



Red Hat Enterprise Linux 8

カーネルの管理、監視、および更新

Red Hat Enterprise Linux 8 上で Linux カーネルを管理するためのガイド

Red Hat Enterprise Linux 8 カーネルの管理、監視、および更新

Red Hat Enterprise Linux 8 上で Linux カーネルを管理するためのガイド

法律上の通知

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

システム管理者は、オペレーティングシステムを最適化するように Linux カーネルを設定できます。Linux カーネルを変更すると、システムのパフォーマンス、セキュリティー、および安定性が向上するだけでなく、システムを監査して問題をトラブルシューティングする能力も向上します。

目次

RED HAT ドキュメントへのフィードバック (英語のみ)	7
第1章 LINUX カーネル	8
1.1. カーネルとは	8
1.2. RPM パッケージ	8
1.3. LINUX カーネル RPM パッケージの概要	9
1.4. カーネルパッケージの内容の表示	9
1.5. 特定のカーネルバージョンのインストール	10
1.6. カーネルの更新	11
1.7. カーネルのデフォルトとしての設定	11
第2章 カーネルモジュールの管理	13
2.1. モジュールの紹介	13
2.2. カーネルモジュールの依存関係	13
2.3. インストール済みカーネルモジュールのリスト表示	14
2.4. 現在読み込み済みカーネルモジュールのリスト表示	14
2.5. インストール済みカーネルのリスト表示	15
2.6. モジュール情報の表示	16
2.7. システムランタイム時のカーネルモジュールの読み込み	16
2.8. システムランタイム時のカーネルモジュールのアンロード	17
2.9. 起動プロセスの初期段階でのカーネルモジュールのアンロード	19
2.10. システムの起動時に自動的にカーネルモジュールを読み込む	20
2.11. システムの起動時にカーネルモジュールが自動的にロードされないようにする	21
2.12. カスタムカーネルモジュールのコンパイル	23
第3章 セキュアブート用のカーネルとモジュールの署名	26
3.1. 前提条件	26
3.2. UEFI セキュアブートとは	27
3.3. UEFI セキュアブートのサポート	28
3.4. X.509 鍵でカーネルモジュールを認証するための要件	28
3.5. 公開鍵のソース	29
3.6. 公開鍵と秘密鍵の生成	31
3.7. システムキーリングの出力例	32
3.8. 公開鍵を MOK リストに追加することでターゲットシステムで公開鍵を登録する手順	33
3.9. 秘密鍵でカーネルに署名する	34
3.10. 秘密鍵で GRUB ビルドに署名する	35
3.11. 秘密鍵を使用したカーネルモジュールの署名	36
3.12. 署名済みカーネルモジュールの読み込み	38
第4章 カーネルコマンドラインパラメーターの設定	40
4.1. カーネルコマンドラインパラメーターの設定	40
4.2. ブートエントリーについて	40
4.3. すべてのブートエントリーでカーネルコマンドラインパラメーターの変更	41
4.4. 1つのブートエントリーでカーネルコマンドラインパラメーターの変更	42
4.5. 起動時の一時的なカーネルコマンドラインパラメーターの変更	43
4.6. シリアルコンソール接続を有効にする GRUB 設定	44
第5章 ランタイム時のカーネルパラメーターの設定	46
5.1. カーネルパラメーターとは	46
5.2. SYSCTL でカーネルパラメーターの一時的な設定	47
5.3. SYSCTL を使用したカーネルパラメーターの永続的な設定	47
5.4. /ETC/SYSCTL.D/ の設定ファイルでカーネルパラメーターの調整	48

5.5. /PROC/SYS/ でカーネルパラメーターの一時的な設定	49
5.6. 関連情報	49
第6章 GRUB メニューを一時的に変更する	50
6.1. GRUB について	50
6.2. ブートローダー仕様の概要	51
6.3. レスキューモードでの起動	52
6.4. 緊急モードでの起動	52
6.5. デバッグシェルのブート	53
6.6. デバッグシェルへの接続	54
6.7. インストールディスクを使用した ROOT パスワードのリセット	54
6.8. RD.BREAK を使用した ROOT パスワードのリセット	57
6.9. 関連情報	58
第7章 GRUB ブートローダーに永続的な変更を加える	59
7.1. 前提条件	59
7.2. デフォルトのカーネルの一覧表示	59
7.3. カーネルの GRUB メニューエントリーの表示	59
7.4. カーネル引数の編集	60
7.5. GRUB メニューエントリーに対する引数の追加および削除	60
7.6. 新しいブートエントリーの追加	61
7.7. GRUBBY でデフォルトのブートエントリーを変更する	62
7.8. 同じ引数を使用したすべてのカーネルメニューの更新	62
7.9. 現在および将来のカーネルのデフォルトカーネルオプションの変更	62
7.10. 関連情報	63
第8章 カスタマイズされたブートメニューの構築	64
8.1. GRUB 設定ファイル	64
8.2. 起動可能なカーネルのリストを非表示にする	64
8.3. GRUB 設定ファイルを使用してデフォルトのブートエントリーを変更する	65
第9章 GRUB の再インストール	67
9.1. BIOS ベースマシンへの GRUB の再インストール	67
9.2. UEFI ベースマシンへの GRUB の再インストール	67
9.3. IBM POWER マシンへの GRUB の再インストール	68
9.4. GRUB のリセット	68
第10章 GRUB をパスワードで保護する	70
10.1. メニューエントリーを変更するためだけにパスワード保護を設定する	70
10.2. メニューエントリーの変更および起動のためのパスワード保護の設定	70
第11章 仮想化環境でカーネルパニックのパラメーターを無効のままにする	72
11.1. ソフトロックアップとは	72
11.2. カーネルパニックを制御するパラメーター	72
11.3. 仮想化環境で誤ったソフトロックアップ	73
第12章 データベースサーバーのカーネルパラメーターの調整	74
12.1. データベースサーバーの概要	74
12.2. データベースアプリケーションのパフォーマンスに影響するパラメーター	74
第13章 カーネルロギングの使用	77
13.1. カーネルリングバッファとは	77
13.2. ログレベルおよびカーネルロギングにおける PRINTK のロール	77
第14章 KDUMP のインストール	79
14.1. KDUMP とは	79

14.2. ANACONDA を使用した KDUMP のインストール	79
14.3. コマンドラインで KDUMP のインストール	80
第15章 コマンドラインで KDUMP の設定	81
15.1. KDUMP サイズの見積もり	81
15.2. メモリー使用量の設定	81
15.3. KDUMP ターゲットの設定	83
15.4. KDUMP コアコレクターの設定	86
15.5. KDUMP のデフォルト障害応答の設定	87
15.6. KDUMP の設定ファイル	88
15.7. KDUMP 設定のテスト	89
15.8. システムクラッシュ後に KDUMP によって生成されるファイル	90
15.9. KDUMP サービスの有効化および無効化	90
15.10. カーネルドライバーが KDUMP を読み込まないようにする設定	91
15.11. 暗号化されたディスクがあるシステムでの KDUMP の実行	92
第16章 KDUMP の有効化	94
16.1. インストールされているすべてのカーネルでの KDUMP の有効化	94
16.2. 特定のインストール済みカーネルでの KDUMP の有効化	94
16.3. KDUMP サービスの無効化	95
第17章 WEB コンソールで KDUMP の設定	97
17.1. WEB コンソールで KDUMP メモリーの使用量およびターゲットの場所を設定	97
第18章 サポートされている KDUMP 設定とターゲット	100
18.1. KDUMP メモリー要件	100
18.2. メモリー自動予約の最小しきい値	101
18.3. サポートしている KDUMP のダンプ出力先	102
18.4. 対応している KDUMP のフィルターレベル	104
18.5. 対応しているデフォルトの障害応答	104
18.6. FINAL_ACTION パラメーターの使用	105
18.7. FAILURE_ACTION パラメーターの使用	105
第19章 ファームウェア支援ダンプの仕組み	107
19.1. IBM POWERPC ハードウェアにおけるファームウェア支援ダンプ	107
19.2. ファームウェア支援ダンプメカニズムの有効化	107
19.3. IBM Z ハードウェアにおけるファームウェア支援ダンプの仕組み	108
19.4. FUJITSU PRIMEQUEST システムにおける SADUMP の使用	109
第20章 コアダンプの分析	110
20.1. CRASH ユーティリティのインストール	110
20.2. CRASH ユーティリティの実行および終了	110
20.3. CRASH ユーティリティのさまざまなインジケータの表示	112
20.4. KERNEL OOPS ANALYZER の使用	114
20.5. KDUMP HELPER ツール	115
第21章 EARLY KDUMP を使用した起動時間クラッシュの取得	116
21.1. EARLY KDUMP の概要	116
21.2. EARLY KDUMP の有効化	116
第22章 カーネルライブパッチでパッチの適用	118
22.1. KPATCH の制限	118
22.2. サードパーティーのライブパッチサポート	118
22.3. カーネルライブパッチへのアクセス	119
22.4. カーネルライブパッチのコンポーネント	119

22.5. カーネルライブパッチの仕組み	120
22.6. 現在インストールされているカーネルをライブパッチストリームにサブスクライブする手順	120
22.7. ライブパッチストリームに新しいカーネルを自動的にサブスクライブする手順	122
22.8. ライブパッチストリームへの自動サブスクリプションの無効化	124
22.9. カーネルパッチモジュールの更新	125
22.10. ライブパッチパッケージの削除	125
22.11. カーネルパッチモジュールのアンインストール	126
22.12. KPATCH.SERVICE の無効化	128
第23章 コントロールグループを使用したアプリケーションのシステムリソース制限の設定	130
23.1. コントロールグループの概要	130
23.2. カーネルリソースコントローラーの概要	131
23.3. 名前空間の概要	133
23.4. CGROUPS-V1 を使用したアプリケーションへの CPU 制限の設定	133
第24章 CGROUPS-V2 を使用したアプリケーションへの CPU 時間配分の制御	138
24.1. CGROUPS-V2 のマウント	138
24.2. CPU 時間配分のための CGROUP の準備	139
24.3. CPU 帯域幅の調整によるアプリケーションへの CPU 時間配分の制御	143
24.4. CPU の重みの調整によるアプリケーションへの CPU 時間配分の制御	145
第25章 SYSTEMD によるコントロールグループバージョン1の使用	148
25.1. コントロールグループバージョン1における SYSTEMD のロール	148
25.2. 一時的なコントロールグループの作成	149
25.3. 永続的なコントロールグループの作成	150
25.4. コマンドラインでのメモリーリソース制御の設定	151
25.5. ユニットファイルを使用したメモリーリソース制御の設定	151
25.6. 一時的なコントロールグループの削除	153
25.7. 永続的なコントロールグループの削除	153
25.8. SYSTEMD ユニットのリスト表示	154
25.9. SYSTEMD CGROUPS 階層の表示	155
25.10. リソースコントローラーの表示	157
25.11. リソース消費の監視	158
第26章 CGROUPS-V2 と SYSTEMD を使用したリソース管理の設定	159
26.1. 前提条件	159
26.2. リソース配分モデルの概要	159
26.3. SYSTEMD を使用した CPU リソースの割り当て	160
26.4. SYSTEMD の CPU 時間割り当てポリシーオプション	160
26.5. SYSTEMD を使用したメモリーリソースの割り当て	161
26.6. SYSTEMD のメモリー割り当て設定オプション	162
26.7. SYSTEMD を使用した I/O 帯域幅の設定	163
26.8. SYSTEMD の I/O 帯域幅設定オプション	163
26.9. SYSTEMD を使用した CPuset コントローラーの設定	164
第27章 SYSTEMD を使用した CPU のアフィニティーおよび NUMA ポリシーの設定	166
27.1. SYSTEMD を使用した CPU アフィニティーの設定	166
27.2. SYSTEMD を使用した NUMA ポリシーの設定	167
27.3. SYSTEMD の NUMA ポリシー設定オプション	168
第28章 BPF コンパイラコレクションでシステムパフォーマンスの分析	169
28.1. BCC-TOOLS パッケージのインストール	169
28.2. BCC-TOOLS でパフォーマンスの分析	169
第29章 カーネル整合性サブシステムによるセキュリティーの強化	174

29.1. カーネル整合性サブシステム	174
29.2. 信頼できる鍵および暗号化された鍵	175
29.3. 信頼できる鍵での作業	175
29.4. 暗号化鍵での作業	177
29.5. IMA と EVM の有効化	178
29.6. INTEGRITY MEASUREMENT ARCHITECTURE によるファイルのハッシュの収集	182
第30章 RHEL システムロールを使用したカーネルパラメーターの永続的な設定	183
30.1. KERNEL_SETTINGS RHEL システムロールの概要	183
30.2. KERNEL_SETTINGS RHEL システムロールを使用して選択したカーネルパラメーターの適用	184
第31章 高度なエラー報告の使用	186
31.1. AER の概要	186
31.2. AER メッセージの収集および表示	186

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。

Jira からのフィードバック送信 (アカウントが必要)

1. [Jira](#) の Web サイトにログインします。
2. 上部のナビゲーションバーで **Create** をクリックします。
3. **Summary** フィールドにわかりやすいタイトルを入力します。
4. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも追加してください。
5. ダイアログの下部にある **Create** をクリックします。

第1章 LINUX カーネル

Linux カーネルと、Red Hat が提供および管理する Linux カーネル RPM パッケージ (Red Hat カーネル) について学びます。Red Hat カーネルを最新の状態に保ちます。これにより、オペレーティングシステムに最新のバグ修正、パフォーマンス強化、およびパッチがすべて適用され、新しいハードウェアとの互換性が保たれます。

1.1. カーネルとは

カーネルは Linux オペレーティングシステムのコア部分で、システムリソースを管理し、ハードウェアアプリケーションおよびソフトウェアアプリケーション間のインターフェイスを確立します。

Red Hat カーネルは、アップストリームの Linux メインラインカーネルをベースにしたカスタムカーネルです。Red Hat のエンジニアは、安定性と、最新のテクノロジーおよびハードウェアとの互換性に重点を置き、さらなる開発と強化を行っています。

Red Hat が新しいカーネルバージョンをリリースする前に、カーネルは厳格な品質保証テストをクリアしなければなりません。

Red Hat カーネルは RPM 形式でパッケージ化されているため、**yum** パッケージマネージャーにより簡単にアップグレードおよび検証できます。



警告

Red Hat によってコンパイルされていないカーネルは、Red Hat ではサポートされていません。

1.2. RPM パッケージ

RPM パッケージは、ファイルのアーカイブと、これらのファイルのインストールと消去に使用されるメタデータで構成されます。具体的には、RPM パッケージには次の要素が含まれています。

GPG 署名

GPG 署名は、パッケージの整合性を検証するために使用されます。

RPM ヘッダー (パッケージのメタデータ)

RPM パッケージマネージャーは、このメタデータを使用して、パッケージの依存関係、ファイルのインストール先、その他の情報を確認します。

ペイロード

ペイロードは、システムにインストールするファイルを含む **cpio** アーカイブです。

RPM パッケージには 2 つの種類があります。いずれも、同じファイル形式とツールを使用しますが、コンテンツが異なるため、目的が異なります。

- ソース RPM (SRPM)
SRPM には、ソースコードと、ソースコードをバイナリー RPM にビルドする方法を記述した **spec** ファイルが含まれています。必要に応じて、SRPM にはソースコードへのパッチを含めることができます。

バイナリー RPM

バイナリー RPM には、ソースおよびパッチから構築されたバイナリーが含まれます。

1.3. LINUX カーネル RPM パッケージの概要

カーネル RPM は、ファイルを含まないメタパッケージで、以下の必須サブパッケージが正しくインストールされるようにします。

kernel-core

kernel-core - コア機能を確認するために、カーネルのバイナリーイメージ、システムを起動するためのすべての `initramfs` 関連オブジェクト、および最小限のカーネルモジュールが含まれます。このサブパッケージ単体は、仮想環境およびクラウド環境で使用して、Red Hat Enterprise Linux 8 カーネルのブート時間を短縮し、ディスクサイズを抑えます。

kernel-modules

kernel-core に存在しない残りのカーネルモジュールが含まれます。

上記の **kernel** サブパッケージをいくつか用意することで、特に仮想環境やクラウド環境でのシステム管理者へのメンテナンス面を低減させることを目指します。

任意のカーネルパッケージは、以下の例のようになります。

kernel-modules-extra

まれなハードウェア用のカーネルモジュールと、読み込みがデフォルトで無効になっているモジュールが含まれます。

kernel-debug

カーネル診断ができるように複数のデバッグオプションが有効になっているカーネルが含まれます。デバッグオプションが有効になっているとパフォーマンスが低下します。

kernel-tools

Linux カーネル操作のツールとサポートドキュメントが含まれています。

kernel-devel

kernel パッケージに対して、モジュールをビルドするのに十分なカーネルヘッダーと `makefiles` を含んでいます。

kernel-abi-stablelists

RHEL カーネル ABI に関連する情報が含まれています。これには、強化を支援するための外部 Linux カーネルモジュールおよび `dnf` プラグインに必要なカーネルシンボルのリストが含まれます。

kernel-headers

Linux カーネルと、ユーザー空間ライブラリーおよびプログラムとの間のインターフェイスを指定する C ヘッダーファイルが含まれます。ヘッダーファイルは、ほとんどの標準プログラムを構築するのに必要な構造と定数を定義します。

関連情報

- [kernel-core パッケージ](#)、[kernel-modules パッケージ](#)、および [kernel-modules-extras パッケージは何ですか？](#)

1.4. カーネルパッケージの内容の表示

カーネルパッケージがモジュールなどの特定のファイルを提供しているかどうかを判断するには、リポジトリにクエリーを実行して、アーキテクチャーのパッケージのファイルリストを表示します。ファイルリストを表示するためにパッケージをダウンロードまたはインストールする必要はありません。

dnf ユーティリティーを使用して、たとえば **kernel-core**、**kernel-modules-core**、または **kernel-modules** パッケージのファイルリストをクエリーします。**kernel** パッケージはファイルを含まないメタパッケージであることに注意してください。

手順

1. パッケージの利用可能なバージョンをリスト表示します。

```
$ yum repoquery <package_name>
```

たとえば、**kernel-core** パッケージの利用可能なバージョンをリスト表示します。

```
$ yum repoquery kernel-core
kernel-core-0:4.18.0-147.0.2.el8_1.x86_64
kernel-core-0:4.18.0-147.0.3.el8_1.x86_64
kernel-core-0:4.18.0-147.3.1.el8_1.x86_64
kernel-core-0:4.18.0-147.5.1.el8_1.x86_64
kernel-core-0:4.18.0-147.8.1.el8_1.x86_64
kernel-core-0:4.18.0-147.el8.x86_64
...
```

2. パッケージ内のファイルをリスト表示します。

```
$ yum repoquery -l <package_name>
```

たとえば、**kernel-core-0:5.14.0-162.23.1.el9_1.x86_64** パッケージ内のファイルをリスト表示します。

```
$ yum repoquery -l kernel-core-0:4.18.0-147.8.1.el8_1.x86_64
/boot/vmlinuz-4.18.0-147.8.1.el8_1.x86_64.hmac
/boot/System.map-4.18.0-147.8.1.el8_1.x86_64
/boot/config-4.18.0-147.8.1.el8_1.x86_64
/boot/initramfs-4.18.0-147.8.1.el8_1.x86_64.img
/boot/symvers-4.18.0-147.8.1.el8_1.x86_64.gz
/boot/vmlinuz-4.18.0-147.8.1.el8_1.x86_64
/etc/ld.so.conf.d/kernel-4.18.0-147.8.1.el8_1.x86_64.conf
/lib/modules
/lib/modules/4.18.0-147.8.1.el8_1.x86_64
...
```

関連情報

- [ソフトウェアのパッケージ化および配布](#)

1.5. 特定のカーネルバージョンのインストール

yum パッケージマネージャーを使用して新しいカーネルをインストールします。

手順

- 特定のカーネルバージョンをインストールするには、次のコマンドを実行します。

```
# yum install kernel-{version}
```

関連情報

- [Red Hat Code Browser](#)
- [Red Hat Enterprise Linux Release Dates](#)

1.6. カーネルの更新

`yum` パッケージマネージャーを使用してカーネルを更新します。

手順

1. カーネルを更新するには、次のコマンドを入力します。

```
# yum update kernel
```

このコマンドは、カーネルと、利用可能な最新バージョンへのすべての依存関係を更新します。

2. システムを再起動して、変更を有効にします。



注記

RHEL 7 から RHEL 8 にアップグレードする場合は、[RHEL 7 から RHEL 8 へのアップグレード](#) ドキュメントの関連のセクションを参照してください。

関連情報

- [ソフトウェアパッケージの管理](#)

1.7. カーネルのデフォルトとしての設定

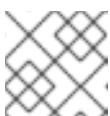
`grubby` コマンドラインツールと GRUB を使用して、特定のカーネルをデフォルトとして設定します。

手順

- `grubby` ツールを使用した、カーネルのデフォルトとしての設定
 - 以下のコマンドを実行し、`grubby` ツールを使用してカーネルをデフォルトとして設定します。

```
# grubby --set-default $kernel_path
```

コマンドは、`.conf` の接尾辞のないマシン ID を引数として使用します。



注記

マシン ID は `/boot/loader/entries/` ディレクトリーにあります。

- `id` 引数を使用したカーネルのデフォルト設定
 - `id` 引数を使用してブートエントリーのリストを表示し、任意のカーネルをデフォルトとして設定します。

-

```
# grubby --info ALL | grep id  
# grubby --set-default /boot/vmlinuz-<version>.<architecture>
```



注記

title 引数を使用してブートエントリーのリストを表示するには、**# grubby --info=ALL | grep title** コマンドを実行します。

- 次回の起動時のみのデフォルトカーネルの設定
 - 次のコマンドを実行し、**grub2-reboot** コマンドを使用して、次回の再起動限定でデフォルトのカーネルを設定します。

```
# grub2-reboot <index|title|id>
```



警告

取り扱いに注意して、次回の起動時限定のデフォルトのカーネルを設定します。新しいカーネル RPM、自己ビルドカーネルをインストールし、エントリーを **/boot/loader/entries/** ディレクトリーに手動で追加すると、インデックス値が変更される可能性があります。

第2章 カーネルモジュールの管理

カーネルモジュール、それらの情報を表示する方法、およびカーネルモジュールを使用して基本的な管理タスクを実行する方法について学びます。

2.1. モジュールの紹介

Red Hat Enterprise Linux カーネルは、システムを再起動しなくても、カーネルモジュールと呼ばれる追加機能で拡張できます。RHEL 8 では、カーネルモジュールは、圧縮された `<KERNEL_MODULE_NAME>.ko.xz` オブジェクトファイルに組み込まれた追加のカーネルコードです。

カーネルモジュールにより有効になっている最も一般的な機能は、以下のとおりです。

- 新しいハードウェアへのサポートを強化するデバイスドライバー
- GFS2 や NFS などのファイルシステムのサポート
- システムコール

最新のシステムでは、必要に応じて自動的にカーネルモジュールが読み込まれます。ただし、場合によっては、モジュールを手動でロードまたはアンロードする必要があります。

モジュールは、カーネル自体と同様に、必要に応じてその動作をカスタマイズするパラメーターを受け取ることができます。

ツールでは、現在実行しているモジュール、カーネルに読み込みできるモジュール、モジュールが受け入れるパラメーターを調べることができます。このツールは、実行中のカーネルに、カーネルモジュールのロードおよびアンロードを行うためのメカニズムを提供します。

2.2. カーネルモジュールの依存関係

特定のカーネルモジュールは、複数の他のカーネルモジュールに依存する場合があります。 `/lib/modules/<KERNEL_VERSION>/modules.dep` ファイルには、各カーネルバージョンに対するカーネルモジュールの依存関係の完全なリストが含まれます。

depmod

依存関係ファイルは、 `kmod` パッケージの一部である `depmod` プログラムにより生成されます。 `kmod` によるユーティリティーの多くは、操作を実行する際にモジュールの依存関係を考慮に入れるため、**手動** で依存関係を追跡する必要はほとんどありません。



警告

カーネルモジュールのコードは、制限のないモードのカーネルスペースで実行されます。そのため、読み込むモジュールに注意してください。

weak-modules

`depmod` に加えて、Red Hat Enterprise Linux は、同じく `kmod` パッケージに同梱されている `weak-`

modules スクリプトを提供します。**weak-modules** は、どのモジュールがインストールされたカーネルと kABI 互換であるかを決定します。モジュールカーネルの互換性をチェックしている間、**weak-modules** はモジュールシンボルの依存関係を、それらがビルドされたカーネルの上位リリースから下位リリースへと処理します。これは、**weak-modules** がビルド対象のカーネルリリースとは無関係に各モジュールを処理することを意味します。

関連情報

- **modules.dep (5)** man ページ
- **depmod (8)** man ページ
- [Red Hat Enterprise Linux に同梱されている weak-modules スクリプトの目的は何ですか?](#)
- [カーネルアプリケーションバイナリーインターフェイス \(kABI\) とは何ですか?](#)

2.3. インストール済みカーネルモジュールのリスト表示

grubby --info=ALL コマンドは、**!BLS** インストールおよび **BLS** インストールにインストールされたカーネルのインデックスリストを表示します。

手順

- 以下のコマンドを使用して、インストールされているカーネルをリスト表示します。

```
# grubby --info=ALL | grep title
```

インストールされているカーネルのリストは、以下のようになります。

```
title=Red Hat Enterprise Linux (4.18.0-20.el8.x86_64) 8.0 (Ootpa)
title=Red Hat Enterprise Linux (4.18.0-19.el8.x86_64) 8.0 (Ootpa)
title=Red Hat Enterprise Linux (4.18.0-12.el8.x86_64) 8.0 (Ootpa)
title=Red Hat Enterprise Linux (4.18.0) 8.0 (Ootpa)
title=Red Hat Enterprise Linux (0-rescue-2fb13ddde2e24fde9e6a246a942caed1) 8.0 (Ootpa)
```

上記の例では、GRUB メニューから **grubby-8.40-17** のインストール済みカーネルの一覧を表示します。

2.4. 現在読み込み済みカーネルモジュールのリスト表示

現在ロードされているカーネルモジュールを表示します。

前提条件

- **kmod** パッケージがインストールされている。

手順

- 現在読み込み済みのカーネルモジュールの一覧を表示するには、以下のコマンドを実行します。

```
$ lsmod
```

```

Module          Size Used by
fuse            126976 3
uinput         20480 1
xt_CHECKSUM    16384 1
ipt_MASQUERADE 16384 1
xt_contrack    16384 1
ipt_REJECT     16384 1
nft_counter    16384 16
nf_nat_tftp    16384 0
nf_contrack_tftp 16384 1 nf_nat_tftp
tun            49152 1
bridge        192512 0
stp           16384 1 bridge
llc           16384 2 bridge,stp
nf_tables_set 32768 5
nft_fib_inet  16384 1
...

```

上記の例では、以下ようになります。

- a. **Module** 列は、現在読み込まれているモジュールの **名前** を示します。
- b. **Size** 列は、モジュールごとの **メモリー** 容量をキロバイト単位で表示します。
- c. **Used by** 列には、特定のモジュールに **依存する** モジュールの数と、オプションで名前が表示されます。

関連情報

- `/usr/share/doc/kmod/README` ファイル
- `lsmod(8)` の man ページ

2.5. インストール済みカーネルのリスト表示

`grubby` ユーティリティを使用して、システムにインストールされているすべてのカーネルを一覧表示します。

前提条件

- root 権限がある。

手順

- インストールされているすべてのカーネルを一覧表示するには、次のコマンドを実行します。

```

# grubby --info=ALL | grep ^kernel

kernel="/boot/vmlinuz-4.18.0-305.10.2.el8_4.x86_64"
kernel="/boot/vmlinuz-4.18.0-240.el8.x86_64"
kernel="/boot/vmlinuz-0-rescue-41eb2e172d7244698abda79a51778f1b"

```

この出力には、インストールしたすべてのカーネルのパスと、各バージョンが表示されます。

2.6. モジュール情報の表示

modinfo コマンドを使用して、指定したカーネルモジュールに関する詳細情報を表示します。

前提条件

- **kmod** パッケージがインストールされている。

手順

- カーネルモジュールの情報を表示するには、以下を実行します。

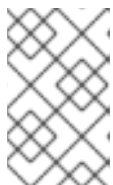
```
$ modinfo <KERNEL_MODULE_NAME>
```

以下に例を示します。

```
$ modinfo virtio_net

filename:    /lib/modules/4.18.0-94.el8.x86_64/kernel/drivers/net/virtio_net.ko.xz
license:    GPL
description: Virtio network driver
rhelversion: 8.1
srcversion: 2E9345B281A898A91319773
alias:      virtio:d00000001v*
depends:     net_failover
intree:     Y
name:       virtio_net
vermagic:   4.18.0-94.el8.x86_64 SMP mod_unload modversions
...
parm:      napi_weight:int
parm:      csum:bool
parm:      gso:bool
parm:      napi_tx:bool
```

読み込まれているかどうかに関わらず、利用可能なすべてのモジュールの情報を照会できます。**parm** エントリは、ユーザーがモジュールに設定できるパラメーターと、期待される値のタイプを示します。



注記

カーネルモジュールの名前を入力する際には、**.ko.xz** 拡張子は名前の末尾に追加しないでください。カーネルモジュール名には拡張子はありません。ただし、対応するファイルには拡張子があります。

関連情報

- **modinfo(8)** の man ページ

2.7. システムランタイム時のカーネルモジュールの読み込み

Linux カーネルの機能を拡張する最適な方法は、カーネルモジュールを読み込むことです。**modprobe** コマンドを使用して、カーネルモジュールを検出し、現在実行しているカーネルに読み込みます。



重要

この手順で説明されている変更は、システムを再起動は維持されません。システムの再起動後も設定を維持するようにカーネルモジュールを読み込む方法は、[システムの起動時に自動的にカーネルモジュールを読み込む](#)を参照してください。

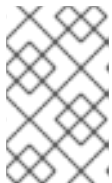
前提条件

- root 権限がある。
- **kmod** パッケージがインストールされている。
- 関連のカーネルモジュールが読み込まれていない。これを確認するには、[読み込まれているカーネルモジュール](#)をリスト表示します。

手順

1. 読み込むカーネルモジュールを選択します。
モジュールは `/lib/modules/$(uname -r)/kernel/<SUBSYSTEM>/` ディレクトリーにあります。
2. 関連するカーネルモジュールを読み込みます。

```
# modprobe <MODULE_NAME>
```



注記

カーネルモジュールの名前を入力する際には、**.ko.xz** 拡張子は名前の末尾に追加しないでください。カーネルモジュール名には拡張子はありません。ただし、対応するファイルには拡張子があります。

検証

- 必要に応じて、関連モジュールが読み込まれたことを確認します。

```
$ lsmod | grep <MODULE_NAME>
```

モジュールが正しく読み込まれた場合、このコマンドは関連するカーネルモジュールを表示します。以下に例を示します。

```
$ lsmod | grep serio_raw
serio_raw      16384 0
```

関連情報

- [modprobe\(8\)](#) の man ページ

2.8. システムランタイム時のカーネルモジュールのアンロード

時折、実行中のカーネルから特定のカーネルモジュールをアンロードする必要性に駆られることがあります。**modprobe** コマンドを使用して、現在読み込まれているカーネルから、システムの実行時にカーネルモジュールを見つけてアンロードします。



警告

実行中のシステムで使用される場合は、カーネルモジュールをアンロードしないでください。これを行うと、システムが不安定になったり、動作しなくなったりすることがあります。



重要

この手順を終了すると、システムの起動時に自動的に読み込まれるように定義したカーネルモジュールは、システムを再起動してもアンロードされません。この結果を追跡する方法は、[システムの起動時にカーネルモジュールが自動的にロードされないようにする](#)を参照してください。

前提条件

- root 権限がある。
- **kmod** パッケージがインストールされている。

手順

1. ロード済みの全カーネルモジュールをリスト表示します。

```
# lsmod
```

2. アンロードするカーネルモジュールを選択します。
カーネルモジュールに依存関係がある場合は、カーネルモジュールをアンロードする前に、これらをアンロードします。依存関係のあるモジュールを特定する方法は、[Listing currently loaded kernel modules](#) および [Kernel module dependencies](#) を参照してください。
3. 関連するカーネルモジュールをアンロードします。

```
# modprobe -r <MODULE_NAME>
```

カーネルモジュールの名前を入力する際には、**.ko.xz** 拡張子は名前の末尾に追加しないでください。カーネルモジュール名には拡張子はありません。ただし、対応するファイルには拡張子があります。

検証

- 必要に応じて、関連モジュールがアンロードされたことを確認します。

```
$ lsmod | grep <MODULE_NAME>
```

モジュールが正常にアンロードされた場合、このコマンドは出力を表示しません。

関連情報

- **modprobe(8)** の man ページ

2.9. 起動プロセスの初期段階でのカーネルモジュールのアンロード

特定の状況では、起動プロセスの初期段階でカーネルモジュールのアンロードが必要になります。たとえば、カーネルモジュールにコードが含まれているとシステムが応答しなくなり、ユーザーがステージに到達して不正なカーネルモジュールを永続的に無効にすることができません。その場合は、ブートローダーを使用して、カーネルモジュールの読み込みを一時的にブロックできます。

ブートシーケンスが実行する前に、関連するブートローダーエントリを編集して、必要なカーネルモジュールをアンロードできます。



重要

この手順で説明されている変更は、システムを再起動すると**維持されません**。起動プロセス時にカーネルモジュールが自動的に読み込まれないように、denylist にカーネルモジュールを追加する方法は、[システムの起動時にカーネルモジュールが自動的にロードされないようにする](#) を参照してください。

前提条件

- なんらかの理由で読み込みを阻止する必要のある、読み込み可能なカーネルモジュールがある。

手順

1. システムをブートローダーで起動します。
2. カーソルキーを使用して、関連するブートローダーエントリを強調表示します。
3. **e** キーを押してエントリを編集します。

図2.1 カーネルブートメニュー

```

Red Hat Enterprise Linux (4.18.0-305.10.2.el8_4.x86_64) 8.4 (Ootpa)
Red Hat Enterprise Linux (4.18.0-305.3.1.el8_4.x86_64) 8.4 (Ootpa)
Red Hat Enterprise Linux (4.18.0-240.22.1.el8_3.x86_64) 8.3 (Ootpa)
Red Hat Enterprise Linux (0-rescue-bfeb014063334b2da66b978e95804445) 8.3->

Use the ↑ and ↓ keys to change the selection.
Press 'e' to edit the selected item, or 'c' for a command prompt.

```

4. カーソルキーを使用して、linux で始まる行に移動します。
5. **modprobe.blacklist=module_name** を行末に追加します。

図2.2 カーネルブートエントリー

```

load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-305.10.2.el8_4.x86_64 root=/dev/mapper/rhel-root \
ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv\
=rhel/swap rhgb quiet modprobe.blacklist=serio_raw
initrd ($root)/initramfs-4.18.0-305.10.2.el8_4.x86_64.img $tuned_initrd

Press Ctrl-x to start, Ctrl-c for a command prompt or Escape to
discard edits and return to the menu. Pressing Tab lists
possible completions.

