Capítulo 34

Virtualização na prática

34.1 Aplicações da virtualização

Por permitir o acoplamento entre componentes de sistema com interfaces distintas, a virtualização tem um grande número de aplicações possíveis. As principais delas serão brevemente discutidas nesta seção:

Construção de binários portáveis: este uso da virtualização começou na década de 1970, com o compilador UCSD Pascal, que traduzia o código fonte Pascal em um código binário *P-Code*, para uma máquina virtual chamada *P-Machine*. A execução do código binário ficava então a cargo de uma implementação da *P-Machine* sobre a máquina alvo. Esse esquema foi posteriormente adotado por linguagens como Java, C#, Perl e Python, nas quais o código fonte é compilado em *bytecodes* para uma máquina virtual. Assim, uma aplicação Java compilada em *bytecode* pode executar em qualquer plataforma onde uma implementação da máquina virtual Java (*JVM - Java Virtual Machine*) esteja disponível.

Compartilhamento de hardware: executar simultaneamente várias instâncias de sistema operacional sobre a mesma plataforma de hardware. Uma área de aplicação dessa possibilidade é a chamada *consolidação de servidores*, que consiste em agrupar vários servidores de rede (web, e-mail, proxy, banco de dados, etc.) sobre o mesmo computador: ao invés de instalar vários computadores fisicamente isolados para abrigar cada um dos serviços, pode ser instalado um único computador, com maior capacidade, para suportar várias máquinas virtuais, cada uma abrigando um sistema operacional convidado e seu respectivo serviço.

Suporte a aplicações legadas: pode-se preservar ambientes virtuais com sistemas operacionais antigos para a execução de aplicações legadas, sem a necessidade de manter computadores reservados para isso.

Experimentação em redes: é possível construir uma rede de máquinas virtuais, comunicando por protocolos de rede como o TCP/IP, sobre um único computador hospedeiro. Isto torna possível o desenvolvimento e implantação de serviços de rede e de sistemas distribuídos sem a necessidade de uma rede real, o que é especialmente interessante em ensino e pesquisa.

Ensino: em disciplinas de rede e de sistema, um aluno deve ter a possibilidade de modificar as configurações da máquina para poder realizar seus experimentos. Essa possibilidade é uma verdadeira "dor de cabeça" para os administradores de laboratórios de ensino. Todavia, um aluno pode lançar uma máquina virtual e ter controle completo sobre ela, mesmo não tendo acesso às configurações da máquina real subjacente.

Segurança: a propriedade de isolamento provida pelo hipervisor torna esta abordagem útil para isolar domínios, usuários e/ou aplicações não confiáveis. As máquinas virtuais de sistema operacional (Seção 32.3) foram criadas justamente com o objetivo de isolar subsistemas particularmente críticos, como servidores Web, DNS e de e-mail. Pode-se também usar máquinas virtuais como plataforma de execução de programas suspeitos, para inspecionar seu funcionamento e seus efeitos sobre o sistema operacional convidado.

Desenvolvimento de baixo nível: o uso de máquinas virtuais para o desenvolvimento partes do núcleo do sistema operacional, módulos e protocolos de rede, tem vários benefícios com o uso de máquinas virtuais. Por exemplo, o desenvolvimento e os testes podem ser feitos sobre a mesma plataforma. Outra vantagem visível é o menor tempo necessário para instalar e lançar um núcleo em uma máquina virtual, quando comparado a uma máquina real. Por fim, a execução em uma máquina virtual pode ser melhor acompanhada e depurada que a execução equivalente em uma máquina real.

Tolerância a faltas: muitos hipervisores oferecem suporte ao *checkpointing*, ou seja, à possibilidade de salvar o estado interno de uma máquina virtual e de poder restaurá-lo posteriormente. Com *checkpoints* periódicos, torna-se possível retornar a execução de uma máquina virtual a um estado salvo anteriormente, em caso de falhas ou incidentes de segurança.

