Capítulo 31

O conceito de virtualização

As tecnologias de virtualização do ambiente de execução de aplicações ou de plataformas de hardware têm sido objeto da atenção crescente de pesquisadores, fabricantes de hardware/software, administradores de sistemas e usuários avançados. A virtualização de recursos é um conceito relativamente antigo, mas os recentes avanços nessa área permitem usar máquinas virtuais com os mais diversos objetivos, como a segurança, a compatibilidade de aplicações legadas ou a consolidação de servidores. Este capítulo apresenta os principais conceitos relacionados à virtualização.

31.1 Um breve histórico

O conceito de máquina virtual não é recente. Os primeiros passos na construção de ambientes de máquinas virtuais começaram na década de 1960, quando a IBM desenvolveu o sistema operacional experimental M44/44X. A partir dele, a IBM desenvolveu vários sistemas comerciais suportando virtualização, entre os quais o famoso OS/370 [Goldberg, 1973; Goldberg and Mager, 1979]. A tendência dominante nos sistemas naquela época era fornecer a cada usuário um ambiente monousuário completo, com seu próprio sistema operacional e aplicações, completamente independente e desvinculado dos ambientes dos demais usuários.

Na década de 1970, os pesquisadores Popek & Goldberg formalizaram vários conceitos associados às máquinas virtuais, e definiram as condições necessárias para que uma plataforma de hardware suporte de forma eficiente a virtualização [Popek and Goldberg, 1974]; essas condições são discutidas em detalhe no Capítulo 33.1. Nessa mesma época surgem as primeiras experiências concretas de utilização de máquinas virtuais para a execução de aplicações, com o ambiente *UCSD p-System*, no qual programas Pascal eram compilados para execução sobre um hardware virtual denominado *P-Machine*.

Na década de 1980, com a popularização de plataformas de hardware baratas como o PC, a virtualização perdeu importância. Afinal, era mais barato, simples e versátil fornecer um computador completo a cada usuário, que investir em sistemas de grande porte, caros e complexos. Além disso, o hardware do PC tinha desempenho modesto e não provia suporte adequado à virtualização, o que inibiu o uso de ambientes virtuais nessas plataformas.

Com o aumento de desempenho e funcionalidades do hardware PC e o surgimento da linguagem Java, nos anos 90, o interesse pelas tecnologias de virtualização voltou à tona. Apesar da plataforma PC Intel na época não oferecer um suporte ade-

quado à virtualização, soluções engenhosas como as adotadas pela empresa VMware permitiram a virtualização nessa plataforma, embora com desempenho relativamente modesto. Nos anos 2000 as soluções de virtualização de plataformas despertou grande interesse do mercado, devido à consolidação de servidores e à construção das nuvens computacionais. Hoje, várias linguagens são compiladas para máquinas virtuais portáveis e os processadores mais recentes trazem um suporte nativo à virtualização do hardware.

31.2 Interfaces de sistema

Uma máquina real é formada por vários componentes físicos que fornecem operações para o sistema operacional e suas aplicações. Iniciando pelo núcleo do sistema real, o processador central (CPU) e o *chipset* da placa-mãe fornecem um conjunto de instruções e outros elementos fundamentais para o processamento de dados, alocação de memória e processamento de entrada/saída. Os sistemas de computadores são projetados com basicamente três componentes: hardware, sistema operacional e aplicações. O papel do hardware é executar as operações solicitadas pelas aplicações através do sistema operacional. O sistema operacional recebe as solicitações das operações (por meio das chamadas de sistema) e controla o acesso ao hardware – principalmente nos casos em que os componentes são compartilhados, como a memória e os dispositivos de entrada/saída.