```

serio_raw カーネルモジュールは、起動プロセスの初期段階でアンロードする不正なモジュールを示しています。

6. **Ctrl+X** を押して、変更した設定を使用して起動します。

検証

- システムが完全に起動したら、関連するカーネルモジュールが読み込まれていないことを確認します。

```
# lsmod | grep serio_raw
```

関連情報

- [カーネルモジュールの管理](#)

2.10. システムの起動時に自動的にカーネルモジュールを読み込む

ブートプロセス中に自動的に読み込まれるようにカーネルモジュールを設定します。

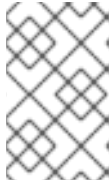
前提条件

- root 権限がある。
- **kmod** パッケージがインストールされている。

手順

1. 起動プロセス中に読み込むカーネルモジュールを選択します。
モジュールは `/lib/modules/$(uname -r)/kernel/<SUBSYSTEM>/` ディレクトリーにあります。
2. モジュールの設定ファイルを作成します。

```
# echo <MODULE_NAME> > /etc/modules-load.d/<MODULE_NAME>.conf
```

注記

カーネルモジュールの名前を入力する際には、**.ko.xz** 拡張子は名前の末尾に追加しないでください。カーネルモジュール名には拡張子はありません。ただし、対応するファイルには拡張子があります。

- 必要に応じて、関連モジュールが読み込まれたことを確認します。

```
$ lsmod | grep <MODULE_NAME>
```

上記のコマンド例は成功し、関連するカーネルモジュールを表示します。



重要

この手順で説明している変更は、システムを再起動しても**持続**されます。

関連情報

- **modules-load.d(5)** の man ページ

2.11. システムの起動時にカーネルモジュールが自動的にロードされないようにする

対応するコマンドを使用して、**modprobe** 設定ファイルにモジュールを一覧表示することで、起動プロセス中にシステムが自動的にカーネルモジュールを読み込むことを阻止できます。

前提条件

- この手順のコマンドには root 権限が必要です。 **su -** を使用して root ユーザーに切り替えるか、コマンドの前に **sudo** を付けます。
- **kmod** パッケージがインストールされている。
- 現在のシステム設定に、拒否する予定のカーネルモジュールが必要ないことを確認する。

手順

1. **lsmod** コマンドを使用して、現在実行中のカーネルに読み込まれているモジュールを一覧表示します。

```
$ lsmod
Module          Size Used by
tls             131072 0
uinput          20480 1
snd_seq_dummy   16384 0
snd_hrtimer     16384 1
...
```

出力で、読み込みを阻止するモジュールを特定します。

- または、**/lib/modules/<KERNEL-VERSION>/kernel/<SUBSYSTEM>/** ディレクトリーに読み込まれないようにするアンロードしたカーネルモジュールを特定します。以下に例を示します。

■

```
$ ls /lib/modules/4.18.0-477.20.1.el8_8.x86_64/kernel/crypto/
ansi_cprng.ko.xz      chacha20poly1305.ko.xz md4.ko.xz
serpent_generic.ko.xz
anubis.ko.xz         cmac.ko.xz...
```

- 拒否リストとして機能する設定ファイルを作成します。

```
# touch /etc/modprobe.d/denylist.conf
```

- 任意のテキストエディターで、カーネルへの自動読み込みから除外するモジュール名を **blacklist** 設定コマンドと組み合わせます。以下に例を示します。

```
# Prevents <KERNEL-MODULE-1> from being loaded
blacklist <MODULE-NAME-1>
install <MODULE-NAME-1> /bin/false

# Prevents <KERNEL-MODULE-2> from being loaded
blacklist <MODULE-NAME-2>
install <MODULE-NAME-2> /bin/false
...
```

blacklist コマンドは、モジュールが、拒否リストにない別のカーネルモジュールの依存関係として読み込まれることを阻止しないため、**install** 行も定義する必要があります。この場合、システムはモジュールをインストールする代わりに **/bin/false** を実行します。ハッシュ記号で始まる行は、ファイルをより読みやすくするために使用可能なコメントです。



注記

カーネルモジュールの名前を入力する際には、**.ko.xz** 拡張子は名前の末尾に追加しないでください。カーネルモジュール名には拡張子はありません。ただし、対応するファイルには拡張子があります。

- 再構築を行う前に、現在の初期 RAM ディスクイメージのバックアップコピーを作成します。

```
# cp /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r).bak.$(date +%m-%d-%H%M%S).img
```

- または、カーネルモジュールの自動読み込みを阻止するカーネルバージョンに対応する初期 RAM ディスクイメージのバックアップコピーを作成します。

```
# cp /boot/initramfs-<VERSION>.img /boot/initramfs-<VERSION>.img.bak.$(date +%m-%d-%H%M%S)
```

- 新しい初期 RAM ディスクイメージを生成して、変更を適用します。

```
# dracut -f -v
```

- システムで現在使用中のものとは異なるカーネルバージョンの初期 RAM ディスクイメージを構築する場合は、ターゲット **initramfs** とカーネルバージョンの両方を指定します。

```
# dracut -f -v /boot/initramfs-<TARGET-VERSION>.img <CORRESPONDING-TARGET-KERNEL-VERSION>
```

6. システムを再起動します。

```
$ reboot
```



重要

この手順で説明している変更は、システムを再起動しても**持続されません**。拒否リストにキーカーネルモジュールを誤って記載した場合、システムを不安定または操作不能な状態に切り換えることができます。

関連情報

- ソリューション記事 [How do I prevent a kernel module from loading automatically?](#) を参照してください。
- **modprobe.d(5)** および **dracut(8)** の man ページ を参照してください。

2.12. カスタムカーネルモジュールのコンパイル

ハードウェアおよびソフトウェアレベルで、さまざまな設定による要求に応じて、サンプリングカーネルモジュールを構築できます。

前提条件

- **kernel-devel** パッケージ、**gcc** パッケージ、および **elfutils-libelf-devel** パッケージをインストールしている。

```
# dnf install kernel-devel-$(uname -r) gcc elfutils-libelf-devel
```

- root 権限がある。
- カスタムカーネルモジュールをコンパイルする **/root/testmodule/** ディレクトリを作成している。

手順

1. 以下の内容で **/root/testmodule/test.c** を作成します。

```
#include <linux/module.h>
#include <linux/kernel.h>

int init_module(void)
{ printk("Hello World\n This is a test\n"); return 0; }

void cleanup_module(void)
{ printk("Good Bye World"); }
```

test.c ファイルは、カーネルモジュールに主な機能を提供するソースファイルです。このファイルは、組織的な目的で、専用の **/root/testmodule/** ディレクトリに作成されています。モジュールをコンパイルすると、**/root/testmodule/** ディレクトリには複数のファイルが含まれます。

test.c ファイルには、システムライブラリーから次のものが含まれます。

- サンプルコードの `printk()` 機能には、`linux/kernel.h` ヘッダーファイルが必要です。
 - `linux/module.h` ファイルには、C 言語で記述した複数のソースファイルで共有する関数宣言とマクロ定義が含まれています。
- 次に、`init_module()` 関数および `cleanup_module()` 関数に従い、テキストを出力するカーネルロギング機能 `printk()` を起動および終了します。
 - 以下の内容で `/root/testmodule/Makefile` を作成します。

```
obj-m := test.o
```

Makefile には、コンパイラーが、`test.o` という名前のオブジェクトファイルを作成する必要がある指示が含まれています。`obj-m` ディレクティブは、生成される `test.ko` ファイルを、読み込み可能なカーネルモジュールとしてコンパイルすることを指定します。もしくは、`obj-y` ディレクティブは、組み込みカーネルモジュールとして `test.ko` をビルドするように指示します。

- カーネルモジュールをコンパイルします。

```
# make -C /lib/modules/$(uname -r)/build M=/root/testmodule modules
make: Entering directory '/usr/src/kernels/4.18.0-305.el8.x86_64'
  CC [M] /root/testmodule/test.o
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /root/testmodule/test.o
see include/linux/module.h for more information
  CC /root/testmodule/test.mod.o
  LD [M] /root/testmodule/test.ko
make: Leaving directory '/usr/src/kernels/4.18.0-305.el8.x86_64'
```

コンパイラーは、各ソースファイル (`test.c`) のオブジェクトファイル (`test.o`) を中間手順として作成してから、それらを最終カーネルモジュール (`test.ko`) にリンクします。

コンパイルが成功すると、`/root/testmodule/` には、コンパイル済みカスタムカーネルモジュールに関連する追加ファイルが含まれます。コンパイル済みモジュール自身は、`test.ko` ファイルで表されます。

検証

- 必要に応じて、`/root/testmodule/` ディレクトリーのコンテンツを確認します。

```
# ls -l /root/testmodule/
total 152
-rw-r--r--. 1 root root 16 Jul 26 08:19 Makefile
-rw-r--r--. 1 root root 25 Jul 26 08:20 modules.order
-rw-r--r--. 1 root root 0 Jul 26 08:20 Module.symvers
-rw-r--r--. 1 root root 224 Jul 26 08:18 test.c
-rw-r--r--. 1 root root 62176 Jul 26 08:20 test.ko
-rw-r--r--. 1 root root 25 Jul 26 08:20 test.mod
-rw-r--r--. 1 root root 849 Jul 26 08:20 test.mod.c
-rw-r--r--. 1 root root 50936 Jul 26 08:20 test.mod.o
-rw-r--r--. 1 root root 12912 Jul 26 08:20 test.o
```

- カーネルモジュールを `/lib/modules/$(uname -r)/` ディレクトリーにコピーします。

```
# cp /root/testmodule/test.ko /lib/modules/$(uname -r)/
```

3. モジュールの依存関係のリストを更新します。

```
# depmod -a
```

4. カーネルモジュールを読み込みます。

```
# modprobe -v test  
insmod /lib/modules/4.18.0-305.el8.x86_64/test.ko
```

5. カーネルモジュールが正常に読み込まれたことを確認します。

```
# lsmod | grep test  
test                16384 0
```

6. カーネルリングバッファから最新のメッセージを読み込みます。

```
# dmesg  
[74422.545004] Hello World  
                This is a test
```

関連情報

- [カーネルモジュールの管理](#)

第3章 セキュアブート用のカーネルとモジュールの署名

署名済みカーネルと署名済みカーネルモジュールを使用して、システムのセキュリティーを強化できます。セキュアブートが有効になっている UEFI ベースのビルドシステムでは、プライベートにビルドされたカーネルまたはカーネルモジュールに自己署名できます。さらに、カーネルまたはカーネルモジュールをデプロイするターゲットシステムに公開鍵をインポートすることもできます。

セキュアブートが有効な場合、次のすべてのコンポーネントを秘密鍵で署名し、対応する公開鍵で認証する必要があります。

- UEFI オペレーティングシステムのブートローダー
- Red Hat Enterprise Linux カーネル
- すべてのカーネルモジュール

これらのコンポーネントのいずれかが署名および認証されていない場合、システムは起動プロセスを完了できません。

RHEL 8 には以下が含まれます。

- 署名済みブートローダー
- 署名済みカーネル
- 署名済みカーネルモジュール

また、署名された第1ステージのブートローダーと署名されたカーネルには、組み込み Red Hat 公開鍵が含まれています。これらの署名済み実行可能バイナリーと組み込みキーにより、RHEL 8 は、UEFI セキュアブートをサポートするシステム上の UEFI ファームウェアによって提供される Microsoft UEFI セキュアブート認証局キーでインストール、起動、および実行できます。



注記

- セキュアブートのサポートは、すべての UEFI ベースのシステムに含まれるわけではありません。
- カーネルモジュールを構築、署名するビルドシステムは、UEFI セキュアブートを有効にする必要がなく、UEFI ベースのシステムである必要すらありません。

3.1. 前提条件

- 外部でビルドされたカーネルモジュールに署名できるようにするには、次のパッケージからユーティリティーをインストールします。

```
# yum install pesign openssl kernel-devel mokutil keyutils
```

表3.1 必要なユーティリティー

ユーティリ ティー	提供するパッ ケージ	使用対象	目的
efikeygen	pesign	ビルドシステム	公開および秘密 X.509 鍵のペアを生成

ユーティリ ティ	提供するパッ ケージ	使用対象	目的
openssl	openssl	ビルドシステ ム	暗号化されていない秘密鍵をエクスポート します。
sign-file	kernel-devel	ビルドシステ ム	秘密鍵でカーネルモジュールに署名するた めに使用する実行ファイル
mokutil	mokutil	ターゲットシ ステム	公開鍵を手動で登録する際に使用するオブ ションのユーティリティー
keyctl	keyutils	ターゲットシ ステム	システムキーリングへの公開鍵の表示時に 使用するオプションのユーティリティー

3.2. UEFI セキュアブートとは

Unified Extensible Firmware Interface(UEFI) セキュアブートテクノロジーを使用すると、信頼できる鍵によって署名されていないカーネル空間コードの実行を防ぐことができます。システムブートルoaderは暗号鍵で署名されています。ファームウェアに含まれる公開鍵のデータベースは、署名鍵を承認します。その後、次のステージのブートルoaderとカーネルで署名を検証できます。

UEFI セキュアブートは、以下のようにファームウェアから署名済みドライバーおよびカーネルモジュールへの信頼チェーンを確立します。

- UEFI 秘密鍵が **shim** 第1ステージブートルoaderに署名し、それを公開鍵が認証します。認証局 (CA) は公開鍵に署名します。CA はファームウェアのデータベースに保存されます。
- **shim** ファイルには、GRUB ブートルoaderとカーネルを認証するための Red Hat 公開鍵 **Red Hat Secure Boot (CA key 1)** が含まれています。
- カーネルには、ドライバーおよびモジュールを認証する公開鍵が含まれます。

セキュアブートは、UEFI 仕様のブートパス検証コンポーネントです。この仕様は、以下を定義します。

- 揮発性ではないストレージでの暗号で保護された UEFI 変数用のプログラミングインターフェイス
- UEFI 変数での信頼できる X.509 ルート証明書の保存
- ブートルoaderやドライバーなどの UEFI アプリケーションの検証
- 既知の問題のある証明書およびアプリケーションハッシュを無効にする手順

UEFI セキュアブートは、不正な変更の検出には役立ちますが、以下を行うことは **できません**。

- 第2ステージブートルoaderのインストールまたは削除を防止する。
- このような変更について、ユーザーによる明示的な確認を要求する。
- ブートパスの操作を停止する。署名は、ブートルoaderのインストールや更新時ではなく、起動時に検証されます。

ブートローダーまたはカーネルがシステムの信頼された鍵で署名されていない場合、セキュアブートにより起動が妨げられます。

3.3. UEFI セキュアブートのサポート

カーネルとロードされたすべてのドライバーが信頼できるキーで署名されている場合、UEFI セキュアブートが有効になっているシステムに RHEL 8 をインストールして実行できます。Red Hat は、関連する Red Hat キーによって署名および認証されたカーネルとドライバーを提供します。

外部でビルドされたカーネルまたはドライバーをロードする場合は、それらにも署名する必要があります。

UEFI セキュアブートによる制限

- システムは、署名が適切に認証された後にのみ、カーネルモードコードを実行します。
- GRUB モジュールの署名および検証を行うインフラストラクチャーがないため、GRUB モジュールの読み込みは無効です。モジュールの読み込みを許可すると、セキュアブートが定義するセキュリティー境界内で信頼できないコードが実行されることになります。
- Red Hat は、RHEL 8 でサポートされるすべてのモジュールが含まれる署名済み GRUB バイナリーを提供します。

関連情報

- [UEFI セキュアブートによる制限](#)

3.4. X.509 鍵でカーネルモジュールを認証するための要件

RHEL 8 では、カーネルモジュールがロードされると、カーネルはカーネルシステムキーリング (`.builtin_trusted_keys`) およびカーネルプラットフォームキーリング (`.platform`) からの公開 X.509 キーに対してモジュールの署名をチェックします。`.platform` キーリングには、サードパーティーのプラットフォームプロバイダーおよびカスタム公開鍵からのキーが含まれます。カーネルシステムの `.blacklist` キーリングからの鍵は検証から除外されます。

UEFI セキュアブート機能が有効になっているシステムでカーネルモジュールをロードするには、特定の条件を満たす必要があります。

- UEFI セキュアブートが有効な場合、または `module.sig_enforce` カーネルパラメーターが指定されている場合:
 - 署名がシステムキーリング (`.builtin_trusted_keys`) およびプラットフォームキーリング (`.platform`) からの鍵に対して認証されている署名済みのカーネルモジュールだけを読み込みできます。
 - 公開鍵は、システムで拒否されたキーのキーリング (`.blacklist`) に配置できません。
- UEFI セキュアブートが無効で `module.sig_enforce` カーネルパラメーターが指定されていない場合:
 - 公開鍵なしで、未署名のカーネルモジュールと署名済みカーネルモジュールを読み込むことができます。
- システムが UEFI ベースでない場合、または UEFI セキュアブートが無効になっている場合:
 - カーネルに埋め込まれた鍵のみが `.builtin_trusted_keys` および `.platform` に読み込まれます。

す。

- カーネルの再構築なしでキーセットを拡張することはできません。

表3.2 カーネルモジュールの読み込み認証要件

モジュールの署名	公開鍵ありおよび署名が有効	UEFI セキュアブートの状態	sig_enforce	モジュールの読み込み	カーネルのテイント
署名なし	-	有効でない	有効でない	成功	はい
		有効でない	有効	失敗	-
		有効	-	失敗	-
署名あり	いいえ	有効でない	有効でない	成功	はい
		有効でない	有効	失敗	-
		有効	-	失敗	-
署名あり	はい	有効でない	有効でない	成功	いいえ
		有効でない	有効	成功	いいえ
		有効	-	成功	いいえ

3.5. 公開鍵のソース

カーネルは、起動時に X.509 キーを永続キーストアから以下のキーリングに読み込みます。

- システムキーリング (`.builtin_trusted_keys`)
- `.platform` キーリング
- システムの `.blacklist` キーリング

表3.3 システムキーリングのソース

X.509 鍵のソース	ユーザーによるキーの追加	UEFI セキュアブートの状態	ブート中に読み込まれる鍵
カーネルに埋め込み	いいえ	-	<code>.builtin_trusted_keys</code>
UEFI <code>db</code>	限定的	有効でない	いいえ

X.509 鍵のソース	ユーザーによるキーの追加	UEFI セキュアブートの状態	ブート中に読み込まれる鍵
		有効	.platform
shim ブートローダーに埋め込み	いいえ	有効でない	いいえ
		有効	.platform
Machine Owner Key (MOK) リスト	はい	有効でない	いいえ
		有効	.platform

.builtin_trusted_keys

- 起動時にビルドされるキーリング
- 信頼できる公開鍵が含まれています。
- 鍵を表示するには **root** 権限が必要

.platform

- 起動時にビルドされるキーリング
- サードパーティーのプラットフォームプロバイダーからのキーとカスタムの公開鍵が含まれています。
- 鍵を表示するには **root** 権限が必要

.blacklist

- 失効した X.509 キーを含むキーリング
- 公開鍵が **.builtin_trusted_keys** にある場合でも、**.blacklist** からのキーで署名されたモジュールは認証に失敗します。

UEFI セキュアブート db

- 署名データベース
- UEFI アプリケーション、UEFI ドライバー、およびブートローダーのキー (ハッシュ) を保存します。
- キーはマシンにロードできます。

UEFI セキュアブート dbx

- 失効した署名データベース
- キーが読み込まれないようにします。
- このデータベースからの失効したキーは、**.blacklist** キーリングに追加されます。

3.6. 公開鍵と秘密鍵の生成

セキュアブート対応システムでカスタムカーネルまたはカスタムカーネルモジュールを使用するには、X.509 の公開鍵と秘密鍵のペアを生成する必要があります。生成された秘密鍵を使用して、カーネルまたはカーネルモジュールに署名できます。また、対応する公開鍵をセキュアブートの Machine Owner Key (MOK) に追加することで、署名済みのカーネルまたはカーネルモジュールを検証できます。



警告

強力なセキュリティー対策とアクセスポリシーを適用して、秘密鍵の内容を保護します。悪用すれば、この鍵は、一致する公開鍵で認証されるシステムのセキュリティーに危害を与えるために使用できます。

手順

- X.509 の公開鍵と秘密鍵のペアを作成します。
 - カスタムカーネル **モジュール** に署名するだけの場合:

```
# efkeygen --dbdir /etc/pki/pesign \
  --self-sign \
  --module \
  --common-name 'CN=Organization signing key' \
  --nickname 'Custom Secure Boot key'
```

- カスタム **カーネル** に署名する場合:

```
# efkeygen --dbdir /etc/pki/pesign \
  --self-sign \
  --kernel \
  --common-name 'CN=Organization signing key' \
  --nickname 'Custom Secure Boot key'
```

- RHEL システムが FIPS モードを実行している場合:

```
# efkeygen --dbdir /etc/pki/pesign \
  --self-sign \
  --kernel \
  --common-name 'CN=Organization signing key' \
  --nickname 'Custom Secure Boot key' \
  --token 'NSS FIPS 140-2 Certificate DB'
```



注記

FIPS モードでは、**efkeygen** が PKI データベース内でデフォルトの NSS Certificate DB トークンを検索できるように、**--token** オプションを使用する必要があります。

公開鍵と秘密鍵は `/etc/pki/pesign/` ディレクトリーに保存されます。



重要

セキュリティ上、署名鍵の有効期間内にカーネルとカーネルモジュールに署名することが推奨されます。ただし、**sign-file** ユーティリティーは警告を表示せず、キーは有効期限に関係なく RHEL 8 で使用できます。

関連情報

- [openssl\(1\) の man ページ](#)
- [RHEL Security Guide](#)
- [公開鍵を MOK リストに追加することでターゲットシステムで公開鍵を登録する手順](#)

3.7. システムキーリングの出力例

keyutils パッケージからの **keyctl** ユーティリティーを使用して、システムのキーリングの鍵に関する情報を表示できます。

前提条件

- root 権限がある。
- **keyutils** パッケージから **keyctl** ユーティリティーをインストールしました。

例3.1 キーリング出力

以下は、UEFI セキュアブートが有効になっている RHEL 8 システムからの **.builtin_trusted_keys**、**.platform**、および **.blacklist** キーリングの短縮された出力例です。

```
# keyctl list %:.builtin_trusted_keys
6 keys in keyring:
...asymmetric: Red Hat Enterprise Linux Driver Update Program (key 3): bf57f3e87...
...asymmetric: Red Hat Secure Boot (CA key 1): 4016841644ce3a810408050766e8f8a29...
...asymmetric: Microsoft Corporation UEFI CA 2011: 13adbf4309bd82709c8cd54f316ed...
...asymmetric: Microsoft Windows Production PCA 2011: a92902398e16c49778cd90f99e...
...asymmetric: Red Hat Enterprise Linux kernel signing key: 4249689eefc77e95880b...
...asymmetric: Red Hat Enterprise Linux kpatch signing key: 4d38fd864ebe18c5f0b7...

# keyctl list %:.platform
4 keys in keyring:
...asymmetric: VMware, Inc.: 4ad8da0472073...
...asymmetric: Red Hat Secure Boot CA 5: cc6fafa72...
...asymmetric: Microsoft Windows Production PCA 2011: a929f298e1...
...asymmetric: Microsoft Corporation UEFI CA 2011: 13adbf4e0bd82...

# keyctl list %:.blacklist
4 keys in keyring:
...blacklist: bin:f5ff83a...
...blacklist: bin:0dfdbec...
...blacklist: bin:38f1d22...
...blacklist: bin:51f831f...
```

この例の `.builtin_trusted_keys` キーリングは、UEFI セキュアブート `db` キーからの2つのキー、および `shim` ブートローダーに組み込まれている **Red Hat Secure Boot (CA key 1)** の追加を表しています。

例3.2 カーネルコンソール出力

以下の例は、カーネルコンソールの出力を示しています。このメッセージでは、UEFI セキュアブートに関連するソースの鍵を特定します。これらには、UEFI セキュアブート `db`、組み込みの `shim`、および MOK リストが含まれます。

```
# dmesg | egrep 'integrity.*cert'
[1.512966] integrity: Loading X.509 certificate: UEFI:db
[1.513027] integrity: Loaded X.509 cert 'Microsoft Windows Production PCA 2011: a929023...
[1.513028] integrity: Loading X.509 certificate: UEFI:db
[1.513057] integrity: Loaded X.509 cert 'Microsoft Corporation UEFI CA 2011: 13adbf4309...
[1.513298] integrity: Loading X.509 certificate: UEFI:MokListRT (MOKvar table)
[1.513549] integrity: Loaded X.509 cert 'Red Hat Secure Boot CA 5: cc6fa5e72868ba494e93...
```

関連情報

- `keyctl(1)`、`dmesg(1)` の man ページ

3.8. 公開鍵を MOK リストに追加することでターゲットシステムで公開鍵を登録する手順

カーネルまたはカーネルモジュールを認証およびロードするすべてのシステムに、公開鍵を登録する必要があります。さまざまな方法でターゲットシステムに公開鍵をインポートして、プラットフォームキーリング (`.platform`) が公開鍵を使用してカーネルまたはカーネルモジュールを認証できるようにすることができます。

セキュアブートが有効になっている UEFI ベースのシステムで RHEL 8 が起動すると、カーネルはセキュアブート `db` キーデータベースにあるすべての公開鍵をプラットフォームキーリング (`.platform`) にロードします。同時に、カーネルは失効したキーの `dbx` データベース内のキーを除外します。

Machine Owner Key (MOK) 機能を使用して、UEFI セキュアブートキーデータベースを拡張できます。セキュアブートが有効な UEFI 対応システムで RHEL 8 が起動すると、キーデータベースの鍵に加えて、MOK リストの鍵もプラットフォームキーリング (`.platform`) に追加されます。MOK リストの鍵は、セキュアブートデータベースの鍵と同様に永続的かつ安全な方法で保存されますが、これらは別個の機能です。MOK 機能は、`shim`、`MokManager`、`GRUB`、および `mokutil` ユーティリティーでサポートされています。



注記

システムでカーネルモジュールの認証を実現するために、ファクトリーファームウェアイメージで公開鍵を UEFI セキュアブート鍵データベースに組み入れるようシステムベンダーに要求することを検討します。

前提条件

- 公開鍵と秘密鍵のペアを生成し、公開鍵の有効期限を知っています。詳細については、[公開鍵と秘密鍵のペアの生成](#) を参照してください。

手順

1. 公開鍵を **sb_cert.cer** ファイルにエクスポートします。

```
# certutil -d /etc/pki/pesign \
  -n 'Custom Secure Boot key' \
  -Lr \
  > sb_cert.cer
```

2. 公開鍵を MOK リストにインポートします。

```
# mokutil --import sb_cert.cer
```

3. この MOK 登録要求の新しいパスワードを入力してください。
4. マシンを再起動します。
shim ブートローダーは、保留中の MOK キー登録要求を認識し、**MokManager.efi** を起動して、UEFI コンソールから登録を完了できるようにします。
5. **Enroll MOK** を選択し、プロンプトが表示されたら、この要求に関連付けたパスワードを入力し、登録を確認します。
公開鍵が MOK リストに永続的に追加されます。

キーが MOK リストに追加されると、UEFI セキュアブートが有効になっている場合は、このブートおよび後続のブートで **.platform** キーリングに自動的に伝達されます。

3.9. 秘密鍵でカーネルに署名する

UEFI セキュアブート機能が有効になっている場合は、署名済みカーネルをロードすると、システムのセキュリティを強化できます。

前提条件

- 公開鍵と秘密鍵のペアを生成し、公開鍵の有効期限を知っています。詳細については、[公開鍵と秘密鍵のペアの生成](#) を参照してください。
- ターゲットシステムに公開鍵を登録しています。詳細については、[公開鍵を MOK リストに追加して、ターゲットシステムに公開鍵を登録する](#) を参照してください。
- 署名に使用できる ELF 形式のカーネルイメージがあります。

手順

- x64 アーキテクチャーの場合:
 - a. 署名済みイメージを作成します。

```
# pesign --certificate 'Custom Secure Boot key' \
  --in vmlinuz-version \
  --sign \
  --out vmlinuz-version.signed
```

version を **vmlinuz** ファイルのバージョン接尾辞に置き換え、**Custom Secure Boot key** を以前に選択した名前に置き換えます。

b. オプション: 署名を確認します。

```
# pesign --show-signature \  
--in vmlinuz-version.signed
```

c. 未署名イメージを署名済みイメージで上書きします。

```
# mv vmlinuz-version.signed vmlinuz-version
```

• 64 ビット ARM アーキテクチャーの場合:

a. **vmlinuz** ファイルを解凍します。

```
# zcat vmlinuz-version > vmlinux-version
```

b. 署名済みイメージを作成します。

```
# pesign --certificate 'Custom Secure Boot key' \  
--in vmlinux-version \  
--sign \  
--out vmlinux-version.signed
```

c. オプション: 署名を確認します。

```
# pesign --show-signature \  
--in vmlinux-version.signed
```

d. **vmlinux** ファイルを圧縮します。

```
# gzip --to-stdout vmlinux-version.signed > vmlinuz-version
```

e. 圧縮されていない **vmlinux** ファイルを削除します。

```
# rm vmlinux-version*
```

3.10. 秘密鍵で GRUB ビルドに署名する

UEFI セキュアブート機能が有効になっているシステムでは、カスタムの既存の秘密鍵で GRUB ビルドに署名できます。カスタム GRUB ビルドを使用している場合、またはシステムから Microsoft トラストアンカーを削除した場合は、これを行う必要があります。

前提条件

- 公開鍵と秘密鍵のペアを生成し、公開鍵の有効期限を知っています。詳細については、[公開鍵と秘密鍵のペアの生成](#) を参照してください。
- ターゲットシステムに公開鍵を登録しています。詳細については、[公開鍵を MOK リストに追加して、ターゲットシステムに公開鍵を登録する](#) を参照してください。
- 署名に使用できる GRUB EFI バイナリーがあります。

手順

- x64 アーキテクチャーの場合:
 - a. 署名済み GRUB EFI バイナリーを作成します。

```
# pesign --in /boot/efi/EFI/redhat/grubx64.efi \
--out /boot/efi/EFI/redhat/grubx64.efi.signed \
--certificate 'Custom Secure Boot key' \
--sign
```

Custom Secure Boot key を以前に選択した名前に置き換えます。

- b. オプション: 署名を確認します。

```
# pesign --in /boot/efi/EFI/redhat/grubx64.efi.signed \
--show-signature
```

- c. 署名されていないバイナリーを署名済みバイナリーで上書きします。

```
# mv /boot/efi/EFI/redhat/grubx64.efi.signed \
/boot/efi/EFI/redhat/grubx64.efi
```

- 64 ビット ARM アーキテクチャーの場合:

- a. 署名済み GRUB EFI バイナリーを作成します。

```
# pesign --in /boot/efi/EFI/redhat/grubaa64.efi \
--out /boot/efi/EFI/redhat/grubaa64.efi.signed \
--certificate 'Custom Secure Boot key' \
--sign
```

Custom Secure Boot key を以前に選択した名前に置き換えます。

- b. オプション: 署名を確認します。

```
# pesign --in /boot/efi/EFI/redhat/grubaa64.efi.signed \
--show-signature
```

- c. 署名されていないバイナリーを署名済みバイナリーで上書きします。

```
# mv /boot/efi/EFI/redhat/grubaa64.efi.signed \
/boot/efi/EFI/redhat/grubaa64.efi
```

3.11. 秘密鍵を使用したカーネルモジュールの署名

UEFI セキュアブートメカニズムが有効になっている場合は、署名済みカーネルモジュールをロードすることでシステムのセキュリティーを強化できます。

署名済みカーネルモジュールは、UEFI セキュアブートが無効になっているシステムまたは非 UEFI システムでもロードできます。そのため、カーネルモジュールの署名済みバージョンと未署名バージョンの両方を提供する必要はありません。

前提条件

- 公開鍵と秘密鍵のペアを生成し、公開鍵の有効期限を知っています。詳細については、[公開鍵と秘密鍵のペアの生成](#)を参照してください。
- ターゲットシステムに公開鍵を登録しています。詳細については、[公開鍵を MOK リストに追加して、ターゲットシステムに公開鍵を登録する](#)を参照してください。
- ELF イメージ形式で署名できるカーネルモジュールがある。

手順

1. 公開鍵を **sb_cert.cer** ファイルにエクスポートします。

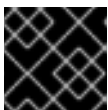
```
# certutil -d /etc/pki/pesign \  
-n 'Custom Secure Boot key' \  
-Lr \  
> sb_cert.cer
```

2. NSS データベースからキーを PKCS #12 ファイルとして抽出します。

```
# pk12util -o sb_cert.p12 \  
-n 'Custom Secure Boot key' \  
-d /etc/pki/pesign
```

3. 前のコマンドでプロンプトが表示されたら、秘密鍵を暗号化する新しいパスワードを入力します。
4. 暗号化されていない秘密鍵をエクスポートします。

```
# openssl pkcs12 \  
-in sb_cert.p12 \  
-out sb_cert.priv \  
-nocerts \  
-nodes
```



重要

暗号化されていない秘密鍵は慎重に取り扱ってください。

5. カーネルモジュールに署名します。次のコマンドは、カーネルモジュールファイル内の ELF イメージに署名を直接追加します。

```
# /usr/src/kernels/$(uname -r)/scripts/sign-file \  
sha256 \  
sb_cert.priv \  
sb_cert.cer \  
my_module.ko
```

これでカーネルモジュールの読み込み準備が完了しました。



重要

RHEL 8 では、キーペアの有効期限が重要です。キーの有効期限はありませんが、カーネルモジュールはその署名キーの有効期間内に署名する必要があります。**sign-file** ユーティリティーでは、これに関する警告は表示されません。たとえば、2019 でのみ有効な鍵を使用して、その鍵で 2019 で署名されたカーネルモジュールを認証できます。ただし、ユーザーはこの鍵を使用して 2020 でカーネルモジュールに署名することはできません。

検証

1. カーネルモジュールの署名に関する情報を表示します。

```
# modinfo my_module.ko | grep signer
signer:    Your Name Key
```

生成時に入力した名前が署名に記載されていることを確認します。



注記

この追加された署名は ELF イメージセクションには含まれず、また ELF イメージの正式な一部ではありません。したがって、**readelf** などのユーティリティーは、カーネルモジュールの署名を表示できません。

2. モジュールをロードします。

```
# insmod my_module.ko
```

3. モジュールを削除 (アンロード) します。

```
# modprobe -r my_module.ko
```

関連情報

- [モジュール情報の表示](#)

3.12. 署名済みカーネルモジュールの読み込み

公開鍵がシステムキーリング (**.builtin_trusted_keys**) と MOK リストに登録され、秘密鍵でそれぞれのカーネルモジュールに署名した後、**modprobe** コマンドで署名済みのカーネルモジュールをロードできます。

前提条件

- 公開鍵と秘密鍵のペアを生成しました。詳細については、[公開鍵と秘密鍵のペアの生成](#) を参照してください。
- 公開鍵をシステムのキーリングに登録している。詳細については、[公開鍵を MOK リストに追加して、ターゲットシステムに公開鍵を登録する](#) を参照してください。
- 秘密鍵でカーネルモジュールに署名している。詳細については、[秘密鍵を使用したカーネルモジュールの署名](#) を参照してください。

- `/lib/modules/$(uname -r)/extra/` ディレクトリーを作成する `kernel-modules-extra` パッケージをインストールします。

```
# yum -y install kernel-modules-extra
```

手順

1. 公開鍵がシステムキーリング上にあることを確認します。

```
# keyctl list %:.platform
```

2. カーネルモジュールを必要なカーネルの `extra/` ディレクトリーにコピーします。

```
# cp my_module.ko /lib/modules/$(uname -r)/extra/
```

3. モジュールの依存関係のリストを更新します。

```
# depmod -a
```

4. カーネルモジュールを読み込みます。

```
# modprobe -v my_module
```

5. 必要に応じて、起動時にモジュールをロードするには、モジュールを `/etc/modules-loaded.d/my_module.conf` ファイルに追加します。

```
# echo "my_module" > /etc/modules-load.d/my_module.conf
```

検証

- モジュールが正常にロードされたことを確認します。

```
# lsmod | grep my_module
```

関連情報

- [カーネルモジュールの管理](#)

第4章 カーネルコマンドラインパラメーターの設定

カーネルコマンドラインパラメーターを使用すると、ブート時に Red Hat Enterprise Linux カーネルの特定の側面の動作を変更できます。システム管理者は、システムの起動時に設定されるオプションを完全に制御できます。特定のカーネルの動作はシステムの起動時にのみ設定できるため、このような変更を行う方法を理解することが管理スキルの鍵となります。



重要

カーネルコマンドラインパラメーターを変更してシステムの動作を変更すると、システムに悪影響が及ぶ可能性があります。変更を実稼働環境にデプロイする前に、必ず変更をテストしてください。詳細なガイダンスは、Red Hat サポートまでご連絡ください。

4.1. カーネルコマンドラインパラメーターの設定

カーネルコマンドラインパラメーターを使用すると、デフォルト値を上書きしたり、特定のハードウェア設定を指定したりできます。ブート時に、次の機能を設定できます。

- Red Hat Enterprise Linux カーネル
- 初期 RAM ディスク
- ユーザー領域機能

デフォルトでは、GRUB ブートローダーを使用するシステムのカーネルコマンドラインパラメーターは、カーネルブートエントリーごとに `/boot/grub2/grubenv` ファイルの `kernelopts` 変数で定義されません。



注記

IBM Z では、`zipl` ブートローダーは環境変数に対応していないため、カーネルコマンドラインパラメーターはブートエントリー設定ファイルに保存されます。したがって、`kernelopts` 環境変数は使用できません。

`grubby` ユーティリティを使用すると、ブートローダー設定ファイルを操作できます。`grubby` を使用すると、次のアクションを実行できます。

- デフォルトのブートエントリーの変更
- GRUB メニューエントリーに対する引数の追加または削除

関連情報

- `kernel-command-line(7)`、`bootparam(7)`、および `dracut.cmdline (7)` の man ページ
- [How to install and boot custom kernels in Red Hat Enterprise Linux 8](#)
- `grubby(8)` の man ページ

4.2. ブートエントリーについて

ブートエントリーは設定ファイルに格納され、特定のカーネルバージョンに関連付けられるオプションの集合です。実際には、ブートエントリーは、システムにカーネルがインストールされているのと同じ数だけあります。ブートエントリーの設定ファイルは、`/boot/loader/entries/` ディレクトリーにあり、