Computação sob demanda: o desacoplamento entre o hardware real e o sistema operacional proporcionado pelas máquinas virtuais as tornou um suporte adequado para a oferta de serviços através da rede. Na computação em nuvem, imensas centrais de processamento de dados alugam máquinas virtuais que podem ser instanciadas, configuradas e destruídas por seus clientes sob demanda, conforme sua necessidade no momento.

Virtualização de redes: máquinas virtuais têm sido usadas para implementar dispositivos de rede como roteadores, *switches* e *firewalls*, em uma abordagem chamada *Virtualização de Funções de Rede* (NFV - *Network Function Virtualization*), diminuindo a quantidade de hardware proprietário na infraestrutura de rede e agilizando operações de reconfiguração da rede (que passam a ser feitas exclusivamente por software).

34.2 Ambientes de máquinas virtuais

Esta seção apresenta alguns exemplos de sistemas de máquinas virtuais de uso corrente. Entre eles há máquinas virtuais de aplicação e de sistema, com virtualização total ou paravirtualização, além de abordagens híbridas. Eles foram escolhidos por estarem entre os mais representativos de suas respectivas classes.

34.2.1 VMware

O hipervisor da *VMware* é um dos mais difundidos, provendo uma implementação completa da interface x86 ao sistema convidado. Embora essa interface seja extremamente genérica para o sistema convidado, acaba conduzindo a um hipervisor mais complexo. Como podem existir vários sistemas operacionais em execução sobre mesmo hardware, o hipervisor tem que emular certas instruções para representar corretamente um processador virtual em cada máquina virtual, fazendo uso intensivo dos mecanismos de tradução dinâmica [VMware, 2000; Newman et al., 2005]. Atualmente, a empresa *VMware* produz vários hipervisores, entre eles:

- *VMware Workstation*: hipervisor convidado para ambientes *desktop*;
- *VMware ESXi Server*: hipervisor nativo para servidores de grande porte, possui um núcleo proprietário chamado *vmkernel* e utiliza *Linux* para prover outros serviços, tais como a gerência de usuários.

O *VMware Workstation* utiliza as estratégias de virtualização total, tradução dinâmica (Seção 33.4) e o suporte de hardware, quando disponível. O *VMware ESXi Server* implementa também a paravirtualização. Por razões de desempenho, o hipervisor do *VMware* utiliza uma abordagem híbrida (Seção 33.5) para implementar a interface do hipervisor com as máquinas virtuais [Sugerman et al., 2001]. O controle de exceção e o gerenciamento de memória são realizados por acesso direto ao hardware, mas o controle de entrada/saída usa o sistema hospedeiro. Para garantir que não ocorra nenhuma colisão de memória entre o sistema convidado e o real, o hipervisor *VMware* aloca uma parte da memória para uso exclusivo de cada sistema convidado.

Para controlar o sistema convidado, o *VMware Workstation* intercepta todas as interrupções do sistema convidado. Sempre que uma exceção é causada no convidado, é examinada primeiro pelo hipervisor. As interrupções de entrada/saída são remetidas para o sistema hospedeiro, para que sejam processadas corretamente. As exceções geradas pelas aplicações no sistema convidado (como as chamadas de sistema, por exemplo) são remetidas para o sistema convidado.

34.2.2 Xen

O ambiente *Xen* é um hipervisor nativo para a plataforma *x86* que implementa a paravirtualização. Ele permite executar sistemas operacionais como Linux e Windows especialmente modificados para executar sobre o hipervisor [Barham et al., 2003]. Versões mais recentes do sistema *Xen* utilizam o suporte de virtualização disponível nos processadores atuais, o que torna possível a execução de sistemas operacionais convidados sem modificações, embora com um desempenho ligeiramente menor que no caso de sistemas paravirtualizados. De acordo com seus desenvolvedores, o custo e impacto das alterações nos sistemas convidados são baixos e a diminuição do custo da virtualização compensa essas alterações: a degradação média de desempenho observada em sistemas virtualizados sobre a plataforma *Xen* não excede 5%. As principais modificações impostas pelo ambiente *Xen* a um sistema operacional convidado são:

