Virtualização em sistemas embarcados: é o futuro?

Alexandra Aguiar, Fabiano Hessel

Faculdade de Informática – PUCRS – Av. Ipiranga 6681, Porto Alegre, Brasil

alexandra.aguiar@pucrs.br, fabiano.hessel@pucrs.br

Abstract. *Traditionally, virtualization has been adopted by enterprise industry aiming to make better use of general purpose multi-core processors and its use in embedded systems (ES) seemed to be both a distant and unnecessary reality. However, with the rise of each more powerful multiprocessed ESs, virtualization brings an opportunity to use simultaneously several operating systems (OS) besides offering more secure systems and even an easier way to reuse legacy software. Although ESs have increasingly bigger computational power, they are still far more restricted than general purpose computers, especially in terms of area, memory and power consumption. Therefore, this paper studies the possibilities of using virtualization - a technique that typically demands robust systems - in powerful yet restricted current embedded systems.*

Resumo. *Tradicionalmente, virtualização tem sido adotada por diversas empresas como uma forma de aproveitar melhor o grande poder computacional oferecido por arquiteturas com vários processadores e seu uso em sistemas embarcados parece desnecessário e distante da realidade. Entretanto, com a utilização de sistemas embarcados multiprocessados, a virtualização oferece uma oportunidade para a utilização de diferentes sistemas operacionais simultaneamente em um único processador, possibilitando o desenvolvimento de aplicações onde a segurança dos dados é importante e possibilitando a reutilização de software de forma mais fácil. Atualmente, os sistemas embarcados são mais abrangentes, multifuncionais, com maior poder de processamento e convergentes, porém, ainda possuem muitas restrições comparadas aos sistemas de propósito geral. Nesse contexto, o presente artigo aborda os motivos que tornam a intersecção entre virtualização e sistemas embarcados um grande desafio a ser superado.*

1. Introdução

Virtualização de sistemas computacionais consiste em criar um grupo lógico de recursos que se assemelham aos recursos físicos oferecidos por um ambiente computacional [Popek e Goldberg, 1974]. Essa técnica tem sido adotada amplamente no mundo empresarial, especialmente para explorar o potencial de sistemas multiprocessados, além de oferecer outras vantagens, tais como:

- permitir que vários sistemas operacionais (SO) sejam executados em uma única máquina;
- prover isolamento de uma máquina virtual para outra, aumentando a segurança;
- aumentar a flexibilidade do sistema;
- melhorar o gerenciamento da carga de trabalho, e:

- permitir a independência de hardware.

Por outro lado, a virtualização pode ser considerada uma técnica que demanda alto poder computacional, já que, normalmente, requer um grande espaço em disco e muito uso de memória RAM, além de inserir uma camada extra de gerenciamento: o monitor de máquinas virtuais (do inglês, *Virtual Machine Monitor* – VMM), também conhecido como *hypervisor*, camada essa responsável por permitir que instruções executadas pela máquina virtual sejam executadas normalmente pela máquina hospedeira.

Em servidores comerciais, a virtualização permite que um único servidor físico sirva como múltiplos servidores lógicos além de prover múltiplas instâncias de diferentes Sistemas Operacionais (SO), como Windows, Linux e outros. Frequentemente, esses sistemas são empregados em processadores *multi-core* da Intel e AMD, tendência essa que é adotada atualmente pela maioria dos fabricantes de processadores, cujos projetos ultrapassam os quatro *cores* para um futuro próximo.

Já no mundo dos Sistemas Embarcados (SE), assim como na computação de propósito geral, o uso de plataformas multiprocessadas tem se mostrado uma forte tendência nos últimos anos [Jerraya *et.al.*2005]. Dispositivos que utilizam tais plataformas forçam uma mudança na maneira pela qual os desenvolvedores de SE's realizam a concepção de seus sistemas, já que técnicas antes vistas na computação multiprocessada de propósito geral precisam ser reavaliadas antes de ser empregadas em SE's [Martin, 2006].

Enquanto a virtualização possibilita a execução de múltiplas instâncias de sistemas operacionais em um único processador (*mono-* ou *multi-core*), a sua utilização em sistemas embarcados não é trivial, pois são muito diferentes de sistemas empresariais [Waldspurger, 2002]. Desse modo, para que a virtualização possa ser empregada de maneira vantajosa em sistemas embarcados, muito esforço deve ser realizado para que se entenda como deve-se adaptá-la às necessidades e características dos SE's, sistemas normalmente restritos com relação ao consumo de energia, quantidade de memória, restrições temporais e tamanho de área.

