1.3

REDHAWK *KVM-RT™ User's Guide*

concurrent REAL-TIME

0898604-1.3
October 2021

本書は当社製品を利用する社員、顧客、エンドユーザーを対象とします。
本書に含まれる情報は、本書発行時点での正確な情報ですが、予告なく変更されることがあります。
当社は、明示的、暗示的に関わらず本書に含まれる情報に対して保障できかねます。

誤字・誤記の報告または本書の特定部分への意見は、当該ページをコピーし、コピーに修正またはコメントを記述してコンカレント日本株式会社まで郵送またはメールしてください。

http://www.concurrent-rt.co.jp/company/

Revision History	Level	Effective With
July 2019	1.0	RedHawk Linux 7.5
January 2020	1.1	RedHawk Linux 8.0
February 2021	1.2	RedHawk Linux 8.2
October 2021	1.3	RedHawk Linux 8.4

注意事項:

本書は、Concurrent Real-Time, Inc.より発行された「RedHawk KVM-RT User's Guide」を日本語に翻訳した資料です。英文と表現が異なる文章については英文の内容が優先されます。

前書き

マニュアルの範囲

本書はCocurrent Real-TimeのRedHawk KVM-RT™を利用するための情報と取扱説明について提供します。

マニュアルの構成

本書は以下のセクションで構成されます:

- 1章 KVM-RTを紹介します。
- 2章 KVM-RTで仮想マシーンをセットアップおよび起動する手順について説明します。
- 3章 KVM-RTの構成する方法を取り上げます。
- 4章 全てのKVM-RTツールを要約します。
- 5章 時刻の同期を説明します。
- 6章 KVM-RTのゲストVMを解析およびデバッグする方法を説明します。

構文記法

本書を通して使用される表記法は以下のとおりとなります。

斜体
: ユーザーが特定する書類、参照カード、参照項目は、*斜体*にて表記します。特殊用語も*斜体*にて表記します。

太字
: ユーザー入力は**太字**形式にて表記され、指示されたとおりに入力する必要があります。ディレクトリ名、ファイル名、コマンド、オプション、manページの引用も**太字**形式にて表記します。

`list`
: プロンプト、メッセージ、ファイルやプログラムのリストのようなオペレーティング・システムおよびプログラムの出力は`list`形式にて表記します。

`[]`
: ブラケット(大括弧)はコマンドオプションやオプションの引数を囲みます。もし、これらのオプションまたは引数を入力する場合、ブラケットをタイプする必要はありません。

ハイパーテキスト・リンク
: 本資料を見ている時に項、図、テーブル・ページ番号照会をクリックすると対応する本文を表示します。青字で提供されるインターネットURLをクリックするとWebブラウザを起動してそのWebサイトを表示します。赤字の出版名称および番号をクリックすると(アクセス可能であれば)対応するPDFのマニュアルを表示します。

関連図書

下の表にConcurrent Real-Timeの文書を記載します。文書にもよりますがRedHawk Linuxシステム上、またはConcurrent Real-Timeの文書Webサイト**http://redhawk.concurrent-rt.com/docs** からでオンラインで利用可能です。

RedHawk KVM-RT	**文書番号**
RedHawk KVM-RT Release Notes	0898603
RedHawk KVM-RT User's Guide	0898604
RedHawk Architect	
RedHawk Architect Release Notes	0898600
RedHawk Architect User's Guide	0898601
RedHawk Linux	
RedHawk Linux Release Notes	0898003
RedHawk Linux User's Guide	0898004
RedHawk Linux Cluster Manager User's Guide	0898016
RedHawk Linux FAQ	N/A
NightStar RT Development Tools	
NightView User's Guide	0898395
NightTrace User's Guide	0898398
NightProbe User's Guide	0898465
NightTune User's Guide	0898515

目次

1
KVM-RTの概要

本章は、RedHawk KVM-RTを使用に関する一般的な概要と要件を提供します。

概要

RedHawk KVM-RTは、ゲストのRedHawk仮想マシーンに対してRedHawkのリアルタイム・デターミニズムを拡張するためにQEMU/KVMとRedHwkのリアルタイム特性を利用するリアルタイム・ハイパーバイザー・ソリューションです。

これはホスト・システム上の仮想マシーン内で複数のゲスト(リアルタイムと非リアルタイムの両方)の実行をサポートします。

ホスト・システムの要件とインストール

ハードウェア・ホスト・システムの要件とソフトウェアのインストール手順については *RedHawk KVM-RT Release Notes*を参照して下さい。

要件ではありませんが、ホスト・システム全体をリアルタイム・ハイパーバイザーの実行に専念させることを強く推奨します。KVM-RTホスト・システムの管理者はシステム上のCPUシールディングまたはCPUアフィニティを阻害しないように注意する必要があります。さもないと仮想マシーンのリアルタイム性能が犠牲になる可能性があります。

KVM-RTがインストールされたら、ホスト・システムの適合性を分析するために次のコマンドを実行することが可能です。

```
$ sudo kvmrt-validate-host
```

ホスト・カーネル構成

KVM-RTが使用される間はRedHawkカーネルがホスト・システムで起動されていることをKVM-RTは要求します。追加のシステム構成が必要になる可能性があります。

カーネル起動パラメータ

これらのパラメータは、7.xリリースでは**/usr/src/linux-<***kernel-name***>/Documentation**以下の**kernel-parameters.txt**ファイル内、8.xリリースではそのパスの下のサブディレクトリ**admin-guide**以下にも記載されています。

これらのパラメータは、RedHawk Version 8.0以降では**`blscfg(1)`**コマンド、UbuntuリリースおよびRedHawk Version 7.xでは**`ccur-grub2(1)`**を使いRedHawkシステムの起動時パラメータに追加することが可能です。パラメータを有効にするには再起動が必要であること、お気付きのように全てのリリースで利用可能ではないことに注意して下さい。

`intel_iommu` = on / *`amd_iommu`* = on

本パラメータは仮想マシーンのDMAで使用されるメモリ領域のデバイス・レベルの再マッピングを有効にします。本パラメータはいずれの仮想マシーンが物理PCIデバイスのPCIパススルーを使用する場合には必要となります。

`workqueue.pri` = 3

リアルタイム仮想マシーンとして有効になっている仮想マシーンは物理CPUを事実上「所有」します。ホストに仮想マシーンの制御を保持させるために本パラメータはホストのデーモンおよびプロセスをより高い優先度で実行することを許可します。本パラメータはリアルタイム仮想マシーンにリアルタイム性能を保証するために必要となります。

一部のPCI-eカードはintel_iommuとは互換性がありません。ユーザーがIOMMUを有効にした時に適切に動作しないデバイスを持っている場合、次の2つのオプションが役立つ可能性があります。これらのオプションはRedHawkリリース7.5以降で利用可能であることに注意して下さい。同様にRedHawkリリース7.5では、以下に記載されている起動パラメータ*`intel_iommu.exception_ids`*は*`intel_iommu.blacklist_ids`*という名前になっていることにも注意して下さい。これは8.0以降のリリースで名称が変更されました。

`intel_iommu` = plx_off

PLXブリッジの後ろにある全てのデバイスをIOMMUが再マッピングするのを防ぐにはこれを有効にして下さい。これらのデバイスはPCIパススルーで使用することは出来ません。

