Capítulo 32

Tipos de máquinas virtuais

O principal uso da virtualização de interfaces é a construção de *máquinas virtuais*. Uma máquina virtual é um ambiente de suporte à execução de software, construído usando uma ou mais técnicas de virtualização. Existem vários tipos de máquinas virtuais, que serão discutidos neste capítulo.

32.1 Critérios de classificação

Conforme as características do ambiente virtual proporcionado, as máquinas virtuais podem ser classificadas em três categorias, representadas na Figura 32.1:

Máquinas virtuais de sistema: são ambientes de máquinas virtuais construídos para emular uma plataforma de hardware completa, com processador e periféricos. Este tipo de máquina virtual suporta sistemas operacionais convidados com aplicações convidadas executando sobre eles. Como exemplos desta categoria de máquinas virtuais temos os ambientes *KVM*, *VMware* e *VirtualBox*.

Máquinas virtuais de sistema operacional: são construídas para suportar espaços de usuário distintos sobre um mesmo sistema operacional. Embora compartilhem o mesmo núcleo, cada ambiente virtual possui seus próprios recursos lógicos, como espaço de armazenamento, mecanismos de IPC e interfaces de rede distintas. Os sistemas *Docker*, *Solaris Containers* e *FreeBSD Jails* implementam este conceito.

Máquinas virtuais de processo: também chamadas de máquinas virtuais de aplicação ou de linguagem, são ambientes construídos para prover suporte de execução a apenas um processo ou aplicação convidada específica. A máquina virtual Java e o ambiente de depuração *Valgrind* são exemplos deste tipo de ambiente.

Os ambientes de máquinas virtuais também podem ser classificados de acordo com o nível de similaridade entre as interfaces de hardware do sistema convidado e do sistema real (ISA - *Instruction Set Architecture*, Seção 31.2):

Interfaces equivalentes: a interface virtual oferecida ao ambiente convidado reproduz a interface de hardware do sistema real, permitindo a execução de aplicações construídas para o sistema real. Como a maioria das instruções do sistema convidado pode ser executada diretamente pelo processador (com exceção das

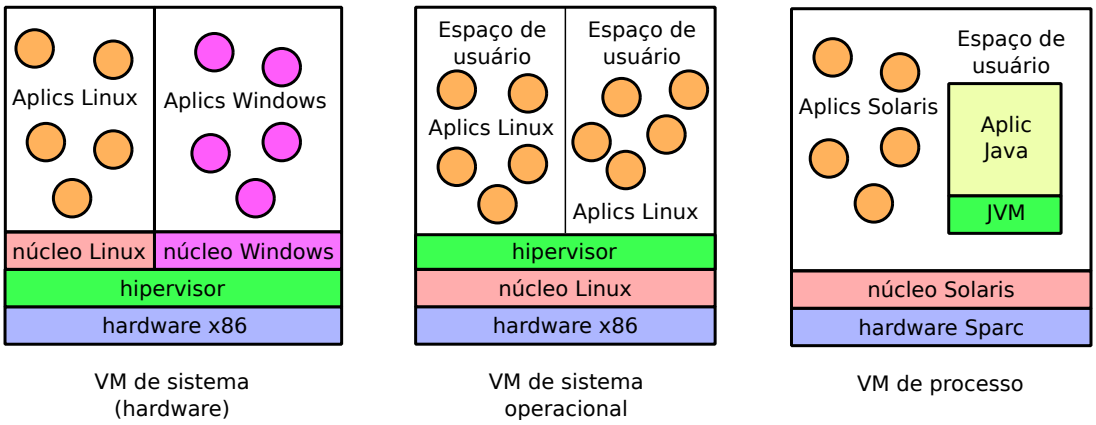

Figura 32.1: Máquinas virtuais de processo, de sistema operacional e de hardware.

instruções sensíveis), o desempenho obtido pelas aplicações convidadas pode ser próximo do desempenho de execução no sistema real. Ambientes como *VMware* são exemplos deste tipo de ambiente.

Interfaces distintas: a interface virtual não tem nenhuma relação com a interface de hardware do sistema real, ou seja, implementa um conjunto de instruções distinto, que deve ser totalmente traduzido pelo hipervisor. Conforme visto na Seção 33.1, a interpretação de instruções impõe um custo de execução significativo ao sistema convidado. A máquina virtual Java e o ambiente *QEmu* são exemplos dessa abordagem.