Adicionalmente, o uso de técnicas previamente otimizadas para sistemas comerciais Windows e Linux não são as mais adequadas em SE. Ao invés disso, técnicas existentes de virtualização devem ser adaptadas aos sistemas embarcados, verificando a relação custo benefício de seu uso. Além disso, deve-se verificar qual o tipo de sistema embarcado teria mais vantagens ao se utilizar a virtualização.

Nesse contexto, o presente trabalho visa discutir os principais conceitos envolvendo virtualização, alem de explorar as possibilidades de uso dessa técnica em sistemas embarcados. Para isso, são descritas as maneiras mais adequadas de se desfrutar da virtualização em sistemas embarcados de acordo com a meta a ser atingida. Ainda, apresentam-se as principais desvantagens e problemas esperados durante a aplicação da técnica em SE além de possíveis soluções ou formas para contornar esses problemas.

O artigo está organizado como segue. A próxima seção exibe alguns conceitos básicos adotados durante o artigo. Já a Seção 3 discute as vantagens da virtualização em SE seguida da seção 4 que mostra os principais problemas esperados nessa adaptação.

Na Seção 5 discute-se como se pode aplicar virtualização em SE e, finalmente, a Seção 6 conclui o trabalho.

2. Conceitos básicos

Nesta seção são apresentados os conceitos básicos sobre virtualização e sistemas embarcados. Destacam-se aqueles conceitos necessários para melhor entender o restante do artigo.

2.1 Sistemas Embarcados

Nos últimos anos, pode-se afirmar que sistemas embarcados eram comumente caracterizados por dispositivos simples, com severas restrições, tais como uso de memória, poder de processamento e vida útil da bateria. Sua funcionalidade era determinada, tipicamente, via hardware, sendo que seu software usualmente era composto por *drivers* de dispositivos, escalonador de tarefas e alguma lógica de controle, causando uma baixa dinamicidade das tarefas ofertadas por um dado sistema ao longo do seu tempo de vida [Hansson *et.al.* 2005].

No entanto, este cenário encontra-se diferente. Cada vez mais sistemas híbridos, convergentes e não críticos, com inúmeras características dos sistemas de propósito geral estão surgindo. A maior mudança, sem dúvidas, é o fato de se oferecer um número antes inimaginável de funções que afeta e aumenta, dramaticamente, a complexidade do software desses sistemas. Também, é muito comum que aplicações de propósito geral necessitem de ser executadas nessas plataformas. Aplicações as quais nem sempre foram desenvolvidas por pessoas com o conhecimento adequado acerca da programação para dispositivos embarcados [Wolf, 2003].

Nesse contexto, API's de alto nível, fornecidas pelos sistemas operacionais (SO's) são cada vez mais importantes. Além disso, torna-se muito comum a disponibilização dessas API's pela própria indústria, como incentivo ao desenvolvimento de aplicações embarcadas, como no caso das API's para desenvolvimento de aplicações para telefones celulares. Esse fato, porém, traz à tona o problema da segurança. No passado, somente os fabricantes tinham acesso a determinadas partes do sistema. Atualmente, com o fornecimento das API's, os desenvolvedores passam a ter acesso a partes sensíveis do sistema, o que pode torná-lo vulnerável a ataques visando à captura de dados confidenciais como, por exemplo, senhas bancárias.

Apesar de todo avanço conquistado, os sistemas embarcados continuam sendo muito diferentes dos sistemas de propósito geral: ainda possuem requisitos de tempo real estritos, além de necessitar customizações que visem à economia no consumo de energia, já que a vida útil da bateria deve ser o mais longa possível [Lavagno e Passerone, 2005]. Isso impacta na frequência de processador possível de ser adotada: normalmente frequências mais baixas são obrigatórias para que se possam atingir determinadas metas de consumo de energia.

Outra restrição comum diz respeito ao uso de memória, já que os usuários de dispositivos modernos anseiam por mais memória apesar de, tipicamente, exigirem dispositivos a preços acessíveis e com baixo consumo de energia (condição que pode facilmente ser violada pelo uso de mais memória) [Hohmuth *et.al.*2005].

Finalmente, é possível afirmar que sistemas embarcados são, cada vez mais, parte do cotidiano das pessoas, além de ser usado em sistemas especificamente críticos [Tanenbaum, 2007]. Apesar de essa ser uma classe de SE com muito mais restrições do que a classe *moderna* de SE's, o uso da virtualização em seus sistemas poderia trazer vantagens, como aumento da segurança e confiabilidade.