`workqueue.exception_ids` = [vendor:device, …]

IOMMU再マッピングから除外されるカンマ区切りのデバイスのリスト。これらのデバイスはPCIパススルーで使用することは出来ません。ベンダーとデバイスは0xの接頭語なしの16進数として指定されます。

ユーザーはシステム内の重複するデバイスの一部だけを仮想マシーンに専念してもらいたいと望んでいるかもしれません。起動時にVFIOドライバーに対して個々のデバイスを予約するには本オプションを使用して下さい。これは起動中の初期に他の動的モジュールがデバイスを獲得するのを防止します。本オプションはRedHawkリリース8.0以降でサポートされます。

`vfio-pci.addrs` = [BUS:SLOT.FUNCTION, …]

VFIOドライバーに割り当てられるカンマ区切りのPCIデバイスのリスト。

管理IRQの移動

CPU毎の割込みは管理割込みに分類することが可能です。最新のNIC、RAID、NVMEデバイスは管理割込みを生成します。RedHawkリリースVersion 8.2では、管理割込みを他のCPUへ移動することが可能となるように変更されました。本変更はリリース7.5以降に対してバックポートされています。最新のリリース・アップデートをお持ちである場合は本変更を得ています。管理割込みは7.5以前のリリースでは存在していなかったことに注意して下さい。

管理割込みの移動はVMのリアルタイム性能に影響を及ぼす可能性がありますのでKVM-RTでは必要です。また、KVM-RTはハイパースレッド化されたCPUを停止しようとします。IRQがCPUと関連付けられている場合はCPUを停止させることが出来ません。目標は全てのIRQをvCPUに対する責任を持つCPUから移動すること、およびエミュレーションとVirtIO操作を担当するCPUにそれらを移動する事です。

KVM-RTツール(**`irq-affinity`**と**`task-affinity`**)はIRQとタスクのCPUアフィニティをそれぞれ表示し、特定のCPUにバインドしたIRQやタスクを探すのに非常に便利です。これらのコマンドの詳細および使用方法については**`--help`**オプションを使用して下さい。

管理割込みはCPUアフィニティ・マスクに1つのCPUしか持てないため、それらを移動するのに**`shield(1)`**コマンドを使用することは出来ません。以下は管理IRQをCPUから移動するための方法の一部です。

1. **`/proc/irq/`**<*irq-no*>**`/smp_affinity_list`**ファイルに書き込むことでIRQのアフィニティ・マスクをリセットします。変更はブートを超えて持続しないことに注意して下さい。従って後述のカーネル起動パラメータ***`msi_affinity_mask`***を設定することを推奨します。次の例では、IRQ番号11はCPU 0-11と15-25で実行するように設定されます。

   `echo “0-11,15-25” > /proc/irq/11/smp_affinity_list`

2. カーネル起動パラメータ***`msi_affinity_mask`***に設定すると、起動時に全てのMSI(X)管理割込みにアフィニティ・マスクを設定します。現在、パラメータ***`irqaffinity`***も同じ値が設定される必要がありますが、将来のリリースでは***`msi_affinity_mask`***だけが設定される必要があることに注意して下さい。また、本機能はRedHawk 8.Xリリースでのみ利用可能です。

   `msi_affinity_mask` = [cpulist]
   `irqaffinity` = [cpulist]

   cpulistはCPU 0を含む必要のあるCPUのリストが設定される必要があります。リストは0-5のような範囲、または0,3,4,5のようなカンマ区切りのリストを含めることが可能です。

3. **`system shield`**サービスは選択したCPUのシールド属性を設定するために使用することが可能です。変更は次のファイルに対して行います：
   `/etc/sysconfig/shield`

   例えば、enp4s0f0の割込みをCPU 0から4に割り当て、または割込み番号55, 60, 61をCPU 0と2に割り当てるにはこれらの行を**`shield`**サービス構成ファイルに追加することで可能となります。

   IRQ_ASSIGN+=”0-4:enp4s0f0;”
   IRQ_ASSIGN+=”0,2:55; 0,2:60, 0,2:61;”

   ファイルを編集後、次のコマンドを使ってサービスを再開することが可能です：
   `systemctl restart shield`

   続いてコマンドのステータスを確認することが可能です：
   `systemctl status shield`

2
はじめに

本章ではKVM-RTで仮想マシーンをセットアップおよび起動する手順について説明します。また、各仮想マシーンのホスト上で実行する様々なQEMU/KVMスレッドについても説明します。

仮想マシーンの構築

KVM-RTはlibvirtフレームワーク内で生成および構成された仮想マシーンと連動します。仮想マシーンは次を含むいくつかの方法でlibvirt内に生成し構成することが可能です：

- 仮想マシーン・マネージャーを使用
- RedHawk Architectを使用
- 他の仮想マシーンのクローン作成

仮想マシーンを構築する詳細な手順は本書の範囲を超えますが、十分に文書化されています。汎用的な手順書および文書の参照は次項で提供されます。

リアルタイム仮想マシーンはRedHawk Linux 7.0以降のゲストOSが含まれている必要があります。ゲストCPUアーキテクチャはホストと一致している必要があります。

仮想マシーンを生成するために仮想マシーン・マネージャーを使用

仮想マシーン・マネージャーはlibvirtフレームワーク内の仮想マシーンを生成、構成、管理するために使用することが可能なGUIツールです。

次を実行して仮想マシーン・マネージャーを開始して下さい：

```
$ sudo run virt-manager
```

詳細についてはのmanページを参照して下さい。

仮想マシーンを生成するために**RedHawk Architect**を使用

RedHawk Architectは、RedHawk Linuxのディスク・イメージを生成、カスタマイズ、展開することに特化したConcurrent Real-Timeが提供するオプション製品です。

ArchitectはRedHawkの仮想マシーンを生成し、それを仮想マシーン・マネージャーにエクスポートするために使用することが可能です。詳細な手順についてはRedHawk Architectに付属する文書内で見ることが可能です。以下は必要となる一般的な手順となります：

- Architectを実行
- 新しいセッションを生成し、望むようなイメージを構成
- イメージを構築
- 仮想マシーンにイメージを展開
- 仮想マシーン・マネージャーに仮想マシーンをエクスポート

仮想マシーン・イメージのクローン作成

libvirtフレームワーク内の既存の仮想マシーンはvirt-cloneコマンドを使用することでクローンを作成することが可能です。実行例：

```
$ sudo virt-clone -o old_vm -n new_vm
```

詳細については**`virt-clone(1)`**のmanページを参照して下さい。

仮想マシーンを**KVM-RT**にインポート

仮想マシーンがlibvirtフレームワーク内に生成されたら、KVM-RTにインポートすることが可能です。

全てのlibvirtの仮想マシーンは次のコマンドを使ってKVM-RTにインポートすることが可能です：

```
$ sudo kvmrt-import
```

本コマンドは新しいVMが生成された時にいつでも実行することが可能です。詳細およびオプションについては**`kvmrt-import --help`**を実行して下さい。

VMがKVM-RTにインポートされた時、VMの構成設定はlibvirtから継承します。これが終了するとVMは必要に応じてKVM-RTを使って更に構成することが可能です。詳細については3章の「仮想マシーンの構成」を参照して下さい。

