# KVMのサーバ仮想化技術

## Kernel-Based Virtual Machine Virtualization

● 五島康文

### あらまし

KVM（Kernel-based Virtual Machine）はオープンソースのサーバ仮想化機能として，近年急速に注目されるようになってきた。KVMは2006年10月にその実装アイデアが紹介されると，そのシンプルなアイデアからLinux Kernelの開発者の間で支持を集めた。その結果，多くの開発者が集まり，急速に機能を拡張しており，ついにはRed Hat Enterprise Linuxが正式にサポートする機能となり，富士通もRed Hat Enterprise Linux 6からサポートを開始している。

本稿ではKVMについて，まずその基本的な内部の仕組みについて解説する。また関連するコンポーネントについて紹介する。そしてハードウェア・ソフトウェアによる，KVMの仮想化を支援する機能について紹介する。さらに，KVMが基幹業務での使用に耐えられるようにするための，富士通としての取組みについて，いくつかの予定を簡単に紹介する。

### Abstract

Kernel-based virtual machine (KVM) virtualization has been attracting attention in recent years as an open source server virtualization technique. Since its introduction in October 2006, this idea has aroused the interest of Linux kernel developers because of its simplicity, which has resulted in rapid extension of KVM functionality. KVM is now formally supported by Red Hat Enterprise Linux and has also been supported by Fujitsu since version 6 of Red Hat Enterprise Linux. This paper begins by explaining the internal structure of KVM and then describes the relevant components. It then introduces hardware and software support for KVM virtualization and briefly describes some enhancements planned by Fujitsu to enable KVM to be used in mission-critical work.

## まえがき

近年，x86系CPUにおけるサーバの仮想化技術は，様々な理由で注目されるようになっている。サーバの仮想化という技術自体は古くから実現されていた技術であるが，Intel/AMDのCPUにおいて，サーバの仮想化を支援する機能であるIntel VT（Intel Virtualization Technology）(1)/AMD-V（AMD-Virtualization）(2)が提供されることにより，x86系サーバを使って比較的安価に現実的な性能で実現できる環境が整ってきた。また，サーバの仮想化を実現するソフトウェアも様々なものが出そろうようになった。

この中で，OSS（Open Source Software）のサーバ仮想化機能として近年急速に台頭してきたのがKVM（Kernel-based Virtual Machine）(3)である。KVMはIntel VT-x/AMD-Vの機能を使うことを前提に設計され，比較的シンプルな構造で仮想化機能を実現した。2006年10月にQumranet（当時）のAvi KivityからKVMの実装のアイデアが公表(4)されると，シンプルな設計アプローチにLinux Kernel開発者の支持が集まり，その結果急速に機能が強化されてきた。このため，現在ではRed Hatから公式にサポートされるようになり，富士通もRed Hat Enterprise Linux 6（以下，RHEL6）からKVMのサポートを開始している。

本稿では，KVMのそのシンプルな内部構造について概説する。また，KVMを支えるソフトウェア・ハードウェアの支援機能についても簡単に紹介する。なお，CPUの機能について本稿ではIntel VT-xを取り上げる。

## KVMの仕組み

本章では，KVMの基本的な仕組みを理解いただくため，まずIntel VT-xを説明し，つぎにKVMの重要なコンポーネントであるQEMUについて解説する。

### ● Intel VT-xとセンシティブ命令

KVMはIntel VT-xの機能を使うことを前提に設計し，その機能の利用箇所をLinuxのカーネルの1機能として実現（注）している。このため，KVMを本質的に理解いただくために，まずIntel VT-xについて説明する。

Intel VT-xとは，「CPUがセンシティブ命令の実行を検出して，ハイパーバイザに処理を切り替えるようにした機能」であると言える。ここで，ハイパーバイザとは，**図-1**のように，実計算機上で動作する仮想計算機（すなわちゲストシステム）を実現するための制御プログラムである。またセンシティブ命令は，以下のような命令として定義されている。(5)

(1) システムリソースの状態を変えてしまう命令：Control Sensitive Instructions
(2) システムリソースの状態によって動作が変わってしまう命令：Behavior Sensitive Instructions

これらの命令を概念的に説明すると，以下であるとも言える。

(1) 仮想計算機上のプログラムが実行すると，実計算機の動作に影響を与えてしまう命令
(2) 実行すると，実計算機上で実行したときと結果が異なるため，仮想計算機上で実行していることが分かってしまう命令