Nesse contexto, a virtualização pode ajudar os sistemas embarcados em muitos aspectos: aumentar o poder de processamento, aumentar a segurança além de permitir que usuários carreguem suas aplicações sem comprometer o sistema como um todo. Apesar disso, é mandatório que se estude como realizar a virtualização sem violar as diversas restrições existentes nos sistemas embarcados.

2.2 Virtualização

Como dito anteriormente, a virtualização permite que um único computador hospede múltiplas máquinas virtuais, sendo que existe isolamento entre elas com a possibilidade de que executem sistemas operacionais diferentes. A principal vantagem é que, se uma máquina virtual falha, as outras são mantidas funcionando a um custo razoável [Heiser, 2008].

No âmbito empresarial, apesar de isso significar que o sistema possui um único ponto de falha, uma vez que muitos servidores podem ser colocados em um hardware único, pode ser considerado que essa é uma abordagem mais segura porque a maioria das interrupções de serviço não é causada por falhas no hardware, mas sim, por problemas do software, que tipicamente é grande e complexo demais, o que atrapalha na sua manutenibilidade. Além disso, o software que normalmente contém grande parte desses erros é o sistema operacional. Nesse sentido, a virtualização utiliza um único elemento de software que é executado no modo *kernel*: o *hypervisor*, que tipicamente é – pelo menos – duas ordens de magnitude menor do que um sistema operacional e, portanto, menos suscetível a erros [Tanenbaum, 2007].

Adicionalmente, deve-se observar o funcionamento desse componente. De acordo com [Popek e Goldberg, 1974], existem dois tipos diferentes de *hypervisor*:

- tipo 1, conhecido como virtualização no nível de hardware, onde se considera que o *hypervisor* é um sistema operacional por si só, já que somente ele opera em modo *kernel*, como pode ser observado no lado esquerdo da Figura 1. Sua principal tarefa, além de controlar a máquina real, é prover a noção de máquinas virtuais, e;
- tipo 2, ou virtualização no nível de SO, onde o *hypervisor* é como qualquer outra aplicação de usuário e não tem acesso direto ao hardware (deve passar antes pelo sistema operacional da máquina). Nesse caso, perde-se uma das principais vantagens da virtualização que é justamente o uso de SO's diferentes. Por esse motivo, considera-se, neste trabalho, somente a virtualização no nível de hardware.

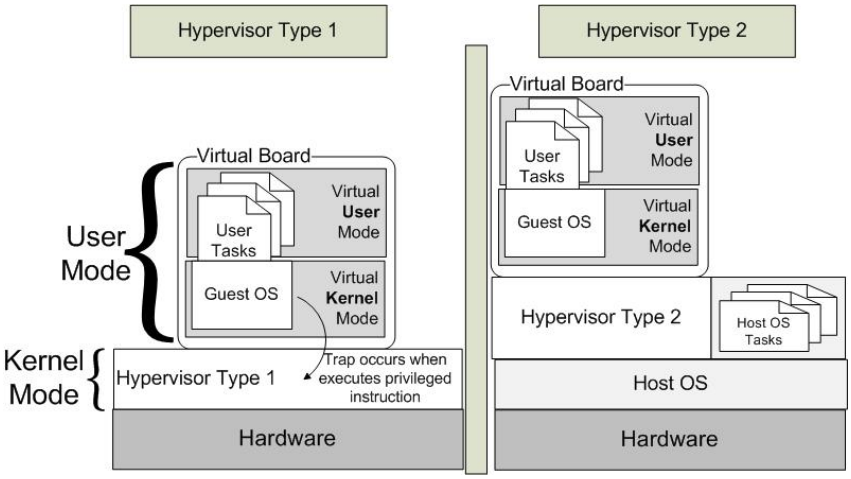

Figura 1 – Hypervisors tipos 1 e 2

Em ambos os casos a máquina virtual deve ter o mesmo comportamento do hardware verdadeiro. Mais especificamente, deve ser possível iniciá-la e reiniciá-la como um computador real, bem como instalar diferentes sistemas operacionais. A criação deste cenário é responsabilidade do hypervisor.

O hypervisor (independente de seu tipo) juntamente com o hardware, é responsável por lidar com as instruções vindas da máquina virtual. É importante destacar que uma vez que a máquina virtual "imita" o hardware real, também deve separar a execução nos modos *kernel* e usuário.

