Virtualização de aplicativos, passado e futuro
M. Tim Jones

Atualmente, quando você ouve a expressão máquina virtual , provavelmente pensa em virtualização e hypervisors. Entretanto, as VMs são simplesmente um conceito mais antigo de abstração, um método comum de abstrair uma entidade a partir de outra. Este artigo explora duas das várias tecnologias mais recentes de VM de software livre: Dalvik (o núcleo de VM do sistema operacional Android) e Parrot (uma tecnologia de VM de software livre para executar linguagens dinâmicas com eficiência).

Virtualização de plataforma versus virtualização de aplicativos

As máquinas virtuais (VMs), em sua primeira "encarnação", foram criadas pela IBM há 60 anos como uma forma de compartilhar sistemas de mainframe grandes e caros. E, embora o conceito ainda seja aplicado aos sistemas IBM atuais, o conceito bastante difundido de VM foi ampliado e vem sendo aplicado a várias áreas fora da virtualização.

Origens da máquina virtual

O primeiro sistema operacional a suportar virtualização total para VMs foi o Conversational Monitor System (CMS). O CMS suportava tanto a virtualização completa quanto a paravirtualização. No início da década de 70, a IBM apresentou a família VM de sistemas, que executava vários sistemas operacionais de usuário único sobre a base do VM Control Program — um dos primeiros hypervisors de tipo 1.

A área da virtualização que a IBM popularizou na década de 60 é conhecida como virtualização de plataforma (ou sistema) . Nessa forma de virtualização, a plataforma de hardware subjacente é virtualizada para compartilhá-la com diversos sistemas operacionais e usuários diferentes.

Outra aplicação da VM é o fornecimento da propriedade de independência de máquina. Essa forma, chamada de virtualização de aplicativo (ou processo)) cria um ambiente abstraído (para um aplicativo), tornando-o independente do seu ambiente físico.

Aspectos das máquinas virtuais de aplicativos

Na área da virtualização de aplicativos, as VMs são usadas para fornecer um ambiente independente de hardware para a execução de aplicativos. Por exemplo, observe a Figura 1. Na parte superior, está a linguagem de alto nível, que os desenvolvedores usam para desenvolver aplicativos. Por meio de um processo de compilação, esse código de alto nível é compilado para uma representação intermediária chamada código do objeto. Em um ambiente não virtualizado, esse código de objeto (que é independente da máquina) é compilado no código nativo da máquina para execução na plataforma física. Contudo, em um ambiente de virtualização de aplicativos, o código do objeto é interpretado dentro de uma máquina abstrata para fornecer a execução. A principal vantagem disso é que o mesmo código de objeto pode ser executado em qualquer plataforma de hardware que suporte a máquina abstrata (o intérprete).

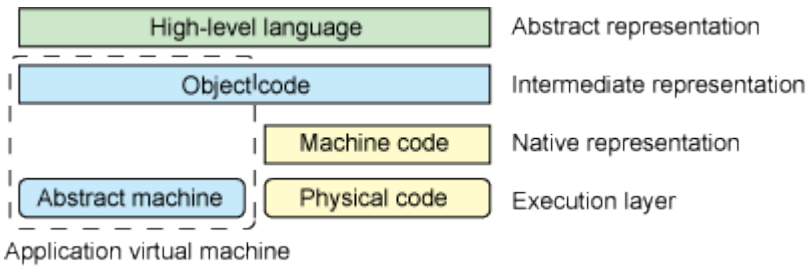

Figura 1. VM de aplicativo para independência de plataforma

Além de criar um ambiente portátil no qual o código do objeto será executado, a virtualização de aplicativos fornece um ambiente em que a VM ficará isolada dos outros aplicativos que executam no host. Essa configuração tem várias vantagens, como gerenciamento detalhado de recursos e segurança.

O código de objeto de uma VM, também chamado de bytecode , define especificamente um conjunto de instruções que o intérprete executa. O termo bytecode evoluiu a partir de implementações que implementavam com eficiência os seus conjuntos virtuais de instruções como bytes únicos para proporcionar simplicidade e desempenho.

Agora, vamos analisar alguns dos usos históricos da virtualização de aplicativos e explorar alguns de seus usos modernos.