そのような命令をゲストシステム上のプログラムによって勝手に実施されてしまうと仮想計算機としては成り立たないため，CPUがこのような命令が実行されたことを検出し，ハイパーバイザが代行して処理する必要がある。

しかし，x86系のCPUはもともと仮想化を想定し

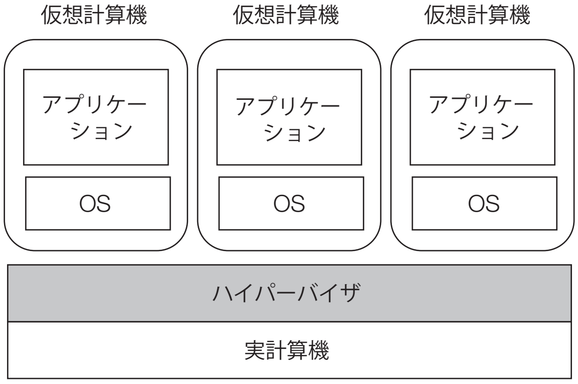


図-1　仮想計算機とハイパーバイザ
Fig.1-Hypervisor and virtual machine.

---

（注）　この利点として，カーネルとハイパーバイザとの親和性が高いため，ハイパーバイザの動作の支援にカーネルの機能を使いやすいという点が挙げられる。

ていなかったため，センシティブ命令をゲストシステムが勝手に実行しても，CPUが検出できない命令が存在した。このような命令の実行は，ハイパーバイザが代行処理できないため，これらの命令をすべて検出してハイパーバイザが処理できるモードに切り替えられるようにCPUを拡張した機能がIntel VT-xである。

Intel VT-xでは，VMX root operationモードとVMX non root operationモードの二つのプログラム実行モードが追加された。**図-2**のようにゲストシステムのための実行モードであるVMX non root operationモードのときにセンシティブ命令を実行すると，CPUがそれを検出し，ハイパーバイザ用の実行モードであるVMX root operationモードに移行するようになった。これをVM Exitと呼ぶ。このVM Exitを契機にハイパーバイザに制御を移し，ハイパーバイザがセンシティブ命令を代行処理することができるようになった。

VMX non root operationモードになるために，VMLAUNCH，VMRESUMEの二つのCPU命令が追加されている。また，これらの命令を実行してVMX non root operation モードになることをVM Entryと呼んでいる。

KVMの主な役割は，これらのVM Exitのハンドルと，VM Entryの実行であると言え，その機能をLinuxのカーネルモジュールとして実現している。

## ● KVMとQEMU

KVMのカーネルモジュールは，それ単体で仮想計算機を実現しているわけではなく，QEMU(6)というユーザ空間のプロセスと連携して実現している。

QEMUとはもともとはハードウェアエミュレータであり，一般的なx86系PCやほかの様々なアーキテクチャをエミュレーションするソフトウェアであり，オープンソースである。KVMの発表以前から存在するものであり，KVMがなくても単体で動作可能である。

QEMUはソフトウェアによるエミュレータである以上，CPU命令をソフトウェアで解釈して一つずつ実行するため，性能面ではあまり期待できなかった。しかし，直接実行できる命令であれば，そのままVMX non root operationモードで直接CPUに実行させ，直接実行できない処理のみを検出してQEMUにエミュレート処理を行わせるようにできれば，QEMUの性能を大きく改善でき，また仮想計算機としての機能を実現できる。KVMのアイデアとはこのような構想に基づいている。

またこのアイデアは，既存のOSSの資産を最大限に生かしつつ，最小限の改造で仮想計算機を実現するものであった。このために，Linux Kernel開発者に広く支持されたのである。

QEMU/KVMの大枠の実行フロー（**図-3**）を説明すると次のようになる。KVMカーネルモジュールは，導入すると/dev/kvmというファイルを作成する ｛図-3（0)｝。QEMUは起動すると，この/dev/

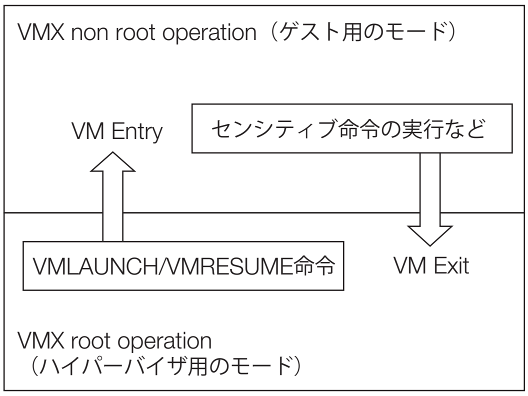


図-2　ハイパーバイザとIntel VT-xのプログラム実行モードの関係
Fig.2-Relationship between Hypervisor and program execution modes of Intel VT-x.