Nesse sentido, estudos clássicos de Popek e Goldberg [1974] introduzem uma classificação dos ISA (*Instruction Set Architecture*) em três grupos diferentes:

1. *instruções privilegiadas:* aquelas que causam em uma *trap* quando executadas em modo usuários mas que não causam *trap* se empregadas no modo *kernel*;
2. *instruções sensitivas de controle:* aquelas que tentam modificar a configuração dos recursos no sistema, e;
3. *instruções sensitivas de comportamento:* aquelas cujo comportamento ou resultado depende da configuração de recursos (o conteúdo do registrador de relocação ou o modo do processador).

Assim sendo, de acordo com os Popek e Goldberg [1974], para que a virtualização de uma dada máquina seja possível, as instruções sensitivas (de controle e comportamento) devem ser um subconjunto das instruções privilegiadas. Isso não é realidade em muitos processadores, como os da família Intel x86, e, nesse caso, a solução comumente perpassa por adotar suporte em nível de hardware por parte do processador. A Intel possui o *IntelVT (Virtualization Technology)* e a AMD possui o SVM (*Secure Virtual Machine*). O suporte pelo hardware pode não ser a melhor solução no caso dos sistemas embarcados, já que é interessante que a virtualização consiga lidar com o hardware existente, especialmente para acelerar o restrito *time-to-market.*

Conforme dito anteriormente, neste trabalho considera-se apenas a virtualização em nível de hardware. Detalhando a maneira pela qual essa técnica é empregada sem o suporte de hardware (como aquele fornecido pela Intel e pela AMD), é possível

observar que, inicialmente, sempre que a máquina virtual tentar executar uma instrução privilegiada (requisição de E/S, escrita em memória etc.), ocorre uma *trap* para o *hypervisor*. Isso é conhecido como virtualização pura e é normalmente uma forma muito ineficiente de se aplicar essa técnica [Waldspurger, 2002].

Outra opção no nível de hardware é conhecida como virtualização impura e requer que as instruções sensíveis (aquelas que causam uma *trap* no *hypervisor*) sejam removidas do código a ser executado na máquina virtual. Isso pode ser feito tanto em tempo de compilação (através de pré-virtualização) ou por reescrita de código binário, em tempo de execução, onde o código é vistoriado com o intuito de substituir tais instruções. O problema de ambas essas abordagens é que causam perda de desempenho.

Alternativamente, a técnica de *para-virtualização* pode ser empregada para substituir as instruções sensitivas do código original por chamadas explícitas ao *hypervisor* (*hypercalls*). Na verdade, o sistema operacional da máquina virtual está agindo como uma aplicação normal de usuário sendo executada sobre um sistema operacional normal[1], com a diferença que o sistema operacional convidado está sendo executado sobre o *hypervisor*. Quando a para-virtualização é adotada, o *hypervisor* deve definir uma interface composta por chamadas de sistemas que possam ser usadas pelo sistema operacional convidado. Ainda, é possível remover todas as instruções sensitivas do SO convidado, forçando-o a usar comente as *hypercalls* o que torna o *hypervisor* mais parecido com um *microkernel*, o que pode aumentar o desempenho da virtualização.

As diferenças entre virtualização pura e para-virtualização são mostradas na Figura 2. Na parte A dessa figura, a virtualização pura é mostrada. Nesse caso, sempre que o SO convidado utiliza uma instrução sensitiva, uma *trap* é causada no hypervisor, o que emula o comportamento de uma instrução com os resultados apropriados. Na parte B, a para virtualização é mostrada. O SO convidado deve ser modificado para efetuar as *hypercalls* ao invés de conter instruções sensitivas. Nesse caso, a *trap* é similar ao que ocorre em sistemas não virtualizados quando uma aplicação de usuário faz uma chamada de sistema do SO.

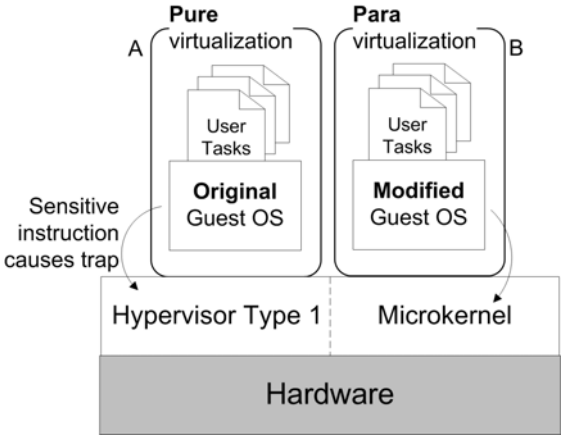

Figura 2 – Controle do *hypervisor* na virtualização pura (parte A) e na para-virtualização (parte B)

[1] Neste contexto, normal significa não virtualizado.