Os sistemas de computação convencionais são caracterizados por níveis de abstração crescentes e interfaces bem definidas entre eles. As abstrações oferecidas pelo sistema às aplicações são construídas de forma incremental, em níveis separados por interfaces bem definidas e relativamente padronizadas. Cada interface encapsula as abstrações dos níveis inferiores, permitindo assim o desenvolvimento independente dos vários níveis, o que simplifica a construção e evolução dos sistemas. As interfaces existentes entre os componentes de um sistema de computação típico são:

Conjunto de instruções (*ISA – Instruction Set Architecture*): é a interface básica entre o hardware e o software, sendo constituída pelas instruções de máquina aceitas pelo processador e as operações de acesso aos recursos do hardware (acesso físico à memória, às portas de entrada/saída, ao relógio do hardware, etc.). Essa interface é dividida em duas partes:

- **Instruções de usuário (*User ISA*):** compreende as instruções do processador e demais itens de hardware acessíveis aos programas do usuário, que executam com o processador operando em modo usuário;
- **Instruções de sistema (*System ISA*):** compreende as instruções do processador e demais itens de hardware unicamente acessíveis ao núcleo do sistema operacional, que executa em modo privilegiado;

Chamadas de sistema (*syscalls*): é o conjunto de operações oferecidas pelo núcleo do sistema operacional aos processos no espaço de usuário. Essas chamadas permitem o acesso controlado das aplicações aos dispositivos periféricos, à memória e às instruções privilegiadas do processador.

Chamadas de bibliotecas (*libcalls*): as bibliotecas oferecem um grande número de funções para simplificar a construção de programas; além disso, muitas chamadas de biblioteca encapsulam chamadas do sistema operacional, para tornar seu uso mais simples. Cada biblioteca possui uma interface própria, denominada *Interface de Programação de Aplicações* (*API – Application Programming Interface*). Exemplos típicos de bibliotecas são a *LibC* do UNIX (que oferece funções como `fopen` e `printf`), a GTK+ (*Gimp ToolKit*, que permite a construção de interfaces gráficas) e a SDL (*Simple DirectMedia Layer*, para a manipulação de áudio e vídeo).

A Figura 31.1 apresenta essa visão conceitual da arquitetura de um sistema computacional, com seus vários componentes e as respectivas interfaces entre eles.

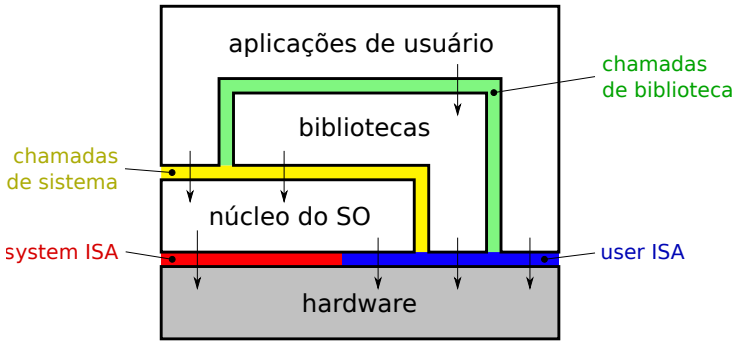

Figura 31.1: Componentes e interfaces de um sistema computacional.

31.3 Compatibilidade entre interfaces

Para que programas e bibliotecas possam executar sobre uma determinada plataforma, é necessário que tenham sido compilados para ela, respeitando o conjunto de instruções do processador em modo usuário (*User ISA*) e o conjunto de chamadas de sistema oferecido pelo sistema operacional. A visão conjunta dessas duas interfaces (*User ISA* + *syscalls*) é denominada *Interface Binária de Aplicação* (*ABI – Application Binary Interface*). Da mesma forma, um sistema operacional só poderá executar sobre uma plataforma de hardware se tiver sido construído e compilado de forma a respeitar sua interface ISA (*User/System ISA*). A Figura 31.2 representa essas duas interfaces.