以下のようになります。

```
6f9cc9cb7d7845d49698c9537337cedc-4.18.0-5.el8.x86_64.conf
```

上記のファイル名は、`/etc/machine-id` ファイルに保存されているマシン ID と、カーネルバージョンから設定されます。

ブートエントリーの設定ファイルには、カーネルバージョン、初期 ramdisk イメージ、および **kernelopts** 環境変数 (カーネルコマンドラインパラメーターを含む) に関する情報が含まれます。ブートエントリー設定例の内容は、以下のようになります。

```
title Red Hat Enterprise Linux (4.18.0-74.el8.x86_64) 8.0 (Ootpa)
version 4.18.0-74.el8.x86_64
linux /vmlinuz-4.18.0-74.el8.x86_64
initrd /initramfs-4.18.0-74.el8.x86_64.img $tuned_initrd
options $kernelopts $tuned_params
id rhel-20190227183418-4.18.0-74.el8.x86_64
grub_users $grub_users
grub_arg --unrestricted
grub_class kernel
```

kernelopts 環境変数は `/boot/grub2/grubenv` ファイルで定義されます。

関連情報

- [How to install and boot custom kernels in Red Hat Enterprise Linux 8](#)

4.3. すべてのブートエントリーでカーネルコマンドラインパラメーターの変更

システム上のすべてのブートエントリーのカーネルコマンドラインパラメーターを変更します。

前提条件

- **grubby** ユーティリティーがシステムにインストールされていることを確認してください。
- **zipl** ユーティリティーが IBM Z システムにインストールされていることを確認してください。

手順

- パラメーターを追加するには、以下を行います。

```
# grubby --update-kernel=ALL --args="<NEW_PARAMETER>"
```

GRUB ブートローダーを使用するシステムの場合は、`/boot/grub2/grubenv` の **kernelopts** 変数に新しいカーネルパラメーターを追加して、ファイルを更新します。

- IBM Z で、ブートメニューを更新します。

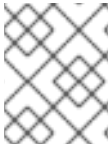
```
# zipl
```

- パラメーターを削除するには、次のコマンドを実行します。

```
# grubby --update-kernel=ALL --remove-args="<PARAMETER_TO_REMOVE>"
```

- IBM Z で、ブートメニューを更新します。

```
# zipl
```



注記

新しくインストールされたカーネルは、以前に設定されたカーネルからカーネルコマンドラインパラメーターを継承します。

関連情報

- [カーネルコマンドラインパラメーターの設定](#)
- [grubby\(8\)](#) および [zipl\(8\)](#) の man ページ
- [grubby ツール](#)

4.4.1つのブートエントリーでカーネルコマンドラインパラメーターの変更

システム上の単一のブートエントリーのカーネルコマンドラインパラメーターを変更します。

前提条件

- **grubby** ユーティリティおよび **zipl** ユーティリティがシステムにインストールされている。

手順

- パラメーターを追加するには、以下を行います。

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="<NEW_PARAMETER>"
```

- IBM Z で、ブートメニューを更新します。

```
# zipl
```

- パラメーターを削除するには、次のコマンドを実行します。

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --remove-args="<PARAMETER_TO_REMOVE>"
```

- IBM Z で、ブートメニューを更新します。

```
# zipl
```



注記

grub.cfg ファイルを使用するシステムでは、デフォルトで、カーネルブートエントリーごとに **options** パラメーターがあり、これは **kernelopts** 変数に設定されます。この変数は、**/boot/grub2/grubenv** 設定ファイルで定義されます。

 **重要**

GRUB2 システムの場合:

- すべてのブートエントリーに対してカーネルコマンドラインパラメーターを変更する場合には、**grubby** ユーティリティーは `/boot/grub2/grubenv` ファイルの **kernelopts** 変数を更新します。
- 1つのブートエントリーのカーネルコマンドラインパラメーターが変更されると、**kernelopts** 変数の拡張やカーネルパラメーターの変更が行われ、得られた値は各ブートエントリーの `/boot/loader/entries/<RELEVANT_KERNEL_BOOT_ENTRY.conf>` ファイルに保存されます。

ziPL システムの場合:

- **grubby** は、個別のカーネルブートエントリーのカーネルコマンドラインパラメーターを変更して、`/boot/loader/entries/<ENTRY>.conf` ファイルに保存します。

関連情報

- [カーネルコマンドラインパラメーターの設定](#)
- **grubby(8)** および **zipl(8)** の man ページ
- [grubby ツール](#)

4.5. 起動時の一時的なカーネルコマンドラインパラメーターの変更

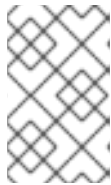
1回の起動プロセス中にのみカーネルパラメーターを変更することで、カーネルメニューエントリーを一時的に変更します。

 **注記**

この手順は単一ブートにのみ適用され、変更は永続的に行われません。

手順

1. GRUB 2 ブートメニューを起動します。
2. 起動するカーネルを選択します。
3. **e** キーを押してカーネルパラメーターを編集します。
4. カーソルを下に移動してカーネルコマンドラインを見つけます。カーネルコマンドラインは、64 ビット IBM Power シリーズおよび x86-64 BIOS ベースのシステムの場合は **linux** で始まり、UEFI システムの場合は **linuxefi** で始まります。
5. カーソルを行の最後に移動します。



注記

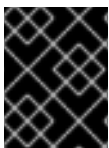
行の最初に移動するには **Ctrl+a** を押します。行の最後に移動するには **Ctrl+e** を押します。システムによっては、**Home** キーおよび **End** キーも機能する場合があります。

- 必要に応じてカーネルパラメーターを編集します。たとえば、緊急モードでシステムを実行するには、**linux** 行の最後に **emergency** パラメーターを追加します。

```
linux ($root)/vmlinuz-4.18.0-348.12.2.el8_5.x86_64 root=/dev/mapper/rhel-root ro
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap
rhgb quiet emergency
```

システムメッセージを有効にするには、**rhgb** および **quiet** パラメーターを削除します。

- Ctrl+x** を押して、選択したカーネルと変更したコマンドラインパラメーターで起動します。



重要

Esc キーを押してコマンドラインの編集を終了すると、ユーザーの加えた変更はすべて破棄されます。

4.6. シリアルコンソール接続を有効にする GRUB 設定

シリアルコンソールは、ネットワークがダウンしている場合にヘッドレスサーバーまたは埋め込みシステムに接続する際に便利です。あるいは、セキュリティールールを回避し、別のシステムへのログインアクセスを取得する必要がある場合などです。

シリアルコンソール接続を使用するように、デフォルトの GRUB 設定の一部を設定する必要があります。

前提条件

- root 権限がある。

手順

- /etc/default/grub** ファイルに以下の 2 つの行を追加します。

```
GRUB_TERMINAL="serial"
GRUB_SERIAL_COMMAND="serial --speed=9600 --unit=0 --word=8 --parity=no --stop=1"
```

最初の行は、グラフィカルターミナルを無効にします。**GRUB_TERMINAL** キーは、**GRUB_TERMINAL_INPUT** および **GRUB_TERMINAL_OUTPUT** の値を上書きします。

2 行目は、ボーレート (**--speed**)、パリティ、および他の値を使用中の環境とハードウェアに適合するように調整します。以下のログファイルのようなタスクには、115200 のように非常に高いボーレートが推奨されます。

- GRUB 設定ファイルを更新します。

- BIOS ベースのマシンの場合:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```


- UEFI ベースのマシンの場合:

```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

3. システムを再起動して、変更を有効にします。

第5章 ランタイム時のカーネルパラメーターの設定

システム管理者は、ランタイム時に Red Hat Enterprise Linux カーネルの動作を多数変更できます。 **sysctl** コマンドを使用し、 **/etc/sysctl.d/** および **/proc/sys/** ディレクトリー内の設定ファイルを変更して、実行時にカーネルパラメーターを設定します。



重要

プロダクションシステムでカーネルパラメーターを設定するには、慎重なプランニングが必要です。プランニングが欠如した変更では、カーネルが不安定になり、システムの再起動が必要とすることがあります。カーネル値を変更する前に、有効なオプションを使用していることを確認してください。

5.1. カーネルパラメーターとは

カーネルパラメーターは、システムの実行中に調整できる調整可能な値です。変更を有効にする場合でも、カーネルの再起動や再コンパイルは不要です。

以下を使用してカーネルパラメーターに対応できます。

- **sysctl** コマンド
- **/proc/sys/** ディレクトリーにマウントされている仮想ファイルシステム
- **/etc/sysctl.d/** ディレクトリー内の設定ファイル

調整可能パラメーターは、カーネルサブシステムでクラスに分割されます。Red Hat Enterprise Linux には、以下の調整可能なクラスがあります。

表5.1 sysctl クラスの表

調整パラメーターのクラス	サブシステム
abi	実行ドメインおよびパーソナリティー
crypto	暗号化インターフェイス
debug	カーネルのデバッグインターフェイス
dev	デバイス固有の情報
fs	グローバルおよび固有の調整可能なファイルシステム
kernel	グローバルなカーネルの設定項目
net	ネットワークの設定項目
sunrpc	Sun Remote Procedure Call (NFS)
user	ユーザー名前空間の制限

調整パラメーターのクラス	サブシステム
vm	メモリー、バッファ、およびキャッシュのチューニングと管理

関連情報

- [sysctl\(8\)](#) および [sysctl.d\(5\)](#) の man ページ

5.2. SYSCTL でカーネルパラメーターの一時的な設定

sysctl コマンドを使用して、実行時に一時的にカーネルパラメーターを設定します。このコマンドは、調整可能パラメーターのリスト表示およびフィルタリングにも便利です。

前提条件

- root 権限がある。

手順

1. すべてのパラメーターとその値をリストします。

```
# sysctl -a
```



注記

sysctl -a コマンドは、ランタイム時およびシステムの起動時に調整できるカーネルパラメーターを表示します。

2. パラメーターを一時的に設定するには、次のように入力します。

```
# sysctl <TUNABLE_CLASS>.<PARAMETER>=<TARGET_VALUE>
```

上記のサンプルコマンドは、システムの実行中にパラメーター値を変更します。この変更は、再起動なしですぐに適用されます。



注記

変更は、システムの再起動後にデフォルトに戻ります。

関連情報

- [sysctl\(8\)](#) の man ページ
- [sysctl](#) を使用したカーネルパラメーターの永続的な設定
- [/etc/sysctl.d/](#) の設定ファイルでカーネルパラメーターの調整

5.3. SYSCTL を使用したカーネルパラメーターの永続的な設定

sysctl コマンドを使用して、カーネルパラメーターを永続的に設定します。

前提条件

- root 権限がある。

手順

1. すべてのパラメーターをリストします。

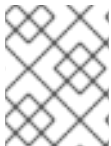
```
# sysctl -a
```

このコマンドは、ランタイム時に設定できるカーネルパラメーターをすべて表示します。

2. パラメーターを永続的に設定します。

```
# sysctl -w <TUNABLE_CLASS>.<PARAMETER>=<TARGET_VALUE> >> /etc/sysctl.conf
```

サンプルコマンドは、調整可能な値を変更して、`/etc/sysctl.conf` ファイルに書き込みます。これにより、カーネルパラメーターのデフォルト値が上書きされます。変更は、再起動なしで即座に永続的に反映されます。



注記

カーネルパラメーターを永続的に変更するには、`/etc/sysctl.d/` ディレクトリーの設定ファイルに手動で変更を行ってください。

関連情報

- [sysctl\(8\)](#) および [sysctl.conf\(5\)](#) の man ページ
- [/etc/sysctl.d/ の設定ファイルでカーネルパラメーターの調整](#)

5.4. /ETC/SYSCTL.D/ の設定ファイルでカーネルパラメーターの調整

`/etc/sysctl.d/` ディレクトリーの設定ファイルを手動で変更して、カーネルパラメーターを永続的に設定します。

前提条件

- root 権限がある。

手順

1. `/etc/sysctl.d/` に新しい設定ファイルを作成します。

```
# vim /etc/sysctl.d/<some_file.conf>
```

2. カーネルパラメーターを1行に1つずつ含めます。

```
<TUNABLE_CLASS>.<PARAMETER>=<TARGET_VALUE>
<TUNABLE_CLASS>.<PARAMETER>=<TARGET_VALUE>
```

3. 設定ファイルを作成します。

4. マシンを再起動して、変更を有効にします。
 - また、再起動せずに変更を適用するには、以下を実行します。

```
# sysctl -p /etc/sysctl.d/<some_file.conf>
```

このコマンドにより、以前に作成した設定ファイルから値を読み取ることができます。

関連情報

- [sysctl\(8\)](#)、[sysctl.d\(5\)](#) の man ページ

5.5. /PROC/SYS/ でカーネルパラメーターの一時的な設定

`/proc/sys/` 仮想ファイルシステムディレクトリー内のファイルを使用して、一時的にカーネルパラメーターを設定します。

前提条件

- root 権限がある。

手順

1. 設定するカーネルパラメーターを特定します。

```
# ls -l /proc/sys/<TUNABLE_CLASS>/
```

コマンドが返した書き込み可能なファイルは、カーネルの設定に使用できます。読み取り専用権限を持つユーザーは、現在の設定についてフィードバックを提供します。

2. カーネルパラメーターにターゲットの値を割り当てます。

```
# echo <TARGET_VALUE> > /proc/sys/<TUNABLE_CLASS>/<PARAMETER>
```

このコマンドでは、システムが再起動すると設定変更が消えます。

3. 必要に応じて、新しく設定した設定したカーネルパラメーターの値を確認します。

```
# cat /proc/sys/<TUNABLE_CLASS>/<PARAMETER>
```

関連情報

- [sysctl](#) を使用したカーネルパラメーターの永続的な設定
- [/etc/sysctl.d/](#) の設定ファイルでカーネルパラメーターの調整

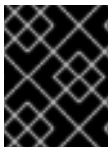
5.6. 関連情報

- [Tuning Red Hat Enterprise Linux for IBM DB2](#)

第6章 GRUB メニューを一時的に変更する

GRUB メニューエントリを変更したり、現在のブートにのみ適用される引数をカーネルに渡したりできます。ブートローダーメニューで選択したメニューエントリでは、次のことができます。

- **e** キーを押して、メニューエントリエディターインターフェイスを表示します。
- 変更を破棄し、**Esc** キーを押して標準メニューインターフェイスを再読み込みします。
- **c** キーを押してコマンドラインインターフェイスをロードします。
- 関連する GRUB コマンドを入力し、**Enter** キーを押して入力します。
- **Tab** キーを押して、コンテキストに基づいてコマンドを完了します。
- **Ctrl+a** キーの組み合わせを押して、行の先頭に移動します。
- **Ctrl+e** キーの組み合わせを押して、行末に移動します。



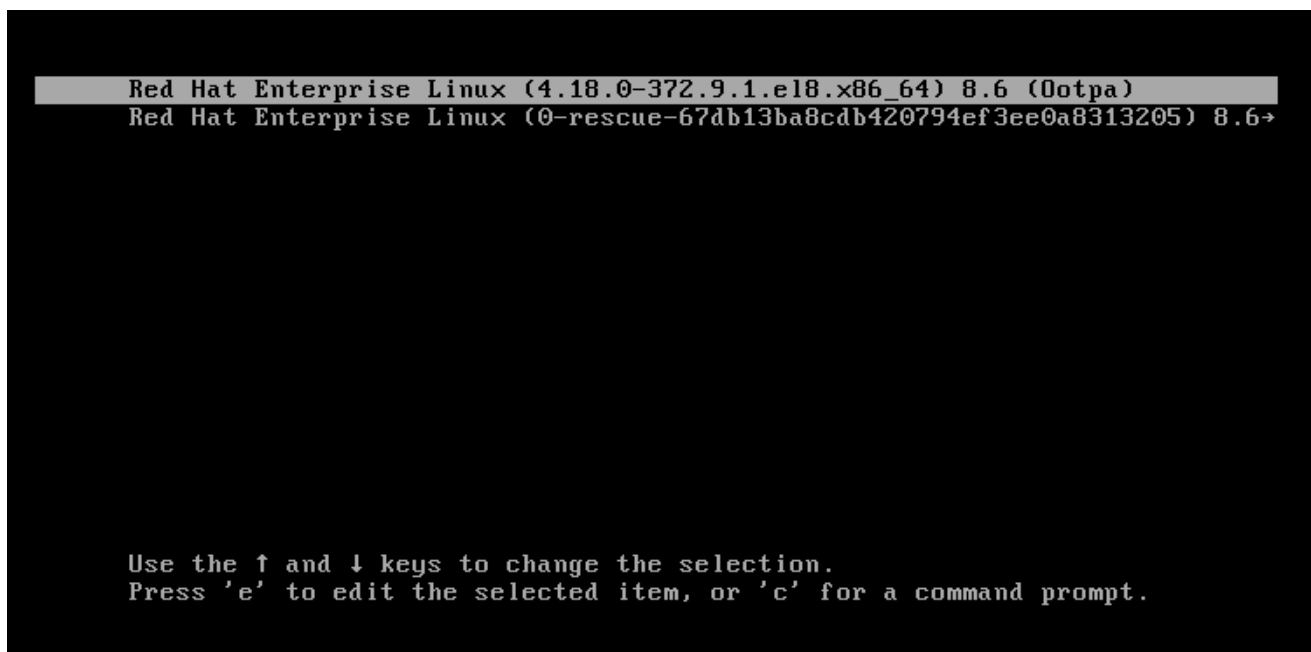
重要

次の手順では、1回の起動プロセス中に GRUB メニューを変更する方法について説明します。

6.1. GRUB について

GRUB は **GNU GRand Unified Bootloader** の略です。GRUB を使用すると、システムの起動時にロードするオペレーティングシステムまたはカーネルを選択できます。また、カーネルに引数を渡すこともできます。

GRUB で起動する場合は、メニューインターフェイスまたはコマンドラインインターフェイス (**GRUB command shell**) のいずれかを使用できます。システムを起動すると、メニューインターフェイスが表示されます。



c キーを押すと、コマンドラインインターフェイスに切り替えることができます。

```
Minimal BASH-like line editing is supported. For the first word,
TAB lists possible command completions. Anywhere else TAB lists
possible device or file completions. ESC at any time exits.
```

```
grub>
```

exit と入力して Enter キーを押すと、メニューインターフェイスに戻ることができます。

GRUB BLS ファイル

ブートローダーメニューエントリーは、ブートローダー仕様 (**BLS**) ファイルとして定義されます。このファイル形式は、ブートローダー設定ファイル进行操作することなく、ドロップインディレクトリー内の各ブートオプションのブートローダー設定を管理します。**grubby** ユーティリティは、これらの **BLS** ファイルを編集できます。

GRUB 設定ファイル

`/boot/grub2/grub.cfg` 設定ファイルは、メニューエントリーを定義しません。

関連情報

- [起動メニュー](#)
- [ブートローダー仕様の概要](#)

6.2. ブートローダー仕様の概要

BootLoader Specification (BLS) は、ブートローダー設定ファイル进行操作せずに、ドロップインディレクトリーの各起動オプションのブートローダー設定を管理するスキームとファイル形式を定義します。前述の方法とは異なり、各ブートエントリーはドロップインディレクトリーの個別の設定ファイルで表されるようになりました。ドロップインディレクトリーは、設定ファイルを編集または再生成せずに設定を拡張します。BLS は、ブートメニューエントリーのこの概念を拡張します。

BLS を使用すると、ディレクトリー内の個々のブートエントリーファイルを追加、削除、または編集して、ブートローダーメニューのオプションを管理できます。これにより、カーネルのインストールプロセスがはるかに簡素化され、異なるアーキテクチャーの間でこのプロセスの一貫性を保つことができます。

grubby ツールは、BLS に関するシンラッパースクリプトで、同じ **grubby** 引数およびオプションをサポートします。**dracut** を実行して、初期 ramdisk イメージを作成します。この設定では、コアブートローダーの設定ファイルは静的で、カーネルのインストール後に変更されません。

RHEL 8 では、同じブートローダーがすべてのアーキテクチャーで使用されているわけではないため、RHEL 8 には特に該当します。GRUB は、64 ビット ARM などのほとんどのアーキテクチャーに使用されますが、Open Power Abstraction Layer (OPAL) を使用する IBM Power Systems のリトルエンディアソバリエーションは **Petitboot** を使用し、IBM Z アーキテクチャーは **zipl** を使用します。

関連情報

- [「ブートエントリーについて」](#)
- [grubby\(8\) man ページ](#)

6.3. レスキューモードでの起動

レスキューモードは、便利なシングルユーザー環境を提供し、通常の起動プロセスを完了できない状況においてシステムの修復を可能にします。レスキューモードでは、システムはすべてのローカルファイルシステムをマウントし、いくつかの重要なシステムサービスを開始しようとします。ただし、ネットワークインターフェイスをアクティブにしたり、より多くのユーザーが同時にシステムにログインしたりすることはできません。

手順

1. GRUB ブート画面で、**e** キーを押して編集します。
2. **Linux** 行の末尾に次のパラメーターを追加します。

```
systemd.unit=rescue.target
```

```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-372.9.1.el8.x86_64 root=/dev/mapper/rhel-root ro \
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rh\
el/swap rhgb quiet zswap.enabled=0 systemd.unit=rescue.target
initrd ($root)/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd
```

3. **Ctrl+x** を押して、レスキューモードで起動します。

```
You are in rescue mode. After logging in, type "journalctl -xb" to view
system logs, "systemctl reboot" to reboot, "systemctl default" or "exit"
to boot into default mode.
Give root password for maintenance
(or press Control-D to continue):
```

6.4. 緊急モードでの起動

緊急モードは、可能な限り最小限の環境を提供し、システムがレスキューモードに入れない状態でもシステムの修復を可能にします。

緊急モードでは、システムは次のことを行います。

- **root** ファイルシステムを読み取り専用でマウントします
- いくつかの重要なサービスを開始します

ただし、システムは次のことを **行いません**。

- 他のローカルファイルシステムのマウントを試みる
- ネットワークインターフェイスをアクティブにする

手順

1. GRUB ブート画面で、**e** キーを押して編集します。
2. **Linux** 行の末尾に次のパラメーターを追加します。


```
systemd.unit=emergency.target
```

```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-372.9.1.el8.x86_64 root=/dev/mapper/rhel-root ro \
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rh\
el/swap rhgb quiet zswap.enabled=0 systemd.unit=emergency.target
initrd ($root)/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd
```

3. **Ctrl+x** を押して緊急モードで起動します。

```
You are in emergency mode. After logging in, type "journalctl -xb" to view
system logs, "systemctl reboot" to reboot, "systemctl default" or "exit"
to boot into default mode.
Give root password for maintenance
(or press Control-D to continue): _
```

6.5. デバッグシェルへのブート

systemd デバッグシェルは、起動プロセスの非常に早い段階でシェルを提供します。デバッグシェルに入ったら、**systemctl list-jobs** や **systemctl list-units** などの **systemctl** コマンドを使用して、**systemd** 関連の起動問題の原因を検索できます。

手順

1. GRUB ブート画面で、**e** キーを押して編集します。
2. **Linux** 行の末尾に次のパラメーターを追加します。

```
systemd.debug-shell
```

```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-372.9.1.el8.x86_64 root=/dev/mapper/rhel-root ro \
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rh\
el/swap rhgb quiet systemd.debug-shell
initrd ($root)/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd
```

3. 必要に応じて、**デバッグ オプション**を追加します。



注記

カーネルコマンドラインに **debug** オプションを追加すると、ログメッセージの数が増加します。**systemd** では、カーネルコマンドラインオプションの **debug** が、**systemd.log_level=debug** のショートカットになりました。

4. **Ctrl+x** を押して、デバッグシェルを起動します。



警告

デバッグシェルを使用するには認証が必要ないため、デバッグシェルを永続的に有効にすることにはセキュリティー上のリスクがあります。デバッグシェルは、デバッグセッションが終了したときに無効にしてください。

6.6. デバッグシェルへの接続

起動プロセス中に、**systemd-debug-generator** は TTY9 でデバッグシェルを設定します。

前提条件

- デバッグシェルが正常に起動しました。[デバッグシェルのブート](#) を参照してください。

手順

1. **Ctrl+Alt+F9** を押してデバッグシェルに接続します。
仮想マシンを使用している場合は、このキーの組み合わせを送信するには、仮想化アプリケーションからのサポートが必要です。たとえば、**Virtual Machine Manager** を使用している場合は、メニューから **Send Key** → **Ctrl+Alt+F9** を選択します。
2. デバッグシェルは認証を必要としないため、TTY9 で次のようなプロンプトが表示されます。

```
sh-4.4#
```

検証手順

- 以下のようなコマンドを入力します。

```
sh-4.4# systemctl status $$
```

```
sh-4.4# systemctl status $$
■ debug-shell.service - Early root shell on /dev/tty9 FOR DEBUGGING ONLY
   Loaded: loaded (/usr/lib/systemd/system/debug-shell.service; enabled-runtime; vendor preset: disabled)
   Active: active (running) since Tue 2022-08-02 08:40:09 EDT; 45s ago
     Docs: man:systemd-debug-generator(8)
   Main PID: 735 (sh)
    Tasks: 3 (limit: 17257)
   Memory: 2.3M
    CGroup: /system.slice/debug-shell.service
            └─ 735 /bin/sh
               └─ 2092 systemctl status 735
                  └─ 2093 less

sh-4.4#
```

- デフォルトのシェルに戻るには、ブートが成功した場合に **Ctrl+Alt+F1** を押します。

関連情報

- [systemd-debug-generator\(8\) man ページ](#)

6.7. インストールディスクを使用した ROOT パスワードのリセット

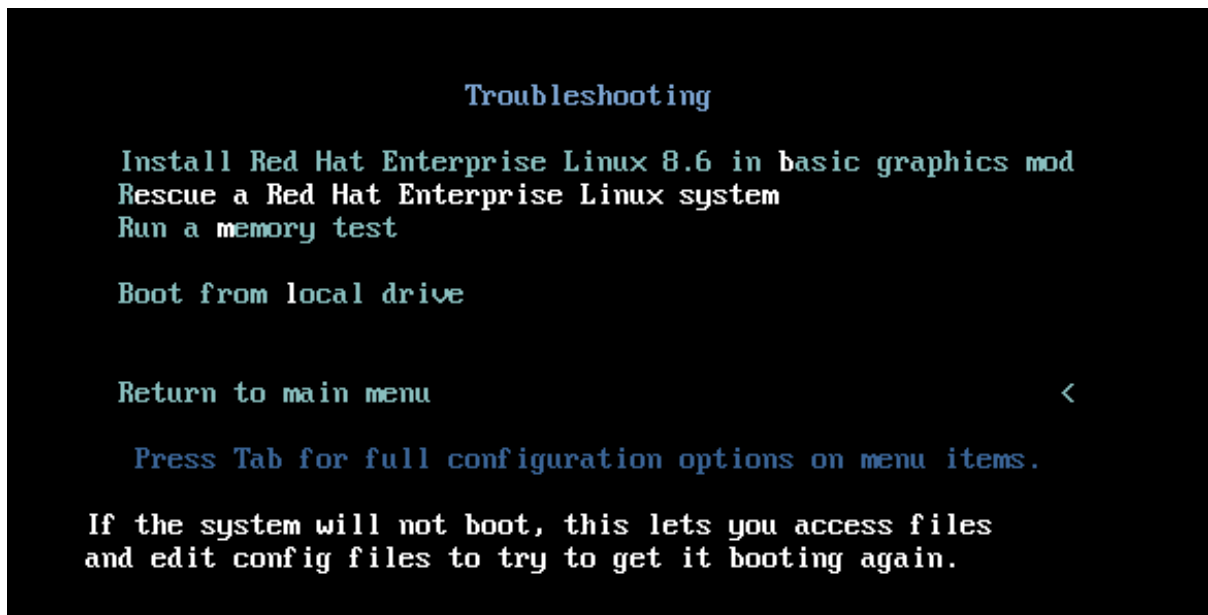
root パスワードを忘れた場合や紛失した場合は、リセットできます。

手順

1. インストールソースからホストを起動します。
2. インストールメディアのブートメニューで、**Troubleshooting** を選択します。



3. Troubleshooting メニューで **Rescue a Red Hat Enterprise Linux system** オプションを選択します。



4. Rescue メニューで **1** を選択し、**Enter** キーを押して続行します。

```

=====
Rescue

The rescue environment will now attempt to find your Linux installation and
mount it under the directory : /mnt/sysroot. You can then make any changes
required to your system. Choose '1' to proceed with this step.
You can choose to mount your file systems read-only instead of read-write by
choosing '2'.
If for some reason this process does not work choose '3' to skip directly to a
shell.

1) Continue
2) Read-only mount
3) Skip to shell
4) Quit (Reboot)

Please make a selection from the above: 1_

```

5. 以下のようにファイルシステム **root** を変更します。

```
sh-4.4# chroot /mnt/sysimage
```

```

When finished, please exit from the shell and your system will reboot.
Please press ENTER to get a shell:
sh-4.4# chroot /mnt/sysimage
bash-4.4#

[anaconda11:main* 2:shell 3:log 4:storage-log 5:program-log

```

6. **passwd** コマンドを入力し、コマンドラインに表示される指示にしたがって **root** パスワードを変更します。

```

bash-4.4# passwd
Changing password for user root.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
bash-4.4#

[anaconda11:main* 2:shell 3:log 4:storage-log 5:program-log

```

7. 時間がかかるディスクの SELinux の再ラベルを防ぐために、**autorelabel** ファイルを削除します。

```
sh-4.4# rm -f /.autorelabel
```

8. **exit** コマンドを入力して、**chroot** 環境を終了します。

9. **exit** コマンドを再び実行して初期化を再開し、システム起動を完了します。

6.8. RD.BREAK を使用した ROOT パスワードのリセット

root パスワードを忘れた場合や紛失した場合は、リセットできます。

手順

1. システムを起動し、GRUB ブート画面で **e** キーを押して編集を行います。
2. **Linux** 行の末尾に **rd.break** パラメーターを追加します。

```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-372.9.1.el8.x86_64 root=/dev/mapper/rhel-root ro \
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rh\
el/swap rhgb quiet rd.break
initrd ($root)/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd
```

3. **Ctrl+x** を押して変更したパラメーターでシステムを起動します。

```
Generating "/run/initramfs/rdsosreport.txt"

Entering emergency mode. Exit the shell to continue.
Type "journalctl" to view system logs.
You might want to save "/run/initramfs/rdsosreport.txt" to a USB stick or /boot
after mounting them and attach it to a bug report.

switch_root:/# _
```

4. ファイルシステムを書き込み可能で再マウントします。

```
switch_root:/# mount -o remount,rw /sysroot
```

5. ファイルシステムの **root** を変更します。

```
switch_root:/# chroot /sysroot
```

6. **passwd** コマンドを入力し、コマンドラインに表示される指示に従います。

```
bash-4.4# passwd
Changing password for user root.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
bash-4.4#
```

```
[anaconda]1:main* 2:shell 3:log 4:storage-log 5:program-log
```

7. 次のシステム起動時にすべてのファイルにラベルを付け直します。

```
sh-4.4# touch /.autorelabel
```

8. ファイルシステムを **読み取り専用** で再マウントします。

```
sh-4.4# mount -o remount,ro /
```

9. **exit** コマンドを入力して、**chroot** 環境を終了します。
10. **exit** コマンドを再び実行して初期化を再開し、システム起動を完了します。



注記

SELinux の再ラベル付けプロセスには時間がかかる場合があります。プロセスが完了すると、システムが自動的に再起動します。

ヒント

enforcing=0 オプションを追加すると、時間のかかる SELinux の再ラベル付けプロセスを省略できます。

手順

1. **linux** 行の最後に **rd.break** パラメーターを追加するときは、**enforcing=0** も追加します。

```
rd.break enforcing=0
```

2. **/etc/shadow** ファイルの SELinux セキュリティーコンテキストを復元します。

```
# restorecon /etc/shadow
```

3. SELinux ポリシーの適用をオンに戻し、オンになっていることを確認します。

```
# setenforce 1
# getenforce
Enforcing
```

手順 3 で **enforcing=0** オプションを追加した場合は、手順 8 で **touch /.autorelabel** コマンドの入力を省略できることに注意してください。

6.9. 関連情報

- **/usr/share/doc/grub2-common** ディレクトリー。
- **info grub2** コマンド。

第7章 GRUB ブートローダーに永続的な変更を加える

grubby ツールを使用して、GRUB で永続的な変更を行います。

7.1. 前提条件

- システムに RHEL が正常にインストールされました。
- root 権限がある。

7.2. デフォルトのカーネルの一覧表示

デフォルトのカーネルを一覧表示すると、デフォルトのカーネルのファイル名およびインデックス番号を見つけて、GRUB ブートローダーに永続的な変更を加えることができます。

手順

- デフォルトのカーネルのファイル名を確認するには、次のように入力します。

```
# grubby --default-kernel
/boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
```

- デフォルトのカーネルのインデックス番号を調べるには、次のように入力します。

```
# grubby --default-index
0
```

7.3. カーネルの GRUB メニューエントリーの表示

すべてのカーネルメニューエントリーを一覧表示するか、特定のカーネルの GRUB メニューエントリーを表示できます。

手順

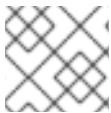
- すべてのカーネルメニューエントリーを一覧表示するには、次のように入力します。

```
# grubby --info=ALL
index=0
kernel="/boot/vmlinuz-4.18.0-372.9.1.el8.x86_64"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet $tuned_params zswap.enabled=1"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd"
title="Red Hat Enterprise Linux (4.18.0-372.9.1.el8.x86_64) 8.6 (Ootpa)"
id="67db13ba8cdb420794ef3ee0a8313205-4.18.0-372.9.1.el8.x86_64"
index=1
kernel="/boot/vmlinuz-0-rescue-67db13ba8cdb420794ef3ee0a8313205"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-0-rescue-67db13ba8cdb420794ef3ee0a8313205.img"
```

```
title="Red Hat Enterprise Linux (0-rescue-67db13ba8cdb420794ef3ee0a8313205) 8.6
(Ootpa)"
id="67db13ba8cdb420794ef3ee0a8313205-0-rescue"
```

- 特定のカーネルの GRUB メニューエントリを表示するには、次のように入力します。

```
# grubby --info /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
grubby --info /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
index=0
kernel="/boot/vmlinuz-4.18.0-372.9.1.el8.x86_64"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet $tuned_params zswap.enabled=1"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd"
title="Red Hat Enterprise Linux (4.18.0-372.9.1.el8.x86_64) 8.6 (Ootpa)"
id="67db13ba8cdb420794ef3ee0a8313205-4.18.0-372.9.1.el8.x86_64"
```



注記

タブ補完を試行して **/boot** ディレクトリー内の利用可能なカーネルを確認します。

7.4. カーネル引数の編集

既存のカーネル引数の値を変更できます。たとえば、仮想コンソール (画面) のフォントとサイズを変更できます。

手順

- 仮想コンソールのフォントを、サイズが **32** の **latarcyrheb-sun** に変更します。

```
# grubby --args=vconsole.font=latarcyrheb-sun32 --update-kernel /boot/vmlinuz-
4.18.0-372.9.1.el8.x86_64
```

7.5. GRUB メニューエントリに対する引数の追加および削除

GRUB メニューから引数を追加、削除、または同時に追加および削除できます。

手順

- GRUB メニューエントリに引数を追加するには、**--update-kernel** オプションを **--args** と組み合わせて使用します。たとえば、次のコマンドはシリアルコンソールを追加します。

```
# grubby --args=console=ttyS0,115200 --update-kernel /boot/vmlinuz-4.18.0-
372.9.1.el8.x86_64
```

コンソールの引数は行末に付けられ、新しいコンソールは他の設定済みコンソールよりも優先されます。

- GRUB メニューエントリから引数を削除するには、**--update-kernel** オプションを **--remove-args** と組み合わせて使用します。以下に例を示します。

```
# grubby --remove-args="rhgb quiet" --update-kernel /boot/vmlinuz-4.18.0-
372.9.1.el8.x86_64
```


■

このコマンドは、Red Hat グラフィカルブート引数を削除し、詳細モードであるログメッセージを有効にします。

- 引数を同時に追加および削除するには、次のように入力します。

```
# grubby --remove-args="rhgb quiet" --args=console=ttyS0,115200 --update-kernel
/boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
```

検証手順

- 行った永続的な変更を確認するには、次のように入力します。

```
# grubby --info /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
index=0
kernel="/boot/vmlinuz-4.18.0-372.9.1.el8.x86_64"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap $tuned_params zswap.enabled=1 console=ttyS0,115200"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-4.18.0-372.9.1.el8.x86_64.img $tuned_initrd"
title="Red Hat Enterprise Linux (4.18.0-372.9.1.el8.x86_64) 8.6 (Ootpa)"
id="67db13ba8cdb420794ef3ee0a8313205-4.18.0-372.9.1.el8.x86_64"
```

7.6. 新しいブートエントリーの追加

ブートローダーメニューエントリーに新しいブートエントリーを追加できます。

手順

1. すべてのカーネル引数をデフォルトカーネルからこの新しいカーネルエントリーにコピーします。

```
# grubby --add-kernel=new_kernel --title="entry_title" --initrd="new_initrd" --copy-
default
```

2. 利用可能なブートエントリーの一覧を取得します。