História da máquina virtual

Um dos primeiros usos da virtualização de aplicativos ocorreu na década de 60, para a Basic Combined Programming Language (BCPL). A BCPL era uma linguagem imperativa que foi desenvolvida por Martin Richards, da Universidade de Cambridge, e foi precursora da linguagem B, que evoluiu para a linguagem C que usamos hoje.

Embora a BCPL fosse uma linguagem de alto nível (semelhante à linguagem C), o código intermediário que o compilador gerava era chamado de código O (código de objeto). O código O podia ser interpretado em uma máquina física (como uma VM) ou compilado a partir do código O para a linguagem de máquina nativa do host. Essa funcionalidade fornecia várias vantagens no contexto da independência de máquina. Primeiro, ao abstrair o código O a partir da máquina física, ele podia ser interpretado facilmente em diversos hosts. Em segundo lugar, o código O podia ser compilado para a máquina nativa, o que permitiu o desenvolvimento de um compilador e dos vários compiladores que convertiam o código O para as instruções nativas da máquina (uma tarefa mais simples). Essa independência de máquina tornou a linguagem móvel de uma máquina para outra e, portanto, popular, por causa de sua disponibilidade.

No começo da década de 70, a Universidade da Califórnia em San Diego implementou a abordagem da VM para a execução da linguagem Pascal compilada. Eles chamavam a representação intermediária de código p , que buscava independência do hardware subjacente para simplificar o desenvolvimento do compilador de Pascal (em vez de se basear em uma arquitetura de pseudomáquina abstrata). A linguagem Forth também aplicava VMs, ou seja, arquiteturas de endereço zero ou baseadas em pilha.

Em 1972, a Xerox PARC apresentou a linguagem Smalltalk, cuja execução se baseava em uma VM. A Smalltalk foi uma das primeiras linguagens desenvolvias com base no conceito de objeto. Tanto a Smalltalk quanto o código p influenciaram uma das mais importantes linguagens baseadas em VM da atualidade: a linguagem Java. O Java surgiu em 1995, desenvolvido pela Sun Microsystems, e desenvolveu a ideia da programação independente de plataforma por meio da Java Virtual Machine. Desde então, a tecnologia Java se tornou um elemento de desenvolvimento dos aplicativos da Web. De scripts no lado do servidor a applets no lado do cliente, a tecnologia Java atraiu atenção para as tecnologias de VM e apresentou tecnologias novas que uniam a interpretação e a execução nativa usando técnicas de compilação just-in-time (JIT).

Diversas outras linguagens incluem o conceito de VMs. A linguagem Erlang (desenvolvida pela Ericsson) usa uma VM para executar bytecodes de Erlang e também para interpretar o Erlang a partir da árvore de sintaxe abstrata da origem. A linguagem leve Lua (desenvolvida pela Pontifícia Universidade Católica do Rio de Janeiro, Brasil) inclui uma VM baseada em registro. Quando um programa em Lua é executado, ele é convertido para bytecodes e, em seguida, executado na VM. Mais adiante, este artigo trata de um padrão de bytecode que pode ser usado com qualquer linguaguem.

Máquinas virtuais hoje

O uso de VMs para fornecer uma abstração para o host físico é um método historicamente comum e, atualmente, evolui e encontra aplicações. Vamos analisar algumas das novas soluções de software livre que levam o conceito das VMs parao futuro.

VM Dalvik

Dalvik é uma tecnologia de VM de software livre desenvolvida pela Google para o sistema operacional Android. O Android é um kernel Linux modificado que incorpora uma pilha de software para dispositivos remotos (veja a Figura 2). Ao contrário de muitas tecnologias de VM que usam arquiteturas baseadas em pilha, a VM Dalvik é uma arquitetura virtual baseada em registro (consulte recursos para obter mais informações sobre a arquitetura e o conjunto de instruções). Embora as arquiteturas baseadas em pilha sejam conceitualmente simples e eficientes, elas podem introduzir novas ineficiências, como programas de tamanho maior (por causa da manutenção da pilha).