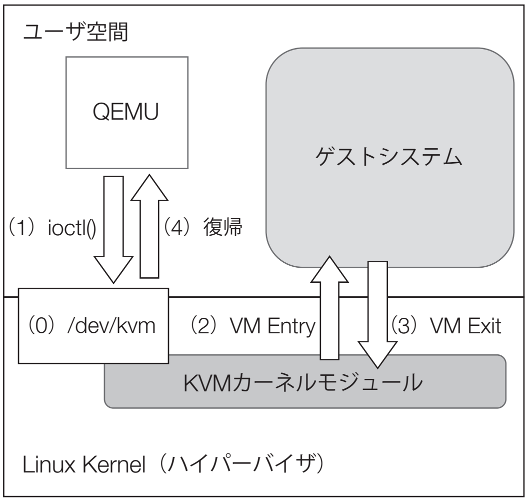


図-3　QEMU/KVMの大枠の実行フロー
Fig.3-Outline of flow of QEMU/KVM execution.

kvm，あるいはここを基点に作成されたファイルディスクリプタに対してioctl()システムコールでアクセスし，KVMに対してハイパーバイザとしての様々な要求を伝える。ゲストシステムの実行を開始するときもこのioctl()でKVMカーネルモジュールに指示を出す｛同図（1）｝。KVMカーネルモジュールはVM Entry｛同図（2）｝してゲストシステムの実行を開始する。しばらくしてゲストシステムがセンシティブ命令を実行してVM Exitする｛同図（3）｝と，KVMはそのVM Exitの原因を評価する。その結果，I/Oの実行などのようにQEMUの介入が必要であればQEMUプロセスに制御を移し｛同図（4）｝，QEMUが処理を行う。完了すれば，再びioctl()を実行してゲストの処理の継続をKVMに依頼する。すなわち同図（1）に戻る。QEMU/KVMは基本的にこの繰返しで仮想計算機を動作させている。

また，これによりQEMU/KVMは以下のように比較的シンプルな構造となっている。

（1）KVMカーネルモジュールを導入することで，Linux Kernelそのものがハイパーバイザとなる。

（2）QEMUというプロセス一つがゲストシステム一つに対応する。複数のゲストシステムを起動させるなら，それと同数のQEMUプロセスが起動するということである。

（3）QEMUはマルチスレッドプログラムであり，ゲストシステムの一つの仮想CPUは，QEMUプロセス内の一つのスレッドである。前述の図-3の（1）～（4）の動作もこのスレッド単位で動作する。

（4）QEMUのプロセス・スレッドは，Linux Kernelからは，ほかの一般的なユーザプロセスと同様に扱われる。ゲストシステムのVCPU（Virtual CPU）に対応するスレッドなどのスケジューリングもほかのプロセス・スレッドと同様にLinux Kernelのスケジューラが担当する。

## KVMを構成するコンポーネント

本章では，KVMに関係するQEMU以外のコンポーネントについて紹介し，KVMの全体像について説明する。

前章で紹介したQEMU自体は，実はCUI（Character User Interface）で起動するただのコマンドである。ゲストシステムの実行中にpsコマンドを表示すると，QEMUが起動していることは以下のように観察できる。

```
[goto@lifua ~]$ ps auxw |grep qemu
qemu        ......        /usr/bin/qemu-kvm -S -M
 fedora-13 -enable-kvm -m 1024 -smp 1, sockets=1,
cores=1, threads=1 .....
 -drive file=/home/goto/kvm_image/fedora13.img,
 .....    -device    rtl8139,    vlan=0,    id=net0,
mac=52:54:00:65:03:a0 ......
```

ここで，コマンドのオプション-m 1024はゲストシステムのメモリ量，-smp 1はゲストシステムのCPU数の指定である。このコマンドの出力では省略して1/3程度の行数にしているが，デバイスの設定など，QEMUのすべての設定がQEMUコマンドのパラメータとして渡されている。