Através dos conceitos apresentados nesta seção, são discutidas a seguir as vantagens, desvantagens e maneiras para se usar a virtualização em sistemas embarcados.

3. Vantagens da virtualização em sistemas embarcados

Neste trabalho, considera-se que os sistemas embarcados são multiprocessados, não críticos, mas contêm restrições temporais. Sendo assim, nesta seção destacam-se possíveis casos de uso para a virtualização em tais sistemas.

Uma das vantagens mais apropriadas e diretas da virtualização em sistemas embarcados consiste em permitir que SO's diferentes possam coexistir na mesma máquina. Nesse caso, pode-se atacar dois problemas diferentes:

- o uso de software legado, já que é possível a criação (ou manutenção) de um sistema operacional compatível com esse software e outro mais moderno, que permita que novos recursos sejam explorados, e;
- dividir o sistema em uma parte onde o usuário tem acesso, com chamadas específicas conhecidas por ele, separadas da parte crítica, responsável por manter o dispositivo funcionando. Nesse caso, dois sistemas operacionais, um de usuário e outro de sistema, podem ser empregados simultaneamente.

Quando a virtualização for empregada com esses objetivos, o *hypervisor* deve ter controle total do hardware além de criar diferentes máquinas virtuais, uma por SO. Como pode ser observado na Figura 3, essa abordagem pode ser usada tanto em máquinas mono- ou multi-core. Ainda, permite que se aumente a qualidade de desenvolvimento de software, uma vez que o projetista pode escolher entre diversos SO's aquele mais adequado à sua aplicação. Além disso, o tempo requerido para desenvolver uma aplicação pode ser reduzido drasticamente, já que a reusabilidade das aplicações cresce sensivelmente [Shen e Petrot, 2009].

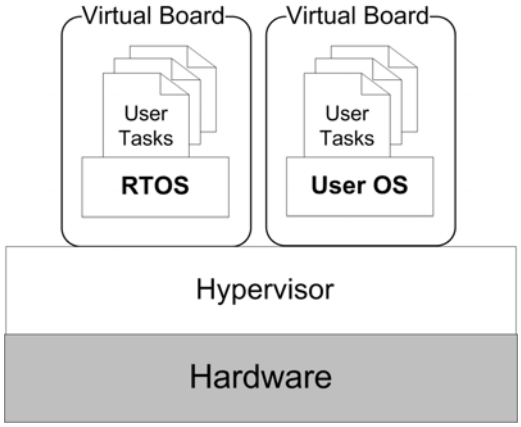

Figura 3 – Hypervisor para separação de máquinas com vários SO's

Além disso, essa abordagem oferece a vantagem de se alcançar uma arquitetura de software unificada que pode ser executada em múltiplas plataformas de hardware. Nesse caso, um problema atual e recorrente nos sistemas embarcados – a portabilidade de software – pode ser amplamente afetado e os projetistas têm o potencial de satisfazer mais rapidamente o *time-to-market* cada vez mais restrito.

Adicionalmente, a segurança do sistema embarcado também é um forte apelo para a virtualização, já que através dela pode se prevenir que ataques ocorridos ao sistema atinjam o SO principal, como pode ser visto na Figura 4. Nesse caso, o código malicioso fica restrito à maquina virtual, sem contaminar o resto do sistema, pois não possui o conhecimento necessário do *hypervisor* para poder explorar os seus pontos fracos. Ainda, o *hypervisor* pode detectar a ocorrência de um ataque e reinicializar a maquina virtual, sem prejudicar o resto do sistema.