仮想マシーンの起動とシャットダウン

仮想マシーンがKVM-RTにインポートされると次のKVM-RTツールがVMを起動、シャットダウン、ステータス表示するために使用することが可能です。

構成されている全てのVMを開始するには次のコマンドを実行して下さい：

```
$ sudo kvmrt-boot
```

構成されている全てのVMをシャットダウンするには次のコマンドを実行して下さい：

```
$ sudo kvmrt-shutdown
```

全てのVMのステータスを問合せるには次のコマンドを実行して下さい：

```
$ sudo kvmrt-stat
```

個々のVMは**kvmrt-boot**および**kvmrt-shutdown**コマンドを使って起動またはシャットダウンすることが可能です。実行例：

```
$ sudo kvmrt-boot RedHawk-8.4
$ sudo kvmrt-shutdown RedHawk-8.4
```

詳細およびオプションについては上記のコマンドのいずれかに**--help**オプションを付けて実行して下さい。

QEMU/KVMスレッドの理解

QEMU/KVMは各仮想マシーンに対して複数のスレッドを実行します。これらのスレッドの名称と目的は次のとおりです：

qemu-kvm

エミュレータ・スレッドです。これらは2つ以上になる可能性があります。

qemu-system-x86

一部のディストリビューションでは*qemu-kvm*の別名です。

worker

エミュレータにより実行される長いI/O操作用に動的に生成されたスレッドです。

SPICE Worker

仮想コンソール用のスレッドです。

IO mon_ioth

一部のI/Oで使用されるオプションのスレッドです。

CPU n/KVM

仮想CPU(vCPU)スレッドです。仮想CPUごとに1個あり、*n*はvCPU IDです。

現在実行中の全てのスレッドに関する情報を表示するには**kvmrt-stat -t**コマンドを使用して下さい。

3
仮想マシーンの構成

libvirtフレームワーク内に構成された仮想マシーンは、仮想マシーンの全ての属性を制御するXML構成ファイルを持っています。

本ファイルは通常、任意のVMドメイン名を表す**/etc/libvirt/qemu/{DOMAIN}.xml**として存在し、VMがlibvirtフレームワーク内に生成またはインポートされた時に生成されます。本ファイルはVM構成の変更が仮想マシーン・マネージャーで行われた時に更新されます。

KVM-RTは複数のVMを管理するために後述する簡易化された構成ファイルを使用します。KVM-RTは2つのファイルの同期を維持するために必要に応じてlivvirt XML構成ファイルを更新します。

KVM-RT構成ファイル

KVM-RT構成ファイルのデフォルトの保管場所は**/etc/kvmrt.cfg**ですが、構成ファイルを使用する全ての**kvmrt-***ツールはユーザーが代替の構成ファイルを指定することを許可する**-f**オプションを受け付けます。

KVM-RT構成ファイルはINIファイルの書式を使用しており、各セクションでVMについて説明します。各セクションの最初の行は、libvirtにより生成された一意的なVMの識別番号であるUUIDです。構成の実例を以下に示します：

```
[aeec46cc-0638-4949-ac04-146b233194a9]
name = RedHawk-8.4
title = RedHawk 8.4
description = A RedHawk 8.4 VM.
nr_vcpus = 2
cpu_topology = auto
cpuset =
rt = False
rt_memory = auto
numatune = auto
hide_kvm = False
autostart = True
comments = This VM tends to run out of memory;
       remember to clean up

[fde74e84-0e1b-404e-90e7-72101e79c48a]
name = RedHawk-8.4-RT
title = Real-Time RedHawk 8.4
description = A RedHawk 8.4 VM configured real-time.
nr_vcpus = 4
cpu_topology = auto
cpuset = 1-5
rt = True
```

```
rt_memory = auto
numatune = auto
hide_kvm = False
autostart = True
comments = remember to change back autostart = False
       after testing
```

以下に定義されているのは後述する属性の説明内で使用されているフィールドの型です：

{ string }: 任意の文字列

{ int }: 任意の整数

{ bool }: **`true | false | on | off | yes | no | 1 | 0`**
(大文字と小文字の区別なし)

{ ID-set }: 「0,2,4-7,12-15」などの形式で人間が解読可能な整数の範囲のセットを説明する文字列

各VMは次の属性を使って構成されます。属性が設定されていないもしくはファイルからなくなっている場合、デフォルト値が使用されることに注意して下さい。

`name` = { string }

本属性はVMの名称を設定します。これは任意ですが、libvirtに対して一意である必要があるユーザーが指定する名称です。
デフォルトの値はなく、本属性は設定されている必要はありますが変更することが可能です。

`title` = { string }

本属性はVMのタイトルを設定します。
デフォルトの値は””です。

`description` = { string }

本属性はVMの説明を設定します。
デフォルトの値は””です。

`nr_vcpus` = { int }

本属性はVM内の仮想CPUの数を定義します。
デフォルトの値は**`1`**です。

`cpu_topology` = { int },{ int },{ int } | **`auto`**

本属性はVMに認識されるCPUトポロジーを定義します。

`auto`ではない場合、値はCPUトポロジーを表現するためにカンマで区切られた3つの正の整数の文字列(ソケット、コア、スレッド)である必要があります。ソケットはCPUソケットの数、コアはソケットあたりのコアの数、スレッドはコアあたりのスレッドの数となります。

値が**`auto`**である場合、トポロジーは1個のソケット、ソケットあたり*`nr_vcpus`*個のコア、コアあたり1個のスレッドが設定されます。

デフォルトの値は**`auto`**です。

NOTE

ゲストの仮想マシーンがWindowsオペレーティング・システムを実行する場合、***cpu_topology***属性がKVM-RTで正しく動作しないデフォルト値に設定されている可能性があります。本設定を**auto**に変更するのが最適です。KVM-RT Release Notesの既知の問題項にある「Windowsオペレーティング・システムが動作するVM」のラベルの付いた項目を参照して下さい。

cpuset = { ID-set }

本属性は全てのVMスレッドがバイアスされるホストのCPU IDを定義します。デフォルトの値は""(CPUバイアスなし)です。詳細については本章で後述する「cpuset構成属性の理解」を参照して下さい。

rt_memory = { bool } | **auto**

本属性はVMで使用される全ページのメモリ・ロックを有効にします。

値が**auto**である場合、本オプションは***rt***属性が有効化されていれば有効、***rt***属性が無効化されていれば無効となります。

デフォルトの値は**auto**です。

numatune = { ID-set } | **auto**

本属性はVMへのメモリ割り当てで使用されるホストのNUMAノードを設定します。

autoではない場合、この値はホストのNUMAノードのセットを記述する必要があります。設定が空である場合、メモリはいずれのホストのNUMAノードにも制限されません。***cpuset***が空であった場合もメモリはいずれのホストのNUMAノードにも制限されません。