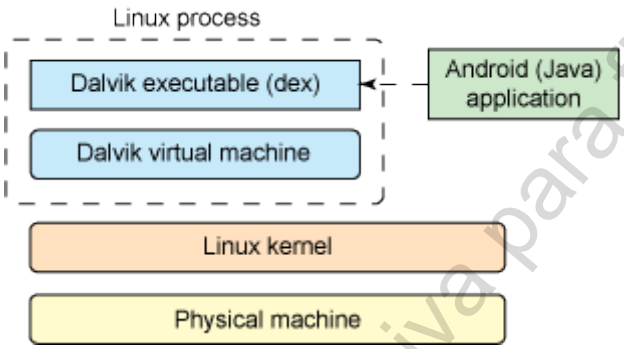

Figura 2. Arquitetura simples de uma pilha de software de Dalvik

Como o Dalvik é a arquitetura da VM, ele se baseia em uma linguagem de alto nível compilada nos bytecodes que a VM entende. Em vez de reinventar tudo, o Dalvik se baseia na linguagem Java como a linguagem de alto nível para o desenvolvimento de aplicativos. O Dalvik também usa uma ferramenta especial chamada dx para converter arquivos de classe Java em executáveis da VM Dalvik. Para obter desempenho, a VM pode modificar ainda mais um Dalvik executável (dex) para mais otimizações, como compilação de JIT, que converte as instruções em dex para instruções nativas, para um desempenho nativo. Esse processo também é conhecido como conversão dinâmica e é uma técnica bastante difundida para melhorar o desempenho das tecnologias de VM.

Como mostra a Figura 2, um Dalvik executável (juntamente com uma instância da VM) é isolado como um único processo no espaço do usuário do Linux. A VM Dalvik foi projetada para suportar a execução de várias VMs (em processos independentes) simultaneamente.

A VM Dalvik não é implementada no Java Runtime padrão e, portanto, não herda as licenças relacionadas a ele. Em vez disso, o Dalvik é uma implementação limpa publicada sob a licença do Apache 2.0.

Parrot

O Parrot é outro projeto interessante de VM de software livre. O Parrot é outra tecnologia de VM baseada em registro que foi projetada para executar linguagens dinâmicas com eficiência (linguagens que realizam certas operações no tempo de execução, que normalmente são realizadas no tempo de compilação, como a alteração do sistema de tipos).

O Parrot foi projetado originalmente como um tempo de execução para o Perl6, mas é um ambiente flexível para a execução de bytecodes para várias linguagens (veja a Figura 3). O Parrot suporta várias formas de entrada, como o Parrot Abstract Syntax Tree (PAST), que é útil para escritores de compilador, o Parrot Intermediate Representation (PIR), uma representação de alto nível que pode ser escrita por pessoas ou por compiladores, de forma

automática, o Parrot Assembly (PASM), que fica abaixo da representação intermediária, mas é útil para pessoas e compiladores. Cada forma é convertida e executada no bytecode do Parrot, na VM Parrot.

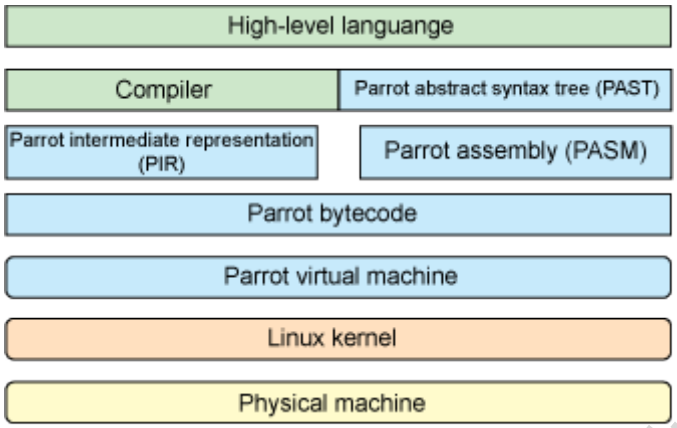

Figura 3. Arquitetura simples da VM Parrot

O Parrot suporta um grande número de linguagens, mas um aspecto que o torna muito interessante é o suporte para linguagens estáticas e dinâmicas, inclusive o suporte para linguagens funcionais. A Listagem 1 mostra um uso simples do PASM. Para instalar o Parrot com o Ubuntu, basta usar apt-get:

```
sudo apt-get install parrot
```

A sessão a seguir ilustra um programa simples de manipulação de cadeias de caracteres no Parrot. Observe que, embora o Parrot implemente seu código como montagem, ele tem muito mais recursos do que a montagem com a qual você talvez esteja acostumado. As instruções no Parrot usam a sintaxe dest, src, portanto, a Listagem 1 mostra um registro de cadeia de caracteres sendo carregado com texto. A instrução length determina o comprimento da cadeia de caracteres e a carrega em um registro de números inteiros. A instrução print emite o argumento para a saída padrão (stdout), e concat implementa a concatenação de cadeias de caracteres.

```
$ more test.pasm
set     S1, "Parrot"
set     S2, "VM"
length  I1, S1
print   I1
print   "n"

concat  S3, S1, S2
print   S3
print   "n"

end$ parrot test.pasm
6
ParrotVM
$
```

Listagem 1. Exemplo do PASM

No Parrot, é possível encontrar um conjunto amplo de instruções (consulte Recursos para obter detalhes adicionais). Os autores preferem a variedade de recursos ao minimalismo, facilitando a codificação e o desenvolvimento de compiladores para a VM Parrot.