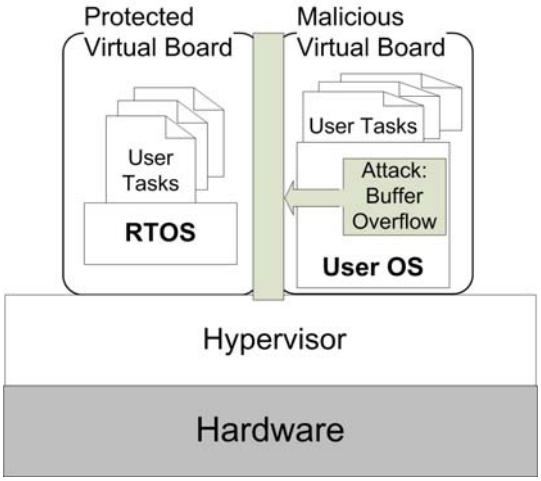

Figura 4 – Ataque de usuário bloqueado através do isolamento das máquinas virtuais

Consequentemente, o uso dos *hypervisors* previamente conhecidos, exatamente como aqueles adotados em sistemas de propósito geral, provavelmente não será possível. Com o objetivo de manter o hypervisor pequeno (para que ataques e falhas sejam mais difíceis), restrições específicas dos sistemas embarcados devem ser levadas em consideração, o que afeta todo o ciclo de desenvolvimento do software.

Em arquiteturas multi-core, existem maneiras diferentes de se utilizar os diversos processadores do sistema. Uma forma bastante comum e usual de se fazer isso é executar um único SO sobre todos os processadores, criando o que se conhece como configuração multiprocessada simétrica (SMP). Essa abordagem traz a vantagem principal de oferecer balanceamento de carga entre os processadores de forma direta. Entretanto, não permite o uso de SO's diferentes em um mesmo MPSoC, o que já foi argumentado anteriormente como sendo uma atrativa opção. Além disso, faz com que exista somente um ponto único de falha no sistema inteiro e – sempre que houver uma falha no sistema – todos os *cores* devem ser reiniciados.

Nesse caso, pode-se adotar uma configuração multiprocessada assimétrica (AMP) onde cada processador possui seu próprio SO, responsável por escalonar suas próprias tarefas. A configuração AMP consegue aproveitar diversas vantagens oferecidas pela virtualização, uma vez que provê a arbitragem do uso de recursos entre as máquinas virtuais, evitando que o SO de usuário cause um comportamento inesperado no RTOS [4]. Se nenhuma técnica de virtualização é empregada, a única forma de se conseguir tal separação é fazendo isso manualmente, o que pode ser mais complicado e mais propenso a erros. O *hypervisor* pode mapear cada máquina virtual em um *core* do sistema multi-procesado ou até mesmo mapear um único SO em múltiplos *cores*, criando um subconjunto SMP de *cores* [11].

Para o mundo empresarial, a redução de custos trazida pela virtualização além das possibilidades de atualizações facilitadas são vantagens promissoras que encorajam seu uso.

Sumarizando, as principais vantagens da utilização da virtualização em sistemas embarcados são:

- o uso de diferentes SO's;
- segurança e confiabilidade do sistema são aumentadas, e;
- diferentes alternativas para configuração de um ambiente multi-core.

4. Barreiras para o uso de virtualização em sistemas embarcados

Enquanto a virtualização embarcada pode trazer inúmeras vantagens, é importante que se esclareça a que custo esses benefícios podem ser alcançados. Algumas das limitações já estão presentes na virtualização de propósito geral, enquanto que outras surgem do seu uso em ambientes tão severamente restritos, como os sistemas embarcados.

Um dos principais problemas a ser atacado está relacionado com o escalonamento das tarefas realizado pelo *hypervisor*. Sistemas embarcados tipicamente têm restrições temporais e, por isso, qualquer deslize do *hypervisor* pode comprometer o sistema.

Pode-se ainda considerar o caso onde um dado multi-core apresenta um comportamento multi-processado assimétrico, com dois SO's: um de usuário e um RTOS. Nesse caso, cada SO é tratado como uma máquina virtual separada e, em sistemas embarcados, é desejável que o RTOS seja priorizado em relação ao SO de usuário, assim como tarefas de tempo-real que eventualmente sejam executadas no SO de usuário (como aplicações multimídia) também devem ter preferência. Esse escalonamento com prioridades vai de encontro com os princípios das máquinas virtuais, nos quais todas as máquinas virtuais devem dividir o hardware real em proporções iguais.

Além disso, a heterogeneidade típica de sistemas embarcados pode representar um grande desafio, já que o *hypervisor* tem de, teoricamente, conseguir comunicar-se com o maior número possível de arquiteturas. Enquanto que na computação de propósito geral a arquitetura Intel x86 é amplamente usada, por exemplo, em sistemas embarcados existe uma grande variedade de arquiteturas empregadas, desde DSP's a processadores ARM, passando ainda por arquiteturas PowerPC e MIPS.