さすがにこれらのオプションすべてを指定しながらQEMUを直接使うというのは使いにくい。このため，RHEL（Red Hat Enterprise Linux）などではvirt-manager（7）というGUI（Graphical User Interface）が用意されており，ゲストシステムの操作や管理を行うことができるようになっている。

ゲストシステムの起動中の画面を**図-4**に示す。左上のウインドウがvirt-managerの画面，左下と右のウインドウはゲストシステムのコンソールであり，二つのゲストシステムが起動している。なお，本画面はFedoraでの実行例であるため，RHEL6とは表示が若干異なる可能性がある。

また，virt-managerはこれらのGUIのほか，virshと呼ばれるCUIも持っており，virshからゲストシステムの操作も行うことができる。

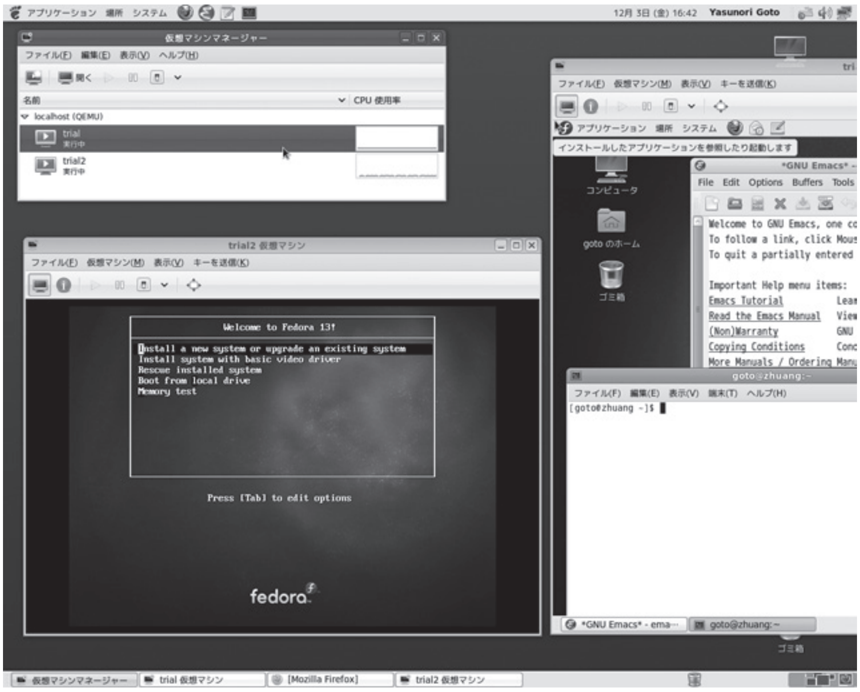
これらのようなユーザインタフェースからLinux Kernelまで含めたKVMは，以下で構成されていることになる。

（1）virt-manager

GUI・CUIのユーザインタフェースであり，仮想計算機の管理を担当する。つぎに述べるlibvirtを使って仮想計算機の機能を呼び出している。

（2）libvirt

XenやVMware ESX/GSXなどにも対応した，サーバ仮想化ソフトウェア共通のツールやインタ



```
Welcome to Fedora 13!

Install a new system or upgrade an existing system
Install system with basic video driver
Rescue installed system
Boot from local drive
Memory test

Press [Tab] to edit options
```

```
Welcome to GNU Emacs, one co
To follow a link, click Mous
To quit a partially entered

Important Help menu items:
Emacs Tutorial          Lear
Read the Emacs Manual   View
(Non)Warranty           GNU
Copying Conditions      Cond
More Manuals / Ordering Manu
```

```
[goto@zhuang ~]$ 
```

図-4　ゲストシステムの実行中の画面
Fig.4-Screen when guest systems are executing.

