executar sistemas operacionais convencionais como Linux e Windows, modificados para executar sobre um hipervisor.

Embora exija que o sistema convidado seja adaptado ao hipervisor, o que diminui sua portabilidade, a paravirtualização permite que o sistema convidado acesse alguns recursos do hardware diretamente, sem a intermediação ativa do hipervisor. Nesses casos, o acesso ao hardware é apenas monitorado pelo hipervisor, que informa ao sistema convidado seus limites, como as áreas de memória e de disco disponíveis. O acesso aos demais dispositivos, como mouse e teclado, também é direto: o hipervisor apenas

gerencia os conflitos, no caso de múltiplos sistemas convidados em execução simultânea. Apesar de exigir modificações nos sistemas operacionais convidados, a paravirtualização teve sucesso, por conta do desempenho obtido nos sistemas virtualizados, além de simplificar a interface de baixo nível dos sistemas convidados.

33.5 Aspectos de desempenho

De acordo com os princípios de Goldberg e Popek, o hipervisor deve permitir que a máquina virtual execute diretamente sobre o hardware sempre que possível, para não prejudicar o desempenho dos sistemas convidados. O hipervisor deve retomar o controle do processador somente quando a máquina virtual tentar executar operações que possam afetar o correto funcionamento do sistema, o conjunto de operações de outras máquinas virtuais ou do próprio hardware. O hipervisor deve então simular com segurança a operação solicitada e devolver o controle à máquina virtual.

Na prática, os hipervisores nativos e convidados raramente são usados em sua forma conceitual. Várias otimizações são inseridas nas arquiteturas apresentadas, com o objetivo principal de melhorar o desempenho das aplicações nos sistemas convidados. Como os pontos cruciais do desempenho dos sistemas de máquinas virtuais são as operações de entrada/saída, as principais otimizações utilizadas em sistemas de produção dizem respeito a essas operações. Quatro formas de otimização são usuais:

- Em hipervisores nativos (Figura 33.5, esquerda):

  1. O sistema convidado (*guest system*) acessa diretamente o hardware. Essa forma de acesso é implementada por modificações no núcleo do sistema convidado e no hipervisor. Essa otimização é implementada, por exemplo, no subsistema de gerência de memória do ambiente Xen [Barham et al., 2003].

- Em hipervisores convidados (Figura 33.5, direita):

  2. O sistema convidado (*guest system*) acessa diretamente o sistema nativo (*host system*). Essa otimização é implementada pelo hipervisor, oferecendo partes da API do sistema nativo ao sistema convidado. Um exemplo dessa otimização é a implementação do sistema de arquivos no *VMware* [VMware, 2000]: em vez de reconstruir integralmente o sistema de arquivos sobre um dispositivo virtual provido pelo hipervisor, o sistema convidado faz uso da implementação de sistema de arquivos existente no sistema nativo.
  3. O sistema convidado (*guest system*) acessa diretamente o hardware. Essa otimização é implementada parcialmente pelo hipervisor e parcialmente pelo sistema nativo, pelo uso de um *device driver* específico. Um exemplo típico dessa otimização é o acesso direto a dispositivos físicos como leitor de CDs, hardware gráfico e interface de rede provida pelo sistema *VMware* aos sistemas operacionais convidados [VMware, 2000].
  4. O hipervisor acessa diretamente o hardware. Neste caso, um *device driver* específico é instalado no sistema nativo, oferecendo ao hipervisor uma interface de baixo nível para acesso ao hardware subjacente. Essa abordagem,

também ilustrada na Figura 33.6, é usada pelo sistema *VMware* [VMware, 2000].

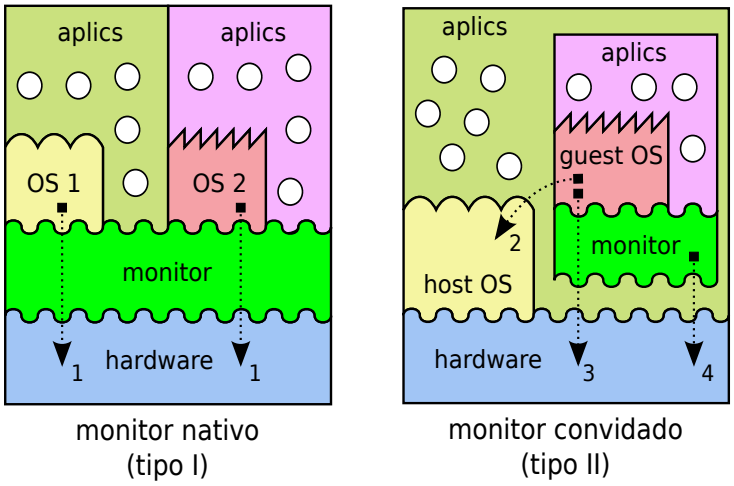

Figura 33.5: Otimizações em sistemas de máquinas virtuais.