Ainda, o isolamento excessivo e absoluto trazido pelas máquinas virtuais – que aumenta os níveis de segurança e confiabilidade – podem causar dificuldades para que os diversos subsistemas embarcados cooperem entre si, o que altamente desejável em sistemas embarcados.

5. Possíveis casos de uso para virtualização embarcada

Após considerar os conceitos previamente explanados, descrevem-se, nesta seção, alguns cenários onde a virtualização embarcada pode ser adotada.

No *Cenário 1* pretende-se reduzir o número total de processadores em um sistema, colocando-os em diversas máquinas virtuais sobre um único processador (seja

ele mono- ou multi-core). Na Figura 5 pode-se observar uma possibilidade de tal configuração.

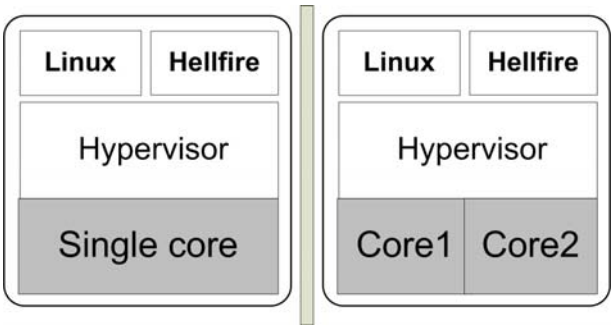

Figura 5 – Cenário 1: redução do número total de processadores

Mesmo no caso onde o processador oferece suporte de hardware à virtualização (como VT ou SVM), é desejável se adotar a para-virtualização, o que permite que o hypervisor torne-se independente de suporte além de trazer um aumento de desempenho.

No *Cenário 2* a confiabilidade de sistemas AMP pode ser aumentada através da separação dos recursos, com a capacidade de se reiniciar as máquinas virtuais, como pode ser visto na Figura 6.

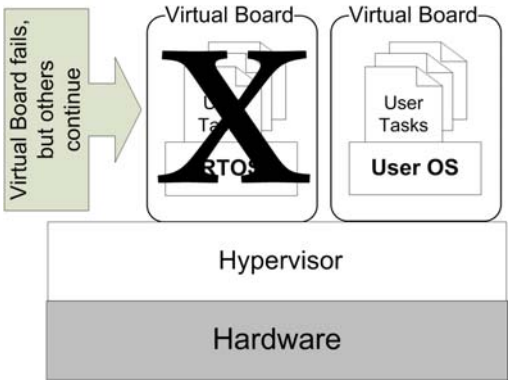

Figura 6 – Cenário 2: confiabilidade aumentada

No *Cenário 3* é possível migrar sistemas existentes em uma máquina virtual e adicionar novas funcionalidades a elas, provendo assim, oportunidade para reuso e inovação. Além disso, a migração de tarefas entre máquinas virtuais é facilitada. Esse cenário pode ser visualizado na Figura 7. É válido lembrar que as vantagens da migração em sistemas embarcados tem sido objeto amplo de estudo ao longo dos anos [Shen e Petrot, 2009], [Bertozzi *et. al.*, 2006], [Nollet *et. al.*, 2006].

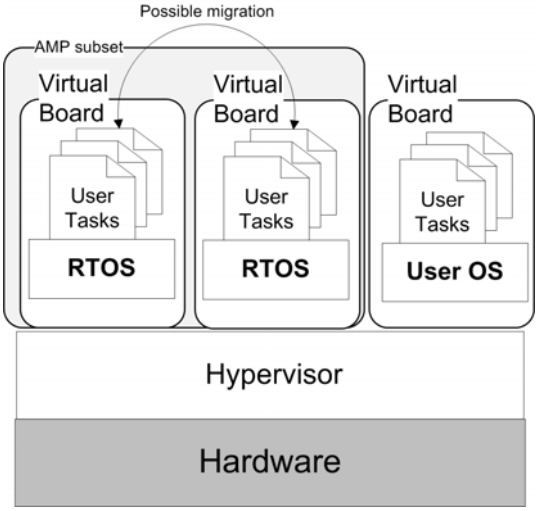

Figura 7 – Cenário 3: migração entre máquinas virtuais

No *Cenário 4*, a combinação de sistemas de tempo-real, legados e SO's de propósito geral no mesmo dispositivo é alcançada através da virtualização, como mostra a Figura 8.