```
# ls -l /boot/loader/entries/*
-rw-r--r--. 1 root root 408 May 27 06:18
/boot/loader/entries/67db13ba8cdb420794ef3ee0a8313205-0-rescue.conf
-rw-r--r--. 1 root root 536 Jun 30 07:53
/boot/loader/entries/67db13ba8cdb420794ef3ee0a8313205-4.18.0-372.9.1.el8.x86_64.conf
-rw-r--r-- 1 root root 336 Aug 15 15:12
/boot/loader/entries/d88fa2c7ff574ae782ec8c4288de4e85-4.18.0-193.el8.x86_64.conf
```

3. 新たなブートエントリーを作成します。たとえば、4.18.0-193.el8.x86_64 カーネルの場合は、次のようにコマンドを発行します。

```
# grubby --grub2 --add-kernel=/boot/vmlinuz-4.18.0-193.el8.x86_64 --title="Red Hat
Enterprise 8 Test" --initrd=/boot/initramfs-4.18.0-193.el8.x86_64.img --copy-default
```

検証

- 新しく追加されたブートエントリーが、使用可能なブートエントリーの一覧に表示されていることを確認します。

```
# ls -l /boot/loader/entries/*
-rw-r--r--. 1 root root 408 May 27 06:18
/boot/loader/entries/67db13ba8cdb420794ef3ee0a8313205-0-rescue.conf
-rw-r--r--. 1 root root 536 Jun 30 07:53
/boot/loader/entries/67db13ba8cdb420794ef3ee0a8313205-4.18.0-372.9.1.el8.x86_64.conf
-rw-r--r-- 1 root root 287 Aug 16 15:17
/boot/loader/entries/d88fa2c7ff574ae782ec8c4288de4e85-4.18.0-
193.el8.x86_64.0~custom.conf
-rw-r--r-- 1 root root 287 Aug 16 15:29
/boot/loader/entries/d88fa2c7ff574ae782ec8c4288de4e85-4.18.0-193.el8.x86_64.conf
```

7.7. GRUBBY でデフォルトのブートエントリーを変更する

grubby ツールを使用すると、デフォルトのブートエントリーを変更できます。

手順

- デフォルトのカーネルとして指定されたカーネルに永続的な変更を加えるには、次のように入力します。

```
# grubby --set-default /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
The default is /boot/loader/entries/67db13ba8cdb420794ef3ee0a8313205-4.18.0-
372.9.1.el8.x86_64.conf with index 0 and kernel /boot/vmlinuz-4.18.0-372.9.1.el8.x86_64
```

7.8. 同じ引数を使用したすべてのカーネルメニューの更新

すべてのカーネルメニューエントリーに同じカーネルブート引数を追加できます。

手順

- すべてのカーネルメニューエントリーに同じカーネルブート引数を追加するには、**--update-kernel=ALL** パラメーターをアタッチします。たとえば、次のコマンドはシリアルコンソールをすべてのカーネルに追加します。

```
# grubby --update-kernel=ALL --args=console=ttyS0,115200
```



注記

--update-kernel パラメーターには、**DEFAULT** またはカーネルインデックス番号のコンマ区切りリストも指定できます。

7.9. 現在および将来のカーネルのデフォルトカーネルオプションの変更

kernelopts 変数を使用すると、現在および将来のカーネルのデフォルトカーネルオプションを変更できます。

手順

1. **kernelopts** 変数を使用してカーネルパラメーターをリスト表示します。

```
# grub2-editenv - list | grep kernelopts
```

```
kernelopts=root=/dev/mapper/rhel-root ro crashkernel=auto resume=/dev/mapper/rhel-swap
rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet
```

- カーネルのコマンドラインパラメーターを変更します。パラメーターを追加、削除、または変更できます。たとえば、**debug** パラメーターを追加するには、次のように入力します。

```
# grub2-editenv - set "$(grub2-editenv - list | grep kernelopts) <debug>"
```

- オプション:** 新しく追加したパラメーターが **kernelopts** に追加されていることを確認します。

```
# grub2-editenv - list | grep kernelopts
```

```
kernelopts=root=/dev/mapper/rhel-root ro crashkernel=auto resume=/dev/mapper/rhel-swap
rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet debug
```

- システムを再起動して、変更を有効にします。



注記

代わりに、**grubby** コマンドを使用して、現在および将来のカーネルに引数を渡すこともできます。

```
# grubby --update-kernel ALL --args="<PARAMETER>"
```

7.10. 関連情報

- `/usr/share/doc/grub2-common` ディレクトリー。
- `info grub2` コマンド。

第8章 カスタマイズされたブートメニューの構築

特定のエントリーを含むブートメニューを作成したり、エントリーの順序を変更したりできます。このようなタスクには、GRUB、**grubby**、および Boot Loader Specification (**BLS**) ファイルを使用できます。

次のセクションでは、GRUB と **grubby** を使用してブートメニューの基本的なカスタマイズを行う方法について説明します。

8.1. GRUB 設定ファイル

BIOS ベースのマシンでは **/boot/grub2/grub.cfg** であり、UEFI ベースのマシンでは **/boot/efi/EFI/redhat/grub.cfg** であるブートローダー設定ファイルについて学びます。

GRUB スクリプトはユーザーのコンピューターを検索して、スクリプトが見つけたオペレーティングシステムに基づくブートメニューを構築します。最新のシステムブートオプションを反映させるために、カーネルが更新されるか新規カーネルが追加されると、ブートメニューは自動的に再構築されます。

GRUB は、一連のスクリプトを使用してメニューを構築します。これらは **/etc/grub.d/** ディレクトリーにあります。このスクリプトは **/etc/grub.d/** ディレクトリーに格納されており、以下のファイルが含まれます。

- **00_header**: **/etc/default/grub** ファイルから GRUB 設定を読み込みます。
- **01_users**: **user.cfg** ファイルから root パスワードを読み取ります。
- **10_linux**: Red Hat Enterprise Linux のデフォルトのパーティションでカーネルを見つけます。
- **30_os-prober**: 別のパーティションで見つかったオペレーティングシステム用にエントリーを構築します。
- **40_custom**: 追加のメニューエントリー作成に使用可能なテンプレートです。

GRUB はスクリプトを **/etc/grub.d/** ディレクトリーからアルファベット順に読み取るため、スクリプトの名前を変更して、特定のメニューエントリーの起動順序を変更できます。

8.2. 起動可能なカーネルのリストを非表示にする

システムの起動時に GRUB が起動可能なカーネルのリストを表示しないようにすることができます。

手順

1. **/etc/default/grub** ファイルで **GRUB_TIMEOUT_STYLE** オプションを次のように設定します。

```
GRUB_TIMEOUT_STYLE=hidden
```

2. 変更を有効にするために **grub.cfg** ファイルを再ビルドします。

- BIOS ベースのマシンで、次のように入力します。

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

- UEFI ベースのマシンでは、次のように入力します。

```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

3. 起動時に **Esc** キーを押すと、起動可能なカーネルのリストが表示されます。



重要

`/etc/default/grub` ファイルで **GRUB_TIMEOUT** を **0** に設定して、起動可能なカーネルのリストを非表示にしないでください。このように設定すると、システムが常にデフォルトのメニューエントリーで直ちに起動し、デフォルトのカーネルが起動に失敗した場合に、以前のカーネルを起動することができなくなります。

8.3. GRUB 設定ファイルを使用してデフォルトのブートエントリーを変更する

デフォルトのカーネルパッケージタイプを指定して、デフォルトのブートエントリーを変更できます。

手順

1. **grub2-set-default** コマンドにインデックスを渡して、デフォルトでロードするオペレーティングシステムまたはカーネルを指定します。以下に例を示します。

```
# grubby --set-default-index=1
The default is /boot/loader/entries/d5151aa93c444ac89e78347a1504d6c6-4.18.0-348.el8.x86_64.conf with index 1 and kernel /boot/vmlinuz-4.18.0-348.el8.x86_64
```

GRUB は、`/boot/grub2/grubenv` の **saved_entry** ディレクティブのキーとして数値を使用して、オペレーティングシステムがロードされるデフォルトの順序を変更することをサポートしています。



注記

インデックスカウントはゼロから始まります。したがって、前の例では、GRUB は 2 番目のエントリーをロードします。次にインストールされたカーネルで、インデックス値は上書きされます。



注記

grubby を使用して、カーネルのインデックスを見つけることもできます。詳細は、[カーネルの GRUB メニューエントリーの表示](#) を参照してください。

2. **オプション**: システムが常に特定のメニューエントリーを使用するように強制します。
 - a. 利用可能なメニューエントリーを一覧表示します。

```
# grubby --info=ALL
```

- b. `/etc/default/grub` ファイルの **GRUB_DEFAULT** ディレクティブのキーとして、メニューエントリー名またはリスト内のメニューエントリーの位置番号を使用します。以下に例を示します。

```
GRUB_DEFAULT=example-gnu-linux
```

3. 変更を有効にするために **grub.cfg** ファイルを再ビルドします。

- BIOS ベースのマシンで、次のように入力します。

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

- UEFI ベースのマシンでは、次のように入力します。

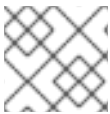
```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

第9章 GRUB の再インストール

GRUB ブートローダーを再インストールすると、GRUB の誤ったインストール、ファイルの欠落、またはシステムの破損によってよく発生する一部の問題を修正できます。これらの問題は、不足しているファイルを復元し、ブート情報を更新することで解決できます。

GRUB を再インストールする理由:

- GRUB ブートローダーパッケージをアップグレードする。
- 別のドライブにブート情報を追加する。
- インストール済みのオペレーティングシステムを制御するために、ユーザーが GRUB ブートローダーを必要としている。ただし、一部のオペレーティングシステムには独自のブートローダーがインストールされており、GRUB を再インストールすると、目的のオペレーティングシステムに制御権限が戻されます。

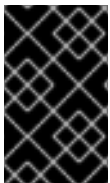


注記

GRUB は、ファイルが破損していない場合にのみファイルを復元します。

9.1. BIOS ベースマシンへの GRUB の再インストール

BIOS ベースのシステムに GRUB ブートローダーを再インストールできます。GRUB パッケージを更新した後は必ず GRUB を再インストールしてください。



重要

これにより、既存の GRUB が上書きされ、新しい GRUB がインストールされます。インストール中にシステムでデータの破損やブートクラッシュが発生しないようにしてください。

手順

1. GRUB がインストールされているデバイスに GRUB を再インストールします。たとえば、**sda** がデバイスの場合は、以下のようになります。

```
# grub2-install /dev/sda
```

2. システムを再起動して、変更を有効にします。

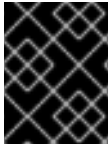
```
# reboot
```

関連情報

- `grub-install(1)` man ページ

9.2. UEFI ベースマシンへの GRUB の再インストール

UEFI ベースのシステムに GRUB ブートローダーを再インストールできます。



重要

インストール中にシステムでデータの破損やブートクラッシュが発生しないようにしてください。

手順

1. **grub2-efi** および **shim** ブートローダーファイルを再インストールします。

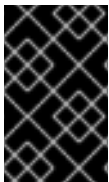
```
# yum reinstall grub2-efi shim
```

2. システムを再起動して、変更を有効にします。

```
# reboot
```

9.3. IBM POWER マシンへの GRUB の再インストール

IBM Power システムの Power PC Reference Platform (PReP) ブートパーティションに GRUB ブートローダーを再インストールできます。GRUB パッケージを更新した後は必ず GRUB を再インストールしてください。



重要

これにより、既存の GRUB が上書きされ、新しい GRUB がインストールされます。インストール中にシステムでデータの破損やブートクラッシュが発生しないようにしてください。

手順

1. GRUB が格納されているディスクパーティションを特定します。

```
# bootlist -m normal -o  
sda1
```

2. ディスクパーティションに GRUB を再インストールします。

```
# grub2-install partition
```

partition は、前のステップで特定した GRUB パーティション (**/dev/sda1** など) に置き換えます。

3. システムを再起動して、変更を有効にします。

```
# reboot
```

関連情報

- **grub-install(1)** man ページ

9.4. GRUB のリセット

GRUB をリセットすると、すべての GRUB 設定ファイルとシステム設定が完全に削除され、ブートローダーが再インストールされます。すべての構成設定をデフォルト値にリセットして、破損したファイルや不適切な設定によって引き起こされた障害を修正できます。



重要

次の手順では、ユーザーが行ったすべてのカスタマイズを削除します。

手順

1. 設定ファイルを削除します。

```
# rm /etc/grub.d/*  
# rm /etc/sysconfig/grub
```

2. パッケージを再インストールします。

- BIOS ベースのマシンで、次のように入力します。

```
# yum reinstall grub2-tools
```

- UEFI ベースのマシンでは、次のように入力します。

```
# yum reinstall grub2-efi shim grub2-tools
```

3. 変更を有効にするために **grub.cfg** ファイルを再ビルドします。

- BIOS ベースのマシンで、次のように入力します。

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

- UEFI ベースのマシンでは、次のように入力します。

```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

4. [GRUB の再インストール](#) 手順に従って、**/boot/**パーティションに GRUB を復元します。

第10章 GRUB をパスワードで保護する

GRUB をパスワードで保護するには、次の2つの方法があります。

- メニューエントリーの修正にはパスワードが必要ですが、既存のメニューエントリーの起動には必要ありません。
- メニューエントリーを変更したり、既存のメニューエントリーを起動するには、パスワードが必要です。

10.1. メニューエントリーを変更するためだけにパスワード保護を設定する

GRUB メニューエントリーを変更するためのパスワード認証をサポートするように GRUB を設定できます。この手順では、ハッシュ形式のパスワードを含む `/boot/grub2/user.cfg` ファイルを作成します。



重要

`grub2-setpassword` コマンドを使用してパスワードを設定すると、権限のない修正からメニューエントリーは保護されますが、権限のない起動からは保護されません。

手順

1. root で `grub2-setpassword` コマンドを実行します。

```
# grub2-setpassword
```

2. ユーザーのパスワードを入力し、**Enter** キーを押してパスワードを確認します。

```
Enter password:
Confirm the password:
```



注記

root ユーザーは、`/boot/grub2/grub.cfg` ファイルで定義され、パスワードが変更されません。したがって、ブート中にブートエントリーを変更するには、root ユーザー名とパスワードが必要です。

10.2. メニューエントリーの変更および起動のためのパスワード保護の設定

GRUB を設定して、メニューエントリーが不正に変更されたり、不正に起動されたりするのを防ぐことができます。



警告

GRUB パスワードを忘れると、再設定したエントリーを起動できなくなります。

手順

1. `/boot/loader/entries/` ディレクトリーから、変更するブートエントリーのブートローダー仕様 (**BLS**) ファイルを開きます。
2. `grub_users` で始まる行を見つけます。このパラメーターは追加の引数を `menuentry` に渡します。
3. `grub_users` 属性を、スーパーユーザー以外にエントリーを起動できるユーザー名に設定します。デフォルトでは、このユーザーは `root` です。サンプル設定ファイルは次のとおりです。

```
title Red Hat Enterprise Linux (4.18.0-221.el8.x86_64) 8.3
(Ootpa)
version 4.18.0-221.el8.x86_64
linux /vmlinuz-4.18.0-221.el8.x86_64
initrd /initramfs-4.18.0-221.el8.x86_64.img $tuned_initrd
options $kernelopts $tuned_params
id rhel-20200625210904-4.18.0-221.el8.x86_64
grub_users root
grub_arg --unrestricted
grub_class kernel
```

4. **BLS** ファイルを保存して閉じます。



注記

すべてのメニューエントリーを起動から保護したい場合は、`grub_users` 属性を直接設定できます。たとえば、`root` がユーザーの場合は、以下を行います。

```
# grub2-editenv - set grub_users="root"
```

第11章 仮想化環境でカーネルパニックのパラメーターを無効のままにする

RHEL 8 に仮想マシンを設定する場合は、仮想マシンで偽ソフトロックアップが発生する可能性があるため、カーネルパラメーター `softlockup_panic` および `nmi_watchdog` を有効にしないでください。また、カーネルパニックは必要ありません。

以下のセクションで、このアドバイスの背後にある理由を見つけてください。

11.1. ソフトロックアップとは

ソフトロックアップは、通常、タスクが再スケジュールされずに CPU のカーネル領域で実行しているときにバグによって生じる状況です。また、このタスクは、その他のタスクがその特定の CPU で実行することを許可しません。これにより、警告が、システムコンソールを介してユーザーに表示されます。この問題は、ソフトロックアップの発生 (fire) とも呼ばれます。

関連情報

- [CPU ソフトロックアップとは。](#)

11.2. カーネルパニックを制御するパラメーター

ソフトロックの検出時にシステムの動作を制御する、以下のカーネルパラメーターを設定できます。

`softlockup_panic`

ソフトロックアップが検出されたときにカーネルでパニックを発生させるどうかを制御します。

タイプ	値	Effect
整数	0	カーネルが、ソフトロックアップでパニックにならない
整数	1	カーネルが、ソフトロックアップでパニックになる

RHEL 8 では、この値のデフォルトは 0 です。

システムでパニックを発生させるには、その前にハードロックアップを検出する必要があります。検出は、`nmi_watchdog` パラメーターで制御されます。

`nmi_watchdog`

ロックアップ検出メカニズム (`watchdogs`) がアクティブかどうかを制御します。このパラメーターは整数型です。

値	Effect
0	ロックアップ検出を無効にする
1	ロックアップ検出を有効にする

ハードロックアップ検出は、各 CPU で割り込みに応答する機能を監視します。

watchdog_thresh

ウォッチドッグの `hrtimer`、NMI イベント、およびソフトロックアップまたはハードロックアップのしきい値を制御します。

デフォルトのしきい値	ソフトロックアップのしきい値
10 秒	$2 * \text{watchdog_thresh}$

このパラメーターをゼロに設定すると、ロックアップ検出を無効にします。

関連情報

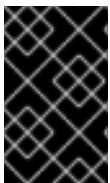
- [Softlockup detector and hardlockup detector](#)
- [カーネル sysctl](#)

11.3. 仮想化環境で誤ったソフトロックアップ

ソフトロックアップとは で説明されている物理ホストでのソフトロックアップの発生は、通常、カーネルまたはハードウェアのバグを示しています。仮想化環境のゲストオペレーティングシステムで同じ現象が発生すると、誤った警告が表示される場合があります。

ホストの負荷が重い場合や、メモリーなどの特定リソースに対する競合が多い場合は、通常、ソフトロックアップが誤って発生します。これは、ホストが 20 秒よりも長い期間、ゲストの CPU をスケジュールすることが原因となる場合があります。その後、再度、ホストで実行するようにゲスト CPU がスケジュールされると、タイマーにより発生する **時間ジャンプ** が発生します。タイマーには、ウォッチドッグの `hrtimer` も含まれます。これにより、ゲスト CPU のソフトロックアップを報告できません。

仮想化環境でのソフトロックアップは誤りである可能性があるため、ゲスト CPU でソフトロックアップが報告されたときにシステムパニックを発生させるカーネルパラメーターは有効にしないでください。



重要

ゲストのソフトロックアップについて理解するには、ホストがゲストをタスクとしてスケジュールしてから、ゲストが独自のタスクをスケジュールしていることを理解することが重要になります。

関連情報

- [ソフトロックアップとは](#)
- [仮想マシンコンポーネントおよびその相互作用](#)
- [仮想マシンが BUG: soft lockup を報告する](#)

第12章 データベースサーバーのカーネルパラメーターの調整

特定のデータベースアプリケーションのパフォーマンスに影響を与える可能性のあるカーネルパラメーターのセットには、様々なものがあります。データベースサーバーとデータベースの効率的な運用を確保するには、それぞれのカーネルパラメーターを適切に設定します。

12.1. データベースサーバーの概要

データベースサーバーは、データベース管理システム (DBMS) の機能を提供するサービスです。DBMS は、データベース管理のためのユーティリティを提供し、エンドユーザー、アプリケーション、およびデータベースと対話します。

Red Hat Enterprise Linux 8 は、以下のデータベース管理システムを提供します。

- MariaDB 10.3
- MariaDB 10.5 - RHEL 8.4 以降で利用できます。
- MariaDB 10.11 - RHEL 9.4 以降で利用可能
- MySQL 8.0
- PostgreSQL 10
- PostgreSQL 9.6
- PostgreSQL 12 - RHEL 8.1.1 以降で利用できます。
- PostgreSQL 13 - RHEL 8.4 以降で利用できます。
- PostgreSQL 15 - RHEL 8.8 以降で利用できます。
- PostgreSQL 16 - RHEL 9.4 以降で利用可能

12.2. データベースアプリケーションのパフォーマンスに影響するパラメーター

次のカーネルパラメーターは、データベースアプリケーションのパフォーマンスに影響します。

fs.aio-max-nr

サーバー上でシステムが処理できる非同期 I/O 操作の最大数を定義します。



注記

fs.aio-max-nr パラメーターを増やしても、aio の制限以上を追加することはありません。

fs.file-max

システムがインスタンスで対応するファイルハンドル (一時ファイル名または開いているファイルに割り当てられた ID) の最大数を定義します。

カーネルは、アプリケーションからファイルハンドルが要求されるたびに、ファイルハンドルを動的に割り当てます。ただし、カーネルは、そのファイルハンドルがアプリケーションによって解放されたときに解放しません。代わりに、カーネルはこれらのファイルハンドルをリサイクルしま

す。これは、現在使用しているファイルハンドルの数が少なくても、時間の経過とともに割り当てられたファイルハンドルの合計が増加することを意味します。

kernel.shmall

システム全体で使用できる共有メモリーページの合計を定義します。メインメモリー全体を使用するには、**kernel.shmall** パラメーターの値が、メインメモリーの合計サイズ以下である必要があります。

kernel.shmmax

Linux プロセスが仮想アドレス空間に割り当てることができる1つの共有メモリーセグメントの最大サイズをバイト単位で定義します。

kernel.shmmni

データベースサーバーが処理できる共有メモリーセグメントの最大数を定義します。

net.ipv4.ip_local_port_range

特定のポート番号なしでデータベースサーバーに接続するプログラムにシステムが使用できるポート範囲を定義します。

net.core.rmem_default

TCP (Transmission Control Protocol) を介してデフォルトの受信ソケットメモリーを定義します。

net.core.rmem_max

TCP (Transmission Control Protocol) による最大受信ソケットメモリーを定義します。

net.core.wmem_default

TCP (Transmission Control Protocol) によるデフォルトの送信ソケットメモリーを定義します。

net.core.wmem_max

TCP (Transmission Control Protocol) による最大送信ソケットメモリーを定義します。

vm.dirty_bytes / vm.dirty_ratio

ダーティーデータを生成するプロセスが **write()** 関数で開始するダーティー可能メモリーの割合 (バイト単位) でしきい値を定義します。

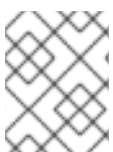


注記

一度に指定できるのは、**vm.dirty_bytes** または **vm.dirty_ratio** のいずれかです。

vm.dirty_background_bytes / vm.dirty_background_ratio

カーネルがダーティーデータをハードディスクにアクティブに書き込もうとする、ダーティー可能なメモリーの割合 (バイト単位) でしきい値を定義します。



注記

一度に指定できるのは、**vm.dirty_background_bytes** または **vm.dirty_background_ratio** のいずれかです。

vm.dirty_writeback_centisecs

ハードディスクへのダーティーデータの書き込みを行うカーネルスレッドの起動を定期的に行う間隔を定義します。

このカーネルパラメーターは、100 分の 1 秒単位で測定されます。

vm.dirty_expire_centisecs

ダーティーデータがハードディスクに書き込まれるまでの時間を定義します。

このカーネルパラメーターは、100 分の 1 秒単位で測定されます。

関連情報

- [Dirty pagecache writeback and vm.dirty parameters](#)

第13章 カーネルロギングの使用

ログファイルは、システム (カーネル、サービス、および実行中のアプリケーションなど) に関するメッセージが含まれるファイルです。Red Hat Enterprise Linux におけるロギングシステムは、組み込みの **syslog** プロトコルに基づいています。さまざまなユーティリティーがこのシステムを使用してイベントを記録し、ログファイルにまとめます。このファイルは、オペレーティングシステムの監査や問題のトラブルシューティングに役に立ちます。

13.1. カーネルリングバッファとは

コンソールは、システムの起動プロセス時に、システム起動の初期段階に関する重要な情報を多数提供します。先に出力されたメッセージが失われないように、カーネルではリングバッファと呼ばれるものが使用されています。このバッファは、カーネルコード内の **printk()** 関数により生成されるブートメッセージなど、すべてのメッセージを格納します。次に、カーネルリングバッファからのメッセージは、**syslog** サービスなどの永続ストレージのログファイルに読み込まれ、保存されます。

上記のバッファは、固定サイズの循環データ構造であり、カーネルにハードコーディングされています。ユーザーは、**dmesg** コマンドまたは `/var/log/boot.log` ファイル介して、カーネルリングバッファに保存されているデータを表示できます。リングバッファが満杯になると、新しいデータにより古いデータが上書きされます。

関連情報

- **syslog(2)** および **dmesg(1)** の man ページ

13.2. ログレベルおよびカーネルロギングにおける PRINTK のロール

カーネルが報告する各メッセージには、メッセージの重要性を定義するログレベルが関連付けられています。カーネルリングバッファは、[カーネルリングバッファとは](#) で説明されているように、すべてのログレベルのカーネルメッセージを収集します。バッファからコンソールに出力されるメッセージを定義するのは **kernel.printk** パラメーターです。

ログレベルの値は、以下の順序で分類されます。

0

カーネルの緊急事態。システムが利用できません。

1

カーネルアラート。すぐに対処する必要があります。

2

重大な問題があると見なされるカーネルの状態。

3

一般的なカーネルのエラー状態。

4

一般的なカーネルの警告状態。

5

正常だが重要な状態に関するカーネル通知。

6

カーネル情報メッセージ。

7

カーネルのデバッグレベルのメッセージ。

RHEL 8 の `kernel.printk` には、デフォルトで以下の 4 つの値が含まれます。

```
# sysctl kernel.printk
kernel.printk = 7 4 1 7
```

この 4 つの値は、順に以下を定義します。

1. コンソールログレベル。コンソールに出力されるメッセージの最低優先度を定義します。
2. 明示的なログレベルが付いていないメッセージのデフォルトのログレベル。
3. コンソールのログレベルに、可能な限り低いログレベル設定を設定します。
4. 起動時のコンソールのログレベルのデフォルト値を設定します。
上記の各値は、エラーメッセージを処理するさまざまなルールを定義します。

重要

デフォルトの `7 4 1 7 printk` 値を使用することで、カーネルアクティビティのデバッグを改善できます。ただし、シリアルコンソールと組み合わせると、この `printk` 設定により激しい I/O バーストが発生し、RHEL システムが一時的に応答しなくなる可能性があります。通常 `4 4 1 7` に `printk` 値を設定するとこのような状況を回避できますが、代わりに追加のデバッグ情報が失われてしまいます。

また、`quiet`、`debug` などの特定のカーネルコマンドラインパラメーターにより、デフォルトの `kernel.printk` 値が変更される点に注意してください。

関連情報

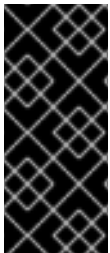
- `syslog(2)` の man ページ

第14章 KDUMP のインストール

RHEL 8 の新しいインストールでは、デフォルトで **kdump** サービスがインストールされアクティベートされます。**kdump** の概要、および **kdump** がデフォルトで有効になっていない場合のインストール方法について、情報を提供して手順を説明します。

14.1. KDUMP とは

kdump は、クラッシュダンプメカニズムを提供し、クラッシュダンプまたは **vmcore** ファイルとして知られるダンプファイルを生成するサービスです。**vmcore** ファイルには、分析とトラブルシューティングに役立つシステムメモリのコンテンツが含まれています。**kdump** は **kexec** システムコールを使用して、再起動せずに **キャプチャーカーネル** である 2 番目のカーネルで起動し、クラッシュしたカーネルメモリの内容をキャプチャしてファイルに保存します。この別のカーネルは、システムメモリの予約部分で使用できます。



重要

カーネルクラッシュダンプは、システム障害時に利用できる唯一の情報になります。したがって、ミッションクリティカルな環境では、**kdump** を稼働させることが重要です。Red Hat は、通常のカーネル更新サイクルで **kexec-tools** を定期的に更新してテストすることを推奨します。これは、新しいカーネル機能をインストールする場合に特に重要です。

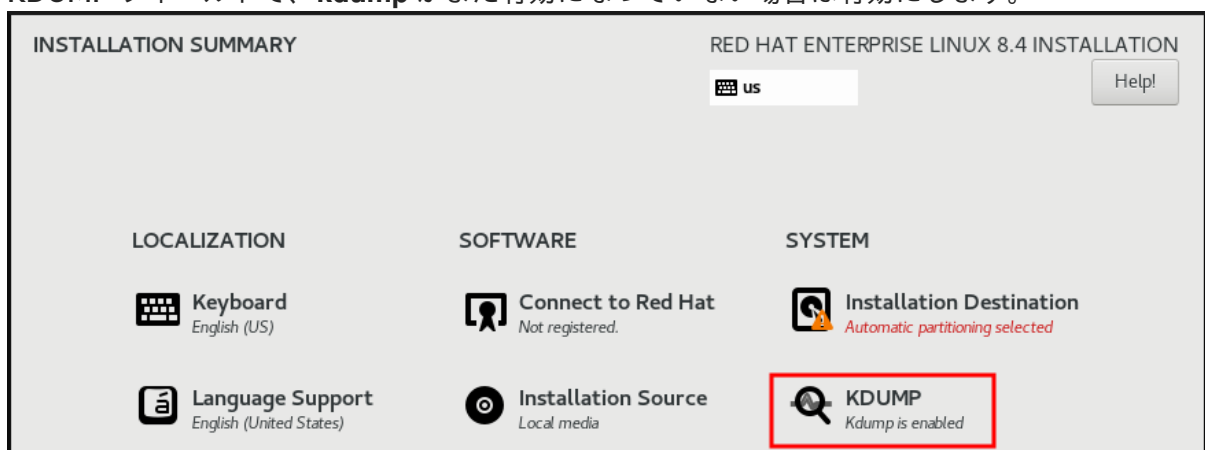
kdump は、マシンにインストールされているすべてのカーネルに対して、または指定したカーネルに対してのみ有効にできます。これは、マシンで複数のカーネルが使用されており、その一部が安定しており、クラッシュの心配がない場合に役立ちます。**kdump** をインストールすると、デフォルトの **/etc/kdump.conf** ファイルが作成されます。**/etc/kdump.conf** ファイルにはデフォルトの最小 **kdump** 設定が含まれており、これを編集して **kdump** 設定をカスタマイズできます。

14.2. ANACONDA を使用した KDUMP のインストール

Anaconda インストーラーでは、対話式インストール時に **kdump** 設定用のグラフィカルインターフェイス画面が表示されます。インストーラー画面のタイトルは **KDUMP** で、メインの **インストールの概要** 画面から利用できます。**kdump** を有効にして、必要な量のメモリーを予約できます。

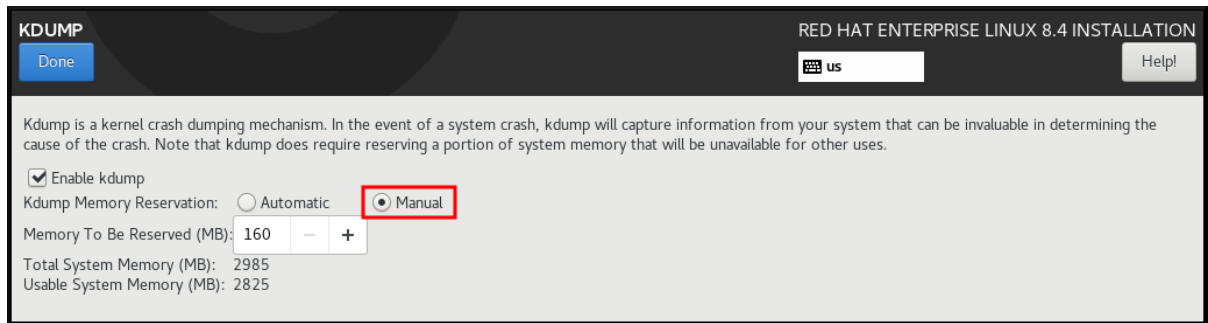
手順

1. **KDUMP** フィールドで、**kdump** がまだ有効になっていない場合は有効にします。



2. メモリー予約をカスタマイズする必要がある場合は、**Kdump Memory Reservation** で **Manual** を選択します。

3. **KDUMP** フィールドの **Memory To Be Reserved (MB)** で、**kdump** に必要なメモリ予約を設定します。



14.3. コマンドラインで KDUMP のインストール

カスタムの Kickstart インストールなどの一部のインストールオプションでは、デフォルトで **kdump** がインストールまたは有効化されない場合があります。この場合は、以下の手順を行ってください。

前提条件

- アクティブな RHEL サブスクリプションがある。
- システムの CPU アーキテクチャー用の **kexec-tools** パッケージを含むリポジトリがある。
- **kdump** 設定とターゲットの要件をすべて満たしている。詳細は [対応している kdump 設定とターゲット](#) を参照してください。

手順

1. **kdump** がシステムにインストールされているかどうかを確認します。

```
# rpm -q kexec-tools
```

このパッケージがインストールされている場合は以下を出力します。

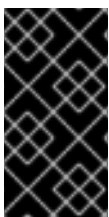
```
kexec-tools-2.0.17-11.el8.x86_64
```

このパッケージがインストールされていない場合は以下を出力します

```
package kexec-tools is not installed
```

2. **kdump** および必要なパッケージをインストールします。

```
# dnf install kexec-tools
```



重要

kernel-3.10.0-693.el7 以降では、**kdump** で **Intel IOMMU** ドライバーがサポートされます。**kernel-3.10.0-514[.XYZ].el7** および初期バージョンの場合は、応答しないキャプチャーカーネルを回避するために **Intel IOMMU** が無効になっていることを確認する必要があります。

第15章 コマンドラインで KDUMP の設定

kdump 用メモリーは、システムの起動時に予約されます。メモリーサイズは、システムの GRUB (Grand Unified Bootloader) 設定ファイルで設定されています。メモリーサイズは、設定ファイルで指定された **crashkernel=** 値と、システムの物理メモリーのサイズにより異なります。

15.1. KDUMP サイズの見積もり

kdump 環境の計画および構築を行う際に、クラッシュダンプファイルに必要な領域を把握しておくことが重要です。

makedumpfile --mem-usage コマンドは、クラッシュダンプファイルに必要な領域を推定し、メモリー使用量に関するレポートを生成します。このレポートは、ダンプレベルと、除外して問題ないページを判断するのに役立ちます。

手順

- 次のコマンドを実行して、メモリー使用量に関するレポートを生成します。

```
# makedumpfile --mem-usage /proc/kcore
```

TYPE	PAGES	EXCLUDABLE	DESCRIPTION
ZERO	501635	yes	Pages filled with zero
CACHE	51657	yes	Cache pages
CACHE_PRIVATE	5442	yes	Cache pages + private
USER	16301	yes	User process pages
FREE	77738211	yes	Free pages
KERN_DATA	1333192	no	Dumpable kernel data

重要

makedumpfile --mem-usage は、必要なメモリーをページ単位で報告します。つまり、カーネルページサイズを元に、使用するメモリーのサイズを計算する必要があります。

デフォルトでは、RHEL カーネルは、AMD64 および Intel 64 の CPU アーキテクチャーで 4KB のサイズのページを使用し、IBM POWER アーキテクチャーで 64KB のサイズのページを使用します。

15.2. メモリー使用量の設定

kdump のメモリー予約は、システムの起動中に行われます。メモリーサイズは、システムの GRUB (Grand Unified Bootloader) 設定で設定されます。メモリーサイズは、設定ファイルで指定された **crashkernel=** オプションの値と、システムの物理メモリーのサイズにより異なります。

crashkernel= オプションはさまざまな方法で定義できます。**crashkernel=** 値を指定するか、**auto** オプションを設定できます。**crashkernel=auto** パラメーターは、システムの物理メモリーの合計量に基づいて、メモリーを自動的に予約します。これを設定すると、カーネルは、キャプチャーカーネルに必要な適切な量のメモリーを自動的に予約します。これにより、OOM (Out-of-Memory) エラーの回避に役立ちます。



注記

kdump の自動メモリー割り当ては、システムのハードウェアアーキテクチャーと利用可能なメモリーサイズによって異なります。

たとえば、AMD64 および Intel 64 の場合には、**crashkernel=auto** パラメーターは、利用可能なメモリーが 1GB を超える場合にのみ機能します。64 ビット ARM アーキテクチャーと IBM Power Systems には、2 GB 以上の使用可能なメモリーが必要です。

システムに、自動割り当ての最小メモリーしきい値より少ないメモリーしかない場合は、手動で予約メモリーの量を設定できます。

前提条件

- システムの root 権限がある。
- **kdump** 設定とターゲットの要件をすべて満たしている。詳細は [対応している kdump 設定とターゲット](#) を参照してください。