32.2 Máquinas virtuais de sistema

Uma máquina virtual de sistema (ou de hardware) provê uma interface de hardware completa para um ou mais sistemas operacionais convidados, com suas respectivas aplicações, que executam de forma isolada e independente. Cada sistema operacional convidado tem a ilusão de executar sozinho sobre uma plataforma de hardware exclusiva.

O hipervisor de sistema fornece aos sistemas operacionais convidados uma interface de sistema ISA virtual, que pode ser idêntica ao hardware real, ou distinta. Além disso, ele intercepta e virtualiza o acesso aos recursos, para que cada sistema operacional convidado tenha um conjunto próprio de recursos virtuais, construído a partir dos recursos físicos existentes na máquina real. Assim, cada máquina virtual terá sua própria interface de rede, disco, memória RAM, etc.

Em um ambiente virtual, os sistemas operacionais convidados são fortemente isolados uns dos outros, e normalmente só podem interagir através dos mecanismos de rede, como se estivessem em computadores separados. Todavia, alguns sistemas de máquinas virtuais permitem o compartilhamento controlado de certos recursos. Por exemplo, os sistemas *VMware Workstation* e *VirtualBox* permitem a definição de diretórios compartilhados no sistema de arquivos da máquina real, que podem ser acessados pelas máquinas virtuais.

As máquinas virtuais de sistema constituem a primeira abordagem usada para a construção de hipervisores, desenvolvida na década de 1960 e formalizada por Popek e Goldberg (conforme apresentado na Seção 33.1). Naquela época, a tendência de

desenvolvimento de sistemas computacionais buscava fornecer a cada usuário uma máquina virtual com seus recursos virtuais próprios, sobre a qual o usuário executava um sistema operacional monotarefa e suas aplicações. Assim, o compartilhamento de recursos não era responsabilidade do sistema operacional convidado, mas do hipervisor subjacente.

Ao longo dos anos 70, com o desenvolvimento de sistemas operacionais multitarefas eficientes e robustos como MULTICS e UNIX, as máquinas virtuais de sistema perderam gradativamente seu interesse. Somente no final dos anos 90, com o aumento do poder de processamento dos microprocessadores e o surgimento de novas possibilidades de aplicação, as máquinas virtuais de sistema foram "redescobertas".

Existem basicamente duas arquiteturas de hipervisores de sistema, apresentados na Figura 32.2:

Hipervisor nativo (ou *de tipo I*): executa diretamente sobre o hardware do computador real, sem um sistema operacional subjacente. A função do hipervisor é virtualizar os recursos do hardware (memória, discos, interfaces de rede, etc.) de forma que cada máquina virtual veja um conjunto de recursos próprio e independente. Assim, cada máquina virtual se comporta como um computador completo que pode executar o seu próprio sistema operacional. Esta é a forma mais antiga de virtualização, encontrada nos sistemas computacionais de grande porte dos anos 1960-70. Alguns exemplos de sistemas que empregam esta abordagem são o *IBM OS/370*, o *VMware ESX Server* e o ambiente *Xen*.

Hipervisor convidado (ou *de tipo II*): executa como um processo normal (ou um componente de núcleo) de um sistema operacional nativo subjacente, que gerencia o hardware. O hipervisor utiliza os recursos oferecidos por esse sistema operacional para oferecer recursos virtuais aos sistemas operacionais convidados. Exemplos de sistemas que adotam esta estrutura incluem o *VMware Workstation*, o *KVM* e o *VirtualBox*.

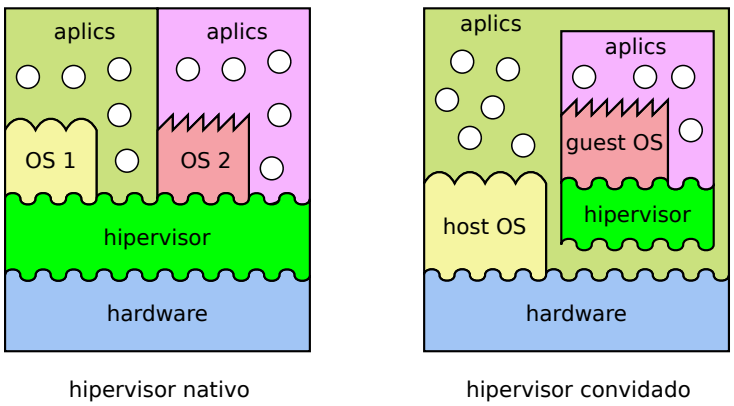

Figura 32.2: Arquiteturas de máquinas virtuais de sistema.