- O mecanismo de entrega de interrupções passa a usar um serviço de eventos oferecido pelo hipervisor; o núcleo convidado deve registrar uma tabela de tratadores de exceções junto ao hipervisor;

- as operações de entrada/saída de dispositivos são feitas através de uma interface simplificada, independente de dispositivo, que usa *buffers* circulares de tipo produtor/consumidor;
- o núcleo convidado pode consultar diretamente as tabelas de segmentos e páginas da memória usada por ele e por suas aplicações, mas as modificações nas tabelas devem ser solicitadas ao hipervisor;
- o núcleo convidado deve executar em um nível de privilégio inferior ao do hipervisor;
- o núcleo convidado deve implementar uma função de tratamento das chamadas de sistema de suas aplicações, para evitar que elas tenham de passar pelo hipervisor antes de chegar ao núcleo convidado.

Como o hipervisor deve acessar os dispositivos de hardware, ele deve dispor dos *drivers* adequados. Já os núcleos convidados não precisam de *drivers* específicos, pois eles acessam dispositivos virtuais através de uma interface simplificada. Para evitar o desenvolvimento de *drivers* específicos para o hipervisor, o ambiente *Xen* usa uma abordagem alternativa: a primeira máquina virtual (chamada VM_0) pode acessar o hardware diretamente e provê os *drivers* necessários ao hipervisor. As demais máquinas virtuais ($VM_i, i > 0$) acessam o hardware virtual através do hipervisor, que usa os *drivers* da máquina VM_0 conforme necessário. Essa abordagem, apresentada na Figura 34.1, simplifica muito a evolução do hipervisor, por permitir utilizar os *drivers* desenvolvidos para o sistema Linux.

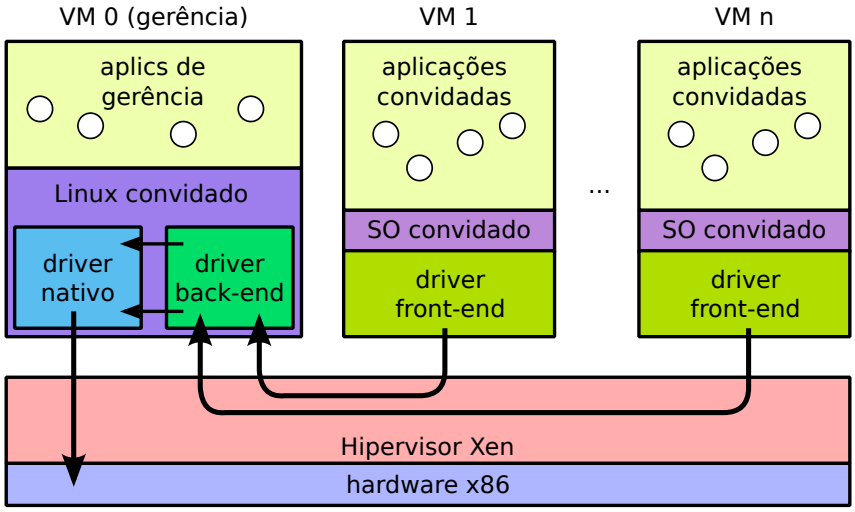

Figura 34.1: O hipervisor *Xen*.

O hipervisor *Xen* pode ser considerado uma tecnologia madura, sendo muito utilizado em sistemas de produção. O seu código fonte está liberado sob a licença *GNU General Public Licence* (GPL). Atualmente, o ambiente *Xen* suporta os sistemas Windows, Linux e NetBSD. Várias distribuições Linux já possuem suporte nativo ao *Xen*.

34.2.3 QEMU

O *QEMU* é um emulador de hardware [Bellard, 2005]. Não requer alterações ou otimizações no sistema hospedeiro, pois utiliza intensivamente a tradução dinâmica

(Seção 33.4) como técnica para prover a virtualização. O *QEMU* oferece dois modos de operação:

- *Emulação total do sistema*: emula um sistema completo, incluindo processador (normalmente um processador no padrão x86) e vários periféricos. Neste modo o emulador pode ser utilizado para executar diferentes sistemas operacionais;
- *Emulação no modo de usuário*: disponível apenas para o sistema Linux. Neste modo o emulador pode executar processos Linux compilados em diferentes plataformas (por exemplo, um programa compilado para um processador *x86* pode ser executado em um processador *PowerPC* e vice-versa).