手順

1. **crashkernel=** オプションを準備してください。

- たとえば、128 MB のメモリーを予約するには、以下を使用します。

```
crashkernel=128M
```

- または、インストールされているメモリーの合計量に応じて、予約メモリーサイズを変数に設定できます。変数へのメモリー予約の構文は **crashkernel=<range1>:<size1>,<range2>:<size2>** です。以下に例を示します。

```
crashkernel=512M-2G:64M,2G-:128M
```

システムメモリーの合計量が 512 MB - 2 GB の範囲にある場合、64 MB のメモリーを予約します。メモリーの合計量が 2 GB を超える場合、メモリー予約は 128 MB になります。

- 予約メモリーのオフセット。
一部のシステムでは、**crashkernel** の予約が早い段階で行われるため、特定の固定オフセットでメモリーを予約する必要があります。また、特別な用途のために、さらに多くのメモリーの予約が必要になることもあります。オフセットを定義すると、予約メモリーはそこから開始されます。予約メモリーをオフセットするには、以下の構文を使用します。

```
crashkernel=128M@16M
```

この例では、**kdump** は 16 MB (物理アドレス **0x01000000**) から始まる 128 MB のメモリーを予約します。offset パラメーターを 0 に設定するか、完全に省略すると、**kdump** は予約メモリーを自動的にオフセットします。変数のメモリー予約を設定する場合は、この構文を使用することもできます。その場合、オフセットは常に最後に指定されます。以下に例を示します。

```
crashkernel=512M-2G:64M,2G-:128M@16M
```

2. **crashkernel=** オプションをブートローダー設定に適用します。

```
# grubby --update-kernel=ALL --args="crashkernel=<value>"
```

<value> は、前のステップで準備した **crashkernel=** オプションの値に置き換えます。

関連情報

- [kdump メモリー要件](#)
- [カーネルコマンドラインパラメーターの設定](#)
- [システムを起動する前に、grub で boot パラメーターを手動で変更する](#)
- [How to install and boot custom kernels in Red Hat Enterprise Linux 8](#)
- [grubby\(8\) の man ページ](#)

15.3. KDUMP ターゲットの設定

クラッシュダンプは通常、ローカルファイルシステムにファイルとして保存され、デバイスに直接書き込まれます。または、**NFS** プロトコルまたは **SSH** プロトコルを使用して、ネットワーク経由でクラッシュダンプを送信するように設定できます。クラッシュダンプファイルを保存するオプションは、一度に1つだけ設定できます。デフォルトの動作では、ローカルファイルシステムの **/var/crash/** ディレクトリに保存されます。

前提条件

- システムの root 権限がある。
- **kdump** 設定とターゲットの要件をすべて満たしている。詳細は [対応している kdump 設定とターゲット](#) を参照してください。

手順

- ローカルファイルシステムの **/var/crash/** ディレクトリにクラッシュダンプファイルを保存するには、**/etc/kdump.conf** ファイルを変更して、パスを指定します。

```
path /var/crash
```

path /var/crash オプションは、**kdump** がクラッシュダンプファイルを保存するファイルシステムへのパスを表します。



注記

- **/etc/kdump.conf** ファイルでダンプターゲットを指定すると、**path** は指定されたダンプ出力先に対する相対パスになります。
- **/etc/kdump.conf** ファイルでダンプターゲットを指定しない場合、パスはルートディレクトリからの **絶対** パスを表します。

現在のシステムにマウントされている内容に応じて、ダンプターゲットと調整されたダンプパスが自動的に適用されます。

kdump によって生成されるクラッシュダンプファイルと付随するファイルを保護するには、ユーザー権限や SELinux コンテキストなど、ターゲットの宛先ディレクトリの属性を適切に

設定する必要があります。さらに、次のように **kdump.conf** ファイルで **kdump_post.sh** などのスクリプトを定義することもできます。

```
kdump_post <path_to_kdump_post.sh>
```

kdump_post ディレクティブは、**kdump** がクラッシュダンプの取得と指定の保存先への保存を完了した後に実行されるシェルスクリプトまたはコマンドを指定するものです。このメカニズムを使用すると、**kdump** の機能を拡張して、ファイル権限の調整などの操作を実行できます。

例15.1 kdump ターゲット設定

```
# grep -v ^# /etc/kdump.conf | grep -v ^$
ext4 /dev/mapper/vg00-varcrashvol
path /var/crash
core_collector makedumpfile -c --message-level 1 -d 31
```

ここでは、ダンプターゲットが指定されているため (**ext4/dev/mapper/vg00-varcrashvol**)、**/var/crash** にマウントされます。**path** オプションも **/var/crash** に設定されているため、**kdump** は **vmcore** ファイルを **/var/crash/var/crash** ディレクトリに保存します。

- クラッシュダンプを保存するローカルディレクトリを変更するには、**root** として **/etc/kdump.conf** 設定ファイルを編集します。
 - a. **#path /var/crash** の行頭にあるハッシュ記号 (#) を削除します。
 - b. 値を対象のディレクトリパスに置き換えます。以下に例を示します。

```
path /usr/local/cores
```



重要

RHEL 8 では、**kdump systemd** サービスの起動時に、**path** ディレクティブを使用して **kdump** ターゲットとして定義されたディレクトリが存在し、失敗を回避する必要があります。この動作は、サービスの起動時にディレクトリが存在しなかった場合はディレクトリが自動的に作成されていた RHEL の以前のバージョンとは異なります。

- ファイルを別のパーティションに書き込むには、**/etc/kdump.conf** 設定ファイルを編集します。
 - a. 必要に応じて **#ext4** の行頭にあるハッシュ記号 (#) を削除します。
 - デバイス名 (**#ext4 /dev/vg/lv_kdump** 行)
 - ファイルシステムラベル (**#0ext4 LABEL=/boot** 行)
 - UUID (**#ext4 UUID=03138356-5e61-4ab3-b58e-27507ac41937** の行)
 - b. ファイルシステムタイプとデバイス名、ラベル、または UUID を必要な値に変更します。UUID 値を指定するための正しい構文は、**UUID="correct-uuid"** と **UUID=correct-uuid** の両方です。以下に例を示します。


```
ext4 UUID=03138356-5e61-4ab3-b58e-27507ac41937
```



重要

LABEL= または **UUID=** を使用してストレージデバイスを指定することが推奨されます。**/dev/sda3** などのディスクデバイス名は、再起動した場合に一貫性が保証されません。

IBM Z ハードウェアで Direct Access Storage Device (DASD) を使用する場合は、**kdump** に進む前に、ダンプデバイスが **/etc/dasd.conf** で正しく指定されていることを確認してください。

- クラッシュダンプを直接書き込むには、**/etc/kdump.conf** 設定ファイルを修正します。
 - a. **#raw /dev/vg/lv_kdump** の行頭にあるハッシュ記号 (#) を削除します。
 - b. 値を対象のデバイス名に置き換えます。以下に例を示します。

```
raw /dev/sdb1
```

- **NFS** プロトコルを使用してクラッシュダンプをリモートマシンに保存するには、次の手順を実行します。
 - a. **#nfs my.server.com:/export/tmp** の行頭にあるハッシュ記号 (#) を削除します。
 - b. 値を、正しいホスト名およびディレクトリーパスに置き換えます。以下に例を示します。

```
nfs penguin.example.com:/export/cores
```

- c. 変更を有効にするには、**kdump** サービスを再起動します。

```
sudo systemctl restart kdump.service
```



注記

NFS ディレクティブを使用して **NFS** ターゲットを指定すると、**kdump.service** が自動的に **NFS** ターゲットをマウントしてディスク容量をチェックしようとします。事前に **NFS** ターゲットをマウントする必要はありません。**kdump.service** によるターゲットのマウントを防ぐには、**kdump.conf** で **dracut_args --mount** ディレクティブを使用して、**kdump.service** が **NFS** ターゲットを指定する **--mount** 引数とともに **dracut** ユーティリティーを呼び出すようにしてください。

- **SSH** プロトコルを使用してクラッシュダンプをリモートマシンに保存するには、次の手順を実行します。
 - a. **#ssh user@my.server.com** の行頭にあるハッシュ記号 (#) を削除します。
 - b. 値を正しいユーザー名およびホスト名に置き換えます。
 - c. **SSH** キーを設定に含めます。
 - i. **#sshkey /root/.ssh/kdump_id_rsa** の行頭にあるハッシュ記号 ("#") を削除します。

- ii. 値を、ダンプ先のサーバー上の正しいキーの場所に変更します。以下に例を示します。

```
ssh john@penguin.example.com
sshkey /root/.ssh/mykey
```

関連情報

[「システムクラッシュ後に kdump によって生成されるファイル」](#)

15.4. KDUMP コアコレクターの設定

kdump では、**core_collector** を使用してクラッシュダンプイメージをキャプチャーします。RHEL では、**makedumpfile** ユーティリティーがデフォルトのコアコレクターです。これは、以下に示すプロセスによりダンプファイルを縮小するのに役立ちます。

- クラッシュダンプファイルのサイズを圧縮し、さまざまなダンプレベルを使用して必要なページのみをコピーする
- 不要なクラッシュダンプページを除外する
- クラッシュダンプに含めるページタイプをフィルタリングする

構文

```
core_collector makedumpfile -l --message-level 1 -d 31
```

オプション

- **-c**、**-l**、または **-p: zlib** (**-c** オプションの場合)、**lzo** (**-l** オプションの場合)、または **snappy** (**-p** オプションの場合) のいずれかを使用して、ページごとに圧縮ダンプファイルの形式を指定します。
- **-d (dump_level)**: ページを除外して、ダンプファイルにコピーされないようにします。
- **--message-level**: メッセージタイプを指定します。このオプションで **message_level** を指定すると、出力の表示量を制限できます。たとえば、**message_level** で 7 を指定すると、一般的なメッセージとエラーメッセージを出力します。**message_level** の最大値は 31 です。

前提条件

- システムの root 権限がある。
- **kdump** 設定とターゲットの要件をすべて満たしている。詳細は [対応している kdump 設定とターゲット](#) を参照してください。

手順

1. root で、**/etc/kdump.conf** 設定ファイルを編集し、**#core_collector makedumpfile -l --message-level 1 -d 31** の行頭にあるハッシュ記号("#")を削除します。
2. クラッシュダンプファイルの圧縮を有効にするには、以下のコマンドを実行します。

```
core_collector makedumpfile -l --message-level 1 -d 31
```

-l オプションにより、**dump** の圧縮ファイル形式を指定します。**-d** オプションで、ダンプレベルを 31 に指定します。**--message-level** オプションで、メッセージレベルを 1 に指定します。

また、**-c** オプションおよび **-p** オプションを使用した以下の例を検討してください。

- **-c** を使用してクラッシュダンプファイルを圧縮するには、以下のコマンドを実行します。

```
core_collector makedumpfile -c -d 31 --message-level 1
```

- **-p** を使用してクラッシュダンプファイルを圧縮するには、以下のコマンドを実行します。

```
core_collector makedumpfile -p -d 31 --message-level 1
```

関連情報

- [makedumpfile\(8\) の man ページ](#)
- [kdump の設定ファイル](#)

15.5. KDUMP のデフォルト障害応答の設定

デフォルトでは、設定したターゲットの場所で **kdump** がクラッシュダンプファイルの作成に失敗すると、システムが再起動し、ダンプがプロセス内で失われます。デフォルトの障害応答を変更し、コアダンプをプライマリーターゲットに保存できない場合に別の操作を実行するように **kdump** を設定できます。追加のアクションは次のとおりです。

dump_to_rootfs

コアダンプを **root** ファイルシステムに保存します。

reboot

システムを再起動します。コアダンプは失われます。

halt

システムを停止します。コアダンプは失われます。

poweroff

システムの電源を切ります。コアダンプは失われます。

shell

initramfs 内からシェルセッションを実行します。コアダンプを手動で記録できます。

final_action

kdump の成功後、またはシェルまたは **dump_to_rootfs** の失敗アクションの完了時に、**reboot**、**halt** および **poweroff** などの追加操作を有効にします。デフォルトは **reboot** です。

failure_action

カーネルクラッシュでダンプが失敗する可能性がある場合に実行するアクションを指定します。デフォルトは **reboot** です。

前提条件

- root 権限
- **kdump** 設定とターゲットの要件をすべて満たしている。詳細は [対応している kdump 設定とターゲット](#) を参照してください。

手順

1. **root** で、`/etc/kdump.conf` 設定ファイルの `#failure_action` の行頭にあるハッシュ記号 (`#`) を削除します。
2. 値を任意のアクションに置き換えます。

```
failure_action poweroff
```

関連情報

- [kdump ターゲットの設定](#)

15.6. KDUMP の設定ファイル

kdump カーネルの設定ファイルは `/etc/sysconfig/kdump` です。このファイルは、**kdump** カーネルコマンドラインパラメーターを制御します。ほとんどの設定では、デフォルトオプションを使用します。ただし、シナリオによっては、**kdump** カーネルの動作を制御するために特定のパラメーターを変更する必要があります。たとえば、**KDUMP_COMMANDLINE_APPEND** オプションを変更して **kdump** カーネルコマンドラインを追加して詳細なデバッグ出力を取得したり、**KDUMP_COMMANDLINE_REMOVE** オプションを変更して **kdump** コマンドラインから引数を削除したりします。

KDUMP_COMMANDLINE_REMOVE

現在の **kdump** コマンドラインから引数を削除します。これにより、**kdump** エラーや **kdump** カーネルブートエラーの原因となるパラメーターが削除されます。このパラメーターは、以前の **KDUMP_COMMANDLINE** プロセスで解析されたか、`/proc/cmdline` ファイルから継承された可能性があります。

この変数が設定されていない場合は、`/proc/cmdline` ファイルからすべての値が継承されます。このオプションを設定すると、問題のデバッグに役立つ情報も提供されます。

特定の引数を削除するには、以下のようにして **KDUMP_COMMANDLINE_REMOVE** に追加します。

```
KDUMP_COMMANDLINE_REMOVE="hugepages hugepagesz slub_debug quiet log_buf_len swiotlb"
```

KDUMP_COMMANDLINE_APPEND

このオプションは、現在のコマンドラインに引数を追加します。これらの引数は、以前の **KDUMP_COMMANDLINE_REMOVE** 変数で解析されている可能性があります。

kdump カーネルの場合は、**mce**、**cgroup**、**numa**、**hest_disable** などの特定のモジュールを無効にすると、カーネルエラーを防ぐのに役立ちます。これらのモジュールは、**kdump** 用に予約されているカーネルメモリーの大部分を消費したり、**kdump** カーネルの起動に失敗する可能性があります。

kdump カーネルコマンドラインでメモリー **cgroup** を無効にするには、以下のコマンドを実行します。

```
KDUMP_COMMANDLINE_APPEND="cgroup_disable=memory"
```

関連情報

- `Documentation/admin-guide/kernel-parameters.txt` ファイル
- `/etc/sysconfig/kdump` ファイル

15.7. KDUMP 設定のテスト

`kdump` を設定したら、システムクラッシュを手動でテストして、定義した `kdump` ターゲットに `vmcore` ファイルが生成されていることを確認する必要があります。`vmcore` ファイルは、新しく起動したカーネルのコンテキストからキャプチャーされるため、カーネルクラッシュのデバッグに役立つ重要な情報を含みます。



警告

アクティブな稼働システムでは `kdump` をテストしないでください。`kdump` をテストするコマンドにより、カーネルがクラッシュし、データが失われます。システムアーキテクチャーに応じて、十分なメンテナンス時間を必ず確保してください。`kdump` のテストでは時間のかかる再起動が数回必要になる場合があります。

`kdump` のテスト中に `vmcore` ファイルが生成されない場合は、`kdump` のテストを成功させるために、再度テストを実行する前に問題を特定して修正してください。

重要

`kdump` のテストでは時間のかかる再起動が数回必要になる場合があるため、十分なメンテナンス時間を確保してください。

手動でシステムを変更した場合は、システム変更の最後に `kdump` 設定をテストする必要があります。たとえば、次のいずれかの変更を行った場合は、最適な `kdump` パフォーマンスを得るために `kdump` 設定を必ずテストしてください。

- パッケージのアップグレード。
- ハードウェアレベルの変更 (ストレージやネットワークの変更など)。
- ファームウェアと BIOS のアップグレード。
- サードパーティーのモジュールを含む新規のインストールおよびアプリケーションのアップグレード。
- ホットプラグメカニズムを使用した、このメカニズムをサポートするハードウェアへのメモリーの追加。
- `/etc/kdump.conf` ファイルまたは `/etc/sysconfig/kdump` ファイルに対する変更。

前提条件

- システムの root 権限がある。
- 重要なデータがすべて保存されている。`kdump` をテストするコマンドにより、カーネルがクラッシュし、データが失われます。

- システムアーキテクチャーに応じて、十分なマシンメンテナンス時間が確保されている。

手順

1. **kdump** サービスを有効にします。

```
# kdumpctl restart
```

2. **kdump** サービスのステータスを確認します。**kdumpctl** コマンドを使用すると、出力をコンソールに出力できます。

```
# kdumpctl status
kdump:Kdump is operational
```

あるいは、**systemctl** コマンドを使用すると、出力は **systemd** ジャーナルに出力されます。

3. カーネルクラッシュを開始して、**kdump** 設定をテストします。**sysrq-trigger** キーの組み合わせによりカーネルがクラッシュし、必要に応じてシステムが再起動します。

```
# echo c > /proc/sysrq-trigger
```

カーネルの再起動時に、**/etc/kdump.conf** ファイルで指定した場所に **address-YYYY-MM-DD-HH:MM:SS/vmcore** ファイルが作成されます。デフォルトは **/var/crash/** です。

関連情報

- [kdump ターゲットの設定](#)

15.8. システムクラッシュ後に KDUMP によって生成されるファイル

システムがクラッシュすると、**kdump** サービスは、カーネルメモリーをダンプファイル (**vmcore**) にキャプチャーします。また、トラブルシューティングと事後分析に役立つ追加の診断ファイルを生成します。

kdump によって生成されるファイル:

- **vmcore** - クラッシュ時のシステムメモリーを含む主なカーネルメモリーダンプファイル。これには、**kdump** 設定で指定されている **core_collector** プログラムの設定に従ってデータが追加されます。デフォルトでは、カーネルデータ構造、プロセス情報、スタックトレース、およびその他の診断情報が含まれます。
- **vmcore-dmesg.txt** - パニックになったプライマリーカーネルからのカーネルリングバッファログ (**dmesg**) の内容。
- **kexec-dmesg.log** - **vmcore** データを収集するセカンダリーの **kexec** カーネルの実行に基づくカーネルおよびシステムログメッセージが含まれます。

関連情報

- [カーネルリングバッファとは](#)
- [kdump とは](#)

15.9. KDUMP サービスの有効化および無効化

kdump 機能は、特定のカーネルまたはインストールされているすべてのカーネルで有効または無効にするように設定できます。**kdump** 機能を定期的にテストし、適切に動作していることを検証する必要があります。

前提条件

- システムの root 権限がある。
- **kdump** の設定とターゲットの要件をすべて満たしている。[サポートされている kdump 設定とターゲット](#) を参照してください。
- **kdump** をインストールするためのすべての設定が、要件に応じてセットアップされている。

手順

- **multi-user.target** の **kdump** サービスを有効にします。

```
# systemctl enable kdump.service
```

- 現在のセッションでサービスを起動します。

```
# systemctl start kdump.service
```

- **kdump** サービスを停止します。

```
# systemctl stop kdump.service
```

- **kdump** サービスを無効にします。

```
# systemctl disable kdump.service
```



警告

kptr_restrict=1 をデフォルトとして設定することが推奨されます。**kptr_restrict** をデフォルトで (1) に設定すると、**kdumpctl** サービスは、Kernel Address Space Layout (KASLR) が有効または無効であるかに拘らず、クラッシュカーネルを読み込みます。

kptr_restrict が 1 に設定されておらず、KASLR が有効になっている場合は、**/proc/kcore** ファイルの内容がすべてゼロとして生成されます。**kdumpctl** サービスは、**/proc/kcore** ファイルにアクセスしてクラッシュカーネルを読み込むことができません。**kexec-kdump-howto.txt** ファイルには、**kptr_restrict=1** に設定することを推奨する警告メッセージが表示されます。**kdumpctl** サービスが必ずクラッシュカーネルを読み込むように、**sysctl.conf** ファイルで次の内容を確認します。

- **sysctl.conf** ファイルでのカーネルの **kptr_restrict=1** 設定

15.10. カーネルドライバーが KDUMP を読み込まないようにする設定

`/etc/sysconfig/kdump` 設定ファイルに `KDUMP_COMMANDLINE_APPEND=` 変数を追加することで、キャプチャーカーネルが特定のカーネルドライバーをロードしないように制御できます。この方法を使用すると、`kdump` 初期 RAM ディスクイメージ `initramfs` が、指定されたカーネルモジュールをロードするのを防ぐことができます。これにより、メモリー不足 (OOM) killer エラーやその他のクラッシュカーネル障害を防ぐことができます。

以下の設定オプションのいずれかを使用して、`KDUMP_COMMANDLINE_APPEND=` 変数を追加することができます。

- `rd.driver.blacklist=<modules>`
- `modprobe.blacklist=<modules>`

前提条件

- システムの root 権限がある。

手順

1. 現在実行中のカーネルに読み込まれるモジュールのリストを表示します。読み込みをブロックするカーネルモジュールを選択します。

```
$ lsmod

Module              Size Used by
fuse                 126976 3
xt_CHECKSUM          16384 1
ipt_MASQUERADE       16384 1
uinput               20480 1
xt_contrack          16384 1
```

2. `/etc/sysconfig/kdump` ファイルの `KDUMP_COMMANDLINE_APPEND=` 変数を更新します。以下に例を示します。

```
KDUMP_COMMANDLINE_APPEND="rd.driver.blacklist=hv_vmbus,hv_storvsc,hv_utils,hv_netvsc,hid-hyperv"
```

`modprobe.blacklist=<modules>` 設定オプションを使用した以下の例も検討してください。

```
KDUMP_COMMANDLINE_APPEND="modprobe.blacklist=emcp modprobe.blacklist=bnx2fc
modprobe.blacklist=libfcoe modprobe.blacklist=fcoe"
```

3. `kdump` サービスを再起動します。

```
# systemctl restart kdump
```

関連情報

- `dracut.cmdline` の man ページ

15.11. 暗号化されたディスクがあるシステムでの KDUMP の実行

LUKS 暗号化パーティションを実行すると、システムで利用可能なメモリーが一定量必要になります。

システムが必要なメモリー量を下回ると、**cryptsetup** ユーティリティーがパーティションのマウントに失敗します。その結果、2 番目のカーネル (キャプチャーカーネル) で、暗号化したターゲットの場所に **vmcore** ファイルをキャプチャーできませんでした。

kdumpctl estimate コマンドは、**kdump** に必要なメモリー容量を予測できます。**kdumpctl estimate** は、推奨の **crashkernel** 値を出力します。この値は **kdump** に必要な最適なメモリーサイズです。

推奨の **crashkernel** 値は、現在のカーネルサイズ、カーネルモジュール、initramfs、および暗号化したターゲットメモリー要件に基づいて計算されます。

カスタムの **crashkernel=** オプションを使用している場合には、**kdumpctl estimate** は **LUKS required size** 値を出力します。この値は、LUKS 暗号化ターゲットに必要なメモリーサイズです。

手順

1. **crashkernel=** の推定値を出力します。

```
# *kdumpctl estimate*

Encrypted kdump target requires extra memory, assuming using the keyslot with minimum
memory requirement
Reserved crashkernel: 256M
Recommended crashkernel: 652M

Kernel image size: 47M
Kernel modules size: 8M
Initramfs size: 20M
Runtime reservation: 64M
LUKS required size: 512M
Large modules: <none>
WARNING: Current crashkernel size is lower than recommended size 652M.
```

2. **crashkernel=** の値を増やして、必要なメモリー量を設定します。
3. システムを再起動します。



注記

それでも **kdump** がダンプファイルを暗号化したターゲットに保存できない場合は、必要に応じて **crashkernel=** を増やしてください。

第16章 KDUMP の有効化

RHEL 8 システムでは、特定のカーネルまたはインストールされているすべてのカーネルで **kdump** 機能を有効または無効にするように設定できます。ただし、**kdump** 機能を定期的にテストし、適切に動作していることを検証する必要があります。

16.1. インストールされているすべてのカーネルでの KDUMP の有効化

kdump サービスは、**kexec** ツールのインストール後に **kdump.service** を有効にすることで起動します。マシンにインストールされているすべてのカーネルに対して、**kdump** を有効にして起動できます。

前提条件

- 管理者権限がある。

手順

1. インストールしたすべてのカーネルに **crashkernel=** コマンドラインパラメーターを追加します。

```
# grubby --update-kernel=ALL --args="crashkernel=xxM"
```

xxM は必要なメモリー (メガバイト単位) です。

2. **kdump** を有効にします。

```
# systemctl enable --now kdump.service
```

検証

- **kdump** が実行されていることを確認します。

```
# systemctl status kdump.service
```

```
○ kdump.service - Crash recovery kernel arming
```

```
Loaded: loaded (/usr/lib/systemd/system/kdump.service; enabled; vendor preset: disabled)
```

```
Active: active (live)
```

16.2. 特定のインストール済みカーネルでの KDUMP の有効化

マシン上の特定カーネルに対して、**kdump** を有効にできます。

前提条件

- 管理者権限がある。

手順

1. マシンにインストールされているカーネルをリスト表示します。

```
# ls -a /boot/vmlinuz-*  
/boot/vmlinuz-0-rescue-2930657cd0dc43c2b75db480e5e5b4a9  
/boot/vmlinuz-4.18.0-330.el8.x86_64  
/boot/vmlinuz-4.18.0-330.rt7.111.el8.x86_64
```

- 特定の **kdump** カーネルを、システムの Grand Unified Bootloader (GRUB) 設定に追加します。以下に例を示します。

```
# grubby --update-kernel=vmlinuz-4.18.0-330.el8.x86_64 --args="crashkernel=xxM"
```

xxM は必要なメモリー予約 (メガバイト単位) です。

- kdump** を有効にします。

```
# systemctl enable --now kdump.service
```

検証

- kdump** が実行されていることを確認します。

```
# systemctl status kdump.service  
  
○ kdump.service - Crash recovery kernel arming  
  Loaded: loaded (/usr/lib/systemd/system/kdump.service; enabled; vendor preset:  
disabled)  
  Active: active (live)
```

16.3. KDUMP サービスの無効化

kdump.service を停止し、RHEL 8 システムでサービスの開始を無効にできます。

前提条件

- kdump** 設定とターゲットの要件をすべて満たしている。詳細は [対応している kdump 設定とターゲット](#) を参照してください。
- kdump** のインストール用のオプションがすべて、要件に応じて設定されている。詳細は、[kdump のインストール](#) を参照してください。

手順

- 現在のセッションで **kdump** を停止するには、以下のコマンドを実行します。

```
# systemctl stop kdump.service
```

- kdump** を無効にするには、以下を行います。

```
# systemctl disable kdump.service
```



警告

kptr_restrict=1 をデフォルトとして設定することが推奨されます。**kptr_restrict** をデフォルトで (1) に設定すると、Kernel Address Space Layout (**KASLR**) が有効か無効かにかかわらず、**kdumpctl** サービスはクラッシュカーネルを読み込みます。

kptr_restrict が **1** に設定されておらず、**KASLR** が有効になっている場合は、**/proc/kcore** ファイルの内容がすべてゼロとして生成されます。**kdumpctl** サービスは、**/proc/kcore** ファイルにアクセスしてクラッシュカーネルを読み込むことができません。**kexec-kdump-howto.txt** ファイルには、**kptr_restrict=1** に設定することを推奨する警告メッセージが表示されます。**kdumpctl** サービスが必ずクラッシュカーネルを読み込むように、**sysctl.conf** ファイルで次の内容を確認します。

- **sysctl.conf** ファイルでのカーネルの **kptr_restrict=1** 設定

関連情報

- [systemd の管理](#)

第17章 WEB コンソールで KDUMP の設定

RHEL 9 Web コンソールを使用して、**kdump** 設定をセットアップおよびテストできます。Web コンソールでは、起動時に **kdump** サービスを有効にすることができます。さらに、Web コンソールを使用すると、**kdump** 用に予約されたメモリーを設定し、**vmcore** の保存場所を非圧縮形式または圧縮形式で選択できます。

17.1. WEB コンソールで KDUMP メモリーの使用量およびターゲットの場所を設定

RHEL Web コンソールインターフェイスを使用して、**kdump** カーネルのメモリー予約を設定し、**vmcore** ダンプファイルをキャプチャーするターゲットの場所を指定することもできます。

前提条件

- Web コンソールがインストールされており、アクセス可能である。
詳細は、[Web コンソールのインストール](#) を参照してください。

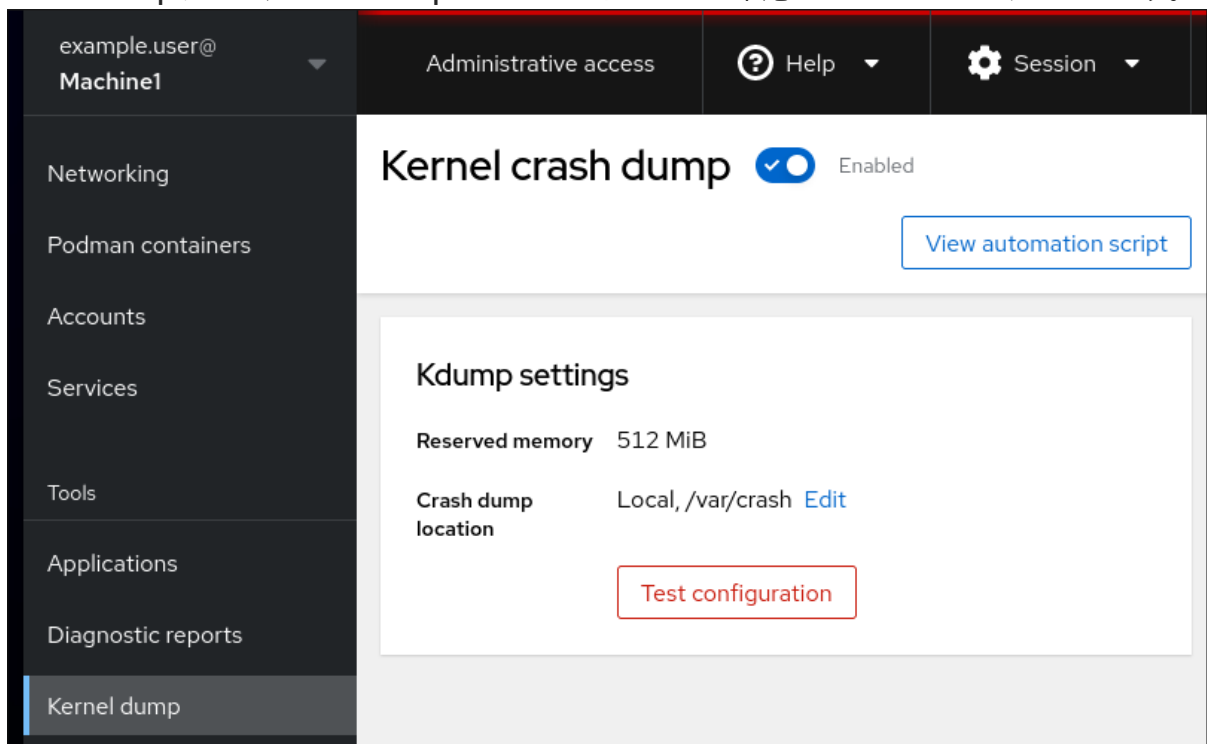
手順

1. Web コンソールで、**Kernel dump** タブを開き、**Kernel crash dump** スイッチをオンに設定して **kdump** サービスを起動します。
2. ターミナルで **kdump** のメモリー使用量を設定します。以下に例を示します。

```
$ sudo grubby --update-kernel ALL --args crashkernel=512M
```

変更を適用するにはシステムを再起動します。

3. **Kernel dump** タブで、**Crash dump location** フィールドの末尾にある **Edit** をクリックします。



4. **vmcore** ダンプファイルを保存するターゲットディレクトリーを指定します。

- ローカルファイルシステムの場合は、ドロップダウンメニューから **Local Filesystem** を選

択します。

Crash dump location

Location Local filesystem ▼

Directory /var/crash

Compression Compress crash dumps to save space

Apply Cancel

- SSH プロトコルを使用したリモートシステムの場合は、ドロップダウンメニューから **Remote over SSH** を選択し、次のフィールドを指定します。
 - **Server** フィールドに、リモートサーバーのアドレスを入力します。
 - **SSH key** フィールドに、SSH キーの場所を入力します。
 - **Directory** フィールドに、ターゲットディレクトリーを入力します。
- NFS プロトコルを使用したリモートシステムの場合は、ドロップダウンメニューから **Remote over NFS** を選択し、次のフィールドを指定します。
 - **Server** フィールドに、リモートサーバーのアドレスを入力します。
 - **Export** フィールドに、NFS サーバーの共有フォルダーの場所を入力します。
 - **Directory** フィールドに、ターゲットディレクトリーを入力します。



注記

Compression チェックボックスをオンにすると、**vmcore** ファイルのサイズを削減できます。

5. **View automation script** をクリックして自動化スクリプトを表示します。
生成されたスクリプトを含むウィンドウが開きます。シェルスクリプトと Ansible Playbook の生成オプションタブ間を移動できます。
6. オプション: **[クリップボードにコピー]** をクリックしてスクリプトをコピーします。
このスクリプトを使用すると、複数のマシンに同じ設定を適用できます。

検証

1. **Test configuration** をクリックします。

Kdump settings

Reserved memory 512 MiB

Crash dump location Local, /var/crash [Edit](#)

[Test configuration](#)

2. Test kdump settings の下にある **Crash system** をクリックします。



警告

システムクラッシュを開始すると、カーネルの動作が停止し、システムがクラッシュしてデータが失われます。

関連情報

- [サポートしている kdump のダンプ出力先](#)

第18章 サポートされている KDUMP 設定とターゲット

kdump メカニズムは、カーネルクラッシュが発生したときにクラッシュダンプファイルを生成する Linux カーネルの機能です。カーネルダンプファイルには、カーネルクラッシュの根本原因を分析して特定するのに役立つ重要な情報が含まれています。クラッシュの原因は、ハードウェアの問題やサードパーティーのカーネルモジュールの問題など、さまざまな要因が考えられます。

下記の情報と手順により、Red Hat Enterprise Linux 9 システムでサポートされている設定とターゲットを理解し、**kdump** を適切に設定して動作を検証できます。

18.1. KDUMP メモリー要件

kdump がカーネルクラッシュダンプをキャプチャーし、さらなる分析のために保存するには、システムメモリーの一部をキャプチャーカーネル用に永続的に予約しておく必要があります。予約されている場合、システムメモリーのこの部分はメインカーネルでは使用できません。

メモリー要件は、特定のシステムパラメーターによって異なります。主な要因は、システムのハードウェアアーキテクチャーです。正確なマシンアーキテクチャー (Intel 64 や AMD64 (x86_64) など) を調べ、それを標準出力に出力するには、以下のコマンドを使用します。

```
$ uname -m
```

下記の最小メモリー要件のリストを使用して、利用可能な最新バージョンで **kdump** 用のメモリーを自動的に予約するための適切なメモリーサイズを設定できます。メモリーサイズは、システムのアーキテクチャーと利用可能な物理メモリーの合計によって異なります。

表18.1 **kdump** 用に必要な最小予約メモリー

アーキテクチャー	使用可能なメモリー	最小予約メモリー
AMD64 と Intel 64 (x86_64)	1 GB から 4 GB	192 MB のメモリー
	4 GB から 64 GB	256 MB のメモリー
	64 GB 以上	512 MB のメモリー
64 ビット ARM アーキテクチャー (arm64)	2 GB 以上	480 MB のメモリー
IBM Power Systems (ppc64le)	2 GB から 4 GB	384 MB のメモリー
	4 GB から 16 GB	512 MB のメモリー
	16 GB から 64 GB	1 GB のメモリー
	64 GB から 128 GB	2 GB のメモリー
	128 GB 以上	4 GB のメモリー
IBM Z (s390x)	1 GB から 4 GB	192 MB のメモリー

アーキテクチャー	使用可能なメモリー	最小予約メモリー
	4 GB から 64 GB	256 MB のメモリー
	64 GB 以上	512 MB のメモリー

多くのシステムでは、**kdump** は必要なメモリー量を予測して、自動的に予約できます。この動作はデフォルトで有効になっていますが、利用可能な合計メモリーサイズが一定以上搭載されているシステムに限られます。この自動割り当て動作に必要なメモリーサイズはシステムのアーキテクチャーによって異なります。