Pode-se afirmar que os hipervisores convidados são mais flexíveis que os hipervisores nativos, pois podem ser facilmente instalados/removidos em máquinas com sistemas operacionais previamente instalados, e podem ser facilmente lançados sob demanda. Por outro lado, hipervisores convidados têm desempenho pior que hipervisores nativos, pois têm de usar os recursos oferecidos pelo sistema operacional

subjacente, enquanto um hipervisor nativo pode acessar diretamente o hardware real. Técnicas para atenuar este problema são discutidas na Seção 33.5.

32.3 Máquinas virtuais de sistema operacional

Em muitas situações, a principal motivação para o uso de máquinas virtuais é o isolamento oferecido pelo ambiente virtual (Seção 33.1). Essa propriedade é importante na segurança de sistemas, por permitir isolar entre si sistemas independentes que executam sobre o mesmo hardware. Por exemplo, a estratégia de *consolidação de servidores* usa máquinas virtuais para abrigar os diversos servidores (de nomes, de arquivos, de e-mail, de Web) de uma empresa em uma mesma máquina física. Dessa forma, pode-se fazer um uso mais eficiente do hardware disponível, preservando o isolamento entre os serviços. Todavia, o custo da virtualização de uma plataforma de hardware ou sistema operacional sobre o desempenho do sistema final pode ser elevado. As principais fontes desse custo são a virtualização dos recursos (periféricos) da máquina real e a eventual necessidade de emular instruções do processador real.

Uma forma simples e eficiente de implementar o isolamento entre aplicações ou subsistemas em um sistema operacional consiste na virtualização do espaço de usuário (*userspace*). Nesta abordagem, denominada *máquinas virtuais de sistema operacional*, *servidores virtuais* ou *contêineres*, o espaço de usuário do sistema operacional é dividido em áreas isoladas denominadas *domínios* ou *contêineres*. A cada domínio é alocada uma parcela dos recursos do sistema operacional, como memória, tempo de processador e espaço em disco.

Para garantir o isolamento entre os domínios, alguns recursos do sistema real são virtualizados, como as interfaces de rede: cada domínio tem sua própria interface de rede virtual e, portanto, seu próprio endereço de rede. Além disso, na maioria das implementações cada domínio tem seu próprio espaço de nomes para os identificadores de usuários, processos e primitivas de comunicação. Assim, é possível encontrar um usuário `pedro` no domínio d_3 e outro usuário `pedro` no domínio d_7, sem relação entre eles nem conflitos. Essa noção de espaços de nomes distintos se estende aos demais recursos do sistema: identificadores de processos, semáforos, árvores de diretórios, etc.

Os processos presentes em um determinado domínio virtual podem interagir entre si, criar novos processos e usar os recursos presentes naquele domínio, respeitando as regras de controle de acesso associadas a esses recursos. Todavia, processos presentes em um domínio não podem ver ou interagir com processos que estiverem em outro domínio, não podem trocar de domínio, criar processos em outros domínios, nem consultar ou usar recursos de outros domínios. Dessa forma, para um determinado domínio, os demais domínios são máquinas distintas, acessíveis somente através de seus endereços de rede.

Para fins de gerência, normalmente é definido um domínio d_0, chamado de *domínio inicial*, *privilegiado* ou *de gerência*, cujos processos têm visibilidade e acesso aos processos e recursos dos demais domínios. Somente processos no domínio d_0 podem migrar para outros domínios; uma vez realizada uma migração, não há possibilidade de retornar ao domínio inicial.

Nesta forma de virtualização, o núcleo do sistema operacional é o mesmo para todos os domínios virtuais, e sua interface (conjunto de chamadas de sistema) é preservada (apenas algumas chamadas de sistema são adicionadas para a gestão dos domínios). A Figura 32.3 mostra a estrutura típica de um ambiente de máquinas

virtuais de sistema operacional. Nela, pode-se observar que um processo pode migrar de d_0 para d_1, mas que os processos em d_1 não podem migrar para outros domínios. A comunicação entre processos confinados em domínios distintos (d_2 e d_3) também é proibida.