フェースライブラリ(8)である。もちろんQEMU/KVMにも対応している。

（3）QEMU

前述のとおり，KVMカーネルモジュールと連携してI/Oの実行などゲストシステムの多くの処理を行う。一つのQEMUプロセスが一つのゲストシステムに相当する。

（4）KVMカーネルモジュール

狭い意味でのKVMとはこのLinuxカーネルモジュールである。ゲストシステムのVM Exitのハンドリングや，VM Entryの実行などを行う。

（5）Linux Kernel

QEMUが一般的なプロセスと同じであることから，ゲストシステムのスケジューリングなどはLinux Kernel本体が行っている。

そして，virt-managerなどのコンポーネントをすべて含めた構成として図-3を書き直すと，**図-5**のようになる。これらはすべてOSSで構成されている。

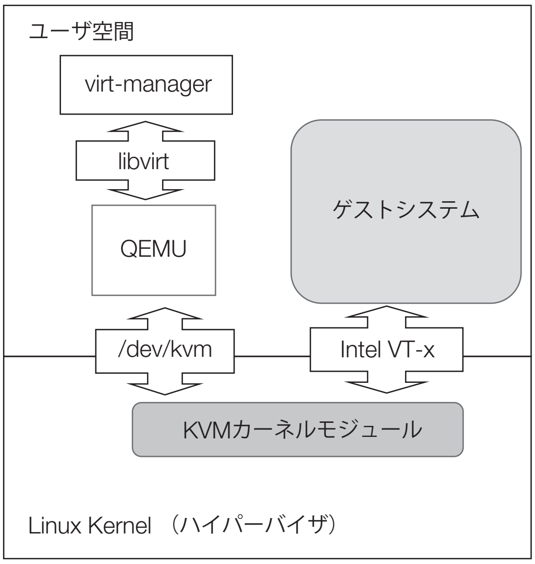


図-5　KVMの全体像
Fig.5-Whole image of KVM.

## KVMを支えるハードウェア・ソフトウェアの支援機能

サーバの仮想化について最も懸念されるのは，その性能である。実計算機と比較して仮想計算機の性能劣化はできる限りないほうがよい。

そこで，これまで述べてきた基本的な仕組みのほかに，仮想化のためにハードウェア・ソフトウェアの様々な支援機能があり，KVMの性能改善を図っている。本章ではそれらについていくつかを簡単に紹介する。

### ● EPT（Extended Page Table）

EPTとは，CPUが提供するアドレス変換機構（MMU：Memory Management Unit）の拡張機能である。

仮想計算機ではハイパーバイザとゲストシステムの2重構造になるため，そのようなことを想定していない従来のMMUの構造をそのまま適用することはできなかった。ゲストシステムに割り当てる物理アドレスは，あくまで仮想的な物理アドレスであり実計算機の物理アドレスとは異なるため，その変換を行う必要がある。

このため，EPTの登場以前はShadow Pagingという技法により，ソフトウェアでアドレス変換処理を行う必要があった。しかし，EPTの登場によって仮想計算機上の物理アドレスから実計算機の物理アドレスへの変換をCPUで処理できるようになり，ソフトウェアによるアドレス変換処理が不要となった。

KVMはEPTに対応済みであり，仮想計算機の性能が大幅に向上している。

### ● VT-d

VT-dはI/Oデバイスのためのアドレス変換機構（IOMMU）である。VT-dは，それぞれのデバイスの機能ごとにメモリアドレスの変換テーブル（データ構造としてはMMUのPagetableに「うり二つ」である）を持つことができる機能である。これを設定することより，デバイスからのデータの転送先として，ゲストシステムのメモリのアドレスを指定することができ，ゲストOSへデータ転送を直接行うことが可能となる。

VT-dはチップセットの機能であり，またファームウェアがVT-dに対応している必要がある。ともに対応していればLinux Kernelが認識して，VT-dが使用されるようになっている。

### ● virtio

ゲストシステムのデバイスは，通常QEMUのデバイスエミュレーションによって実現されている。しかし，このエミュレーション処理のオーバヘッドが大きいため，I/Oの性能は非常に悪い。このエミュレーションを回避する仕組みがvirtioである。

virtioは，ゲストシステムとQEMUからともにアクセスできるバッファを用意する。ゲストシステムは，データをそのバッファに書き込んでQEMUに通知し，データの処理をまとめて行うフレームワークを用意した。これらの仕組みは，ゲストシステムからはvirtioのPCIデバイスとしてアクセスできる。

ゲストOSにvirtio用のドライバ（もちろんWindowsのvirtio用のドライバもある）を導入することで，virtioの効果を得ることができ，I/Oの高速化を行うことができる。