重要

システムのメモリー合計量に基づく予約メモリーの自動設定は、ベストエフォート予測です。実際に必要なメモリーは、I/O デバイスなどの他の要素により異なる場合があります。メモリーが十分でない場合は、カーネルパニックが発生したときにデバッグカーネルがキャプチャーカーネルとして起動できなくなる可能性があります。この問題を回避するには、クラッシュカーネルメモリーを十分なサイズにします。

関連情報

- [RHEL8 マイナーリリース間で crashkernel パラメーターがどのように変化してきたか？](#)
- [テクノロジープレビュー機能および制限の表](#)
- [メモリー自動予約の最小しきい値](#)

18.2. メモリー自動予約の最小しきい値

kexec-tools ユーティリティーは、デフォルトで、**crashkernel** コマンドラインパラメーターを設定し、**kdump** 用に一定量のメモリーを予約します。ただし、一部のシステムでは、ブートローダー設定ファイルで **crashkernel=auto** パラメーターを使用するか、グラフィカル設定ユーティリティーでこのオプションを有効にすることで、**kdump** 用のメモリーを割り当てるのが可能です。この自動予約を機能させるには、システムで一定量の合計メモリーが使用可能である必要があります。メモリー要件はシステムのアーキテクチャーによって異なります。システムのメモリーが指定のしきい値よりも小さい場合は、メモリーを手動で設定する必要があります。

表18.2 自動メモリー予約に必要な最小メモリーサイズ

アーキテクチャー	必要なメモリー
AMD64 と Intel 64 (x86_64)	2 GB
IBM Power Systems (ppc64le)	2 GB
IBM Z (s390x)	4 GB



注記

起動コマンドラインの **crashkernel=auto** オプションは、RHEL 9 以降のリリースでは対応しなくなりました。

18.3. サポートしている KDUMP のダンプ出力先

カーネルクラッシュが発生すると、オペレーティングシステムは、設定したダンプ出力先またはデフォルトのダンプ出力先にダンプファイルを保存します。ダンプファイルは、デバイスに直接保存することも、ローカルファイルシステムにファイルとして保存することも、ネットワーク経由で送信することもできます。以下に示すダンプ出力先のリストを使用すると、**kdump** で現在サポートされているダンプ出力先とサポートされていないダンプ出力先を把握できます。

表18.3 RHEL 9 の **kdump** のダンプ出力先

ダンプ出力先の種類	対応しているダンプ出力先	対応していないダンプ出力先
物理ストレージ	<ul style="list-style-type: none"> ● 論理ボリュームマネージャー (LVM) ● シンプロビジョニングボリューム ● ファイバーチャネル (FC) ディスク (qla2xxx、lpfc、bnx2fc、bfa など) ● ネットワークストレージサーバー上の iSCSI ソフトウェア設定の論理デバイス ● ソフトウェア RAID リューションとしての mdraid サブシステム ● ハードウェア RAID (cciss、hpsa、megaraid_sas、mpt2sas、acraid など)。 ● SCSI および SATA ディスク ● iSCSI および HBA オフロード ● ハードウェア FCoE (qla2xxx、lpfc など) 	<ul style="list-style-type: none"> ● BIOS RAID ● iBFT を使用したソフトウェア iSCSI。現在対応しているトランスポートは、bnx2i、cxgb3i、および cxgb4i です。 ● be2iscsi などのハイブリッドデバイスドライバを使用したソフトウェア iSCSI。 ● イーサネット上ファイバーチャネル (FCoE) ● レガシー IDE ● GlusterFS サーバー ● GFS2 ファイルシステム ● クラスタ化論理ボリュームマネージャー (CLVM) ● 高可用性 LVM ボリューム (HA-LVM)

ダンプ出力先の種類	対応しているダンプ出力先	対応していないダンプ出力先
Network	<ul style="list-style-type: none"> ● カーネルモジュールを使用するハードウェア: tg3、igb、ixgbe、sfc、e1000e、bna、cnic、netxen_nic、qlge、bnx2x、bnx、qlcnic、be2net、enic、virtio-net、ixgbev、igbvf。 ● IPv4 プロトコル。 ● イーサネットデバイスや VLAN など、さまざまなデバイス上のネットワークボンディング。 ● VLAN ネットワーク。 ● ネットワークブリッジ ● ネットワークチーミング。 ● タグ付き VLAN とボンド上の VLAN。 ● ボンド、チーム、および VLAN 上のブリッジネットワーク。 	<ul style="list-style-type: none"> ● IPv6 プロトコル ● ワイヤレス接続 ● InfiniBand ネットワーク ● ブリッジおよびチーム上の VLAN ネットワーク
ハイパーバイザー	<ul style="list-style-type: none"> ● カーネルベースの仮想マシン (KVM) ● Xen Hypervisor (特定の設定を有するもののみ) ● VMware ESXi 4.1 および 5.1。 ● RHEL Gen1 UP ゲスト上の Hyper-V 2012 R2 のみ。 	
ファイルシステム	ext[234]、XFS、および NFS ファイルシステム。	Btrfs ファイルシステム
ファームウェア	<ul style="list-style-type: none"> ● BIOS ベースのシステム ● UEFI セキュアブート 	

関連情報

- [kdump ターゲットの設定](#)

18.4. 対応している KDUMP のフィルターレベル

ダンプファイルのサイズを削減するために、**kdump** は **makedumpfile** コアコレクターを使用してデータを圧縮し、さらに不要な情報を除外します。たとえば、**-8** レベルを使用すると、**hugepages** および **hugetlbfs** ページを削除できます。**makedumpfile** が現在サポートしているレベルは `kdump` のフィルタリングレベルの表で確認できます。

表18.4 kdump のフィルタリングレベル

オプション	説明
1	ゼロページ
2	キャッシュページ
4	キャッシュプライベート
8	ユーザーページ
16	フリーページ

関連情報

- [コアコレクターの設定](#)

18.5. 対応しているデフォルトの障害応答

デフォルトでは、**kdump** がコアダンプを作成できない場合、オペレーティングシステムが再起動します。ただし、コアダンプをプライマリーターゲットに保存できない場合は、**kdump** が別の操作を実行するように設定できます。

表18.5 kdump の障害応答

オプション	説明
dump_to_rootfs	root ファイルシステムにコアダンプの保存を試行します。ネットワーク上のダンプ出力先と併用する場合に特に便利なオプションです。ネットワーク上のダンプ出力先にアクセスできない場合、ローカルにコアダンプを保存するよう kdump の設定を行います。システムは、後で再起動します。
reboot	システムを再起動します。コアダンプは失われます。
halt	システムを停止します。コアダンプは失われます。

オプション	説明
poweroff	システムの電源を切ります。コアダンプは失われます。
shell	initramfs 内から shell セッションを実行して、ユーザーが手動でコアダンプを記録できるようにします。
final_action	kdump の成功後、または shell または dump_to_rootfs の失敗アクションの完了時に、 reboot 、 halt および poweroff アクションなどの追加の操作を有効にします。デフォルトの final_action オプションは reboot です。

関連情報

- [kdump のデフォルト障害応答の設定](#)

18.6. FINAL_ACTION パラメーターの使用

kdump が成功した場合、または **kdump** が設定されたターゲットでの **vmcore** ファイルの保存に失敗した場合は、**final_action** パラメーターを使用して、**reboot**、**halt**、**poweroff** などの追加操作を実行できます。**final_action** パラメーターが指定されていない場合、デフォルトの応答は **reboot** です。

手順

1. **final_action** を設定するには、`/etc/kdump.conf` ファイルを編集して、次のいずれかのオプションを追加します。
 - **final_action reboot**
 - **final_action halt**
 - **final_action poweroff**
2. 変更を有効にするには、**kdump** サービスを再起動します。

```
# kdumpctl restart
```

18.7. FAILURE_ACTION パラメーターの使用

failure_action パラメーターは、カーネルがクラッシュした場合にダンプが失敗したときに実行するアクションを指定します。**failure_action** のデフォルトのアクションは **reboot** で、これはシステムを再起動します。

このパラメーターは、実行する以下のアクションを認識します。

reboot

ダンプが失敗した後、システムを再起動します。

dump_to_rootfs

非ルートダンプターゲットが設定されている場合、ダンプファイルをルートファイルシステムに保存します。

halt

システムを停止します。

poweroff

システムで実行中の操作を停止します。

shell

initramfs 内でシェルセッションを開始します。このセッションから、追加のリカバリーアクションを手動で実行できます。

手順:

1. ダンプが失敗した場合に実行するアクションを設定するには、**/etc/kdump.conf** ファイルを編集して、**failure_action** オプションの1つを指定します。
 - **failure_action reboot**
 - **failure_action halt**
 - **failure_action poweroff**
 - **failure_action shell**
 - **failure_action dump_to_rootfs**
2. 変更を有効にするには、**kdump** サービスを再起動します。

```
# kdumpctl restart
```

第19章 ファームウェア支援ダンプの仕組み

ファームウェア支援ダンプ (fadump) は、IBM POWER システムの **kdump** メカニズムの代わりに提供されるダンプ取得メカニズムです。**kexec** および **kdump** のメカニズムは、AMD64 および Intel 64 システムでコアダンプを取得する際に役立ちます。ただし、最小システムやメインフレームコンピューターなどの一部のハードウェアでは、オンボードファームウェアを活用してメモリー領域を分離して、クラッシュ分析に重要なデータが誤って上書きされないようにします。**fadump** ユーティリティーは、**fadump** メカニズムと IBM POWER システム上の RHEL との統合向けに最適化されています。

19.1. IBM POWERPC ハードウェアにおけるファームウェア支援ダンプ

fadump ユーティリティーは、PCI デバイスおよび I/O デバイスが搭載され、完全にリセットされたシステムから **vmcore** ファイルをキャプチャーします。この仕組みでは、クラッシュするとファームウェアを使用してメモリー領域を保存し、**kdump** ユーザー空間スクリプトをもう一度使用して **vmcore** ファイルを保存します。このメモリー領域には、ブートメモリー、システムレジスター、およびハードウェアのページテーブルエントリー (PTE) を除く、すべてのシステムメモリーコンテンツが含まれます。

fadump メカニズムは、パーティションを再起動し、新規カーネルを使用して以前のカーネルクラッシュからのデータをダンプすることで従来のダンプタイプに比べて信頼性が向上されています。**fadump** には、IBM POWER6 プロセッサベースまたはそれ以降バージョンのハードウェアプラットフォームが必要です。

PowerPC 固有のハードウェアのリセット方法など、**fadump** メカニズムの詳細は、`/usr/share/doc/kexec-tools/fadump-howto.txt` ファイルを参照してください。



注記

保持されないメモリー領域はブートメモリーと呼ばれており、この領域はクラッシュ後にカーネルを正常に起動するのに必要なメモリー容量です。デフォルトのブートメモリーサイズは、256 MB または全システム RAM の 5% のいずれか大きい方です。

kexec で開始されたイベントとは異なり、**fadump** メカニズムでは実稼働用のカーネルを使用してクラッシュダンプを復元します。PowerPC ハードウェアは、クラッシュ後の起動時に、デバイスノード `/proc/device-tree/rtas/ibm.kernel-dump` が **proc** ファイルシステム (**procf**s) で利用できるようにします。**fadump-aware kdump** スクリプトでは、保存された **vmcore** があるかを確認してから、システムの再起動を正常に完了させます。

19.2. ファームウェア支援ダンプメカニズムの有効化

IBM POWER システムのクラッシュダンプ機能は、ファームウェア支援ダンプ (**fadump**) メカニズムを有効にすることで強化できます。

セキュアブート環境では、**GRUB2** ブートローダーは、Real Mode Area (RMA) と呼ばれるブートメモリー領域を割り当てます。RMA のサイズは 512 MB で、ブートコンポーネント間で分割されます。コンポーネントがサイズの割り当てを超えると、**GRUB2** はメモリー不足 (**OOM**) エラーで失敗します。



警告

RHEL 8.7 および 8.6 バージョンのセキュアブート環境では、ファームウェア支援ダンプ (**fadump**) メカニズムを有効にしないでください。**GRUB2** ブートローダーが次のエラーで失敗します。

```
error: ../grub-core/kern/mm.c:376:out of memory.
Press any key to continue...
```

システムは、**fadump** 設定のためにデフォルトの **initramfs** サイズを増やした場合にのみ回復可能です。

システムを回復するための回避策については、記事 [System boot ends in GRUB Out of Memory \(OOM\)](#) を参照してください。

手順

1. **kdump** のインストールと設定
2. **fadump=on** カーネルオプションを有効にします。

```
# grubby --update-kernel=ALL --args="fadump=on"
```

3. (オプション) デフォルトを使用する代わりに、予約済みのブートメモリーを指定する場合は、**crashkernel=xxM** オプションを有効にします。ここで、**xx** は必要なメモリーの量 (メガバイト単位) です。

```
# grubby --update-kernel=ALL --args="crashkernel=xxM fadump=on"
```



重要

ブート設定オプションを指定するときは、実行する前にすべてのブート設定オプションをテストしてください。**kdump** カーネルの起動に失敗した場合は、**crashkernel=** 引数に指定した値を徐々に増やして、適切な値を設定します。

19.3. IBM Z ハードウェアにおけるファームウェア支援ダンプの仕組み

IBM Z システムは、以下のファームウェア支援ダンプメカニズムをサポートします。

- スタンドアロンダンプ (**sadump**)
- **VMDUMP**

IBM Z システムでは、**kdump** インフラストラクチャーはサポート対象で、使用されています。ただし、IBM Z にファームウェア支援ダンプ (**fadump**) を使用すると、さまざまな利点が得られます。

- **sadump** メカニズムはシステムコンソールで開始および制御され、**IPL** 起動可能なデバイスに保存されます。

- **VMDUMP** メカニズムは **sadump** に似ています。このツールもシステムコンソールから開始しますが、ハードウェアから生成されたダンプを取得して解析用にシステムにコピーします。
- (他のハードウェアベースのダンプメカニズムと同様に) これらの手法では、(**kdump** サービスが開始される前の) 起動初期段階におけるマシンの状態をキャプチャーできます。
- **VMDUMP** には、ハードウェアからコピーしたダンプファイルを Red Hat Enterprise Linux システムに格納する仕組みがありますが、IBM Z ハードウェアコンソールから、**VMDUMP** の設定および制御が管理されます。

関連情報

- [スタンドアロンダンププログラム](#)
- [VMDUMP を使用した z/VM でのダンプの作成](#)
- [Red Hat Enterprise Linux 7.4 でのダンプツールの使用](#)

19.4. FUJITSU PRIMEQUEST システムにおける SADUMP の使用

Fujitsu **sadump** メカニズムは、**kdump** が正常に完了しないイベントで **fallback** ダンプキャプチャーを提供するように設計されています。**sadump** メカニズムは、システムの ManageMent Board (MMB) インターフェイスから手動で呼び出します。MMB を使用して、Intel 64 または AMD 64 サーバーの場合と同様に **kdump** を設定し、**sadump** を有効にします。

手順

1. **sadump** に対して **kdump** が予想どおりに起動するように **/etc/sysctl.conf** ファイルで以下の行を追加または編集します。

```
kernel.panic=0
kernel.unknown_nmi_panic=1
```



警告

特に、**kdump** の後にシステムが再起動しないようにする必要があります。**kdump** が **vmcore** ファイルの保存に失敗した後にシステムが再起動すると、**sadump** を呼び出すことができなくなります。

2. **/etc/kdump.conf** の **failure_action** パラメーターを **halt** または **shell** として適切に設定します。

```
failure_action shell
```

関連情報

- FUJITSU Server PRIMEQUEST 2000 Series インストールマニュアル

第20章 コアダンプの分析

システムクラッシュの原因を確認するには、**crash** ユーティリティーを使用します。これにより、GDB (GNU Debugger) と非常によく似たインタラクティブなプロンプトを利用できます。このユーティリティーでは、**kdump**、**netdump**、**diskdump**、または **xendump** によって作成されたコアダンプ、実行中の Linux システムなどをインタラクティブに分析できます。または、Kernel Oops Analyzer または Kdump Helper ツールを使用する選択肢もあります。

20.1. CRASH ユーティリティーのインストール

crash ユーティリティーをインストールするために必要なパッケージと手順を説明します。**crash** ユーティリティーは、RHEL 8 システムにデフォルトでインストールされていない場合があります。クラッシュは、実行中またはカーネルクラッシュが発生してコアダンプファイルが作成された後に、システムの状態をインタラクティブに分析するツールです。コアダンプファイルは、**vmcore** ファイルとも呼ばれます。

手順

1. 関連するリポジトリを有効にします。

```
# subscription-manager repos --enable baseos repository
```

```
# subscription-manager repos --enable appstream repository
```

```
# subscription-manager repos --enable rhel-8-for-x86_64-baseos-debug-rpms
```

2. **crash** パッケージをインストールします。

```
# yum install crash
```

3. **kernel-debuginfo** パッケージをインストールします。

```
# yum install kernel-debuginfo
```

パッケージ **kernel-debuginfo** は実行中のカーネルに対応し、ダンプ分析に必要なデータを提供します。

20.2. CRASH ユーティリティーの実行および終了

提供される情報を使用して、必要なパラメーターと、**crash** ユーティリティーを実行および終了するための手順を理解してください。クラッシュは、実行中またはカーネルクラッシュが発生してコアダンプファイルが作成された後に、システムの状態をインタラクティブに分析するツールです。コアダンプファイルは、**vmcore** ファイルとも呼ばれます。

前提条件

- 現在実行しているカーネルを特定します (4.18.0-5.el8.x86_64 など)。

手順

1. **crash** ユーティリティーを起動するには、以下の2つの必要なパラメーターを渡します。

- `debug-info` (圧縮解除された `vmlinuz` イメージ) (特定の `kernel-debuginfo` パッケージに今

- debuginfo (正確なアドレスを vmlinux ファイル) (特定の kernel-debuginfo パッケージに含まれる `/usr/lib/debug/lib/modules/4.18.0-5.el8.x86_64/vmlinux` など)
- 実際の vmcore ファイル。 `/var/crash/127.0.0.1-2018-10-06-14:05:33/vmcore`
その結果の `crash` コマンドは次のようになります。

```
# crash /usr/lib/debug/lib/modules/4.18.0-5.el8.x86_64/vmlinux /var/crash/127.0.0.1-2018-10-06-14:05:33/vmcore
```

`kdump` で取得したのと同じ `<kernel>` のバージョンを使用します。

例20.1 crash ユーティリティーの実行

以下の例は、4.18.0-5.el8.x86_64 カーネルを使用して、2018年10月6日(14:05 PM)に作成されたコアダンプの分析を示しています。

```
...
WARNING: kernel relocated [202MB]: patching 90160 gdb minimal_symbol values

    KERNEL: /usr/lib/debug/lib/modules/4.18.0-5.el8.x86_64/vmlinux
    DUMPFILE: /var/crash/127.0.0.1-2018-10-06-14:05:33/vmcore [PARTIAL DUMP]
    CPUS: 2
    DATE: Sat Oct 6 14:05:16 2018
    UPTIME: 01:03:57
    LOAD AVERAGE: 0.00, 0.00, 0.00
    TASKS: 586
    NODENAME: localhost.localdomain
    RELEASE: 4.18.0-5.el8.x86_64
    VERSION: #1 SMP Wed Aug 29 11:51:55 UTC 2018
    MACHINE: x86_64 (2904 Mhz)
    MEMORY: 2.9 GB
    PANIC: "sysrq: SysRq : Trigger a crash"
    PID: 10635
    COMMAND: "bash"
    TASK: ffff8d6c84271800 [THREAD_INFO: ffff8d6c84271800]
    CPU: 1
    STATE: TASK_RUNNING (SYSRQ)

crash>
```

2. 対話型プロンプトを終了して `crash` を停止するには、`exit` または `q` と入力します。

例20.2 crash ユーティリティーの終了

```
crash> exit
~]#
```



注記

`crash` コマンドは、ライブシステムをデバッグする強力なツールとして使用することもできます。ただし、システムを破損しないように注意してください。

関連情報

- [予期しないシステムの再起動のガイド](#)

20.3. CRASH ユーティリティーのさまざまなインジケータの表示

crash ユーティリティーを使用して、カーネルメッセージバッファ、バックトレース、プロセスステータス、仮想メモリー情報、開いているファイルなど、さまざまなインジケータを表示します。

メッセージバッファの表示

- カーネルメッセージバッファを表示するには、対話式プロンプトで **log** コマンドを入力します。

```
crash> log
... several lines omitted ...
EIP: 0060:[<c068124f>] EFLAGS: 00010096 CPU: 2
EIP is at sysrq_handle_crash+0xf/0x20
EAX: 00000063 EBX: 00000063 ECX: c09e1c8c EDX: 00000000
ESI: c0a09ca0 EDI: 00000286 EBP: 00000000 ESP: ef4dbf24
DS: 007b ES: 007b FS: 00d8 GS: 00e0 SS: 0068
Process bash (pid: 5591, ti=ef4da000 task=f196d560 task.ti=ef4da000)
Stack:
c068146b c0960891 c0968653 00000003 00000000 00000002 efade5c0 c06814d0
<0> ffffffff c068150f b7776000 f2600c40 c0569ec4 ef4dbf9c 00000002 b7776000
<0> efade5c0 00000002 b7776000 c0569e60 c051de50 ef4dbf9c f196d560 ef4dbfb4
Call Trace:
[<c068146b>] ? __handle_sysrq+0xfb/0x160
[<c06814d0>] ? write_sysrq_trigger+0x0/0x50
[<c068150f>] ? write_sysrq_trigger+0x3f/0x50
[<c0569ec4>] ? proc_reg_write+0x64/0xa0
[<c0569e60>] ? proc_reg_write+0x0/0xa0
[<c051de50>] ? vfs_write+0xa0/0x190
[<c051e8d1>] ? sys_write+0x41/0x70
[<c0409adc>] ? syscall_call+0x7/0xb
Code: a0 c0 01 0f b6 41 03 19 d2 f7 d2 83 e2 03 83 e0 cf c1 e2 04 09 d0 88 41 03 f3 c3 90 c7
05 c8 1b 9e c0 01 00 00 00 0f ae f8 89 f6 <c6> 05 00 00 00 00 01 c3 89 f6 8d bc 27 00 00 00
00 8d 50 d0 83
EIP: [<c068124f>] sysrq_handle_crash+0xf/0x20 SS:ESP 0068:ef4dbf24
CR2: 0000000000000000
```

このコマンドの使用方法についての詳しい情報を参照するには、**help log** と入力してください。



注記

カーネルメッセージバッファには、システムクラッシュに関する最も重要な情報が含まれています。したがって、これは常に最初に **vmcore-dmesg.txt** ファイルにダンプされます。これは、たとえば、ターゲットの場所にスペースがないために、**vmcore** ファイル全体の取得試行に失敗するときに便利です。デフォルトでは、**vmcore-dmesg.txt** は **/var/pkcs/directory/** にあります。

バックトレースの表示

- カーネルスタックトレースを表示するには、**bt** コマンドを使用します。

```

crash> bt
PID: 5591 TASK: f196d560 CPU: 2 COMMAND: "bash"
#0 [ef4dbd0c] crash_kexec at c0494922
#1 [ef4dbe20] oops_end at c080e402
#2 [ef4dbe34] no_context at c043089d
#3 [ef4dbe58] bad_area at c0430b26
#4 [ef4dbe6c] do_page_fault at c080fb9b
#5 [ef4dbee4] error_code (via page_fault) at c080d809
    EAX: 00000063 EBX: 00000063 ECX: c09e1c8c EDX: 00000000 EBP: 00000000
    DS: 007b ESI: c0a09ca0 ES: 007b EDI: 00000286 GS: 00e0
    CS: 0060 EIP: c068124f ERR: ffffffff EFLAGS: 00010096
#6 [ef4dbf18] sysrq_handle_crash at c068124f
#7 [ef4dbf24] __handle_sysrq at c0681469
#8 [ef4dbf48] write_sysrq_trigger at c068150a
#9 [ef4dbf54] proc_reg_write at c0569ec2
#10 [ef4dbf74] vfs_write at c051de4e
#11 [ef4dbf94] sys_write at c051e8cc
#12 [ef4dbfb0] system_call at c0409ad5
    EAX: ffffffff EBX: 00000001 ECX: b7776000 EDX: 00000002
    DS: 007b ESI: 00000002 ES: 007b EDI: b7776000
    SS: 007b ESP: bfc2088 EBP: bfc20b4 GS: 0033
    CS: 0073 EIP: 00edc416 ERR: 00000004 EFLAGS: 00000246

```

bt <pid> を入力して特定のプロセスのバックトレースを表示するか、**help bt** を実行して、**bt** の使用についての詳細を表示します。

プロセスの状態表示

- システム内のプロセスの状態を表示するには、**ps** コマンドを使用します。

```

crash> ps
  PID  PPID  CPU  TASK  ST  %MEM  VSZ  RSS  COMM
>  0    0  0  c09dc560  RU  0.0    0    0 [swapper]
>  0    0  1  f7072030  RU  0.0    0    0 [swapper]
  0    0  2  f70a3a90  RU  0.0    0    0 [swapper]
>  0    0  3  f70ac560  RU  0.0    0    0 [swapper]
  1    0  1  f705ba90  IN  0.0  2828  1424  init
... several lines omitted ...
 5566   1  1  f2592560  IN  0.0  12876   784  auditd
 5567   1  2  ef427560  IN  0.0  12876   784  auditd
 5587  5132  0  f196d030  IN  0.0  11064  3184  sshd
> 5591  5587  2  f196d560  RU  0.0   5084  1648  bash

```

ps <pid> を使用して、単一プロセスのステータスを表示します。**ps** の詳細な使用方法は、**help ps** を使用します。

仮想メモリ情報の表示

- 基本的な仮想メモリ情報を表示するには、対話式プロンプトで **vm** コマンドを入力します。

```

crash> vm
PID: 5591 TASK: f196d560 CPU: 2 COMMAND: "bash"
  MM  PGD  RSS  TOTAL_VM
f19b5900 ef9c6000 1648k  5084k
  VMA  START  END  FLAGS  FILE

```

```

f1bb0310 242000 260000 8000875 /lib/ld-2.12.so
f26af0b8 260000 261000 8100871 /lib/ld-2.12.so
efbc275c 261000 262000 8100873 /lib/ld-2.12.so
efbc2a18 268000 3ed000 8000075 /lib/libc-2.12.so
efbc23d8 3ed000 3ee000 8000070 /lib/libc-2.12.so
efbc2888 3ee000 3f0000 8100071 /lib/libc-2.12.so
efbc2cd4 3f0000 3f1000 8100073 /lib/libc-2.12.so
efbc243c 3f1000 3f4000 100073
efbc28ec 3f6000 3f9000 8000075 /lib/libdl-2.12.so
efbc2568 3f9000 3fa000 8100071 /lib/libdl-2.12.so
efbc2f2c 3fa000 3fb000 8100073 /lib/libdl-2.12.so
f26af888 7e6000 7fc000 8000075 /lib/libtinfo.so.5.7
f26aff2c 7fc000 7ff000 8100073 /lib/libtinfo.so.5.7
efbc211c d83000 d8f000 8000075 /lib/libnss_files-2.12.so
efbc2504 d8f000 d90000 8100071 /lib/libnss_files-2.12.so
efbc2950 d90000 d91000 8100073 /lib/libnss_files-2.12.so
f26afe00 edc000 edd000 4040075
f1bb0a18 8047000 8118000 8001875 /bin/bash
f1bb01e4 8118000 811d000 8101873 /bin/bash
f1bb0c70 811d000 8122000 100073
f26afae0 9fd9000 9ffa000 100073
... several lines omitted ...

```

vm <pid> を実行して、1つのプロセスの情報を表示するか、**help vm** を実行して、**vm** の使用方法を表示します。

オープンファイルの表示

- オープンファイルの情報を表示するには、**files** コマンドを実行します。

```

crash> files
PID: 5591 TASK: f196d560 CPU: 2 COMMAND: "bash"
ROOT: / CWD: /root
FD FILE DENTRY INODE TYPE PATH
0 f734f640 eedc2c6c eecd6048 CHR /pts/0
1 efade5c0 eee14090 f00431d4 REG /proc/sysrq-trigger
2 f734f640 eedc2c6c eecd6048 CHR /pts/0
10 f734f640 eedc2c6c eecd6048 CHR /pts/0
255 f734f640 eedc2c6c eecd6048 CHR /pts/0

```

files <pid> を実行して、選択した1つのプロセスによって開かれたファイルを表示するか、**help files** を実行して、**files** の使用方法を表示します。

20.4. KERNEL OOPS ANALYZER の使用

Kernel Oops Analyzer ツールでは、ナレッジベースの既知の問題で oops メッセージを比較することでクラッシュダンプを分析します。

前提条件

- oops メッセージのセキュリティーを保護し、Kernel Oops Analyzer にフィードしている。

手順

1. Kernel Oops Analyzer ツールにアクセスします。

- カーネルクラッシュの問題を診断するには、**vmcore** に生成されたカーネルの oops ログをアップロードします。
 - テキストメッセージまたは **vmcore-dmesg.txt** を入力として指定して、カーネルクラッシュの問題を診断することも可能です。

The screenshot displays two input options for the Kernel Oops Analyzer. **Option 1: File Input** features a file selection button labeled 'Choose File' with the text 'No file chosen' next to it. Below this, instructions state: 'Choose and upload the kernel oops log generated from a vmcore. Maximum file size for uploaded kernel oops log is 10 MB.' A blue 'Detect' button is at the bottom. **Option 2: Text Input** features a large text area for pasting content. Below the text area are two buttons: a blue 'Detect' button and a white 'Clear' button with a circular icon.

- DETECT** をクリックして、**makedumpfile** からの情報に基づいて既知のソリューションと oops メッセージを比較します。

関連情報

- [Kernel Oops Analyzer](#) の記事
- [予期しないシステムの再起動のガイド](#)

20.5. KDUMP HELPER ツール

Kdump ヘルパーツールは、提供された情報を使用して **kdump** を設定するのに役立ちます。Kdump Helper は、ユーザーの設定に基づいて設定スクリプトを生成します。サーバーでスクリプトを開始して実行すると、**kdump** サービスが設定されます。

関連情報

- [Kdump ヘルパー](#)

第21章 EARLY KDUMP を使用した起動時間クラッシュの取得

Early kdump は、**kdump** メカニズムの機能です。システムサービスが起動する前の起動プロセス初期段階でシステムまたはカーネルのクラッシュが発生した場合に、**vmcore** ファイルをキャプチャーします。Early kdump は、クラッシュカーネルおよびクラッシュカーネルの **initramfs** を早い段階でメモリーにロードします。

21.1. EARLY KDUMP の概要

カーネルクラッシュは、**kdump** サービスが起動する前のブート初期段階で発生することがあります。その場合、kdump はクラッシュしたカーネルメモリーの内容をキャプチャーして保存できません。そのため、トラブルシューティングに重要なクラッシュ関連の情報が失われます。この問題に対処するには、**kdump** サービスの一部である **early kdump** 機能を使用できます。

21.2. EARLY KDUMP の有効化

early kdump 機能は、初期クラッシュの **vmcore** 情報をキャプチャーするため、十分に早めにロードされるように、クラッシュカーネルと初期 RAM ディスクイメージ (**initramfs**) をセットアップします。これにより、初期のブートカーネルクラッシュに関する情報が失われるリスクを排除できます。

前提条件

- アクティブな RHEL サブスクリプションがある。
- システムの CPU アーキテクチャー用の **kexec-tools** パッケージを含むリポジトリがある。
- **kdump** の設定とターゲットの要件を満たしている詳細は、[サポートされている kdump 設定とターゲット](#) を参照してください。

手順

1. **kdump** サービスが有効でアクティブであることを確認します。

```
# systemctl is-enabled kdump.service && systemctl is-active kdump.service
enabled
active
```

kdump が有効ではなく、実行されていない場合は、必要な設定をすべて設定し、**kdump** サービスが有効化されていることを確認します。

2. 起動カーネルの **initramfs** イメージを、**early kdump** 機能で再構築します。

```
# dracut -f --add earlykdump
```

3. **rd.earlykdump** カーネルコマンドラインパラメーターを追加します。

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="rd.earlykdump"
```

4. システムを再起動して、変更を反映させます。

```
# reboot
```


検証手順

- **rd.earlykdump** が正常に追加され、**early kdump** 機能が有効になっていることを確認します。

```
# cat /proc/cmdline
```

```
BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-187.el8.x86_64 root=/dev/mapper/rhel-root ro  
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap  
rhgb quiet rd.earlykdump
```