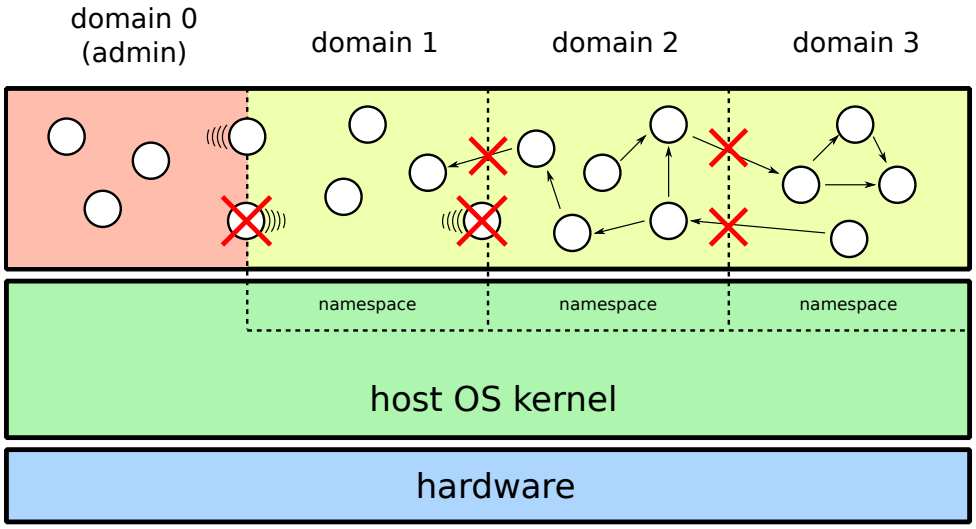

Figura 32.3: Máquinas virtuais de sistema operacional.

Há várias implementações disponíveis de mecanismos para a criação de contêineres. A técnica mais antiga é implementada pela chamada de sistema `chroot()`, disponível na maioria dos sistemas UNIX. Essa chamada de sistema atua sobre o acesso de um processo ao sistema de arquivos: o processo que a executa tem seu acesso ao sistema de arquivos restrito a uma subárvore da hierarquia de diretórios, ou seja, ele fica "confinado" a essa subárvore. Os filhos desse processo herdam essa mesma visão restrita do sistema de arquivos, que não pode ser revertida. Por exemplo, um processo que executa a chamada de sistema `chroot("/var/spool/postfix")` passa a ver somente a hierarquia de diretórios a partir do diretório `/var/spool/postfix`, que se torna o diretório raiz ("/") na visão daquele processo. A Figura 32.4 ilustra essa operação.

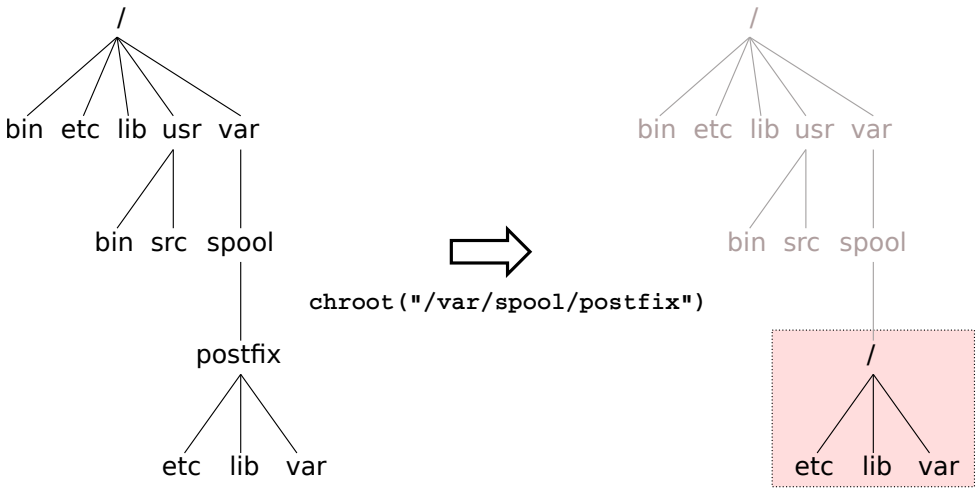

Figura 32.4: Funcionamento da chamada de sistema `chroot`.

A chamada de sistema `chroot` é muito utilizada para isolar processos que oferecem serviços à rede, como servidores DNS e de e-mail. Se um processo servidor

tiver alguma vulnerabilidade e for subvertido por um atacante, este só terá acesso ao conjunto de diretórios visíveis a esse processo, mantendo fora de alcance o restante da árvore de diretórios do sistema.