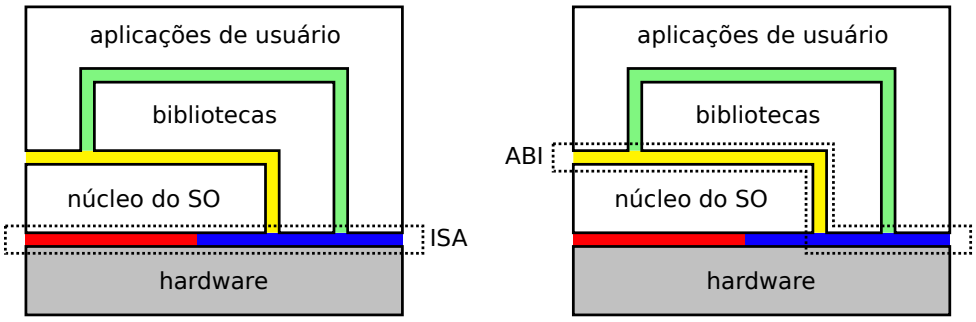

Figura 31.2: Interfaces de sistema ISA e ABI [Smith and Nair, 2004].

Nos sistemas computacionais de mercado atuais, as interfaces de baixo nível ISA e ABI são normalmente fixas, ou pouco flexíveis. Geralmente não é possível criar novas instruções de processador ou novas chamadas de sistema operacional, ou mesmo mudar sua semântica para atender às necessidades específicas de uma determinada aplicação. Mesmo se isso fosse possível, teria de ser feito com cautela, para não comprometer o funcionamento de outras aplicações.

Os sistemas operacionais, assim como as aplicações, são projetados para aproveitar o máximo dos recursos que o hardware fornece. Normalmente os projetistas de hardware, sistema operacional e aplicações trabalham de forma independente (em empresas e tempos diferentes). Por isso, esses trabalhos independentes geraram, ao longo dos anos, várias plataformas computacionais diferentes e incompatíveis entre si.

Observa-se então que, embora a definição de interfaces seja útil, por facilitar o desenvolvimento independente dos vários componentes do sistema, torna pouco flexíveis as interações entre eles: um sistema operacional só funciona sobre o hardware (ISA) para o qual foi construído, uma biblioteca só funciona sobre a ABI para a qual foi projetada e uma aplicação tem de obedecer a ABIs e APIs predefinidas. A Figura 31.3, extraída de [Smith and Nair, 2004], ilustra esses problemas de compatibilidade entre interfaces.

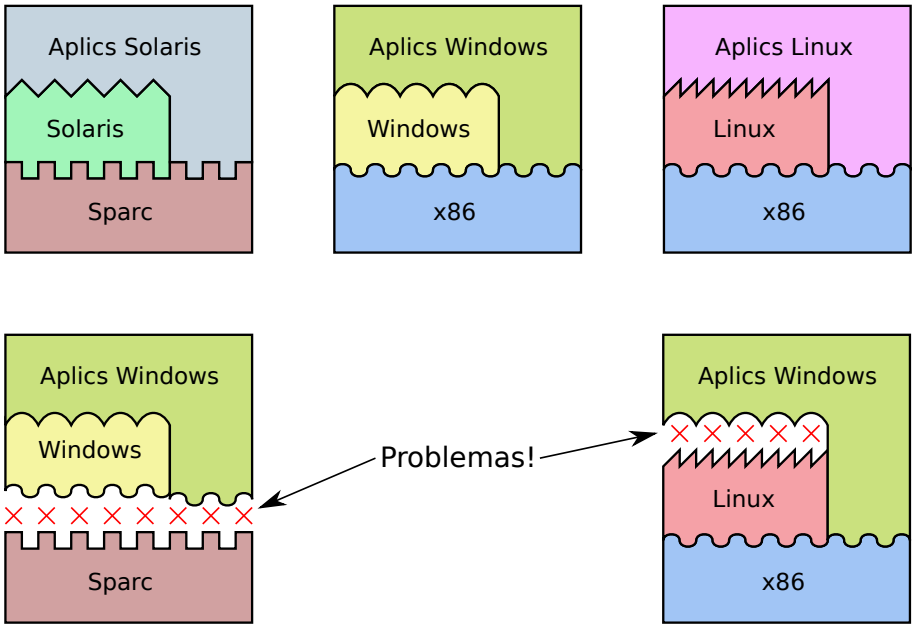

Figura 31.3: Problemas de compatibilidade entre interfaces [Smith and Nair, 2004].