```
# journalctl -x | grep early-kdump
```

```
Mar 20 15:44:41 redhat dracut-cmdline[304]: early-kdump is enabled.
```

```
Mar 20 15:44:42 redhat dracut-cmdline[304]: kexec: loaded early-kdump kernel
```

関連情報

- [/usr/share/doc/kexec-tools/early-kdump-howto.txt](#) ファイル
- [What is early kdump support and how do I configure it?](#)
- [kdump の有効化](#)

第22章 カーネルライブパッチでパッチの適用

Red Hat Enterprise Linux カーネルのライブパッチソリューションを使用して、システムの再起動またはプロセスの再起動を行わずに、実行中のカーネルにパッチを当てることができます。

このソリューションでは、システム管理者は以下を行うことができます。

- 重大なセキュリティーパッチをカーネルに即座に適用することが可能。
- 長時間実行しているタスクの完了、ユーザーのログオフ、スケジュールダウンタイムを待つ必要がない。
- システムのアップタイムをより制御し、セキュリティーや安定性を犠牲にしない。

重要な、重要なすべての CVE は、カーネルライブパッチソリューションで解決されるわけではありません。この目的は、セキュリティー関連パッチに必要な再起動を減らすことであり、完全になくすことではありません。ライブパッチの範囲の詳細は、[RHEL 7 はライブカーネルパッチ \(kpatch\) をサポートしていますか?](#) を参照してください。



警告

カーネルのライブマイグレーションパッチと、その他のカーネルサブコンポーネントとの間に、いくらか非互換性が存在します。以下を参照してください。

カーネルのライブパッチを使用する前に、[kpatch の制限](#) を慎重に確認してください。



注記

カーネルのライブパッチ更新のサポート頻度の詳細は、以下を参照してください。

- [Kernel Live Patch Support Cadence Update](#)
- [Kernel Live Patch life cycles](#)

22.1. KPATCH の制限

- **kpatch** 機能は、汎用のカーネルアップグレードメカニズムではありません。システムをすぐに再起動できない場合など、単純なセキュリティーおよびバグ修正の更新を適用する場合に使用します。
- パッチの読み込み中または読み込み後は、**SystemTap** ツールまたは **kprobe** ツールを使用しないでください。このようなプローブが削除されるまでは、パッチが適用できなくなる可能性があります。

22.2. サードパーティーのライブパッチサポート

kpatch ユーティリティーは、Red Hat リポジトリ提供の RPM モジュールを含む、Red Hat がサポートする唯一のカーネルライブパッチユーティリティーです。Red Hat は、Red Hat 提供でないライブカーネルパッチはサポートしません。

サードパーティーのライブパッチで発生する問題に対応する必要がある場合、Red Hat では、原因発見を必要とする調査のアウトセットで、ライブパッチベンダーにケースを開くことを奨めていますこれにより、ベンダーが許可すれば、ソースコードの供給が可能になり、Red Hat サポートに調査を依頼する前に、サポート組織への原因追及を支援することがになります。

サードパーティーのライブパッチを実行しているシステムの場合、Red Hat は、Red Hat が同梱し、サポートしているソフトウェアの複製を求める権利を有します。これが可能でない場合、Red Hat は、同じ動作が発生するかどうかを確認するために、ライブパッチを適用せずに、お使いのテスト環境で同じようなシステムとワークロードのデプロイメントを求めます。

サードパーティーソフトウェアサポートポリシーの詳細は、[Red Hat グローバルサポートサービスは、サードパーティーのソフトウェア、ドライバー、そして認定されていないハードウェアおよびハイパーバイザー、もしくはゲストのオペレーティングシステムについてどのようなサポートを提供していますか?](#)を参照してください。

22.3. カーネルライブパッチへのアクセス

ライブのカーネルパッチ機能は、RPM パッケージとして提供されるカーネルモジュール (**kmod**) として実装されます。

すべてのお客様は、通常のチャンネルから提供されるカーネルライブパッチにアクセスできます。ただし、延長サポートサービスにサブスクライブしていないお客様は、次のマイナーリリースが利用可能になると、現行のマイナーリリースに対する新しいパッチへのアクセスを失うことになります。たとえば、標準のサブスクリプションを購入しているお客様は、RHEL 8.3 カーネルがリリースされるまで RHEL 8.2 カーネルのライブパッチのみを行うことができます。

22.4. カーネルライブパッチのコンポーネント

カーネルのライブパッチのコンポーネントは、以下のようになります。

カーネルパッチモジュール

- カーネルライブパッチの配信メカニズム
- パッチが適用されるカーネル用に構築したカーネルモジュール。
- パッチモジュールには、カーネルに必要な修正のコードが含まれます。
- パッチモジュールは、**kpatch** カーネルサブシステムで登録し、置き換えられるオリジナル機能の情報を提供します。また、置換される機能に一致するポインターも含まれます。カーネルパッチモジュールは RPM として提供されます。
- 命名規則は、**kpatch_<kernel version>_<kpatch version>_<kpatch release>** です。名前の **kernel version** 部分の **ドット** は、**アンダースコア** に置き換えます。

kpatch ユーティリティー

パッチモジュールを管理するためのコマンドラインユーティリティー。

kpatch サービス

multiuser.target で必要な **systemd** サービス。このターゲットは、システムの起動時にカーネルパッチをロードします。

kpatch-dnf パッケージ

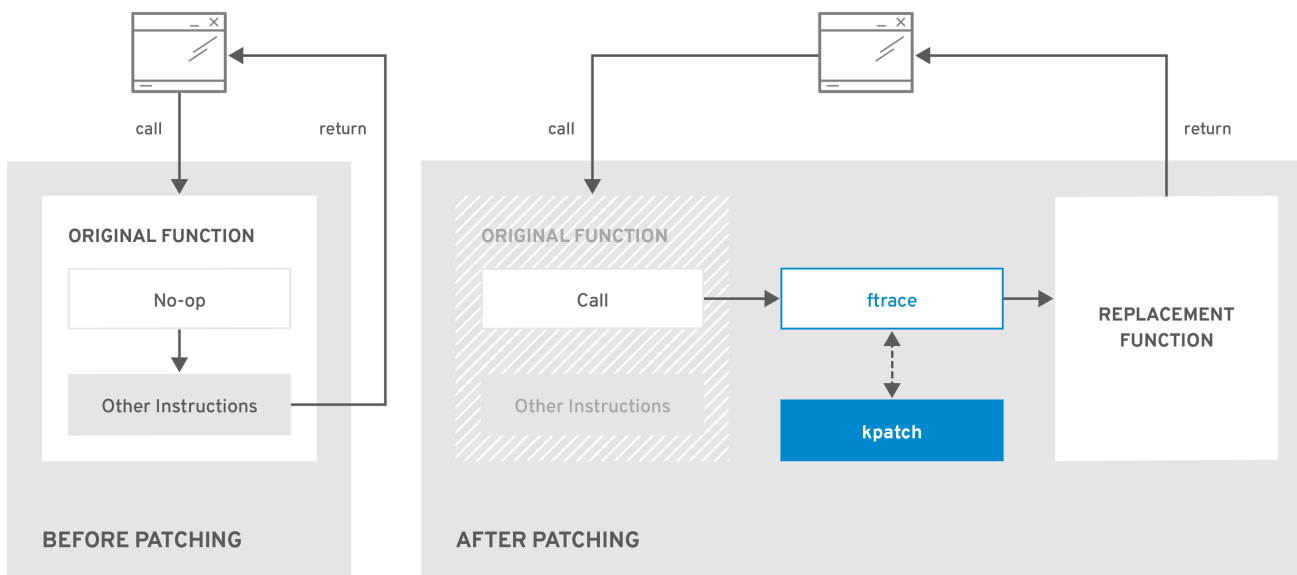
RPM パッケージ形式で配信される DNF プラグイン。このプラグインは、カーネルライブパッチへの自動サブスクリプションを管理します。

22.5. カーネルライブパッチの仕組み

kpatch カーネルパッチソリューションは、**livepatch** カーネルサブシステムを使用して、古い機能の出力先を新しい機能に変更します。ライブカーネルパッチがシステムに適用されると、以下が発生します。

1. カーネルパッチモジュールは、`/var/lib/kpatch/` ディレクトリーにコピーされ、次回の起動時に **systemd** を介して、カーネルへの再適用として登録されます。
2. 実行中のカーネルに kpatch モジュールがロードされ、新しいコードのメモリー内の場所を指定するポインターを使用して、新しい機能が **ftrace** メカニズムに登録されます。
3. パッチが当てられた機能にカーネルがアクセスすると、**ftrace** メカニズムにリダイレクトされます。これにより、元々の機能は回避され、パッチを当てたバージョンの機能にカーネルをリダイレクトします。

図22.1カーネルライブパッチの仕組み



RHEL_424549_0119

22.6. 現在インストールされているカーネルをライブパッチストリームにサブスクライブする手順

カーネルパッチモジュールは RPM パッケージに含まれ、パッチが適用されたカーネルバージョンに固有のものとなります。各 RPM パッケージは、徐々に蓄積されていきます。

以下の手順では、指定のカーネルに対して、今後の累積パッチ更新をすべてサブスクライブする方法を説明します。ライブパッチは累積的であるため、特定のカーネルにデプロイされている個々のパッチを選択できません。



警告

Red Hat は、Red Hat がサポートするシステムに適用されたサードパーティーのライブパッチをサポートしません。

前提条件

- root 権限がある。

手順

1. 必要に応じて、カーネルバージョンを確認します。

```
# uname -r
4.18.0-94.el8.x86_64
```

2. カーネルのバージョンに一致するライブパッチパッケージを検索します。

```
# yum search $(uname -r)
```

3. ライブパッチパッケージをインストールします。

```
# yum install "kpatch-patch = $(uname -r)"
```

上記のコマンドでは、特定カーネルにのみに最新の累積パッチをインストールし、適用します。

ライブパッチパッケージのバージョンが1-1以上である場合には、パッケージにパッチモジュールが含まれます。この場合、ライブパッチパッケージのインストール時に、カーネルにパッチが自動的に適用されます。

カーネルパッチモジュールは、今後の再起動時に **systemd** システムおよびサービスマネージャーにより読み込まれる `/var/lib/kpatch/` ディレクトリーにもインストールされます。



注記

指定のカーネルに利用可能なライブパッチがない場合は、空のライブパッチパッケージがインストールされます。空のライブパッチパッケージには、`kpatch_version-kpatch_release 0-0` (例: `kpatch-patch-4_18_0-94-0-0.el8.x86_64.rpm`) が含まれます。空のRPMのインストールを行うと、指定のカーネルの将来のすべてのライブパッチにシステムがサブスクライブされます。

検証

- インストールされているすべてのカーネルにパッチが当てられていることを確認します。

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_94_1_1 [enabled]
```

```
Installed patch modules:
kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
...
```

この出力は、カーネルパッチモジュールがカーネルに読み込まれていることを示しています。このカーネルには、**kpatch-patch-4_18_0-94-1-1.el8.x86_64.rpm** パッケージで修正された最新のパッチが当てられています。



注記

kpatch list コマンドを入力しても、空のライブパッチパッケージが返されません。代わりに **rpm -qa | grep kpatch** コマンドを使用します。

```
# rpm -qa | grep kpatch
kpatch-patch-4_18_0-477_21_1-0-0.el8_8.x86_64
kpatch-dnf-0.9.7_0.4-2.el8.noarch
kpatch-0.9.7-2.el8.noarch
```

関連情報

- **kpatch(1)** の man ページ
- [ソフトウェアパッケージのインストール](#)

22.7. ライブパッチストリームに新しいカーネルを自動的にサブスクライブする手順

kpatch-dnf YUM プラグインを使用して、カーネルパッチモジュール (別称 カーネルライブパッチ) が提供する修正にシステムをサブスクライブできます。このプラグインは、現在システムが使用するカーネルと、今後インストールされるカーネルの自動サブスクリプションを有効にします。

前提条件

- root 権限がある。

手順

1. 必要に応じて、インストール済みの全カーネルと、現在実行中のカーネルを確認します。

```
# yum list installed | grep kernel
Updating Subscription Management repositories.
Installed Packages
...
kernel-core.x86_64      4.18.0-240.10.1.el8_3      @rhel-8-for-x86_64-baseos-rpms
kernel-core.x86_64      4.18.0-240.15.1.el8_3      @rhel-8-for-x86_64-baseos-rpms
...

# uname -r
4.18.0-240.10.1.el8_3.x86_64
```

2. **kpatch-dnf** プラグインをインストールします。

```
# yum install kpatch-dnf
```

3. カーネルライブパッチの自動サブスクリプションを有効にします。

```
# yum kpatch auto
Updating Subscription Management repositories.
Last metadata expiration check: 19:10:26 ago on Wed 10 Mar 2021 04:08:06 PM CET.
Dependencies resolved.
=====
Package                Architecture
=====
Installing:
kpatch-patch-4_18_0-240_10_1    x86_64
kpatch-patch-4_18_0-240_15_1    x86_64

Transaction Summary
=====
Install 2 Packages
...

```

このコマンドは、現在インストールされているすべてのカーネルをサブスクライブして、カーネルライブパッチを受け取ります。このコマンドは、インストールされている全カーネルに、最新の累積パッチ (存在する場合) をインストールして適用します。

今後、カーネルを更新すると、新しいカーネルのインストールプロセス中にライブパッチが自動的にインストールされます。

カーネルパッチモジュールは、今後の再起動時に **systemd** システムおよびサービスマネージャーにより読み込まれる `/var/lib/kpatch/` ディレクトリーにもインストールされます。



注記

指定のカーネルに利用可能なライブパッチがない場合は、空のライブパッチパッケージがインストールされます。空のライブパッチパッケージには、**kpatch_version-kpatch_release 0-0** (例: **kpatch-patch-4_18_0-240-0.el8.x86_64.rpm**) が含まれます。空の RPM のインストールを行うと、指定のカーネルの将来のすべてのライブパッチにシステムがサブスクライブされます。

検証

- インストールされているすべてのカーネルにパッチが当てられていることを確認します。

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_240_10_1_0_1 [enabled]

Installed patch modules:
kpatch_4_18_0_240_10_1_0_1 (4.18.0-240.10.1.el8_3.x86_64)
kpatch_4_18_0_240_15_1_0_2 (4.18.0-240.15.1.el8_3.x86_64)

```

この出力から、実行中のカーネルとインストールされている他のカーネル両方に **kpatch-patch-4_18_0-240_10_1-0-1.rpm** と **kpatch-patch-4_18_0-240_15_1-0-1.rpm** パッケージそれぞれからの修正が適用されたことが分かります。



注記

kpatch list コマンドを入力しても、空のライブパッチパッケージが返されません。代わりに **rpm -qa | grep kpatch** コマンドを使用します。

```
# rpm -qa | grep kpatch
kpatch-patch-4_18_0-477_21_1-0-0.el8_8.x86_64
kpatch-dnf-0.9.7_0.4-2.el8.noarch
kpatch-0.9.7-2.el8.noarch
```

関連情報

- **kpatch(1)** および **dnf-kpatch(8)** の man ページ

22.8. ライブパッチストリームへの自動サブスクリプションの無効化

カーネルパッチモジュールが提供する修正にシステムをサブスクライブする場合、サブスクリプションは **自動的** です。この機能 (したがって、**kpatch-patch** パッケージの自動インストール) を無効にできません。

前提条件

- root 権限がある。

手順

1. 必要に応じて、インストール済みの全カーネルと、現在実行中のカーネルを確認します。

```
# yum list installed | grep kernel
Updating Subscription Management repositories.
Installed Packages
...
kernel-core.x86_64      4.18.0-240.10.1.el8_3    @rhel-8-for-x86_64-baseos-rpms
kernel-core.x86_64      4.18.0-240.15.1.el8_3    @rhel-8-for-x86_64-baseos-rpms
...

# uname -r
4.18.0-240.10.1.el8_3.x86_64
```

2. カーネルライブパッチへの自動サブスクリプションを無効にします。

```
# yum kpatch manual
Updating Subscription Management repositories.
```

検証手順

- 成功した出力を確認できます。

```
# yum kpatch status
...
Updating Subscription Management repositories.
Last metadata expiration check: 0:30:41 ago on Tue Jun 14 15:59:26 2022.
Kpatch update setting: manual
```


関連情報

- [kpatch\(1\)](#) および [dnf-kpatch\(8\)](#) の man ページ

22.9. カーネルパッチモジュールの更新

カーネルパッチモジュールが配信され、RPM パッケージを通じて適用されているため、累計のカーネルパッチモジュール更新は、他の RPM パッケージの更新と似ています。

前提条件

- [現在インストールされているカーネルをライブパッチストリームにサブスクライブする手順](#) に従って、システムがライブパッチストリームにサブスクライブされている。

手順

- 現在のカーネルの新しい累積バージョンを更新します。

```
# yum update "kpatch-patch = $(uname -r)"
```

上記のコマンドは、現在実行中のカーネルに利用可能な更新を自動的にインストールし、適用します。これには、新たにリリースされた累計なライブパッチが含まれます。

- もしくは、インストールしたすべてのカーネルパッチモジュールを更新します。

```
# yum update "kpatch-patch"
```



注記

システムが同じカーネルで再起動すると、**kpatch.service** systemd サービスにより、カーネルが自動的に再適用されます。

関連情報

- RHEL の [基本的なシステム設定の設定](#)

22.10. ライブパッチパッケージの削除

ライブパッチパッケージを削除して、Red Hat Enterprise Linux カーネルライブパッチソリューションを無効にします。

前提条件

- root 権限がある。
- ライブパッチパッケージがインストールされている。

手順

1. ライブパッチパッケージを選択します。

```
# yum list installed | grep kpatch-patch
kpatch-patch-4_18_0-94.x86_64    1-1.el8    @@commandline
...
```

上記の出力例は、インストールしたライブパッチパッケージをリスト表示します。

2. ライブパッチパッケージを削除します。

```
# yum remove kpatch-patch-4_18_0-94.x86_64
```

ライブパッチパッケージが削除されると、カーネルは次の再起動までパッチが当てられたままになりますが、カーネルパッチモジュールはディスクから削除されます。今後の再起動では、対応するカーネルにはパッチが適用されなくなります。

3. システムを再起動します。
4. ライブパッチパッケージが削除されたことを確認します。

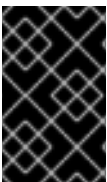
```
# yum list installed | grep kpatch-patch
```

パッケージが正常に削除された場合、このコマンドでは何も出力されません。

5. 必要に応じて、カーネルのライブパッチソリューションが無効になっていることを確認します。

```
# kpatch list
Loaded patch modules:
```

この出力例では、現在読み込まれているパッチモジュールがないため、カーネルにパッチが適用されておらず、ライブパッチソリューションがアクティブでないことが示されています。



重要

現在、Red Hat はシステムの再起動なしで、ライブパッチを元に戻すことはサポートしていません。ご不明な点がございましたら、サポートチームまでお問い合わせください。

関連情報

- [kpatch\(1\) の man ページ](#)
- RHEL での [ソフトウェアパッケージのアンインストール](#)

22.11. カーネルパッチモジュールのアンインストール

Red Hat Enterprise Linux カーネルライブパッチソリューションが、以降の起動時にカーネルパッチモジュールを適用しないようにします。

前提条件

- root 権限がある。
- ライブパッチパッケージがインストールされている。

- カーネルパッチモジュールがインストールされ、ロードされている。

手順

1. カーネルパッチモジュールを選択します。

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_94_1_1 [enabled]

Installed patch modules:
kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
...
```

2. 選択したカーネルパッチモジュールをアンインストールします。

```
# kpatch uninstall kpatch_4_18_0_94_1_1
uninstalling kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
```

- アンインストールしたカーネルモジュールが読み込まれていることに注意してください。

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_94_1_1 [enabled]

Installed patch modules:
<NO_RESULT>
```

選択したモジュールをアンインストールすると、カーネルは次の再起動までパッチが当てられますが、カーネルパッチモジュールはディスクから削除されます。

3. システムを再起動します。
4. 必要に応じて、カーネルパッチモジュールがアンインストールされていることを確認します。

```
# kpatch list
Loaded patch modules:
...
```

上記の出力例では、ロードまたはインストールされたカーネルパッチモジュールが表示されていません。したがって、カーネルにパッチが適用されておらず、カーネルのライブパッチソリューションはアクティブではありません。



重要

現在、Red Hat はシステムの再起動なしで、ライブパッチを元に戻すことはサポートしていません。ご不明な点がございましたら、サポートチームまでお問い合わせください。

関連情報

- `kpatch(1)` の man ページ

22.12. KPATCH.SERVICE の無効化

Red Hat Enterprise Linux カーネルライブパッチソリューションが、以降の起動時にすべてのカーネルパッチモジュールをグローバルに適用しないようにします。

前提条件

- root 権限がある。
- ライブパッチパッケージがインストールされている。
- カーネルパッチモジュールがインストールされ、ロードされている。

手順

1. **kpatch.service** が有効化されていることを確認します。

```
# systemctl is-enabled kpatch.service
enabled
```

2. **kpatch.service** を無効にします。

```
# systemctl disable kpatch.service
Removed /etc/systemd/system/multi-user.target.wants/kpatch.service.
```

- 適用されたカーネルモジュールが依然としてロードされていることに注意してください。

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_94_1_1 [enabled]

Installed patch modules:
kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
```

3. システムを再起動します。
4. 必要に応じて、**kpatch.service** のステータスを確認します。

```
# systemctl status kpatch.service
● kpatch.service - "Apply kpatch kernel patches"
  Loaded: loaded (/usr/lib/systemd/system/kpatch.service; disabled; vendor preset: disabled)
  Active: inactive (dead)
```

この出力テストサンプルでは、**kpatch.service** が無効になっており、実行されていないことを証明しています。したがって、カーネルのライブパッチソリューションはアクティブではありません。

5. カーネルパッチモジュールがアンロードされたことを確認します。

```
# kpatch list
Loaded patch modules:
<NO_RESULT>
```

```
Installed patch modules:  
kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
```

上記の出力例では、カーネルパッチモジュールがインストールされていても、カーネルにパッチが適用されていないことを示しています。



重要

現在、Red Hat はシステムの再起動なしで、ライブパッチを元に戻すことはサポートしていません。ご不明な点がございましたら、サポートチームまでお問い合わせください。

関連情報

- [kpatch\(1\) man ページ](#)、[systemd の管理](#)

第23章 コントロールグループを使用したアプリケーションのシステムリソース制限の設定

コントロールグループ (**cgroups**) カーネル機能を使用すると、アプリケーションのリソース使用状況を制御して、より効率的に使用できます。

cgroups は、以下のタスクで使用できます。

- システムリソース割り当ての制限を設定します。
- 特定のプロセスへのハードウェアリソースの割り当てにおける優先順位を設定する。
- 特定のプロセスをハードウェアリソースの取得から分離する。

23.1. コントロールグループの概要

コントロールグループの Linux カーネル機能を使用して、プロセスを階層的に順序付けされたグループ (**cgroups**) に編成できます。階層 (コントロールグループツリー) は、デフォルトで **/sys/fs/cgroup/** ディレクトリーにマウントされている **cgroups** 仮想ファイルシステムに構造を提供して定義します。

systemd サービスマネージャーは、**cgroups** を使用して、管理するすべてのユニットとサービスを整理します。**/sys/fs/cgroup/** ディレクトリーのサブディレクトリーを作成および削除することで、**cgroups** の階層を手動で管理できます。

続いて、カーネルのリソースコントローラーは、**cgroups** 内のプロセスのシステムリソースを制限、優先順位付け、または割り当てることで、これらのプロセスの動作を変更します。これらのリソースには以下が含まれます。

- CPU 時間
- メモリー
- ネットワーク帯域幅
- これらのリソースの組み合わせ

cgroups の主なユースケースは、システムプロセスを集約し、アプリケーションとユーザー間でハードウェアリソースを分割することです。これにより、環境の効率、安定性、およびセキュリティを強化できます。

コントロールグループ 1

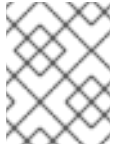
コントロールグループバージョン 1 (**cgroups-v1**) はリソースごとのコントローラー階層を提供します。つまり、各リソース (CPU、メモリー、I/O など) には、独自のコントロールグループ階層があります。各リソースの管理で、1つのコントローラーが別のコントローラーと調整できるように、異なるコントロールグループ階層を組み合わせることができます。ただし、2つのコントローラーが異なるプロセス階層に属する場合、適切な調整が制限されます。

cgroups-v1 コントローラーは、長期間に渡って開発されているため、制御ファイルの動作と命名は均一ではありません。

コントロールグループ 2

Control groups version 2 (**cgroups-v2**) は、すべてのリソースコントローラーがマウントされる単一のコントロールグループ階層を提供します。

コントロールファイルの動作と命名は、さまざまなコントローラーにおいて一貫性があります。



注記

cgroups-v2 は、RHEL 8.2 以降のバージョンで完全にサポートされています。詳細は [Control Group v2 is now fully supported in RHEL 8](#) を参照してください。

関連情報

- [カーネルリソースコントローラーの概要](#)
- [cgroups\(7\) の man ページ](#)
- [コントロールグループ内の systemd のロール](#)

23.2. カーネルリソースコントローラーの概要

カーネルリソースコントローラーは、コントロールグループの機能を有効化します。RHEL 8 は、**コントロールグループバージョン 1 (cgroups-v1)** および **コントロールグループバージョン 2 (cgroups-v2)** のさまざまなコントローラーをサポートします。

コントロールグループサブシステムとも呼ばれるリソースコントローラーは、1つのリソース (CPU 時間、メモリー、ネットワーク帯域幅、ディスク I/O など) を表すカーネルサブシステムです。Linux カーネルは、**systemd** サービスマネージャーによって自動的にマウントされるリソースコントローラーの範囲を提供します。現在マウントされているリソースコントローラーの一覧は、**/proc/cgroups** ファイルで確認できます。

cgroups-v1 で利用可能なコントローラー

blkio

ブロックデバイスへの入出力アクセスを制限します。

cpu

コントロールグループのタスク用の Completely Fair Scheduler (CFS) パラメーターを調整します。**cpu** コントローラーは、同じマウント上の **cpuacct** コントローラーとともにマウントされません。

cpuacct

コントロールグループ内のタスクが使用する CPU リソースに関する自動レポートを作成します。**cpuacct** コントローラーは、同じマウント上の **cpu** コントローラーとともにマウントされません。

cpuset

コントロールグループタスクが、指定された CPU のサブセットでのみ実行されるように制限し、指定されたメモリーノードでのみメモリーを使用するようにタスクに指示します。

devices

コントロールグループ内のタスクのデバイスへのアクセスを制御します。

freezer

コントロールグループ内のタスクを一時停止または再開します。

memory

コントロールグループ内のタスクによるメモリー使用の制限を設定し、それらのタスクが使用したメモリーリソースに関する自動レポートを生成します。

net_cls

特定のコントロールグループタスクから発信されたパケットを識別するために Linux トラフィック

コントローラー (**tc** コマンド) を有効化するクラス識別子 (**classid**) でネットワークパケットをタグ付けします。**net_cls** のサブシステム **net_filter** (iptables) でも、このタグを使用して、そのようなパケットに対するアクションを実行することができます。**net_filter** は、ファイアウォール識別子 (**fwid**) でネットワークソケットをタグ付けします。これにより、Linux ファイアウォールは、(**iptables** コマンドを使用して) 特定のコントロールグループタスクから発信されたパケットを識別できるようになります。

net_prio

ネットワークトラフィックの優先度を設定します。

pids

コントロールグループ内の多数のプロセスとその子に制限を設定します。

perf_event

perf パフォーマンス監視およびレポートユーティリティーにより、監視するタスクをグループ化します。

rdma

コントロールグループ内の Remote Direct Memory Access/InfiniB 固有リソースに制限を設定します。

hugetlb

コントロールグループ内のタスクによる大容量の仮想メモリーページの使用を制限する場合に使用できます。

cgroups-v2 で利用可能なコントローラー

io

ブロックデバイスへの入出力アクセスを制限します。

memory

コントロールグループ内のタスクによるメモリー使用の制限を設定し、それらのタスクが使用したメモリーリソースに関する自動レポートを生成します。

pids

コントロールグループ内の多数のプロセスとその子に制限を設定します。

rdma

コントロールグループ内の Remote Direct Memory Access/InfiniB 固有リソースに制限を設定します。

cpu

コントロールグループのタスクの Completely Fair Scheduler (CFS) パラメーターを調整し、コントロールグループのタスクで使用される CPU リソースに関する自動レポートを作成します。

cpuset

コントロールグループタスクが、指定された CPU のサブセットでのみ実行されるように制限し、指定されたメモリーノードでのみメモリーを使用するようにタスクに指示します。新しいパーティション機能により、コア機能 (**cpus{,.effective}**, **mems{,.effective}**) のみがサポートされます。

perf_event

perf パフォーマンス監視およびレポートユーティリティーにより、監視するタスクをグループ化します。**perf_event** は v2 階層で自動的に有効になります。



重要

リソースコントローラーは、**cgroups-v1** 階層または **cgroups-v2** 階層のいずれかで使用できますが、両方を同時に使用することはできません。

関連情報

- [cgroups\(7\)](#) の man ページ
- [/usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups-v1/](#) ディレクトリー内のドキュメント ([kernel-doc](#) パッケージをインストールした後)。

23.3. 名前空間の概要

名前空間は、ソフトウェアオブジェクトを整理および特定するための最も重要な方法の1つです。

名前空間は、グローバルシステムリソース (マウントポイント、ネットワークデバイス、ホスト名など) を抽象化してラップします。これにより、グローバルリソースの独自の分離されたインスタンスを含む名前空間内のプロセスに表示されます。名前空間を使用する最も一般的なテクノロジーの1つとしてコンテナが挙げられます。

特定のグローバルリソースへの変更は、その名前空間のプロセスにのみ表示され、残りのシステムまたは他の名前空間には影響しません。

プロセスがどの名前空間に所属するかを確認するには、[/proc/<PID>/ns/](#) ディレクトリーのシンボリックリンクを確認します。

表23.1 分離するサポート対象の名前空間およびリソース

名前空間	分離
Mount	マウントポイント
UTS	ホスト名および NIS ドメイン名
IPC	System V IPC、POSIX メッセージキュー
PID	プロセス ID
Network	ネットワークデバイス、スタック、ポートなど。
User	ユーザーおよびグループ ID
Control groups	コントロールグループの root ディレクトリー

関連情報

- [namespaces\(7\)](#) および [cgroup_namespaces\(7\)](#) の man ページ
- [コントロールグループの概要](#)

23.4. CGROUPS-V1 を使用したアプリケーションへの CPU 制限の設定

コントロールグループバージョン 1 ([cgroups-v1](#)) を使用してアプリケーションに対する CPU 制限を設定するには、[/sys/fs/](#) 仮想ファイルシステムを使用します。

前提条件

- root 権限がある。
- CPU 消費を制限するアプリケーションがある。
- **cgroups-v1** コントローラーがマウントされていることを確認している。

```
# mount -l | grep cgroup
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,seclabel,mode=755)
cgroup on /sys/fs/cgroup/systemd type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,xattr,release_agent=/usr/lib/systemd/systemd-
cgroups-agent,name=systemd)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,cpu,cpuacct)
cgroup on /sys/fs/cgroup/perf_event type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,perf_event)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,pids)
...
```

手順

1. CPU 消費を制限するアプリケーションのプロセス ID (PID) を特定します。

```
# top
top - 11:34:09 up 11 min, 1 user, load average: 0.51, 0.27, 0.22
Tasks: 267 total, 3 running, 264 sleeping, 0 stopped, 0 zombie
%Cpu(s): 49.0 us, 3.3 sy, 0.0 ni, 47.5 id, 0.0 wa, 0.2 hi, 0.0 si, 0.0 st
MiB Mem : 1826.8 total, 303.4 free, 1046.8 used, 476.5 buff/cache
MiB Swap: 1536.0 total, 1396.0 free, 140.0 used. 616.4 avail Mem

  PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6955 root    20  0 228440 1752 1472 R 99.3  0.1   0:32.71 sha1sum
 5760 jdoe    20  0 3603868 205188 64196 S  3.7 11.0   0:17.19 gnome-shell
 6448 jdoe    20  0 743648 30640 19488 S  0.7  1.6   0:02.73 gnome-terminal-
    1 root    20  0 245300 6568 4116 S  0.3  0.4   0:01.87 systemd
 505 root    20  0    0    0    0 l 0.3  0.0   0:00.75 kworker/u4:4-events_unbound
...
```

この **top** プログラムの出力例は、説明用のアプリケーション **sha1sum** (PID 6955) が CPU リソースを大量に消費することを示しています。

2. **cpu** リソースコントローラーディレクトリーにサブディレクトリーを作成します。

```
# mkdir /sys/fs/cgroup/cpu/Example/
```

このディレクトリーはコントロールグループを表します。ここに特定のプロセスを配置して、そのプロセスに特定の CPU 制限を適用できます。同時に、いくつかの **cgroups-v1** インターフェイスファイルと **cpu** コントローラー固有のファイルがディレクトリーに作成されます。

3. **オプション**: 新しく作成されたコントロールグループを確認します。

```
# ll /sys/fs/cgroup/cpu/Example/
-rw-r--r--. 1 root root 0 Mar 11 11:42 cgroup.clone_children
-rw-r--r--. 1 root root 0 Mar 11 11:42 cgroup.procs
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.stat
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage
```

```

-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_all
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_sys
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_user
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_sys
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_user
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.cfs_period_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.cfs_quota_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.rt_period_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.rt_runtime_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.shares
-r--r--r--. 1 root root 0 Mar 11 11:42 cpu.stat
-rw-r--r--. 1 root root 0 Mar 11 11:42 notify_on_release
-rw-r--r--. 1 root root 0 Mar 11 11:42 tasks

```

この出力例は、特定の設定や制限を表す **cpuacct.usage**、**cpu.cfs_period_us** などのファイルを示しています。これは、**Example** コントロールグループのプロセスに設定できます。各ファイル名の前に、そのファイルが属するコントロールグループコントローラーの名前が接頭辞として追加されることに注意してください。

デフォルトでは、新しく作成されたコントロールグループは、システムの CPU リソース全体へのアクセスを制限なしで継承します。

4. コントロールグループの CPU 制限を設定します。

```

# echo "1000000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
# echo "200000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us

```

- **cpu.cfs_period_us** ファイルは、コントロールグループの CPU リソースへのアクセスを再割り当てする頻度をマイクロ秒 (μs 、ここでは "us" で表します) 単位で表します。上限は 1,000,000 マイクロ秒で、下限は 1,000 マイクロ秒です。
- **cpu.cfs_quota_us** ファイルは、(**cpu.cfs_period_us** で定義されるように) コントロールグループのすべてのプロセスを 1 期間中に実行できる合計時間をマイクロ秒単位で表します。1 期間中にコントロールグループ内のプロセスがクォータで指定された時間をすべて使いきると、残りの時間はスロットルされて、次の期間まで実行できなくなります。下限は 1000 マイクロ秒です。

上記のコマンド例は、CPU 時間制限を設定して、**Example** コントロールグループでまとめられているすべてのプロセスは、1 秒ごと (**cpu.cfs_period_us** に定義されている) に、0.2 秒間だけ (**cpu.cfs_quota_us** に定義されている) 実行できるようにしています。

5. オプション: 制限を検証します。

```

# cat /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
/sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
1000000
200000

```

6. アプリケーションの PID を **Example** コントロールグループに追加します。

```

# echo "6955" > /sys/fs/cgroup/cpu/Example/cgroup.procs

```

このコマンドによって、特定のアプリケーションが **Example** コントロールグループのメンバーとなり、**Example** コントロールグループに設定された CPU 制限を超えないようになります。PID は、システム内の既存のプロセスを表します。この PID 6955 は、プロセス **sha1sum**

`/dev/zero &` に割り当てられており、**cpu** コントローラーの使用例を示すために使用されています。

検証

1. アプリケーションが指定のコントロールグループで実行されていることを確認します。

```
# cat /proc/6955/cgroup
12:cpuset:/
11:hugetlb:/
10:net_cls,net_prio:/
9:memory:/user.slice/user-1000.slice/user@1000.service
8:devices:/user.slice
7:blkio:/
6:freezer:/
5:rdma:/
4:pids:/user.slice/user-1000.slice/user@1000.service
3:perf_event:/
2:cpu,cpuacct:/Example
1:name=systemd:/user.slice/user-1000.slice/user@1000.service/gnome-terminal-server.service
```

この出力例は、目的のアプリケーションのプロセスが **Example** コントロールグループで実行され、アプリケーションのプロセスに CPU 制限が適用されることを示しています。

2. スロットルしたアプリケーションの現在の CPU 使用率を特定します。

```
# top
top - 12:28:42 up 1:06, 1 user, load average: 1.02, 1.02, 1.00
Tasks: 266 total, 6 running, 260 sleeping, 0 stopped, 0 zombie
%Cpu(s): 11.0 us, 1.2 sy, 0.0 ni, 87.5 id, 0.0 wa, 0.2 hi, 0.0 si, 0.2 st
MiB Mem : 1826.8 total, 287.1 free, 1054.4 used, 485.3 buff/cache
MiB Swap: 1536.0 total, 1396.7 free, 139.2 used. 608.3 avail Mem

  PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6955 root    20  0 228440 1752 1472 R 20.6  0.1  47:11.43 sha1sum
 5760 jdoe    20  0 3604956 208832 65316 R  2.3 11.2   0:43.50 gnome-shell
 6448 jdoe    20  0 743836 31736 19488 S  0.7  1.7   0:08.25 gnome-terminal-
 505 root    20  0    0    0    0 I 0.3  0.0   0:03.39 kworker/u4:4-events_unbound
 4217 root    20  0 74192 1612 1320 S  0.3  0.1   0:01.19 spice-vdagentd
...
```

PID 6955 の CPU 消費が 99% から 20% に減少していることに注意してください。



注記

`cpu.cfs_period_us` および `cpu.cfs_quota_us` に対応する `cgroups-v2` は、`cpu.max` ファイルです。`cpu.max` ファイルは、`cpu` コントローラーから入手できます。

関連情報

- [コントロールグループについて](#)
- [Linux カーネルリソースコントローラーとは](#)

- **cgroups(7)**、**sysfs(5)** の man ページ

第24章 CGROUPS-V2 を使用したアプリケーションへの CPU 時間配分の制御

一部のアプリケーションが CPU 時間を多く使用し過ぎて、環境全体の健全性に悪影響を与える場合があります。アプリケーションを **コントロールグループバージョン 2 (cgroups-v2)** に配置し、それらのコントロールグループの CPU 制限を設定できます。その結果、アプリケーションの CPU 消費を調整できます。

ユーザーには、コントロールグループに割り当てられる CPU 時間の配分を調整する方法が 2 つあります。

- **CPU 帯域幅** の設定 (**cpu.max** コントローラーファイルの編集)
- **CPU の重み** の設定 (**cpu.weight** コントローラーファイルの編集)

24.1. CGROUPS-V2 のマウント

RHEL 8 は、システムの起動プロセス中に、デフォルトで **cgroup-v1** 仮想ファイルシステムをマウントします。**cgroup-v2** 機能を使用してアプリケーションのリソースを制限するには、システムを手動で設定します。

前提条件

- root 権限がある。

手順

1. **systemd** システムおよびサービスマネージャーによるシステムブート中に、デフォルトで **cgroups-v2** をマウントするようにシステムを設定します。

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --  
args="systemd.unified_cgroup_hierarchy=1"
```

これにより、必要なカーネルコマンドラインパラメーターが現在のブートエントリーに追加されます。

systemd.unified_cgroup_hierarchy=1 パラメーターをすべてのカーネルブートエントリーに追加するには、次の手順に従います。

```
# grubby --update-kernel=ALL --args="systemd.unified_cgroup_hierarchy=1"
```

2. システムを再起動して、変更を有効にします。

検証手順

1. 必要に応じて、**cgroups-v2** ファイルシステムがマウントされていることを確認します。

```
# mount -l | grep cgroup  
cgroup2 on /sys/fs/cgroup type cgroup2  
(rw,nosuid,nodev,noexec,relatime,seclabel,nsdelegate)
```

cgroups-v2 ファイルシステムが **/sys/fs/cgroup/** ディレクトリーに正常にマウントされました。

- 必要に応じて、`/sys/fs/cgroup/` ディレクトリーの内容を確認します。