O sistema operacional *FreeBSD* oferece uma implementação de domínios virtuais mais elaborada, conhecida como *Jails* [McKusick and Neville-Neil, 2005] (aqui traduzidas como *celas*). Um processo que executa a chamada de sistema `jail()` cria uma nova cela e é colocado dentro dela, de onde não pode mais sair, nem seus filhos. Os processos dentro de uma cela estão submetidos a várias restrições:

- o processo só vê processos e recursos associados àquela cela; os demais processos, arquivos e outros recursos do sistema não associados à cela não são visíveis;
- somente são permitidas interações (comunicação e coordenação) entre processos dentro da mesma cela;
- de forma similar à chamada `chroot`, cada cela recebe uma árvore de diretórios própria; operações de montagem/desmontagem de sistemas de arquivos são proibidas;
- cada cela tem um endereço de rede associado, que é o único utilizável pelos processos da cela; a configuração de rede (endereço, parâmetros de interface, tabela de roteamento) não pode ser modificada;
- não podem ser feitas alterações no núcleo do sistema, como reconfigurações ou inclusões/exclusões de módulos.

Essas restrições são impostas a todos os processos dentro de uma cela, mesmo aqueles pertencentes ao administrador (usuário *root*). Assim, uma cela constitui uma unidade de isolamento bastante robusta, que pode ser usada para confinar serviços de rede e aplicações ou usuários considerados "perigosos".

Existem implementações de estruturas similares às celas do *FreeBSD* em outros sistemas operacionais. Por exemplo, o sistema operacional *Solaris* implementa o conceito de *zonas* [Price and Tucker, 2004], que oferecem uma capacidade de isolamento similar à celas, além de prover um mecanismo de controle da distribuição dos recursos entre as diferentes zonas existentes. Outras implementações podem ser encontradas para o sistema Linux, como os ambientes *Virtuozzo*, *Vservers*, *LXC* e *Docker*.

32.4 Máquinas virtuais de processo

Uma máquina virtual de processo, de aplicação ou de linguagem (*Process Virtual Machine*) suporta a execução de um processo ou aplicação individual. Ela é criada sob demanda, no momento do lançamento da aplicação convidada, e destruída quando a aplicação finaliza sua execução. O conjunto *hipervisor* + *aplicação* é normalmente visto como um único processo dentro do sistema operacional hospedeiro (ou um pequeno conjunto de processos), submetido às mesmas condições e restrições que os demais processos nativos.

Os hipervisores que implementam máquinas virtuais de processo normalmente permitem a interação entre a aplicação convidada e as demais aplicações do sistema, através dos mecanismos usuais de comunicação e coordenação entre processos, como

mensagens, *pipes* e semáforos. Além disso, também permitem o acesso normal ao sistema de arquivos e outros recursos locais do sistema. Estas características violam a propriedade de *isolamento* descrita na Seção 33.1, mas são necessárias para que a aplicação convidada se comporte como uma aplicação normal aos olhos do usuário.

Ao criar a máquina virtual para uma aplicação, o hipervisor pode implementar a mesma interface de hardware (ISA, Seção 31.2) da máquina real subjacente, ou implementar uma interface distinta. Quando a interface da máquina real é preservada, boa parte das instruções do processo convidado podem ser executadas diretamente sem perda de desempenho, com exceção das instruções sensíveis, que devem ser interpretadas pelo hipervisor.

Os exemplos mais comuns de máquinas virtuais de aplicação que preservam a interface ISA real são os *sistemas operacionais multitarefas*, os *tradutores dinâmicos* e alguns *depuradores de memória*:

Sistemas operacionais multitarefas: os sistemas operacionais que suportam vários processos simultâneos, estudados no Capítulo 4, também podem ser vistos como ambientes de máquinas virtuais. Em um sistema multitarefas, cada processo recebe um *processador virtual* (simulado através das fatias de tempo do processador real e das trocas de contexto), uma *memória virtual* (através do espaço de endereços mapeado para aquele processo) e *recursos físicos* (acessíveis através de chamadas de sistema). Este ambiente de virtualização é tão antigo e tão presente em nosso cotidiano que costumamos ignorá-lo como tal. No entanto, ele simplifica muito a tarefa dos programadores, que não precisam se preocupar com a gestão do isolamento e do compartilhamento de recursos entre os processos.

Tradutores dinâmicos: um tradutor dinâmico consiste em um hipervisor que analisa e otimiza um código executável, para tornar sua execução mais rápida e eficiente. A otimização não muda o conjunto de instruções da máquina real usado pelo código, apenas reorganiza as instruções de forma a acelerar sua execução. Por ser dinâmica, a otimização do código é feita durante a carga do processo na memória ou durante a execução de suas instruções, sendo transparente ao processo e ao usuário. O artigo [Duesterwald, 2005] apresenta uma descrição detalhada desse tipo de abordagem.