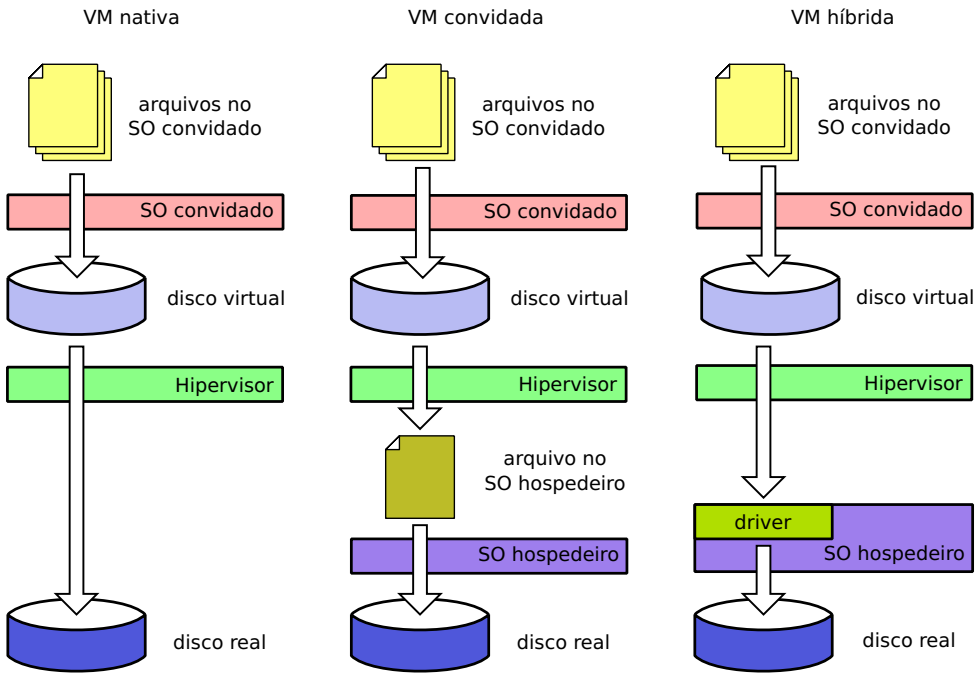

Figura 33.6: Desempenho de hipervisores nativos e convidados.

33.6 Migração de máquinas virtuais

Recentemente, a virtualização vem desempenhando um papel importante na gerência de sistemas computacionais corporativos, graças à facilidade de *migração* de máquinas virtuais implementada pelos hipervisores modernos. Na migração, uma máquina virtual e seu sistema convidado são transferidos através da rede de um hipervisor para outro, executando em equipamentos distintos, sem ter de reiniciá-los. A

máquina virtual tem seu estado preservado e prossegue sua execução no hipervisor de destino assim que a migração é concluída. De acordo com [Clark et al., 2005], as técnicas mais frequentes para implementar a migração de máquinas virtuais são:

- *stop-and-copy*: consiste em suspender a máquina virtual, transferir o conteúdo de sua memória para o hipervisor de destino e retomar a execução em seguida. É uma abordagem simples, mas implica em parar completamente os serviços oferecidos pelo sistema convidado enquanto durar a migração (que pode demorar algumas dezenas de segundos);

- *demand-migration*: a máquina virtual é suspensa apenas durante a cópia das estruturas de memória do núcleo do sistema operacional convidado para o hipervisor de destino, o que dura alguns milissegundos. Em seguida, a execução da máquina virtual é retomada e o restante das páginas de memória da máquina virtual é transferido sob demanda, através dos mecanismos de tratamento de faltas de página. Nesta abordagem a interrupção do serviço tem duração mínima, mas a migração completa pode demorar muito tempo.

- *pre-copy*: consiste basicamente em copiar para o hipervisor de destino todas as páginas de memória da máquina virtual enquanto esta executa; a seguir, a máquina virtual é suspensa e as páginas modificadas depois da cópia inicial são novamente copiadas no destino; uma vez terminada a cópia dessas páginas, a máquina pode retomar sua execução no destino. Esta abordagem, usada no hipervisor *Xen* [Barham et al., 2003], é a que oferece o melhor compromisso entre o tempo de suspensão do serviço e a duração total da migração.

Referências

P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *ACM Symposium on Operating Systems Principles*, pages 164–177, 2003.

C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Symposium on Networked Systems Design and Implementation*, 2005.

F. J. Corbató and V. A. Vyssotsky. Introduction and overview of the Multics system. In *AFIPS Conference Proceedings*, pages 185–196, 1965.

R. Goldberg. Architecture of virtual machines. In *AFIPS National Computer Conference*, 1973.

R. Goldberg and P. Mager. Virtual machine technology: A bridge from large mainframes to networks of small computers. *IEEE Proceedings Compcon Fall 79*, pages 210–213, 1979.

IBM. *Power Instruction Set Architecture – Version 2.04*. IBM Corporation, april 2007.

S. Nanda and T. Chiueh. A survey on virtualization technologies. Technical report, University of New York at Stony Brook, 2005. Departament of Computer Science.

G. Popek and R. Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, July 1974.

J. Robin and C. Irvine. Analysis of the Intel Pentium's ability to support a secure virtual machine monitor. In *9th USENIX Security Symposium*, 2000.

M. Rosenblum. The reincarnation of virtual machines. *Queue Focus - ACM Press*, pages 34–40, 2004.

M. Rosenblum and T. Garfinkel. Virtual machine monitors: Current technology and future trends. *IEEE Computer*, May 2005.

R. Uhlig, G. Neiger, D. Rodgers, A. Santoni, F. Martins, A. Anderson, S. Bennett, A. Kägi, F. Leung, and L. Smith. Intel virtualization technology. *IEEE Computer*, May 2005.

D. Ung and C. Cifuentes. Dynamic re-engineering of binary code with run-time feedbacks. *Science of Computer Programming*, 60(2):189–204, April 2006.

VMware. VMware technical white paper. Technical report, VMware, Palo Alto, CA - USA, 2000.

A. Whitaker, M. Shaw, and S. Gribble. Denali: A scalable isolation kernel. In *ACM SIGOPS European Workshop*, September 2002.

C.-H. Yen. Solaris operating system - hardware virtualization product architecture. Technical Report 820-3703-10, Sun Microsystems, November 2007.