値が**auto**である場合、***cpuset***で使用される全てのNUMAノードが使用されます。

デフォルトの値は**auto**です。

hide_kvm = { bool }

本属性はVM内のゲストOSの表示からKVMを隠します。
デフォルトの値は**false**(無効)です。

rt = { bool }

本属性はリアルタイム用にVMを構成します。
デフォルトの値は**false**(無効)です。

本属性が有効である場合は***cpuset***と***rt_memory***の属性は(有効に)構成されている必要があります。本属性が有効である場合は***numatune***も有効に構成することを推奨します。

autostart = { bool }

本属性は**kvmrt-boot**を使ったVMの自動起動を有効にします。
デフォルトの値は**true**(有効)です。

comments = { string }

ユーザー・コメントの場所となります。複数行のコメントの場合、スペースまたはTABを使って追加行を字下げして下さい。

構成ツール

KVM-RTの構成は次のコマンドを実行することで編集することが可能です：

```
$ sudo kvmrt-edit-config
```

KVM-RT構成ファイルは直接編集すべきではないことに注意して下さい。**kvmrt-edit-config**は妥当性を検証し、また、**libvirt**と構成を同期させます。

KVM-RTによって解釈されるKVM-RT構成は、次のコマンドの実行により表示することが可能です：

```
$ sudo kvmrt-show-config
```

kvmrt-validate-configと**kvmrt-sync-config**のコマンドはそれぞれ構成の妥当性の検証および同期させるために実行することが可能です。**kvmrt-edit-config**を使用する場合、ユーザーは通常これらのコマンドを直接実行する必要はありません。

詳細およびオプションについては上記のコマンドのいずれかに**--help**オプションを付けて実行して下さい。

高度なLibvirt構成

KVM-RT構成ファイルの範囲を超える高度な構成は、仮想マシーン・マネージャーまたは「virsh edit」を使って**libvirt** XMLファイルに対して行うことが可能ですが、追加の同期および妥当性の検証がKVM-RTに必要となります。これは**libvirt**からVMを削除する場合も当てはまります。

一部の構成の組み合わせは無効である可能性があることに注意し、いつであろうともユーザーは**kvmrt-edit-config**を使いKVM-RT構成ファイルを編集して構成を変更することを推奨します。

libvirt XMLファイルをKVM-RTの外でユーザーが変更した場合、次のように**kvmrt-sync-config**および**kvmrt-validate-config**を実行する必要があります：

```
$ sudo kvmrt-sync-config -r
$ sudo kvmrt-validate-config
```

また、次のように**kvmrt-import -u**を**kvmrt-sync-config -r**の代わりに使用することも可能であることに注意して下さい：

```
$ sudo kvmrt-import -u
$ sudo kvmrt-validate-config
```

kvmrt-validate-configコマンドはどの無効な構成に関しても適切なエラーまたは警告を表示します。

詳細およびオプションについては上記のコマンドのいずれかに**--help**オプションを付けて実行して下さい。

cpuset構成属性の理解

*cpuset*属性は仮想マシーンのQEMU/KVMスレッドのためのホストCPUバイアスを制御します。

*cpuset*属性はリアルタイムと非リアルタイムVMの両方で使用することが可能です。*cpuset*が空の場合、VMはどの特定のホストCPUに対しても固定されません。

*cpuset*の最初のCPUは全ての非vCPUスレッドに割り当てられます。*cpuset*の残りのCPUは次のようにvCPUスレッドにより使用されます：

- 全ての仮想CPUスレッドのCPU配列ポリシーは、(1つのCPUに全ての非vCPU VMスレッドを割り当てた後に)*cpuset*により定義されたホストCPU上でラウンド・ロビンとなります。
- ホストCPUの供給過多(*cpuset*内のCPUが*nr_vcpus* + 1以上)は結果的に各仮想CPUが1個以上のホストCPUに固定されます。
- ホストCPUの供給不足(*cpuset*内のCPUが*nr_vcpus* + 1以下)は結果的に各ホストCPUに1個以上の仮想CPUが固定されます。

KVM-RTによるRedHawkリアルタイム機能の使用の理解

構成ファイル内で*rt*構成属性が有効化されている場合、次のRedHawkリアルタイム・システムの機能が実行されます：

- *cpuset*の全てのCPUがシールドされます。**shield(1)**を参照して下さい。
- ハイパースレッド化されたシブリングが停止されます。**cpu(1)**および後述の「KVM-RTによるスレッド化CPUの使用」を参照して下さい。
- メモリ・ロックが有効化されます。**run(1)**の**-L**オプションを参照して下さい。

*rt*構成属性が有効化されている場合には*numatune*も有効にすることを推奨します。*numatune*が有効化されると：

- 指定されたNUMAノードはリアルタイムVMへのメモリ割り当てに使用されます。**NUMA(1)**を参照して下さい。
- メモリ・シールドを試みます。**memory_shielding(7)**を参照して下さい。

NUMAメモリ・シールドのサポートを活用するには、NUMAノード内の全てのCPUがシールドされる必要があります。CPUがリアルタイムVMに割り当てられるとそのCPUはシールドされます。

KVM-RTによるスレッド化CPUの使用

Intelのハイパースレッドのようなスレッド化CPUアーキテクチャを持つホスト・システムにおいて、リアルタイムVMが使用されている場合にKVM-RTはマルチ・スレッド化CPUコアに対して特別な処理を提供します。

CPUコア・リソース(キャッシュ等)の競合を回避するために1つのスレッド化シブリングCPUだけが使用されることをリアルタイムは要求します。これを確実にするため、リアルタイムVMに割り当てられた各CPUコアの1つのスレッド化シブリングCPUを除いて全てをKVM-RTはシャットダウンします。これはVMの*cpuset*を割り当てる時にいくつかの考慮を必要とします。

リアルタイムVMには*cpuset*で指定されたCPUに関連する全てのスレッド化シブリングCPUの所有権が与えられます。これはVMが消費するCPUは*cpuset*で指定された以上は使用しないことになります。スレッド化コア毎に1つのCPUだけがリアルタイムで使用され、他はシャットダウンされます。

非リアルタイムVMをホストするスレッド化コアに対しては特別な処理は提供されません。

リアルタイム仮想マシーンの構成

リアルタイム用VMを構成するには次の手順を実行して下さい：

- *rt*構成属性を有効化
- *rt_memory*属性を有効化 (**auto**を推奨)
- *numatune*属性の有効化を検討 (**auto**を推奨)
- 以下で説明するように*cpuset*属性を構成
- メモリ・バルーニングを無効化することを検討：次の案内を参照して下さい

リアルタイムVMに関する*cpuset*属性を構成するには、ホスト・システムのCPUトポロジーを多少理解していることが求められます。ホスト・システムのCPUトポロジーの表示を見るには**cpu-topology**コマンドを使用して下さい：

```
$ cpu-topology
```

詳細およびオプションについては**cpu-topology --help**を実行して下さい。

cpu-topologyはCPUソケット、NUMAノード、CPUコア、論理CPUのレイアウトを表示します。出力例は次のようになります：

```
NUMA Node 0:
     Core 0:   [Socket 0]
         CPU 0
     Core 1:   [Socket 0]
         CPU 1
     Core 2:   [Socket 0]
         CPU 2
     Core 3:   [Socket 0]
         CPU 3
```