A baixa flexibilidade na interação entre as interfaces dos componentes de um sistema computacional traz vários problemas [Smith and Nair, 2004]:

- *Baixa portabilidade*: a mobilidade de código e sua interoperabilidade são requisitos importantes dos sistemas atuais, que apresentam grande conectividade de rede e diversidade de plataformas. A rigidez das interfaces de sistema atuais dificulta sua construção, por acoplar excessivamente as aplicações aos sistemas operacionais e aos componentes do hardware.

- *Barreiras de inovação*: a presença de interfaces rígidas dificulta a construção de novas formas de interação entre as aplicações e os dispositivos de hardware (e

com os usuários, por consequência). Além disso, as interfaces apresentam uma grande inércia à evolução, por conta da necessidade de suporte às aplicações já existentes.

- *Otimizações intercomponentes*: aplicações, bibliotecas, sistemas operacionais e hardware são desenvolvidos por grupos distintos, geralmente com pouca interação entre eles. A presença de interfaces rígidas a respeitar entre os componentes leva cada grupo a trabalhar de forma isolada, o que diminui a possibilidade de otimizações que envolvam mais de um componente.

Essas dificuldades levaram à investigação de outras formas de relacionamento entre os componentes de um sistema computacional. Uma das abordagens mais promissoras nesse sentido é o uso da virtualização de interfaces, discutida a seguir.

31.4 Virtualização de interfaces

Conforme visto, as interfaces padronizadas entre os componentes do sistema de computação permitem o desenvolvimento independente dos mesmos, mas também são fonte de problemas de interoperabilidade, devido à sua pouca flexibilidade. Por isso, não é possível executar diretamente em um processador Intel/AMD uma aplicação compilada para um processador ARM: as instruções em linguagem de máquina da aplicação não serão compreendidas pelo processador Intel. Da mesma forma, não é possível executar diretamente em Linux uma aplicação escrita para um sistema Windows, pois as chamadas de sistema emitidas pelo programa Windows não serão compreendidas pelo sistema operacional Linux subjacente.

Todavia, é possível contornar esses problemas de compatibilidade através de uma *camada de virtualização* construída em software. Usando os serviços oferecidos por uma determinada interface de sistema, é possível construir uma camada de software que ofereça aos demais componentes uma outra interface. Essa camada de software permitirá o acoplamento entre interfaces distintas, de forma que um programa desenvolvido para a plataforma *A* possa executar sobre uma plataforma distinta *B*.

Usando os serviços oferecidos por uma determinada interface de sistema, a camada de virtualização constrói outra interface de mesmo nível, de acordo com as necessidades dos componentes de sistema que farão uso dela. A nova interface de sistema, vista através dessa camada de virtualização, é denominada *máquina virtual*. A camada de virtualização em si é denominada *hipervisor* (ou *monitor de máquina virtual*).

A Figura 31.4, extraída de [Smith and Nair, 2004], apresenta um exemplo de máquina virtual, onde um hipervisor permite executar um sistema operacional Windows e suas aplicações sobre uma plataforma de hardware Sparc, distinta daquela para a qual esse sistema operacional foi projetado (Intel x86).

Um ambiente de máquina virtual consiste de três partes básicas, que podem ser observadas na Figura 31.4:

- O sistema real, nativo ou hospedeiro (*host system*), que contém os recursos reais de hardware do sistema;

- a camada de virtualização, chamada *hipervisor* ou *monitor* (VMM – *Virtual Machine Monitor*), que constrói a interface virtual a partir da interface real;

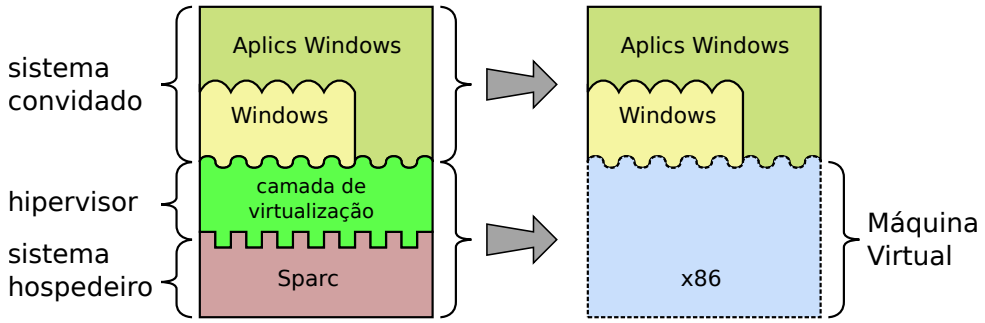

Figura 31.4: Uma máquina virtual [Smith and Nair, 2004].