### ● KSM（Kernel Samepage Merging）

一つの実計算機上の複数のゲストが同じOSやアプリケーションを実行しているというのは，仮想計算機を運用するとよく発生する状況である。このような場合，複数のゲストシステムそれぞれに，同じ内容を持つメモリ領域がある可能性が高い。そこで，メモリ領域の内容が同じものがあれば，それらを一つにまとめてしまうことができれば，メモリの使用量を削減することが期待できる。

そこで，Linux KernelにKVMのためにKSM(9)という機能が追加された。KSMは，カーネルスレッドであるksmdがプロセスの使用メモリを定期的に監視し，重複したページを自動的にマージして共有ページにしてしまう機能である。

無論，メモリの内容が重複しているかどうかは，そのページの内容すべてを比較しなければならず，システムの全プロセスが使用しているページすべてを毎回比較していては非常に効率が悪い。

このため，LinuxではKSMの対象とするかどうかをmadvise()システムコールの第3引数のadviceで指定できるようになっている。QEMUはゲスト用のメモリ確保時にこれらを指定しており，ユーザはKSMの機能の恩恵をそのまま受けることができる。RHEL6でもこの機能は初期状態で有効になっている。

## KVMの今後の強化

基幹業務でKVMを使用できるようにするためには，まだ機能・品質的に強化しなければならないものも多い。そこで本章では，富士通が今後KVMを強化していく予定の開発項目について，そのいくつかを簡単に紹介する。

### ● libvirtなどの機能・品質強化

KVMカーネルモジュールやQEMUの完成度と比べると，libvirtなどの機能はまだまだ不足してお

り，とてもKVMの持つ力を100％発揮できているとは言えない状態である。また，品質的にも安定していないため，機能・品質強化が急務である。

### ● 資源管理機能強化

RHEL6から利用可能となった資源管理機能，Cgroup(10)との連携が期待されている。現在のCgroupでも，一般的なプロセスに対する実計算機のCPUの割当て量など様々な資源の割当てを設定できるようにはなっているが，ゲストシステムの制御という観点でKVMとCgroupを連携させることを考えると，I/Oの制御機能に不十分な点があるなど，いくつかの問題があることが分かっている。これらについて機能を強化していく予定である。

### ● マシンチェック対応

実計算機のハードウェアに訂正不可能なマルチビットエラーなどの障害が発生した場合，それらの異常をゲストシステムに通知する必要がある。基本的な枠組みはすでにKVMに作り込まれているが，ゲストOS上でのメモリエラーリカバリの実現など，KVMに特化したマシンチェック対応の強化が必要である。

## むすび

本稿では，KVMの基本的な仕組みや，サーバの仮想化を支援する機能について概説した。これらの機能は，Red Hat Enterprise Linux 6に標準搭載されている。

また，富士通としての今後のKVMの機能強化・開発項目について紹介した。これらの機能の開発などの活動を通して，KVMは今後のお客様のシステムを支える重要なコンポーネントになっていくと考えている。

## 参考文献

（1）インテル® バーチャライゼーションテクノロジー（インテル® VT）. http://www.intel.com/jp/technology/virtualization/

（2）AMD－Virtualization. http://www.amd.com/jp/products/technologies/virtualization/Pages/virtualization.aspx

（3）Main Page－KVM. http://www.linux-kvm.org/page/Main_Page

（4）［PATCH 0/7］KVM：Kernel-based Virtual Machine. http://marc.info/?l=linux-kernel&m=116126591619631&w=2

（5）Gerald J. Popek，Robert P. Goldberg：Formal Requirements for Virtualizable Third Generation Architectures，（1974）. http://portal.acm.org/ft_gateway.cfm?id=361073&type=pdf&CFID=5009197&CFTOKEN=14838199

（6）About－QEMU. http://wiki.qemu.org/Main_Page

（7）Virtual Machine Manager. http://virt-manager.et.redhat.com/

（8）libvirt：The virtualization API. http://libvirt.org/

（9）［PATCH 0/4］ksm - dynamic page sharing driver for linux. http://marc.info/?l=linux-kernel&m=122640987623679&w=2

（10）石井宏延：PRIMEQUESTを支えるLinux OSの機能強化への取組み. *FUJITSU*, Vol.61, No.6, p.617-624（2010）.

## 著者紹介

**五島康文**（ごとう　やすのり）

プラットフォームソフトウェア事業本部Linux開発統括部 所属

現在，Linux kernel/KVMの開発・サポートに従事。