システムがIntelハイパースレッドなどのスレッド化CPUアーキテクチャを搭載している場合、出力は次のようになります：

```
NUMA Node 0:
     Core 0:           [Socket 0]
         CPU 0
         CPU 4
     Core 1:           [Socket 0]
         CPU 1
         CPU 5
     Core 2:           [Socket 0]
         CPU 2
         CPU 6
     Core 3:           [Socket 0]
         CPU 3
         CPU 7
```

NUMAシステムが複数のNUMAノードを搭載している場合、次のようになります：

```
NUMA Node 0:
     Core 0:           [Socket 0]
         CPU 0
         CPU 16
     Core 1:           [Socket 0]
         CPU 1
         CPU 17
     Core 2:           [Socket 0]
         CPU 2
         CPU 18
     ...
NUMA Node 1:
     Core 8:           [Socket 1]
         CPU 8
         CPU 24
     Core 9:           [Socket 1]
         CPU 9
         CPU 25
     Core 10:          [Socket 1]
         CPU 10
         CPU 26
     ...
```

最適な性能のためにリアルタイムVMを構成する場合は次のルールを遵守する必要があります。いずれかのルールに違反した場合はKVM-RTツールは適切なエラーまたは警告を表示します。エラーは継続するために是正する必要がありますが、警告は構成が最適ではない可能性があることのヒントとなります。

- リアルタイムVMの*cpuset*は、他のどのVMの*cpuset*と重複することも出来ません。
- リアルタイムVMの*cpuset*は、*nr_vcpus*属性で構成されたCPUの数に対して供給不足となってはなりません。
- リアルタイムVMの*cpuset*が複数のNUMAノードに広がる場合、慎重な考慮が必要となります。

- 他のどのVMの*cpuset*がリアルタイムVMとNUMAノードを共有する場合、慎重な考慮が必要となります。
- リアルタイムVMに対して*numatune*が有効ではない場合、または*numatune*ノード・セットが*cpuset*で使用されるNUMAノードの中に含まれていない場合、慎重な考慮が必要となります。
- 他のどのVMの*numatune*ノード・セットがリアルタイムVMの*cpuset*で使用されるNUMAノードと重複している場合、慎重な考慮が必要となります。
- 全てのリアルタイムVMの*cpuset*はホストCPU全てを消費してはなりません。これは一部のCPUはKVM-RTのホストOS用に利用可能である必要があるためです。

次の推奨事項を忠実に守ることはリアルタイムVM構成の簡略化に役立ちます：

- 常時、最小で*nr_vcpus* +1のホストCPUを*cpuset*に構成して下さい。
- 他のいずれのVMと対象のVMの*cpuset*が競合する、または対象のVMで使用するNUMAノード内の他のCPUを使用するような構成をしないで下さい。
- *cpuset*が複数のNUMAノードに広がらないようにして下さい。
- *numatune*は**auto**に設定して下さい。
- 他のいずれのVMの*numatune*が対象のVMで使用するNUMAノードを含むような構成にしないで下さい。
- 全てのVMで構成されるリアルタイム・ポリシーを表示するには**kvmrt-show-config**コマンドを使用して下さい。
- 現在実行中の全てのVMスレッドのCPUバイアス状況を表示するには**kvmrt-stat -t**コマンドを使用して下さい。

VirtIOはホストが仮想マシーンからメモリを取り戻すことが可能なメモリ・バルーニングを提供します。これは仮想マシーンのリアルタイム性能に影響を及ぼすのでこれを無効化することを推奨します。KVM-RTは**libvirt**からこれを無効にしますが、各リアルタイムVMで次の手順を行う必要があります。各々のリアルタイムVMで次が必要です：

1. 次のようにブラックリスト構成ファイルに追加して下さい：

   ```
   echo “blacklist virtio_balloon” >> \
       /etc/modprobe/blacklist.conf
   ```

2. 新しいinitramfsイメージ(initrdとも呼ばれる)を再生成して下さい：

   CentOSシステム：
   ```
   run dracut --regenerate-all
   ```

   Ubuntuシステム：
   ```
   update-initramfs -u -k all
   ```

4
KVM-RTツール

KVM-RTツールは自己文書化されています。詳細についてはコマンドの**--help**オプションを使用して下さい。以下は機能で整理された各ツールの簡単な説明です。

KVM-RTシステム・コマンド

cpu-topology:

現在のシステムのCPUトポロジーをCPUソケット、NUMAノード、CPUコア単位で表示します。

irq-affinity:

現在のシステムにおけるIRQのCPUアフィニティを表示します。いくつかのオプションが検索を絞り込むのに利用可能です。例えば**-c**オプションを加えることで、特定のCPUまたはCPUセットに検索を制限することが可能です。

task-affinity:

現在のシステムで実行中のタスクのCPUアフィニティを表示します。いくつかのオプションが検索を絞り込むのに利用可能です。例えば**-c**オプションを加えることで、特定のCPUまたはCPUセットに検索を制限することが可能です。

KVM-RT開始コマンド

kvmrt-validate-host:

現在のシステム構成がKVM-RTホストとして有効であることを検証します。そうではない場合に行われる変更の提案を提供します。

kvmrt-import:

KVM-RT構成ファイルに**libvirt**仮想マシーンをインポートします。デフォルトで現在のシステム上の全てのlibvirt VMがインポートされますが、代わりに個々のVMを指定することも可能です。いずれのVMが既にKVM-RT構成ファイルに記載されているとスキップされます。

KVM-RT構成コマンド

`kvmrt-edit-config:`

ユーザーがKVM-RT構成ファイルの編集、検証、同期するのを許可します。デフォルト以外の構成ファイルを指定することが可能な**`-f`**オプションを含むいくつかのオプションが利用可能です。

`kvmrt-show-config:`

KVM-RT構成にある仮想マシーンの構成を表示します。デフォルト以外の構成ファイルを指定することが可能な**`-f`**オプションを含むいくつかのオプションが利用可能です。

`kvmrt-sync-config:`

`libvirt`のVM構成の**`XML`**ファイルとKVM-RTファイルを同期します。デフォルトでKVM-RT構成ファイル内の全てのVMが同期されますが、代わりに個々のVMを指定することも可能です。

`kvmrt-validate-config:`

KVM-RT構成ファイルを検証します。**`-f`**オプションはデフォルト以外の構成ファイルを指定するのに利用可能です。

KVM-RT起動/シャットダウン・コマンド

`kvmrt-boot:`

構成を検証した後、KVM-RT構成内の仮想マシーンを起動します。デフォルトで「autostart」構成パラメータが有効である構成内の全てのVMが起動されますが、代わりに個々のVMを指定することも可能です。

`kvmrt-shutdown:`

仮想マシーンをシャットダウンしそれらのVMで使用されていたいずれのリアルタイム・ポリシーも削除します。デフォルトで構成内の全てのVMがシャットダウンされますが、代わりに個々のVMを指定することも可能です。

`kvmrt-stat:`

KVM-RT構成内の仮想マシーンの状態を表示します。デフォルトで全てのVMが表示されますが、代わりに個々のVMを指定することも可能です。

5
時刻の同期

chronyはNTPの万能な時刻同期実装です。これは幅広い条件でも正常に機能するように設計されており、仮想マシーンで実行することが可能です。chronyシステムを構成及び開始する方法について具体的な手順がここに含まれています。詳細については**chronyd(1)**, **chrony.conf**およびオンライン・ドキュメントを参照して下さい。