- o sistema virtual, também chamado *sistema convidado* (*guest system*), que executa sobre a camada de virtualização.

É importante ressaltar a diferença entre os termos *virtualização* e *emulação*. A emulação é na verdade uma forma de virtualização: quando um hipervisor virtualiza integralmente uma interface de hardware ou de sistema operacional, é geralmente chamado de *emulador*. Por exemplo, a máquina virtual Java, que constrói um ambiente completo para a execução de *bytecodes* a partir de um processador real que não executa *bytecodes*, pode ser considerada um emulador.

A virtualização abre uma série de possibilidades interessantes para a composição de um sistema de computação, como por exemplo (Figura 31.5):

- *Emulação de hardware*: um sistema operacional convidado e suas aplicações, desenvolvidas para uma plataforma de hardware A, são executadas sobre uma plataforma de hardware distinta B.

- *Emulação de sistema operacional*: aplicações construídas para um sistema operacional X são executadas sobre outro sistema operacional Y.

- *Otimização dinâmica*: as instruções de máquina das aplicações são traduzidas durante a execução em outras instruções mais eficientes para a mesma plataforma.

- *Replicação de hardware*: são criadas várias instâncias virtuais de um mesmo hardware real, cada uma executando seu próprio sistema operacional convidado e suas respectivas aplicações.

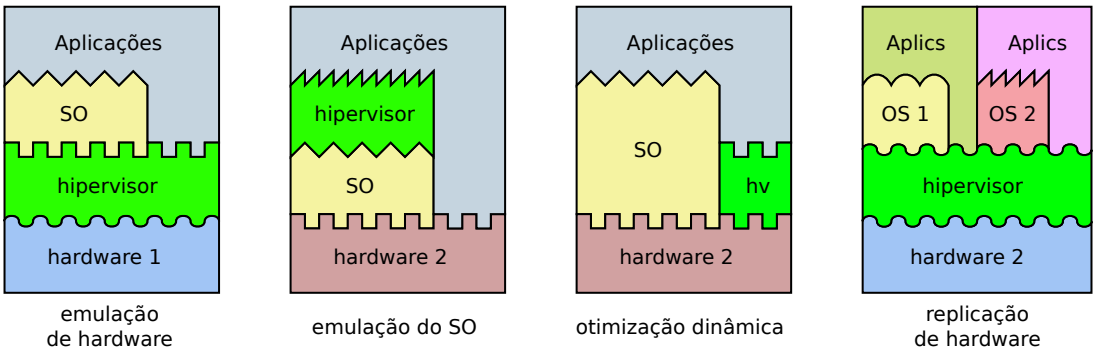

Figura 31.5: Possibilidades de virtualização [Smith and Nair, 2004].

31.5 Virtualização versus abstração

Embora a virtualização possa ser vista como um tipo de abstração, existe uma clara diferença entre os termos "abstração" e "virtualização", no contexto de sistemas operacionais [Smith and Nair, 2004]. Um dos principais objetivos do sistema operacional é oferecer uma visão de alto nível dos recursos de hardware, que seja mais simples de usar e menos dependente das tecnologias subjacentes. Essa visão abstrata dos recursos é construída de forma incremental, em níveis de abstração crescentes.

No subsistema de arquivos, por exemplo, cada nível de abstração trata de um problema: interação com o dispositivo físico, escalonamento de acessos ao dispositivo, gerência de *buffers* e *caches*, alocação de arquivos, diretórios, controle de acesso, etc. Essa estrutura em camadas é discutida em detalhes no capítulo 24.