Apesar da abstração de alto nível que o PASM fornece, o PIR é ainda mais confortável para programadores de alto nível. A Listagem 2 fornece um exemplo de programa escrito em PIR e executado pela VM Parrot. Esse exemplo declara uma sub-rotina chamada square que calcula o quadrado do número e o retorna. Esse processo é chamado pela sub-rotina principal (etiquetada com :main para instruir o Parrot a executá-la primeiro) para imprimir o resultado.

```
$ more test.pir
.sub square
  .param int arg
  arg *= arg
  .return(arg)
.end

.sub main :main
  .local int value
  value = square(19)
  print value
  print "n"
.end
$ parrot test.pir
361
$
```

Listagem 2. Exemplo do PIR

O Parrot fornece um ambiente de virtualização de aplicativos rico para o desenvolvimento de aplicativos independentes de máquinas que também buscam alta eficiência. Existem várias linguagens que suportam front-ends de compilador projetados para o Parrot, como C, Lua, Python, Scheme, Smalltalk e muitas outras.

Outros usos das máquinas virtuais de aplicativos

Até agora, você viu os usos históricos da virtualização de aplicativos, inclusive dois exemplos recentes. O Dalvik está impulsionando o desenvolvimento de aplicativos dentro dos aparelhos de telefone atuais, e o Parrot fornece um framework eficiente para escritores de compiladores para linguagens estáticas e dinâmicas. Entretanto, o conceito da virtualização de aplicativos está sendo implementado em diversas outras áreas fora das abordagens analisadas até agora.

É provável que um uso particularmente interessante esteja executando no computador que você está usando agora. Sistemas que usam a nova Extensible Firmware Interface (EFI), que é substituta do BIOS, podem implementar drivers de firmware no EFI Byte Code (EBC). O firmware de sistemas inclui um intérprete que é chamado quando uma imagem do EBC é carregada. Esse conceito também foi implementado no Open Firmware pela Sun Microsystems usando o Forth (uma linguagem que inclui sua própria VM).

Na área dos jogos, o uso da virtualização de aplicativos não é novidade. Muitos jogos modernos incluem a criação de scripts para personagens que não são o jogador e outros aspectos do jogo usando linguagens que executam bytecodes (como a linguagem Lua). Entretanto, o conceito de virtualização de aplicativos nos jogos é, na verdade, muito anterior a isso.

A Infocom, empresa que apresentou aventuras baseadas em texto como o Zork, percebeu o valor da independência de máquina em 1979. A Infocom criou uma VM chamada Z-machine (nome baseado no Zork). A Z-machine era uma VM que permitia que um jogo de aventura fosse portado para outras arquiteturas com mais facilidade. Em vez de ser necessário portar toda a aventura para um novo sistema, seria portado um intérprete que representava a Z-machine. Essa funcionalidade simplificou o processo de portar para outros sistemas que podem ter um suporte distinto para linguagens e arquiteturas de máquina totalmente diferentes. Embora o objetivo do Infocom fosse a facilitação do processo de portar entre as arquiteturas da época, seu trabalho continua simplificando esse processo e, como resultado, esses jogos ficam acessíveis a uma nova geração (inclusive em plataformas remotas).

O ScummVM, que fornece um ambiente de VM para a linguagem de criação de scripts Script Creation Utility for Maniac Mansion (SCUMM) (criada em 1987) é outra aplicação das VMs aos jogos. O SCUMM foi desenvolvido pela LucasArts para simplificar o desenvolvimento de um jogo de aventura gráfica. Agora, o ScummVM é usado em vários jogos de aventura gráfica e de texto em diversas plataformas.