NOTE

chronydはRedHawkリリース8.0以降でのみサポートされます。以前のリリースについては、ローカル、リモートまたは公開された時刻サーバーに同期された**chrony/ntp**を使用します。

複雑なアプリケーションは、2つ以上のVMまたはホストとの間で同期される時刻に依存する可能性があります。また、これはリアルタイムVMの性能の問題を解析、またはシステムの問題をデバッグするためにRedHawkのトレース機能を使用する場合、仮想ゲストの時刻はホストと同期することが必須となります。

chrony実行の手順

仮想ゲストの時刻クロックを同期するための様々なテクニックがありますが、**ptp_kvm**モジュールを介してchronyと同期するkvm_clockを推奨します。

ptp_kvmを使用するために**chronyd**を構成する過程は、ベース・ディストリビューションにより若干異なります。

ベース・ディストリビューションとしてUbuntuを使用している場合、次の設定を使用して下さい：

```
service=chrony
conf=/etc/chrony/chrony.conf
drift=/var/lib/chrony/chrony.drift
```

ベース・ディストリビューションとしてCentOS互換を使用している場合、次の設定を使用して下さい：

```
service=chronyd
conf=/etc/chrony.conf
drift=/var/lib/chrony/drift
```

次の手順は仮想ゲストでchronyを構成するのに役に立つはずです。適切な上記のディストリビューションの設定を以下の変数設定に置換して下さい。

1. インストールがまだの場合、chronyをインストールして下さい。

   ```
   dnf install chrony
   ```

2. chronyを停止し、無効にして下さい。

```
systemctl stop $service
systemctl disable $service
```

3. 起動時にptp_kvmモジュールをロードして下さい。

```
echo ptp_kvm > /etc/modules-load.d/ptp_kvm.conf
```

4. 適切なchrony構成ファイルを編集し、「refclock」「server」「pool」「peer」を言及するいずれの行もコメント・アウト(先頭に#記号を置きます)して下さい。

```
grep 'refclock|server|pool|peer' $conf && vi $conf
```

5. 「refclock」を構成して下さい。

```
echo "refclock PHC /dev/ptp0 poll 3 dpoll -2 \
     offset 0" >> $conf
```

NOTE

手順6はCentOS互換ディストリビューションのみに適用されます。Ubuntuディストリビューションを使用している場合には手順7にスキップして下さい。

6. /etc/sysconfig/networkファイル内のPEERNTPがある全ての行をコメントアウト(先頭に#を置く)し、PEERNTP=noを付け足して下さい。

```
grep PEERNTP /etc/sysconfig/network && \
     vi /etc/sysconfig/network
echo "PEERNTP=no" >> /etc/sysconfig/network
```

7. 適切な$driftファイルを削除して下さい。

```
rm –f $drift
```

8. 適切なchronydサービスを有効化しますが、開始しないで下さい。

```
systemctl enable $service
```

9. 新しい構成でクリーン・スタートするため再起動して下さい。

```
reboot
```

6
解析およびデバッグ

本章では、仮想化された環境の性能の問題を解析またはシステムの問題をデバッグするために使用可能なシステム・ツールを取り上げます。

新しいマルチ・マージ・トレース機能はRedHawkオペレーティング・システムの最新リリースに含まれています。これはタイムスタンプで整理された1つのビューに複数のシステムのトレース・ダンプを統合することを可能にします。この新しい機能は、リアルタイム・アプリケーションの性能に影響を及ぼす可能性のあるVM-ホスト間の相互作用を頻繁に引き起こす仮想化環境のデバッグでは不可欠です。

マルチ・マージ・トレース機能を活用するには、トレースする全てのゲストVMは時刻クロック(TOD: Time Of Day)を使って同期する必要があります。トレースするために各ゲストVM上でchronyを開始するには5-1ページの「chrony実行の手順」項を参照して下さい。

NOTE

タイムスタンプ・カウンター(TSC)は同期させることが出来ないため、複数のシステムのトレースを行う場合はTODタイムスタンプ型のみを使用する必要があります。トレース・ツールでTODタイムスタンプ・クロック・オプションを必ず選択して下さい。

本章では次の情報を提示します：

- RedHawkでサポートされるKVMトレース・イベント。
- **`xtrace`**と総称するRedHawkトレース・ツールの簡単な説明。これらのツールは簡素なコマンド・ライン・インターフェースを使用します。xtraceおよび新しいマルチ・マージ機能を使ったホストと1つのゲストVMをトレースする実例を含みます。
- KVM-RTゲスト・サービスという名前の新しいサービス。KVM-RTゲスト・サービスは、ゲストのユーザー空間アプリケーションにホスト・ハイパーバイザーにより公開された機能をアクセスする機会を与える新しいアプリケーション・プログラマー・インターフェース群です。

NightTraceはConcurrent Real-Timeが提供するオプション製品です。NightTraceはNightStarファミリーの一部で対話式デバッグや性能解析ツール、トレース・データ収集デーモン、データ値の記録やユーザーまたはカーネルから採取したデータの解析をユーザー・アプリケーションで可能にする2つのアプリケーション・プログラミング・インターフェース(API)で構成されます。

KVM-RTでNightTraceを使用する方法の情報については、NightTrace User's Guideの「Kernel Tracing with KVM-RT」項を参照して下さい。

KVMトレース・イベント

以下はRedHawkオペレーティング・システムでサポートされるKVMのトレース可能なイベントです。

`KVM_ENTER_VM_PID`

これはホスト・カーネルからゲストVMに実行/制御が転送される毎に引き起こされる一般的な包括的イベントです。ホストからゲストへの移行直前にホスト・システム上のKVMモジュールにより生成されます。

`KVM_EXIT_VM_PID`

これはゲストVMからホスト・カーネルに実行/制御が転送される毎に引き起こされる一般的な包括的イベントです。ゲストからホストへの移行直後にホスト・システム上のKVMモジュールにより生成されます。

`KVM_GUEST_HC_START`

本イベントはホストへのハイパーコールを行う直前にゲストVMにより記録されます。

`KVM_GUEST_HC_END`

本イベントは制御がハイパーコールから戻った直後にゲストVMにより記録されます。

`KVM_HOST_HC_ENTER`

本イベントは実行が一般的なハイパーコール・ハンドラーに達する直前にホスト・システムにより記録されます。

`KVM_HOST_HC_EXIT`

本イベントは実行が一般的なハイパーコール・ハンドラーを終了した直後にホスト・システムにより記録されます。

xtraceを使ったカーネル・トレース

`xtrace`はダンプのトレースおよび解析で使用されるコマンド・ライン・インターフェースです。

`xtrace`はRedHawkオペレーティング・システムの**`ccur-xtrace`**パッケージに付属しており、**`xtrace`**-<機能>という名前のいくつかのツールを含んでいます。本パッケージで提供される全てのコマンドとライブラリを参照するには、RedHawkシステムで次を実行して下さい：

```
rpm –ql ccur-xtrace
```

以下は後述する実例で直接呼ばれるツールです。簡単な説明といくつかのオプションのみを以下言及します。詳細および他のオプションを参照するには**`--help`**オプションを使用して下さい：