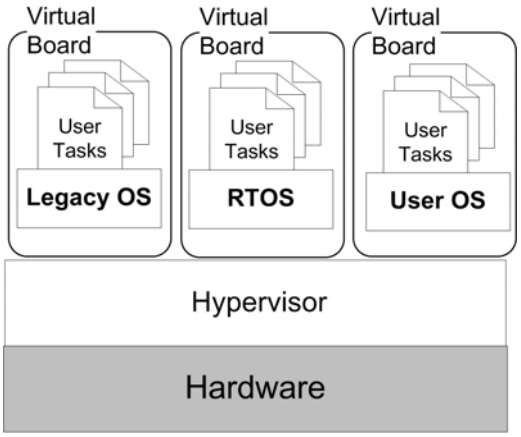

Figura 8 – Cenário 4: uso de software legado com novas aplicações de usuário

É válido lembrar que todos esses cenários são bastante adequados para sistemas embarcados não críticos, como dispositivos de entretenimento móveis. Especialmente, uma vez que a demanda dos usuários continua a crescer, cada vez mais o *time-to-market* está restrito e a virtualização pode ajudar no seu cumprimento.

6. Considerações finais

Virtualização tem sido amplamente usada em sistemas corporativos como forma de aproveitar melhor o poder de processamento das máquinas multi-core. Enquanto isso, sistemas embarcados costumavam ser extremamente restritos e de propósito específico. Esse cenário, no entanto, tem se modificado. Sistemas embarcados cada vez mais fazem parte da vida das pessoas e suas múltiplas funcionalidades levam a um crescimento não linear da complexidade de software. Nesse contexto, muitas soluções têm sido estudadas, entre elas, a virtualização. Os principais apelos para se usar a virtualização em sistemas embarcados são: (i) permitir que diversos SO's sejam executados em um

mesmo processador; (ii) reduzir o custo de fabricação, através do aumento da utilização dos processadores do sistema; (iii) aumentar a confiabilidade e a segurança, e; (iv) ajudar a diminuir a complexidade inerente do desenvolvimento de software em sistemas embarcados.

O desafio, portanto, é adaptar as abordagens nas quais o hypervisor aproxima-se a implementações de um microkernel, com o objetivo de se desenvolver técnicas de virtualização mais leves, adequadas para os sistemas embarcados atuais e futuros.

Referências

Bertozzi, S., Acquaviva, A., Bertozzi, D. e Poggiali, A. Supporting task migration in multi-processor systems-on-chip: A feasibility study. In *Design, Automation and Test in Europe, 2006.* DATE'06. *Proceedings*, pages 1-6, 2006.

Hansson, H., Nolin, M. e Nolte, T. Real-Time in Embedded Systems. In *Richard Zurawski, editors, Embedded Systems Handbook, chapter 2*. CRC press, 2005.

Heiser, G. The role of virtualization in embedded systems. *IIES '08: Proceedings of the 1st workshop on Isolation and integration in embedded systems*, pages 11--16, New York, NY, USA, 2008. ACM.

Hohmuth, M., Peter, M., Härtig H. e Shapiro, J. Reducing TCB size by using untrusted components: small kernels versus virtual-machine monitors. *EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop, pages 22*, New York, NY, USA, 2004. ACM.

Jerraya, A., Tenhunen, H. e Wolf, W. (2005). Multiprocessor systems-on-chips. In *Computer*, 38 (Issue 7), pages 36 – 40.

Lavagno, L. e Passerone, C. (2005). Design of embedded systems. In *Embedded Systems Handbook, chapter 3*, R. Zurawski (editor). CRC press.

Martin, G. (2006). Overview of the mpsoc design challenge. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 274–279, New York, NY, USA. ACM Press.

Nollet, V., Avasare, P., Mignolet, J-Y. e Verkest, D. Low cost task migration initiation in a heterogeneous MPSoC. In *DATE'05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 252-253, Washington, DC, USA, 2005. IEEE Computer Society.

Popek, G. e Goldberg, R. (1974). Formal requirements for virtualizable third generation architectures. In *Communication*. 17(7). pages 412–421. ACM press.

Shen, H e Petrot, F. Novel task migration framework on configurable heterogeneous MPSoC platforms. In *Design Automation Conference, 2009.*ASP-DAC 2009. *Asia and South Pacific*, pages 733-738, Jan. 2009.

Tanenbaum, A. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2007.

Waldspurger, C. (2002). Memory resource management in VMWARE ESX server. In: *SIGOPS Operating Systems Review*, 36(SI). pages 181–194.

Wolf, W (2003). A decade of hardware/software co-design. In: *Computer*, 36(4). pages 38–43.