Depuradores de memória: alguns sistemas de depuração de erros de acesso à memória, como o sistema *Valgrind* [Seward and Nethercote, 2005], executam o processo sob depuração em uma máquina virtual. Todas as instruções do programa que manipulam acessos à memória são executadas de forma controlada, a fim de encontrar possíveis erros. Ao depurar um programa, o sistema *Valgrind* inicialmente traduz seu código binário em um conjunto de instruções interno, manipula esse código para inserir operações de verificação de acessos à memória e traduz o código modificado de volta ao conjunto de instruções da máquina real, para em seguida executá-lo e verificar os acessos à memória realizados.

Contudo, as máquinas virtuais de processo mais populares atualmente são aquelas em que a interface binária de aplicação (ABI, Seção 31.2) requerida pela aplicação é diferente daquela oferecida pela máquina real. Como a ABI é composta pelas chamadas do sistema operacional e as instruções de máquina disponíveis à aplicação (*user ISA*), as

diferenças podem ocorrer em ambos esses componentes. Nos dois casos, o hipervisor terá de fazer traduções dinâmicas (durante a execução) das ações requeridas pela aplicação em suas equivalentes na máquina real. Como visto, um hipervisor com essa função é denominado *tradutor dinâmico*.

Caso as diferenças de interface entre a aplicação e a máquina real se limitem às chamadas do sistema operacional, o hipervisor precisa mapear somente as chamadas de sistema e de bibliotecas usadas pela aplicação sobre as chamadas equivalentes oferecidas pelo sistema operacional da máquina real. Essa é a abordagem usada, por exemplo, pelo ambiente *Wine*, que permite executar aplicações Windows em plataformas Unix. As chamadas de sistema Windows emitidas pela aplicação em execução são interceptadas e convertidas em chamadas Unix, de forma dinâmica e transparente (Figura 32.5).

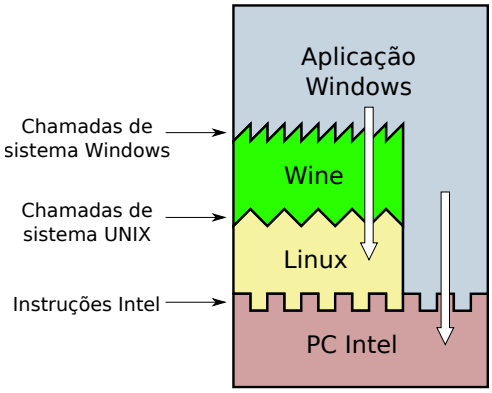

Figura 32.5: Funcionamento do emulador Wine.

Entretanto, muitas vezes a interface ISA utilizada pela aplicação não corresponde a nenhum hardware existente, mas a uma máquina abstrata. Um exemplo típico dessa situação ocorre na linguagem Java: um programa escrito em Java, ao ser compilado, gera um código binário específico para uma máquina abstrata denominada *máquina virtual Java* (*JVM – Java Virtual Machine*). A linguagem de máquina executada pela máquina virtual Java é denominada *bytecode* Java, e não corresponde a instruções de um processador real. A máquina virtual deve então interpretar todas as operações do *bytecode*, utilizando as instruções da máquina real subjacente para executá-las. Além de Java, várias linguagens empregam a mesma abordagem (embora com *bytecodes* distintos), como Lua, Python e C#.

Em termos de desempenho, um programa compilado para um processador abstrato executa mais lentamente que seu equivalente compilado para um processador real, devido ao custo de interpretação do *bytecode*. Todavia, essa abordagem oferece melhor desempenho que linguagens puramente interpretadas. Além disso, técnicas de otimização como a compilação *Just-in-Time* (JIT), na qual blocos de instruções frequentes são traduzidos e mantidos em cache pelo hipervisor, permitem obter ganhos de desempenho significativos.

Referências

E. Duesterwald. Design and engineering of a dynamic binary optimizer. *Proceedings of the IEEE*, 93(2):436–448, Feb 2005.

M. McKusick and G. Neville-Neil. *The Design and Implementation of the FreeBSD Operating System*. Pearson Education, 2005.

D. Price and A. Tucker. Solaris zones: Operating system support for consolidating commercial workloads. In *18th USENIX conference on System administration*, pages 241–254, 2004.

J. Seward and N. Nethercote. Using Valgrind to detect undefined value errors with bit-precision. In *USENIX Annual Technical Conference*, 2005.