Durante a emulação de um sistema completo, o *QEMU* implementa uma MMU (*Memory Management Unit*) totalmente em software, para garantir o máximo de portabilidade. Quando em modo usuário, o *QEMU* simula uma MMU simplificada através da chamada de sistema `mmap` (que permite mapear um arquivo em uma região da memória) do sistema hospedeiro.

O *VirtualBox* [VirtualBox, 2008] é um ambiente de máquinas virtuais construído sobre o hipervisor *QEMU*. Ele é similar ao *VMware Workstation* em muitos aspectos. Atualmente, pode tirar proveito do suporte à virtualização disponível nos processadores Intel e AMD. Originalmente desenvolvido pela empresa *Innotek*, o *VirtualBox* foi depois adquirido pela *Sun Microsystems* e liberado para uso público sob a licença *GPLv2*.

34.2.4 KVM

O KVM (*Kernel-based Virtual Machine*) é um hipervisor convidado (de tipo 2) embutido no núcleo Linux desde sua versão 2.6.20 [Kivity et al., 2007]. Ele foi construído com o objetivo de explorar as extensões de virtualização disponíveis nos processadores modernos para construir máquinas virtuais no *userspace* do Linux. Embora seja construído para hospedeiros Linux, o KVM suporta vários sistemas convidados distintos.

No ambiente KVM, cada máquina virtual dispõe de um conjunto de CPUs virtuais (VCPUs) e de recursos paravirtualizados, como interfaces de rede e de disco, que são implementados pelo hipervisor. Além disso, hipervisor reaproveita grande parte dos mecanismos do núcleo Linux subjacente para realizar o gerenciamento de memória, escalonamento de CPUs e de entrada/saída.

Por default, o KVM realiza a paravirtualização, ou seja, emula as operações de entrada/saída e executa as demais instruções do sistema convidado diretamente pelo processador, usando as extensões de virtualização quando necessário. Entretanto, Se o sistema convidado precisar de um processador distinto do usado pelo sistema hospedeiro, o KVM usa o emulador QEMU para virtualizar a execução das instruções do sistema convidado.

Em termos de implementação, o KVM consiste de um conjunto de módulos de núcleo e de um arquivo de controle (`/dev/kvm`). No espaço de usuário, o KVM pode fazer uso do QEMU e também de uma biblioteca para o gerenciamento de máquinas virtuais chamada *LibVirt*. Os sistemas convidados deve acessar os dispositivos paravirtualizados através de uma biblioteca específica chamada *VirtIO*.

34.2.5 Docker

Docker [Merkel, 2014] é um *framework* para a construção e gestão de contêineres (máquinas virtuais de sistema operacional). Nesse *framework*, uma aplicação corporativa complexa pode ser "empacotada" em um contêiner com todas as suas dependências (bibliotecas, serviços auxiliares, etc), agilizando e simplificando sua distribuição, implantação e configuração. Essa abordagem se mostrou excelente para a distribuição e ativação de serviços em nuvens computacionais.

Para construir e manter o ambiente de virtualização, o *framework* Docker usa várias tecnologias providas pelo núcleo Linux, como os suportes de contêiner LXC (*Linux Containers*) e *systemd-nspawn*, o sistema de arquivos em camadas *OverlayFS* e os mecanismos de isolamento e gestão de recursos *cgroups* e *namespaces*.

34.2.6 JVM

Tendo sido originalmente concebida para o desenvolvimento de pequenos aplicativos e programas de controle de aparelhos eletroeletrônicos, a linguagem Java mostrou-se ideal para ser usada na Internet. O que a torna tão atraente é o fato de programas escritos nessa linguagem de programação poderem ser executados em praticamente qualquer plataforma.