Indo além

Da mesma forma que a virtualização de plataforma (ou de sistema) mudou a maneira de fornecer e gerenciar tanto servidores quanto desktops, a virtualização de aplicativos continua fornecendo mecanismos eficientes para abstrair um aplicativo a partir do seu sistema host. Considerando a popularidade dessa abordagem, será interessante ver uma evolução do software e do hardware para deixar a virtualização de aplicativos ainda mais flexível e eficiente.

Recursos

Aprender

■ Navegue por mais artigos sobre virtualização ou todos os artigos do Tim no developerWorks.

■ A Wikipédia fornece um grande conjunto de recursos para saber mais sobre as VMs (tanto de plataforma quanto de aplicativo). Confira a página sobre máquinas virtuais e uma página dedicada especificamente a máquinas de código p.

■ A BCPL, precursora das linguagens B e, posteriormente, C , originou-se em 1967, criado por Martin Richards. É possível ler o primeiro manual de referência da BCPL como parte do Projeto MAC. Também é possível fazer o download da versão mais recente da BCPL a partir do seu site inicial.

■ O EFI Byte Code (EBC) especifica uma camada interpretativa para drivers de componentes portáteis. Pode-se saber mais sobre o EBC e o UEFI em Boot Loaders: Small, Fast System Initialization (Dr. Dobb's, setembro de 2010).

■ Embora o Forth esteja em uso desde a década de 70, ele continua encontrando aplicações como uma linguagem de VM. Você verá que o Forth é aplicado a ciências espaciais, sistemas embarcados, BIOS e várias outras aplicações que contam com recursos escassos. Saiba mais sobre o Forth no Forth Interest Group.

■ Siga o developerWorks no Twitter, assine um feed de tweets sobre Linux no developerWorks ou siga M. Tim Jones no Twitter.

■ Na zona Linux do developerWorks, você encontra centenas de artigos e tutoriais de instruções além de downloads, fóruns de discussão e vários outros recursos para desenvolvedores e administradores de Linux.

■ Fique por dentro dos eventos técnicos e webcasts do developerWorks com foco em uma variedade de produtos da IBM e tópicos do segmento de mercado de TI.

■ Participe de um briefing ao vivo e gratuito do developerWorks para se atualizar rapidamente sobre produtos e ferramentas IBM, bem como tendências do segmento de mercado de TI.

■ Acompanhe Demos on demand do developerWorks que abrangem desde demos de instalação e configuração de produtos para iniciantes até funcionalidades avançadas para desenvolvedores experientes.

Obter produtos e tecnologias

■O Dalvik é o ambiente de VM para o sistema operacional Android. O Dalvik foi desenvolvido por Dan Bornstein e é mantido pela Google como parte do Android. Saiba mais sobre a máquina Dalvik por meio dos seus bytecodes (disponíveis na documentação para o usuário). Também é possível saber mais sobre o Dalvik em Introdução ao desenvolvimento do Android (Frank Ableson, developerWorks, maio de 2009).

■O Parrot é uma VM projetada para executar linguagens estáticas e dinâmicas com eficiência por meio de uma variedade de representações intermediárias no bytecode do Parrot. O Parrot está disponível como software livre e pode ser usado com várias linguagens. Para saber mais sobre o conjunto de instruções do Parrot, confira os opcodes disponíveis nele.

■ As VMs de aplicativos são bastante utilizadas na área de desenvolvimento de jogos. Uma das primeiras empresas a usá-las foi a Infocom, em seus jogos de aventura em texto (como o Zork). É possível saber mais sobre a VM da Infocom, chamada de Z-machine, e sobre intérpretes disponíveis para várias plataformas. Outra aplicação das VMs foi o SCUMM, usado pela LucasArts em aventuras gráficas. O SCUMM foi implementado como software livre como ScummVM e está trazendo de volta jogos antigos em hardwares novos.

■Avalie produtos IBM da maneira que for melhor para você: faça download da versão de teste de um produto, avalie um produto on-line, use-o em um ambiente de nuvem ou passe algumas horas na SOA Sandbox aprendendo a implementar a Arquitetura Orientada a Serviços de modo eficiente.

Discutir

■ Participe da comunidade do My developerWorks. Entre em contato com outros usuários do developerWorks, enquanto explora os blogs, fóruns, grupos e wikis orientados ao desenvolvedor.

Fonte: iMasters, 12 set. 2011. [Portal]. Disponível em: <http://imasters.com.br>. Acesso em: 14 set. 2011.