`xtrace-run:`

シェル・コマンドの実行中にxtraceデータをキャプチャします。本コマンドはコマンド・ラインで指定する必要があります。コマンド終了時に**`xtrace-run`**は停止します。**`-o`**オプションはxtraceデータが保存される出力ディレクトリの名称を指定します。**`-m`**上書きオプションはトレースが長時間行われxtraceデータが巨大になる場合に使用することが可能です。

`xtrace-multi-merge:`

コマンド・ラインで指定されたxtraceデータ・ディレクトリを1つのマルチ・マージ・ディレクトリに統合します。これらは**`xtrace-run`**が起動された時に生成されたディレクトリです。コマンド・ラインではホスト用に1つ、トレースするゲストVMごとに1つのディレクトリを指定します。**`-o`**オプションは生成するマルチ・マージ・ディレクトリのディレクトリ名称を指定します。時刻クロック(TOD)だけが同期可能であることに注意して下さい。

`xtrace-view:`

ユーザーが理解可能な書式でxtraceデータを総合し表示します。xtraceデータ・ディレクトリを指定する必要があります。

`xtrace-ctl:`

1つまたは複数のCPU上のカーネルxtraceモジュールの制御を提供します。非対話型モードでは、FLUSH, PAUSE, RESUMEのようなコマンドをコマンド・ラインで指定します。

実例：xtraceを使ったマルチ・マージ・トレース

本例ではホスト・システムとゲストVMでトレース・ダンプを同時にキャプチャし、その後2つのトレース・ダンプを1つに統合します。この例はユーザー・アプリケーションが最初の5分以内に失敗することが分かっていることを前提とします。

NOTE

ゲストVMをトレースする前に時刻同期を構成し、実行している必要があります。トレースする各々のVMでchronyを開始するには、5-1ページの「chrony実行の手順」項を参照して下さい。

次の手順1では、ホスト・システムはバックグラウンドでトレースされ、ユーザー・アプリケーションが失敗するのにかかる時間よりも長い時間スリープします。

手順２ではゲストVMのトレースがホストからリモートで開始されます。ユーザー・アプリケーションがゲストVMで失敗した時、トレース・バッファがフラッシュされます。

手順3ではトレース・バッファがフラッシュされ、ホストでトレースが停止されます。

手順4ではゲストVM上のトレース・データ・ディレクトリをホスト・システムにコピーします。手順5では2つのトレース・ディレクトリを1つに統合し、手順6では統合されたトレースをタイムスタンプに従って並べ替え表示します。

1. `rm –rf` *xtrace-host*
   `xtrace-run –m overwrite –t tod –o` *xtrace-host* `sleep 600 &`

2. `ssh` *guest_vm* `"rm -rf` *xtrace-vm*`; \`
   `xtrace-run -m overwrite -t tod -o` *xtrace-vm* `\`
   `bash -c '(` *userapp* `|| xtrace-ctl flush)' "`

3. `xtrace-ctl flush stop`

4. `scp -r` *guest-vm*`:`*xtrace-vm* `.`

5. `xtrace-multi-merge -o` *xtrace-merged* *xtrace-host* *xtrace-vm*

6. `xtrace-view` *xtrace-merged*

表示されるフィールドは**`xtrace-view`**へのオプションで制御されます。次の出力例のフィールドは、タイムスタンプ(TOD)、ホスト名、CPU、イベントです。

CPUは各ホストに対するローカルなので以下の引用では、「vm1 0」はゲストVMのホスト名が「vm1」の仮想CPU 0を意味します。

```
23.404455270 host 3 INTERRUPT_ENTER [apic_timer]
23.404455720 host 3 HRTIMER_CANCEL [0xffffffff8e8f84e0]
23.404455898 host 3 HRTIMER_EXPIRE [0xffffffff8e8f84e0]
23.404456627 host 3 SCHED_WAKEUP [740216]
23.404456854 host 3 HRTIMER_EXPIRE_DONE[0xffffffff8e8f84e0]
23.404456971 host 3 HRTIMER_START [0xffffffff8e8f84e0]
23.407646071 vm1 0 SYSCALL_EXIT [openat]
23.407646321 vm1 0 SYSCALL_ENTER [read]
23.407646512 vm1 0 FILE_READ [3]
23.407647171 vm1 0 SYSCALL_EXIT [read]
```

KVM-RTゲスト・サービス

仮想化環境は、VMで実行しているハード・リアルタイム・アプリケーションの性能に有害な影響を及ぼしかねない複雑なVM-ホスト間の相互作用を引き起こす可能性があります。これらの相互作用の一部は不定期かつ/または再現しにくい可能性があります。これらのケースでは標準的なトレースのアプローチは十分ではありません。

KVM-RTゲスト・サービスは、ゲストのユーザー空間アプリケーションにホスト・ハイパーバイザーにより公開された機能をアクセスする機会を与える新しいアプリケーション・プログラマー・インターフェース群です。

複雑なものを再現する1つの方法は、各アプリケーションの中に含まれている暗示する専門知識を活用することです。アプリケーションが特定の時間であるべき状態やタイミングまたは状態の違反が発生した時の状態をアプリケーションは知っています。そのような背景において、KVM-RTゲスト・サービスはアプリケーション開発者に次のような能力を提供します：

1. ゲストVMで実行中のアプリケーションから直接ホスト上の主要なロギング/トレース機能(例えば、syslog, NightTrace, xtrace)に関連のあるイベント/データを記録します。

2. ホスト上のxtraceバッファをフラッシュします。これはゲストとホストの両方のバッファをほぼ同時にフラッシュするためにゲスト上のxtraceバッファのローカル・フラッシュを組み合わせることが可能です。

3. イベントの順番を成立させるため、ホストのクロックのコンテキストで明示的に事前定義された一連のイベントを記録します。例えば、以下はホスト上の2つの異なるゲストで記録されました。

   VM1上：

   ```
   host: "VM1 is about to start A"
   ...
   host: "VM1 just finished A"
   ...
   ```

   VM2上：

   ```
   host: "VM2 is about to start B"
   ...
   host: "VM2 just finished B"
   ```

   ホストでは、ホストのクロックのコンテキストで一連のイベントを見ることが可能です：

   ```
   host: "VM1 is about to start A"
   ...
   host: "VM2 is about to start B"
   ...
   host: "VM2 just finished B"
   ...
   host: "VM1 just finished A"
   ```

コマンド・ライン・インターフェース**`kvmrt-gs`**およびライブラリ**`libccur_kvmrt_gs`**で提供されるKVM-RTゲスト・サービスの機能は、次項で簡単に説明します。

また、トレース可能なKVM-RTゲスト・サービスのイベントおよびホストとゲストVMで有効化すべきカーネル起動パラメータが後述されています。

KVM-RTゲスト・サービス・ライブラリ・インターフェース

次の機能がライブラリ**`libccur_kvmrt_gs`**を介して提供されます。オプションや利用法に関する詳細については**`libccur_kvmrt_gs(3)`**のmanページを参照して下さい。

manページは以下に記載されたいずれかの機能の名称を使って呼び出すことが可能です。例：
`man kvmrt_gs_available`

```
bool kvmrt_gs_available(void);
bool kvmrt_gs_ping_available(void);
bool kvmrt_gs_log_msg_available(void);
```

```
bool kvmrt_gs_xtrace_flush_available(void);
bool kvmrt_gs_xtrace_log_data_available(void);

long kvmrt_gs_ping(unsigned long cookie);
long kvmrt_gs_log_msg(char * msg);
long kvmrt_gs_xtrace_flush(unsigned long scope);
long kvmrt_gs_xtrace_log_data(void * data, long size);
```