A virtualização é o fator responsável pela independência dos programas Java do hardware e dos sistemas operacionais: 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 nenhum processador real. A máquina virtual deve então interpretar todas as operações do *bytecode*, utilizando as instruções da máquina real subjacente para executá-las.

A vantagem mais significativa da abordagem adotada por Java é a *portabilidade* do código executável: para que uma aplicação Java possa executar sobre uma determinada plataforma, basta que a máquina virtual Java esteja disponível ali (na forma de um suporte de execução denominado JRE - *Java Runtime Environment*). Assim, a portabilidade dos programas Java depende unicamente da portabilidade da própria máquina virtual Java.

O suporte de execução Java pode estar associado a um navegador Web, o que permite que código Java seja associado a páginas Web, na forma de pequenas aplicações denominadas *applets*, que são trazidas junto com os demais componentes de página Web e executam localmente no navegador. A Figura 34.2 mostra os principais componentes da plataforma Java.

É importante ressaltar que a adoção de uma máquina virtual como suporte de execução não é exclusividade da linguagem Java, nem foi inventada por seus criadores. As primeiras experiências de execução de aplicações sobre máquinas abstratas remontam aos anos 1970, com a linguagem *UCSD Pascal*. Hoje, muitas linguagens adotam estratégias similares, como Java, C#, Python, Perl, Lua e Ruby. Em C#, o código fonte é compilado em um formato intermediário denominado CIL (*Common Intermediate Language*), que executa sobre uma máquina virtual CLR (*Common Language Runtime*). CIL e CLR fazem parte da infraestrutura .NET da Microsoft.

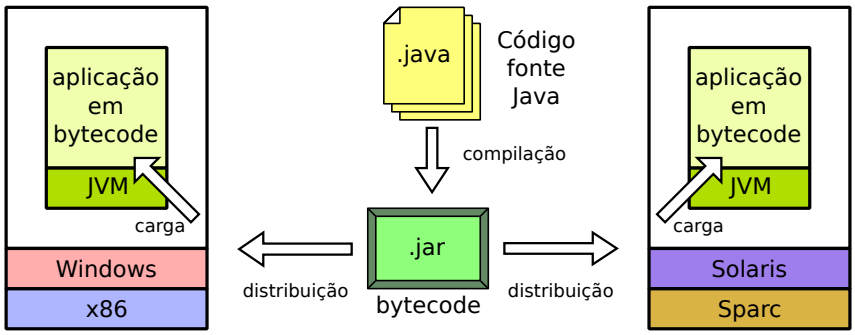

Figura 34.2: Máquina virtual Java.

34.2.7 FreeBSD Jails

O sistema operacional *FreeBSD* oferece um mecanismo de confinamento de processos denominado *Jails*, criado para aumentar a segurança de serviços de rede. Esse mecanismo consiste em criar domínios de execução distintos (denominados *jails* ou celas), conforme descrito na Seção 32.3. Cada cela contém um subconjunto de processos e recursos (arquivos, conexões de rede) que pode ser gerenciado de forma autônoma, como se fosse um sistema separado [McKusick and Neville-Neil, 2005].

Cada domínio é criado a partir de um diretório previamente preparado no sistema de arquivos. 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. Além disso, os processos em um domínio não podem:

- Reconfigurar o núcleo (através da chamada `sysctl`, por exemplo);
- Carregar/retirar módulos do núcleo;
- Mudar configurações de rede (interfaces e rotas);
- Montar/desmontar sistemas de arquivos;
- Criar novos *devices*;
- Realizar modificações de configurações do núcleo em tempo de execução;
- Acessar recursos que não pertençam ao seu próprio domínio.

Essas restrições se aplicam mesmo a processos que estejam executando com privilégios de administrador (*root*).