Por outro lado, a virtualização consiste em criar novas interfaces a partir das interfaces existentes. Na virtualização, os detalhes de baixo nível da plataforma real não são necessariamente ocultos, como ocorre na abstração de recursos. A Figura 31.6 ilustra essa diferença: através da virtualização, um processador Sparc pode ser visto pelo sistema convidado como um processador Intel. Da mesma forma, um disco real no padrão SATA pode ser visto como vários discos menores independentes, com a mesma interface (SATA) ou outra interface (IDE).

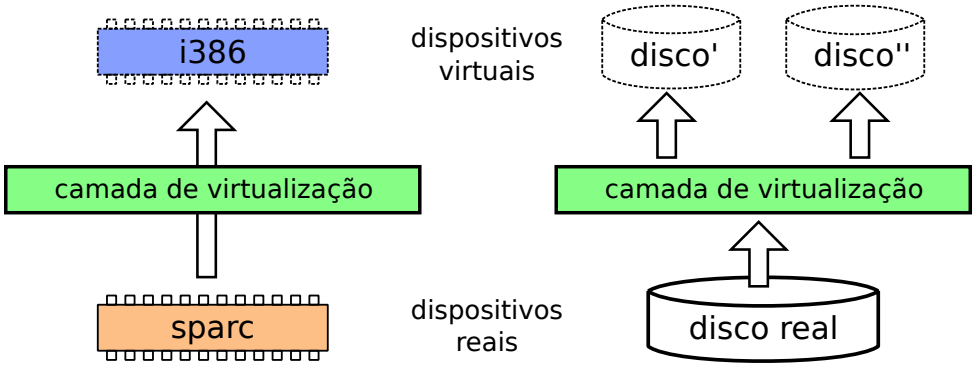

Figura 31.6: Virtualização de recursos do hardware.

A Figura 31.7 ilustra outro exemplo dessa diferença no contexto do armazenamento em disco. A abstração provê às aplicações o conceito de "arquivo", sobre o qual estas podem executar operações simples como `read` ou `write`, por exemplo. Já a virtualização fornece para a camada superior apenas um disco virtual, construído a

partir de um arquivo do sistema operacional real subjacente. Esse disco virtual terá de ser particionado e formatado para seu uso, da mesma forma que um disco real.

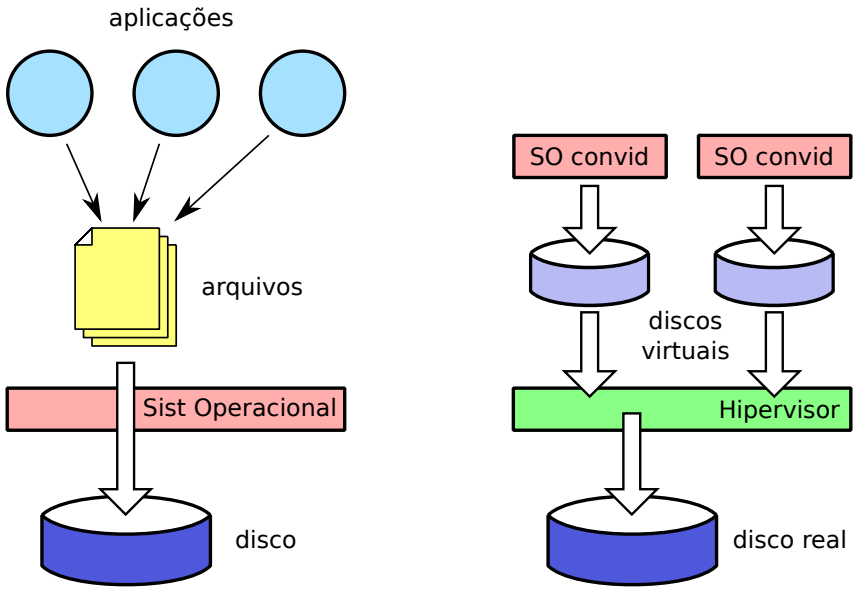

Figura 31.7: Abstração versus virtualização de um disco rígido.

Referências

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.

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

J. Smith and R. Nair. *Virtual Machines: Architectures, Implementations and Applications*. Morgan Kaufmann, 2004.