`kvmrt_gs_available`

KVMRT_GSインターフェースが存在し、有効化され、許可されていればtrueを返します。同様にkvmrt_gs_<function>_availableは、個々のKVMRT_GSの関数が存在し、有効化され、許可されていればtrueを返します。

インターフェースの可用性はいずれの関数の可用性を意味することではないことに注意して下さい。更に、利用可能な関数の呼び出しは様々な理由により常に失敗する可能性があります。

`kvmrt_gs_ping`

*`cookie`*を使ってハイパーバイザーにPingします。本関数の目的は、ゲスト側とホスト側から簡単にトレースし組み合わせることが可能な方法でハイパーバイザー上でVMEXITイベントを明示的に発生させるためにコピーまたは割り当てのない簡単で軽量なメカニズムをゲストに提供することです。本インターフェースはxtraceが利用可能な場合に対応するxtraceイベントを生成します。

`kvmrt_gs_log_msg`

ハイパーバイザー側の標準カーネル・ロギング・メカニズムを介して短いASCIIテキスト・メッセージを記録します。*`msg`*は標準的なNULLで終了するC言語文字列へのポインターです。ハイパーバイザーといずれの中間層も文字列の最大長を制限しますので、さもなければメッセージが切り詰められる可能性があります。以下のkvmrt_gs_xtrace_log_dataも参照して下さい。

`kvmrt_gs_xtrace_flush`

ホストOSのFLUSH xtraceイベントを発生させます。

*`scope`*はFLUSHに影響を受けるCPUを制御します：

```
KVMRT_GS_XTRACE_CPU_CURRENT
KVMRT_GS_XTRACE_CPU_VM
KVMRT_GS_XTRACE_CPU_ALL
```

現在のCPU、現在のVMを処理している全てのCPU、ホスト・システムでアクティブな全てのCPUにそれぞれFLUSHを発行します。

`kvmrt_gs_xtrace_log_data`

ゲスト側とホスト側の2つが一致するxtraceイベントとして、*`size`*バイトを含む任意のバイナリの*`data`*バッファに記録します。ハイパーバイザーといずれの中間層も最大サイズを制限しますので、さもなければ記録されたデータが切り詰められる可能性があります。上述のkvmrt_gs_log_msgも参照して下さい。

KVM-RTゲスト・サービス・コマンド・ライン・インターフェース

次のコマンドが**kvmrt-gs**コマンド・ライン・インターフェースを介して提供されます。オプションと使用方法の詳細については**kvmrt-gs(1)**のmanページを参照して下さい。

```
kvmrt-gs [OPTIONS] [COMMAND [ARGUMENTS] ...] ...
```

`available`

KVM-RTゲスト・サービスが利用可能な場合はSUCCESSを返します。

`ping_available`

「ping」コマンドが利用可能な場合はSUCCESSを返します。

`ping` ***`COOKIE`***

`COOKIE` (任意のユーザが選定した整数(unsigned long int))を使ってハイパーバイザーにpingを実行します。

`log_msg_available`

「log_msg」コマンドが利用可能な場合はSUCCESSを返します。

`log_msg` ***`MESSAGE`***

ハイパーバイザー上のメッセージを記録します。***`MESSAGE`***は通常の引用符付きASCII文字列または16進数でエンコードされたバイト列のどちらかが可能です。

`xtrace_flush_available`

「xtrace_flush」コマンドが利用可能な場合はSUCCESSを返します。

`xtrace_flush` ***`SCOPE`***

ホストOS上のxtraceバッファをフラッシュします。***`SCOPE`***は次のいずれかが可能です：{0: 現在のCPU；1: 全てのVM CPU；2: 全てのホストCPU}

`xtrace_log_data_available`

「xtrace_log_data」コマンドが利用可能な場合はSUCCESSを返します。

`xtrace_log_data` ***`DATA`***

バイナリ・データの状態でxtraceイベントを記録します。***`DATA`***は通常の引用符付きASCII文字列または16進数でエンコードされたバイト列のどちらかが可能です。

KVM-RTゲスト・サービス・トレース・イベント

KVM-RTゲスト・サービスは様々なトレース・イベントを記録します。全てのイベント・タイプがペアとして生じ、ゲスト側では*_GUESTが記録され、ホスト側では*_HOSTが記録されます。

ダブル・ロギングのような目的の背景は、ホストとゲストVMのクロックが同期していない可能性がある、または相互関係が変動した場合にトレース・ログ内で予測可能な基準点を提供することです。

KVMRT_GS_PING_GUEST
KVMRT_GS_PING_HOST

これらはKVM-RTゲスト・サービスの「ping」機能により生成されます。詳細については**kvmrt_gs_ping(3)**を参照して下さい。

KVMRT_GS_FLUSH_GUEST
KVMRT_GS_FLUSH_HOST

これらはKVM-RTゲスト・サービスの「xtrace_flush」機能により生成されます。詳細については**kvmrt_gs_xtrace_flush(3)**を参照して下さい。

KVMRT_GS_LOG_DATA_GUEST
KVMRT_GS_LOG_DATA_HOST

これらはKVM-RTゲスト・サービスの「xtrace_log_data」機能により生成され、XTRACE_EV_CUSTOMに類似しています。詳細については**kvmrt_gs_xtrace_log_data(3)**を参照して下さい。

KVM-RTゲスト・サービス・カーネル起動パラメータ

KVM-RTゲスト・サービスは、起動時に次のカーネル・パラメータが有効化されている必要があることを要求します。1つはホスト・システムに、その他はゲストVMに特化している事に注意して下さい。

kvm.kvmrt_gs_hc_host_enabled=

[KVM, x86] KVMホストでKVM-RTゲスト・サービスのハイパーコールを有効にします。これを1(有効)に設定するとKVMRT_GSハイパーコールおよび関連するGS機能をゲストに提供することをホストに許可します。これはKVMモジュール用のホスト側パラメータです。デフォルトは0(無効)となります。

kvmrt_gs_hc_guest_enabled=

[KVM_GUEST, x86] KVMゲストでKVM-RTゲスト・サービスのハイパーコールを有効にします。これを1(有効)に設定するとKVMRT_GSハイパーコールおよびその機能がホストより提供された場合、検出し使用することをゲスト・カーネルに許可します。これはゲスト側のカーネル・パラメータです。デフォルトは0(無効)となります。

kvmrt_gs_syscall_enabled=

[KVM_GUEST, x86] KVMゲストでKVM-RTゲスト・サービスの間接システムコール(syscall)を有効にします。これを1(有効)に設定するとKVMRT_GS間接システムコールおよびその機能をゲストで実行中のユーザー空間アプリケーションに提供することをゲスト・カーネルに許可します。これはゲスト側のカーネル・パラメータです。デフォルトは0(無効)となります。