Pode-se considerar que o sistema *FreeBSD Jails* virtualiza somente partes do sistema hospedeiro, como a árvore de diretórios (cada domínio tem sua própria visão do sistema de arquivos), espaços de nomes (cada domínio mantém seus próprios identificadores de usuários, processos e recursos de IPC) e interfaces de rede (cada domínio tem sua interface virtual, com endereço de rede próprio). Os demais recursos (como as instruções de máquina e chamadas de sistema) são preservadas, ou melhor, podem ser usadas diretamente. Essa virtualização parcial demanda um custo computacional muito baixo, mas exige que todos os sistemas convidados executem sobre o mesmo núcleo.

34.2.8 Valgrind

O *Valgrind* [Nethercote and Seward, 2007] é uma ferramenta de depuração de uso da memória RAM e problemas correlatos. Ele permite investigar vazamentos de memória (*memory leaks*), acessos a endereços inválidos, padrões de uso dos caches e outras operações envolvendo o uso da memória RAM. O *Valgrind* foi desenvolvido para plataforma *x86 Linux*, mas existem versões experimentais para outras plataformas.

Tecnicamente, o *Valgrind* é um hipervisor de aplicação que virtualiza o processador através de técnicas de tradução dinâmica. Ao iniciar a análise de um programa, o *Valgrind* traduz o código executável do mesmo para um formato interno independente de plataforma denominado IR (*Intermediate Representation*). Após a conversão, o código em IR é instrumentado, através da inserção de instruções para registrar e verificar as operações de alocação, acesso e liberação de memória. A seguir, o programa IR devidamente instrumentado é traduzido no formato binário a ser executado sobre o processador virtual. O código final pode ser até 50 vezes mais lento que o código original, mas essa perda de desempenho normalmente não é muito relevante durante a análise ou depuração de um programa.

34.2.9 User-Mode Linux

O *User-Mode Linux* é um hipervisor simples, proposto por Jeff Dike em 2000 [Dike, 2000]. Nele, o núcleo do Linux foi portado de forma a poder executar sobre si mesmo, como um processo do próprio Linux. O resultado é um *userspace* separado e isolado na forma de uma máquina virtual, que utiliza dispositivos de hardware virtualizados a partir dos serviços providos pelo sistema hospedeiro.

Essa máquina virtual é capaz de executar todos os serviços e aplicações disponíveis para o sistema hospedeiro. Além disso, o custo de processamento e de memória das máquinas virtuais *User-Mode Linux* é geralmente menor que aquele imposto por outros hipervisores mais complexos. O *User-Mode Linux* está integrado ao núcleo Linux desde a versão 2.6 deste.

O UML implementa um hipervisor convidado, que executa na forma de um processo no sistema hospedeiro. Os processos em execução na máquina virtual não têm acesso direto aos recursos do sistema hospedeiro. A implementação do UML se baseia na interceptação das chamadas de sistema emitidas pelo convidado. Essa interceptação é realizada através da chamada de sistema `ptrace`, que permite observar e controlar a execução de outros processos. Assim, o hipervisor recebe o controle de todas as chamadas de sistema de entrada/saída geradas pelas máquinas virtuais. Além disso, todos os sinais gerados ou enviados às máquinas virtuais também são interceptados.

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.

F. Bellard. QEMU, a fast and portable dynamic translator. In *USENIX Annual Technical Conference*, 2005.

J. Dike. A user-mode port of the Linux kernel. In *Proceedings of the 4th Annual Linux Showcase & Conference*, 2000.

A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: the Linux virtual machine monitor. In *Proceedings of the Linux symposium*, volume 1, pages 225–230, 2007.

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

D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

N. Nethercote and J. Seward. Valgrind: A framework for heavyweight dynamic binary instrumentation. In *ACM Conference on Programming Language Design and Implementation*, San Diego - California - USA, june 2007.

M. Newman, C.-M. Wiberg, and B. Braswell. *Server Consolidation with VMware ESX Server*. IBM RedBooks, 2005. http://www.redbooks.ibm.com.

J. Sugerman, G. Venkitachalam, and B. H. Lim. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *USENIX Annual Technical Conference*, pages 1–14, 2001.

I. VirtualBox. The VirtualBox architecture. http://www.virtualbox.org/wiki/VirtualBox_architecture, 2008.

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