```
# ll /sys/fs/cgroup/
-r--r--r--. 1 root root 0 Apr 29 12:03 cgroup.controllers
-rw-r--r--. 1 root root 0 Apr 29 12:03 cgroup.max.depth
-rw-r--r--. 1 root root 0 Apr 29 12:03 cgroup.max.descendants
-rw-r--r--. 1 root root 0 Apr 29 12:03 cgroup.procs
-r--r--r--. 1 root root 0 Apr 29 12:03 cgroup.stat
-rw-r--r--. 1 root root 0 Apr 29 12:18 cgroup.subtree_control
-rw-r--r--. 1 root root 0 Apr 29 12:03 cgroup.threads
-rw-r--r--. 1 root root 0 Apr 29 12:03 cpu.pressure
-r--r--r--. 1 root root 0 Apr 29 12:03 cpuset.cpus.effective
-r--r--r--. 1 root root 0 Apr 29 12:03 cpuset.mems.effective
-r--r--r--. 1 root root 0 Apr 29 12:03 cpu.stat
drwxr-xr-x. 2 root root 0 Apr 29 12:03 init.scope
-rw-r--r--. 1 root root 0 Apr 29 12:03 io.pressure
-r--r--r--. 1 root root 0 Apr 29 12:03 io.stat
-rw-r--r--. 1 root root 0 Apr 29 12:03 memory.pressure
-r--r--r--. 1 root root 0 Apr 29 12:03 memory.stat
drwxr-xr-x. 69 root root 0 Apr 29 12:03 system.slice
drwxr-xr-x. 3 root root 0 Apr 29 12:18 user.slice
```

`/sys/fs/cgroup/` ディレクトリーはルートコントロールグループとも呼ばれ、デフォルトでは、インターフェイスファイル (`cgroup` で始まる) と、`cpuset.cpus.effective` などのコントローラー固有のファイルが含まれます。さらに、`/sys/fs/cgroup/init.scope`、`/sys/fs/cgroup/system.slice`、および `/sys/fs/cgroup/user.slice` など、`systemd` に関連するディレクトリーがいくつかあります。

関連情報

- [コントロールグループについて](#)
- [Linux カーネルリソースコントローラーとは](#)
- [cgroups\(7\)](#)、[sysfs\(5\)](#) の man ページ

24.2. CPU 時間配分のための CGROUP の準備

アプリケーションの CPU 消費を制御するには、特定の CPU コントローラーを有効にして専用のコントロールグループを作成する必要があります。`cgroup` ファイルの組織的な明確さを維持するために、`/sys/fs/cgroup/` ルートコントロールグループ内に少なくとも 2 つのレベルのサブコントロールグループを作成することが推奨されます。

前提条件

- root 権限がある。
- 制御するプロセスの PID を把握している。
- `cgroups-v2` ファイルシステムをマウントしている。詳細は、[cgroups-v2 のマウント](#) を参照してください。

手順

1. CPU 消費を制限するアプリケーションのプロセス ID (PID) を特定します。

```
# top
```

```
Tasks: 104 total, 3 running, 101 sleeping, 0 stopped, 0 zombie
%Cpu(s): 17.6 us, 81.6 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.8 hi, 0.0 si, 0.0 st
MiB Mem : 3737.4 total, 3312.7 free, 133.3 used, 291.4 buff/cache
MiB Swap: 4060.0 total, 4060.0 free, 0.0 used. 3376.1 avail Mem
```

```

PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM  TIME+ COMMAND
34578 root    20  0 18720 1756 1468 R 99.0  0.0  0:31.09 sha1sum
34579 root    20  0 18720 1772 1480 R 99.0  0.0  0:30.54 sha1sum
   1 root    20  0 186192 13940 9500 S  0.0  0.4  0:01.60 systemd
   2 root    20  0    0    0    0 S  0.0  0.0  0:00.01 kthreadd
   3 root     0 -20    0    0    0 I  0.0  0.0  0:00.00 rcu_gp
   4 root     0 -20    0    0    0 I  0.0  0.0  0:00.00 rcu_par_gp
...

```

この出力例は、**PID 34578** および **34579**(**sha1sum** の2つのアプリケーションとして識別) が多くのリソース (具体的には CPU) を消費していることを示しています。いずれも **cgroups-v2** 機能の管理に使用するアプリケーションの例です。

2. **cpu** および **cpuset** コントローラーが **/sys/fs/cgroup/cgroup.controllers** ファイルで利用可能であることを確認します。

```
# cat /sys/fs/cgroup/cgroup.controllers
cpuset cpu io memory hugetlb pids rdma
```

3. CPU 関連のコントローラーを有効にします。

```
# echo "+cpu" >> /sys/fs/cgroup/cgroup.subtree_control
# echo "+cpuset" >> /sys/fs/cgroup/cgroup.subtree_control
```

これらのコマンドにより、**/sys/fs/cgroup/** ルートコントロールグループ直下のサブグループに対して **cpu** および **cpuset** コントローラーが有効になります。**サブグループ** で指定した各プロセスに対して、基準に基づいてコントロールチェックを適用できます。

ユーザーは任意のレベルの **cgroup.subtree_control** ファイルの内容を読み取り、直下のサブグループで有効にするコントローラーについて把握することができます。



注記

デフォルトでは、ルートコントロールグループの **/sys/fs/cgroup/cgroup.subtree_control** ファイルには **memory** と **pids** コントローラーが含まれます。

4. **/sys/fs/cgroup/Example/** ディレクトリーを作成します。

```
# mkdir /sys/fs/cgroup/Example/
```

/sys/fs/cgroup/Example/ ディレクトリーはサブグループを定義します。また、前の手順では、このサブグループの **cpu** コントローラーおよび **cpuset** コントローラーを有効にしました。

/sys/fs/cgroup/Example/ ディレクトリーを作成すると、**cgroups-v2** インターフェイスファイルと **cpu** および **cpuset** のコントローラー固有のファイルがディレクトリーに自動的に作成されます。**/sys/fs/cgroup/Example/** ディレクトリーには、**memory** および **pids** コントローラー

用のコントローラー固有のファイルも含まれます。

5. 必要に応じて、新たに作成されたサブコントロールグループを確認します。

```
# ll /sys/fs/cgroup/Example/
-r--r--r--. 1 root root 0 Jun  1 10:33 cgroup.controllers
-r--r--r--. 1 root root 0 Jun  1 10:33 cgroup.events
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.freeze
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.max.depth
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.max.descendants
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.procs
-r--r--r--. 1 root root 0 Jun  1 10:33 cgroup.stat
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.subtree_control
...
-rw-r--r--. 1 root root 0 Jun  1 10:33 cpuset.cpus
-r--r--r--. 1 root root 0 Jun  1 10:33 cpuset.cpus.effective
-rw-r--r--. 1 root root 0 Jun  1 10:33 cpuset.cpus.partition
-rw-r--r--. 1 root root 0 Jun  1 10:33 cpuset.mems
-r--r--r--. 1 root root 0 Jun  1 10:33 cpuset.mems.effective
-r--r--r--. 1 root root 0 Jun  1 10:33 cpu.stat
-rw-r--r--. 1 root root 0 Jun  1 10:33 cpu.weight
-rw-r--r--. 1 root root 0 Jun  1 10:33 cpu.weight.nice
...
-r--r--r--. 1 root root 0 Jun  1 10:33 memory.events.local
-rw-r--r--. 1 root root 0 Jun  1 10:33 memory.high
-rw-r--r--. 1 root root 0 Jun  1 10:33 memory.low
...
-r--r--r--. 1 root root 0 Jun  1 10:33 pids.current
-r--r--r--. 1 root root 0 Jun  1 10:33 pids.events
-rw-r--r--. 1 root root 0 Jun  1 10:33 pids.max
```

出力例には、**cpuset.cpus** や **cpu.max** などのファイルが表示されます。これらのファイルは、**cpuset** コントローラーおよび **cpu** コントローラーに固有のもので、**cpuset** および **cpu** コントローラーは、**/sys/fs/cgroup/cgroup.subtree_control** ファイルを使用して、ルート (**/sys/fs/cgroup/**) の直下のサブコントロールグループに対して手動で有効にされます。

ディレクトリーには **cgroup.procs** または **cgroup.controllers** などの一般的な **cgroup** コントロールインターフェイスファイルがありますが、これは有効なコントローラーに関係なく、すべてのコントロールグループに共通のもので、

memory.high および **pids.max** などのファイルは、**memory** および **pids** コントローラーに関連し、ルートコントロールグループ (**/sys/fs/cgroup/**) にあり、常にデフォルトで有効になります。

デフォルトでは、新しく作成されたサブグループは、制限なしで、システムのすべての CPU およびメモリーリソースへのアクセスを継承します。

6. **/sys/fs/cgroup/Example/** の CPU 関連のコントローラーを有効にし、CPU にのみ関連するコントローラーを取得します。

```
# echo "+cpu" >> /sys/fs/cgroup/Example/cgroup.subtree_control
# echo "+cpuset" >> /sys/fs/cgroup/Example/cgroup.subtree_control
```

これらのコマンドにより、直下のサブコントロールグループに、(**memory** または **pids** コントローラーではなく) CPU 時間の配分の調整に関するコントローラー **だけ** が設定されるようになります。

7. `/sys/fs/cgroup/Example/tasks/` ディレクトリーを作成します。

```
# mkdir /sys/fs/cgroup/Example/tasks/
```

`/sys/fs/cgroup/Example/tasks/` ディレクトリーは、**cpu** および **cpuset** コントローラーにのみ関連するファイルを持つサブグループを定義します。

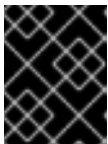
8. 必要に応じて、別のサブコントロールグループを確認します。

```
# ll /sys/fs/cgroup/Example/tasks
-r--r--r--. 1 root root 0 Jun 1 11:45 cgroup.controllers
-r--r--r--. 1 root root 0 Jun 1 11:45 cgroup.events
-rw-r--r--. 1 root root 0 Jun 1 11:45 cgroup.freeze
-rw-r--r--. 1 root root 0 Jun 1 11:45 cgroup.max.depth
-rw-r--r--. 1 root root 0 Jun 1 11:45 cgroup.max.descendants
-rw-r--r--. 1 root root 0 Jun 1 11:45 cgroup.procs
-r--r--r--. 1 root root 0 Jun 1 11:45 cgroup.stat
-rw-r--r--. 1 root root 0 Jun 1 11:45 cgroup.subtree_control
-rw-r--r--. 1 root root 0 Jun 1 11:45 cgroup.threads
-rw-r--r--. 1 root root 0 Jun 1 11:45 cgroup.type
-rw-r--r--. 1 root root 0 Jun 1 11:45 cpu.max
-rw-r--r--. 1 root root 0 Jun 1 11:45 cpu.pressure
-rw-r--r--. 1 root root 0 Jun 1 11:45 cpuset.cpus
-r--r--r--. 1 root root 0 Jun 1 11:45 cpuset.cpus.effective
-rw-r--r--. 1 root root 0 Jun 1 11:45 cpuset.cpus.partition
-rw-r--r--. 1 root root 0 Jun 1 11:45 cpuset.mems
-r--r--r--. 1 root root 0 Jun 1 11:45 cpuset.mems.effective
-r--r--r--. 1 root root 0 Jun 1 11:45 cpu.stat
-rw-r--r--. 1 root root 0 Jun 1 11:45 cpu.weight
-rw-r--r--. 1 root root 0 Jun 1 11:45 cpu.weight.nice
-rw-r--r--. 1 root root 0 Jun 1 11:45 io.pressure
-rw-r--r--. 1 root root 0 Jun 1 11:45 memory.pressure
```

9. CPU 時間を制御するプロセスが同じ CPU で競合していることを確認します。

```
# echo "1" > /sys/fs/cgroup/Example/tasks/cpuset.cpus
```

上記のコマンドは、**Example/tasks** サブコントロールグループに配置されるプロセスが同じ CPU 上で競合するようにします。この設定は、**cpu** コントローラーをアクティベートするのに重要です。



重要

cpu コントローラーは、該当のサブコントロールグループに、同じ CPU の CPU 時間を取り合うプロセスが 2 つ以上ある場合にのみ、有効になります。

検証手順

1. 必要に応じて、CPU 関連のコントローラーが直下のサブ cgroups に対して有効になっていることを確認します。

```
# cat /sys/fs/cgroup/cgroup.subtree_control
/sys/fs/cgroup/Example/cgroup.subtree_control
cpuset cpu memory pids
```

```
cpuset cpu
```

- オプション:CPU 時間を制御するプロセスが同じ CPU で競合していることを確認します。

```
# cat /sys/fs/cgroup/Example/tasks/cpuset.cpus
1
```

関連情報

- [コントロールグループとは](#)
- [Linux カーネルリソースコントローラーとは](#)
- [cgroups-v2 のマウント](#)
- [cgroups\(7\)](#)、[sysfs\(5\)](#) の man ページ

24.3. CPU 帯域幅の調整によるアプリケーションへの CPU 時間配分の制御

特定の cgroup ツリーの下にあるアプリケーションへの CPU 時間の配分を調整するには、**cpu** コントローラーの関連ファイルに値を割り当てる必要があります。

前提条件

- root 権限がある。
- CPU 時間の配分を制御する 2 つ以上のアプリケーションがある。
- [CPU 時間配分のための cgroup の準備](#) に記載するように、該当するアプリケーションが同じ CPU の CPU 時間を取り合っていることを確認している。
- [cgroups-v2 のマウント](#) で説明されているように、**cgroups-v2** ファイルシステムをマウントしている。
- [CPU 時間配分のための cgroup の準備](#) で説明されているのと同様に、親コントロールグループとサブコントロールグループの両方に、**cpu** および **cpuset** コントローラーを有効にしている。
- 以下の例のように、**/sys/fs/cgroup/** ルートコントロールグループ内に 2 つのレベルの **サブコントロールグループ** を作成していること。

```
...
|
|--- Example
|   |
|   |--- tasks
|
...

```

手順

1. コントロールグループ内のリソース制限を実現するために CPU 帯域幅を設定します。

```
# echo "200000 1000000" > /sys/fs/cgroup/Example/tasks/cpu.max
```

最初の値は、一定の期間にサブグループにある全プロセスをまとめて実行できる許容される時間クォータ (マイクロ秒単位) です。2 番目の値は期間の長さを指定します。

一期間中にコントロールグループ内のプロセスが全体としてこのクォータで指定した時間を使い切ってしまうと、残りの時間がスロットルされて、次の期間まで実行できなくなります。

このコマンドは、`/sys/fs/cgroup/Example/tasks` サブグループの全プロセスが1秒ごとに0.2秒間のみCPUで実行されるように、CPU時間の配分の制御を設定します。つまり、毎秒5分の1です。

2. 必要に応じて、時間クォータを確認します。

```
# cat /sys/fs/cgroup/Example/tasks/cpu.max
200000 1000000
```

3. アプリケーションのPIDを **Example/tasks** サブグループに追加します。

```
# echo "34578" > /sys/fs/cgroup/Example/tasks/cgroup.procs
# echo "34579" > /sys/fs/cgroup/Example/tasks/cgroup.procs
```

上記のコマンド例は、目的のアプリケーションが **Example/tasks** サブグループのメンバーになり、このサブグループに設定されたCPU時間の配分を超えないようにします。

検証手順

1. アプリケーションが指定のコントロールグループで実行されていることを確認します。

```
# cat /proc/34578/cgroup /proc/34579/cgroup
0::/Example/tasks
0::/Example/tasks
```

上記の出力は、**Example/tasks** サブグループで実行される指定されたアプリケーションのプロセスを示しています。

2. スロットリングされたアプリケーションの現在のCPU使用率を確認します。

```
# top
top - 11:13:53 up 23:10, 1 user, load average: 0.26, 1.33, 1.66
Tasks: 104 total, 3 running, 101 sleeping, 0 stopped, 0 zombie
%Cpu(s): 3.0 us, 7.0 sy, 0.0 ni, 89.5 id, 0.0 wa, 0.2 hi, 0.2 si, 0.2 st
MiB Mem : 3737.4 total, 3312.6 free, 133.4 used, 291.4 buff/cache
MiB Swap: 4060.0 total, 4060.0 free, 0.0 used. 3376.0 avail Mem

  PID USER  PR  NI  VIRT  RES  SHR S %CPU %MEM  TIME+ COMMAND
 34578 root   20   0 18720 1756 1468 R 10.0  0.0 37:36.13 sha1sum
 34579 root   20   0 18720 1772 1480 R 10.0  0.0 37:41.22 sha1sum
    1 root   20   0 186192 13940 9500 S  0.0  0.4  0:01.60 systemd
    2 root   20   0     0     0  0 S  0.0  0.0  0:00.01 kthreadd
    3 root    0 -20     0     0  0 I  0.0  0.0  0:00.00 rcu_gp
    4 root    0 -20     0     0  0 I  0.0  0.0  0:00.00 rcu_par_gp
...

```

PID 34578 および **PID 34579** のCPU使用率が10%に減少していることが分かります。**Example/tasks** サブグループは、そのプロセスをまとめて20%のCPU時間に調整します。コントロールグループにプロセスが2つあるため、各プロセスはCPU時間の10%を使用できます。

関連情報

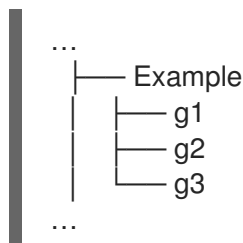
- [コントロールグループについて](#)
- [Linux カーネルリソースコントローラーとは](#)
- [cgroups-v2 のマウント](#)
- [CPU 時間配分のための cgroup の準備](#)
- [cgroups\(7\)、sysfs\(5\) の man ページ](#)

24.4. CPU の重みの調整によるアプリケーションへの CPU 時間配分の制御

特定の cgroup ツリーの下にあるアプリケーションへの CPU 時間の配分を調整するには、**cpu** コントローラーの関連ファイルに値を割り当てる必要があります。

前提条件

- root 権限がある。
- CPU 時間の配分を制御するアプリケーションがある。
- [CPU 時間配分のための cgroup の準備](#) に記載するように、該当するアプリケーションが同じ CPU の CPU 時間を取り合っていることを確認している。
- [cgroups-v2 のマウント](#) で説明されているように、**cgroups-v2** ファイルシステムをマウントしている。
- 次の例のように、**/sys/fs/cgroup/** root control group 内に、**child control groups** グループの 2 階層を作成しました。



- [CPU 時間配分のための cgroup の準備](#) で説明されているのと同様に、親コントロールグループとサブコントロールグループに、**cpu** および **cpuset** コントローラーを有効にしている。

手順

1. コントロールグループ内のリソース制限を実現するために、希望する CPU の重みを設定します。

```
# echo "150" > /sys/fs/cgroup/Example/g1/cpu.weight
# echo "100" > /sys/fs/cgroup/Example/g2/cpu.weight
# echo "50" > /sys/fs/cgroup/Example/g3/cpu.weight
```

2. アプリケーションの PID を **g1**、**g2**、および **g3** サブグループに追加します。

```
# echo "33373" > /sys/fs/cgroup/Example/g1/cgroup.procs
# echo "33374" > /sys/fs/cgroup/Example/g2/cgroup.procs
# echo "33377" > /sys/fs/cgroup/Example/g3/cgroup.procs
```

上記のコマンドの例は、目的のアプリケーションが **Example/g*** 子 cgroup のメンバーになり、それらの cgroup の設定に従って CPU 時間を分散させることを保証します。

実行中のプロセスを持つサブ cgroups(**g1**、**g2**、**g3**) の重みは、親 cgroup のレベルで合算されます (例)。その後、CPU リソースはそれぞれの重みに基づいて相対的に配分されます。

その結果、すべてのプロセスが同時に実行されると、カーネルはそれぞれの cgroup の **cpu.weight** ファイルに基づいて、それぞれのプロセスに比例配分の CPU 時間を割り当てます。

サブ cgroup	cpu.weight ファイル	CPU 時間の割り当て
g1	150	~ 50% (150/300)
g2	100	~ 33% (100/300)
g3	50	~ 16% (50/300)

cpu.weight コントローラーファイルの値はパーセンテージではありません。

1つのプロセスが実行を停止し、cgroup **g2** が実行中のプロセスのない状態になると、計算では cgroup **g2** が省略され、cgroups **g1** および **g3** の重みだけが考慮されます。

サブ cgroup	cpu.weight ファイル	CPU 時間の割り当て
g1	150	~ 75% (150/200)
g3	50	~ 25% (50/200)



重要

サブ cgroup に複数の実行中のプロセスがある場合は、それぞれの cgroup に割り当てられる CPU 時間は、その cgroup のメンバープロセスに均等に配分されます。

検証

1. アプリケーションが指定のコントロールグループで実行されていることを確認します。

```
# cat /proc/33373/cgroup /proc/33374/cgroup /proc/33377/cgroup
0::/Example/g1
0::/Example/g2
0::/Example/g3
```

コマンド出力は、**Example/g*** サブ cgroups で実行される指定されたアプリケーションのプロセスを示しています。

2. スロットリングされたアプリケーションの現在の CPU 使用率を確認します。

```
# top
top - 05:17:18 up 1 day, 18:25, 1 user, load average: 3.03, 3.03, 3.00
```

```
Tasks: 95 total,  4 running, 91 sleeping,  0 stopped,  0 zombie
%Cpu(s): 18.1 us, 81.6 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.3 hi,  0.0 si,  0.0 st
MiB Mem : 3737.0 total, 3233.7 free,  132.8 used,  370.5 buff/cache
MiB Swap: 4060.0 total, 4060.0 free,   0.0 used. 3373.1 avail Mem
```

```

PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM  TIME+ COMMAND
33373 root    20  0 18720 1748 1460 R 49.5  0.0 415:05.87 sha1sum
33374 root    20  0 18720 1756 1464 R 32.9  0.0 412:58.33 sha1sum
33377 root    20  0 18720 1860 1568 R 16.3  0.0 411:03.12 sha1sum
 760 root    20  0 416620 28540 15296 S  0.3  0.7  0:10.23 tuned
   1 root    20  0 186328 14108 9484 S  0.0  0.4  0:02.00 systemd
   2 root    20  0    0    0    0 S  0.0  0.0  0:00.01 kthread

```

...



注記

わかりやすくするために、すべてのサンプルプロセスを単一の CPU で実行するように強制しました。CPU の重みは、複数の CPU で使用される場合にも同じ原則を適用します。

PID 33373、**PID 33374**、および **PID 33377** の CPU リソースは、対応するサブ cgroups に割り当てた重み 150、100、50 に基づいて割り当てられたことが分かります。重みは、各アプリケーションの CPU 時間の約 50%、33%、および 16% の配分に対応します。

関連情報

- [コントロールグループについて](#)
- [Linux カーネルリソースコントローラーとは](#)
- [cgroups-v2 のマウント](#)
- [CPU 時間配分のための cgroup の準備](#)
- [Resource Distribution Models](#)
- [cgroups\(7\)](#)、[sysfs\(5\)](#) の man ページ

第25章 SYSTEMD によるコントロールグループバージョン1の使用

cgroup は、**systemd** システムとサービスマネージャー、およびそれらが提供するユーティリティーを使用して管理できます。これは、**cgroups** 管理の推奨される方法でもあります。

25.1. コントロールグループバージョン1における SYSTEMD のロール

RHEL 8 では、**cgroup** 階層のシステムを **systemd** ユニットツリーにバインドすることにより、リソース管理設定をプロセスレベルからアプリケーションレベルに移行します。したがって、システムリソースは、**systemctl** コマンドを使用するか、**systemd** ユニットファイルを変更して管理できます。

デフォルトでは、**systemd** システムおよびサービスマネージャーは、スライスユニット、スコープユニット、およびサービスユニットを使用して、コントロールグループ内のプロセスを整理および構造化します。**systemctl** コマンドを使用すると、カスタムのスライスを作成してこの構造をさらに変更できます。また、**systemd** は、重要なカーネルリソースコントローラーの階層を **/sys/fs/cgroup/** ディレクトリーに自動的にマウントします。

systemd の3つのユニットタイプは、リソース制御に使用します。

- **Service** - ユニット設定ファイルに従って **systemd** が起動したプロセスまたはプロセスのグループ。サービスは、指定したプロセスをカプセル化して、1つのセットとして起動および停止できるようにします。サービスの名前は以下の方法で指定されます。

```
<name>.service
```

- **スコープ** - 外部で作成されたプロセスのグループ。スコープは、**fork()** 関数を介して任意のプロセスで開始および停止されたプロセスをカプセル化し、ランタイム時に **systemd** で登録します。たとえば、ユーザーセッション、コンテナ、および仮想マシンはスコープとして処理されます。スコープの名前は以下のように指定されます。

```
<name>.scope
```

- **スライス** - 階層的に編成されたユニットのグループ。スライスは、スコープおよびサービスを配置する階層を編成します。実際のプロセスはスコープまたはサービスに含まれます。スライスユニットの名前はすべて、階層内の場所へのパスに対応します。ハイフン (-) 文字は、**-.slice** ルートスライスからのスライスへのパスコンポーネントの区切り文字として機能します。以下の例では、下記の点を前提としています。

```
<parent-name>.slice
```

parent-name.slice は **parent.slice** のサブスライスで、これは **-.slice** root スライスのサブスライスです。**parent-name.slice** には、**parent-name-name2.slice** という名前の独自のサブスライスを指定できます。

サービス、スコープ、スライスユニットは、コントロールグループ階層のオブジェクトに直接マッピングされます。これらのユニットがアクティブになると、ユニット名から構築されるグループパスを制御するように直接マッピングされます。

以下は、コントロールグループ階層の省略形の例です。

```
Control group /:
-.slice
├─user.slice
```



```

| | | | | user-42.slice
| | | | |   | session-c1.scope
| | | | |   |   | 967 gdm-session-worker [pam/gdm-launch-environment]
| | | | |   |   | 1035 /usr/libexec/gdm-x-session gnome-session --autostart
| | | | |   | /usr/share/gdm/greeter/autostart
| | | | |   |   | 1054 /usr/libexec/Xorg vt1 -displayfd 3 -auth /run/user/42/gdm/Xauthority -background none
| | | | |   | -noreset -keeppty -verbose 3
| | | | |   |   | 1212 /usr/libexec/gnome-session-binary --autostart /usr/share/gdm/greeter/autostart
| | | | |   |   | 1369 /usr/bin/gnome-shell
| | | | |   |   | 1732 ibus-daemon --xim --panel disable
| | | | |   |   | 1752 /usr/libexec/ibus-dconf
| | | | |   |   | 1762 /usr/libexec/ibus-x11 --kill-daemon
| | | | |   |   | 1912 /usr/libexec/gsd-xsettings
| | | | |   |   | 1917 /usr/libexec/gsd-a11y-settings
| | | | |   |   | 1920 /usr/libexec/gsd-clipboard
| | | | |
| | | | | ...
| | | | | | init.scope
| | | | | |   | 1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
| | | | | | | system.slice
| | | | | | |   | rngd.service
| | | | | | |   |   | 800 /sbin/rngd -f
| | | | | | |   | systemd-udev.service
| | | | | | |   |   | 659 /usr/lib/systemd/systemd-udev
| | | | | | |   | chronyd.service
| | | | | | |   |   | 823 /usr/sbin/chronyd
| | | | | | |   | auditd.service
| | | | | | |   |   | 761 /sbin/auditd
| | | | | | |   |   | 763 /usr/sbin/sedispach
| | | | | | |   | accounts-daemon.service
| | | | | | |   |   | 876 /usr/libexec/accounts-daemon
| | | | | | |   | example.service
| | | | | | |   |   | 929 /bin/bash /home/jdoe/example.sh
| | | | | | |   |   | 4902 sleep 1
| | | | | | |
| | | | | | | ...

```

上記の例では、サービスおよびスコープにプロセスが含まれており、独自のプロセスを含まないスライスに置かれていることを示しています。

関連情報

- [Linux カーネルリソースコントローラーとは](#)
- [systemd.resource-control\(5\)](#)、[cgroups\(7\)](#)、[fork\(\)](#)、[fork\(2\)](#) man ページ

25.2. 一時的なコントロールグループの作成

一時的な **cgroups** は、ランタイム時にユニット (サービスまたはスコープ) が消費するリソースに制限を設定します。

手順

- 一時的なコントロールグループを作成するには、以下の形式で **systemd-run** コマンドを使用します。

```
# systemd-run --unit=<name> --slice=<name>.slice <command>
```

このコマンドは、一時的なサービスまたはスコープユニットを作成し、開始し、そのユニットでカスタムコマンドを実行します。

- **--unit=<name>** オプションは、ユニットに名前を指定します。**--unit** が指定されていないと、名前は自動的に生成されます。
- **--slice=<name>.slice** オプションは、サービスまたはスコープユニットを指定のスライスのメンバーにします。**<name>.slice** は、既存のスライスの名前 (**systemctl -t slice** の出力に表示) に置き換えるか、一意の名前を指定して新規スライスを作成します。デフォルトでは、サービスおよびスコープは **system.slice** のメンバーとして作成されます。
- **<command>** は、サービスまたはスコープユニットに入力するコマンドに置き換えます。以下のような、サービスまたはスコープが正常に作成され開始したことを確認するメッセージが表示されます。

```
# Running as unit <name>.service
```

- **オプション:** ランタイム情報を収集するため、プロセスが終了した後もユニットを実行したままにします。

```
# systemd-run --unit=<name> --slice=<name>.slice --remain-after-exit <command>
```

このコマンドは、一時的なサービスユニットを作成して起動し、そのユニットでカスタムコマンドを実行します。**--remain-after-exit** オプションを使用すると、プロセスの終了後もサービスが実行し続けます。

関連情報

- [コントロールグループとは](#)
- [コントロールグループ内の systemd のロール](#)
- RHEL での [systemd の管理](#)
- [systemd-run\(1\) の man ページ](#)

25.3. 永続的なコントロールグループの作成

永続的なコントロールグループをサービスに割り当てるには、そのユニット設定ファイルを編集する必要があります。この設定はシステム再起動後も保存されるので、これを使用して自動的に起動されたサービスを管理できます。

手順

- 永続的なコントロールグループを作成するには、以下のように入力します。

```
# systemctl enable <name>.service
```

上記のコマンドは、ユニット設定ファイルを自動的に **/usr/lib/systemd/system/** ディレクトリに作成し、デフォルトで **<name>.service** を **system.slice** ユニットに割り当てます。

関連情報

- [コントロールグループについて](#)

- [コントロールグループ内の systemd のロール](#)
- RHEL での [systemctl によるシステムサービス管理](#)
- [systemd-run\(1\) の man ページ](#)

25.4. コマンドラインでのメモリーリソース制御の設定

プロセスのグループのハードウェアリソースに対するアクセス権限を設定して優先順位を付け、制御する方法の1つとして、コマンドラインインターフェイスでコマンドを実行する方法があります。

手順

- サービスのメモリー使用量を制限するには、以下を実行します。

```
# systemctl set-property example.service MemoryMax=1500K
```

このコマンドは、**example.service** サービスが所属するコントロールグループで実行されるプロセスに対して、1,500 KB のメモリー制限を割り当てます。この設定バリエーションの **MemoryMax** パラメーターは `/etc/systemd/system.control/example.service.d/50-MemoryMax.conf` ファイルで定義さ

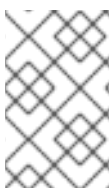
れ、`/sys/fs/cgroup/memory/system.slice/example.service/memory.limit_in_bytes` ファイルの値を制御します。

- 必要に応じて、サービスのメモリー使用量を一時的に制限するには、以下を実行します。

```
# systemctl set-property --runtime example.service MemoryMax=1500K
```

このコマンドは、メモリー制限を **example.service** サービスに即座に割り当てます。 **MemoryMax** パラメーターは、次回起動時まで

`/run/systemd/system.control/example.service.d/50-MemoryMax.conf` ファイルで定義されます。再起動すると、`/run/systemd/system.control/` ディレクトリー全体と **MemoryMax** が削除されます。



注記

50-MemoryMax.conf ファイルは、メモリー制限を 4096 バイトの倍数 (AMD64 および Intel 64 に固有のカーネルページサイズ) として保存します。実際のバイト数は、CPU アーキテクチャーによって異なります。

関連情報

- [コントロールグループとは](#)
- [Linux カーネルリソースコントローラーとは](#)
- [systemd.resource-control\(5\) と cgroups\(7\) man ページ](#)
- [コントロールグループ内の systemd のロール](#)

25.5. ユニットファイルを使用したメモリーリソース制御の設定

各永続的なユニットは **systemd** システムおよびサービスマネージャーによって監視され、`/usr/lib/systemd/system/` ディレクトリーにユニット設定ファイルがあります。永続的なユニット

のリソース制御設定を変更するには、そのユニット設定ファイルをテキストエディターまたはコマンドラインインターフェイスで手動で変更します。

プロセスのグループのハードウェアリソースに対するアクセス権限を設定して優先順位を付け、制御する方法の1つとして、ユニットファイルを手動で変更する方法があります。

手順

1. サービスのメモリー使用量を制限するには、以下のように `/usr/lib/systemd/system/example.service` ファイルを変更します。

```
...
[Service]
MemoryMax=1500K
...
```

上記の設定では、(**example.service** が含まれる) コントロールグループで実行されるプロセスの最大メモリー消費量に制限があります。



注記

測定単位のキロバイト、メガバイト、ギガバイト、またはテラバイトを指定するには、接尾辞 K、M、G、または T を使用します。

2. すべてのユニット設定ファイルを再読み込みします。

```
# systemctl daemon-reload
```

3. サービスを再起動します。

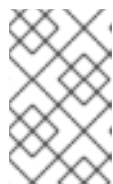
```
# systemctl restart example.service
```

4. システムを再起動します。

5. 必要に応じて、変更が反映されたことを確認します。

```
# cat /sys/fs/cgroup/memory/system.slice/example.service/memory.limit_in_bytes
1536000
```

この出力例では、メモリー消費量が約 1,500 KB に制限されていることを示しています。



注記

memory.limit_in_bytes ファイルは、メモリー制限を 4096 バイトの倍数 (AMD64 および Intel 64 に固有のカーネルページサイズ) として保存します。実際のバイト数は、CPU アーキテクチャーによって異なります。

関連情報

- [コントロールグループについて](#)
- [Linux カーネルリソースコントローラーとは](#)

- [systemd.resource-control\(5\)](#)、[cgroups\(7\)](#) man ページ
- RHEL での [systemctl](#) によるシステムサービス管理
- [コントロールグループバージョン1における systemd のロール](#)

25.6. 一時的なコントロールグループの削除

systemd システムおよびサービスマネージャーを使用して、プロセスのグループのハードウェアリソースへのアクセスを制限して優先順位を付け、制御する必要がなくなった場合に、一時的なコントロールグループ (**cgroup**) を削除できます。

一時的な **cgroups** は、サービスまたはスコープユニットに含まれる全プロセスが完了すると、自動的に解放されます。

手順

- サービスユニットの全プロセスを停止するには、以下を実行します。

```
# systemctl stop <name>.service
```

- ユニットプロセスを1つ以上終了するには、以下を実行します。

```
# systemctl kill <name>.service --kill-who=PID,... --signal=<signal>
```

このコマンドは **--kill-who** オプションを使用して、コントロールグループから終了するプロセスを選択します。複数のプロセスを同時に強制終了するには、PID のコンマ区切りのリストを指定します。**--signal** オプションは、指定されたプロセスに送信する POSIX シグナルのタイプを決定します。デフォルトのシグナルは **SIGTERM** です。

関連情報

- [コントロールグループとは](#)
- [Linux カーネルリソースコントローラーとは](#)
- [systemd.resource-control\(5\)](#) および [cgroups\(7\)](#) man ページ
- [コントロールグループバージョン1における systemd のロール](#)
- RHEL での [systemd の管理](#)

25.7. 永続的なコントロールグループの削除

systemd システムおよびサービスマネージャーを使用して、プロセスのグループのハードウェアリソースへのアクセスを制限して優先順位を付け、制御する必要がなくなった場合に、永続的なコントロールグループおよび永続コントロールグループ (**cgroup**) を削除できます。

永続的な **cgroups** は、サービスまたはスコープユニットが停止または無効化されてその設定ファイルが削除されると解放されます。

手順

1. サービスユニットを停止します。

```
# systemctl stop <name>.service
```

2. サービスユニットを無効にします。

```
# systemctl disable <name>.service
```

3. 関連するユニット設定ファイルを削除します。

```
# rm /usr/lib/systemd/system/<name>.service
```

4. すべてのユニット設定ファイルを再度読み込み、変更を有効にします。

```
# systemctl daemon-reload
```

関連情報

- [コントロールグループとは](#)
- [Linux カーネルリソースコントローラーとは](#)
- [systemd.resource-control\(5\)](#)、[cgroups\(7\)](#)、[systemd.kill\(5\)](#) man ページ
- [コントロールグループ内の systemd のロール](#)
- RHEL での [systemctl によるシステムサービス管理](#)

25.8. SYSTEMD ユニットのリスト表示

systemd システムおよびサービスマネージャーを使用して、そのユニットを一覧表示します。

手順

- **systemctl** ユーティリティを使用して、システム上のすべてのアクティブなユニットをリスト表示します。ターミナルは、次の例のような出力を返します。

```
# systemctl
UNIT                                LOAD  ACTIVE SUB    DESCRIPTION
...
init.scope                          loaded active running  System and Service Manager
session-2.scope                     loaded active running  Session 2 of user jdoe
abrt-ccpp.service                   loaded active exited  Install ABRT coredump hook
abrt-oops.service                   loaded active running  ABRT kernel log watcher
abrt-vmcore.service                 loaded active exited  Harvest vmcores for ABRT
abrt-xorg.service                    loaded active running  ABRT Xorg log watcher
...
-.slice                              loaded active active   Root Slice
machine.slice                        loaded active active   Virtual Machine and Container
Slice system-getty.slice             loaded active active
system-getty.slice
system-lvm2\x2dpvscan.slice          loaded active active   system-
lvm2\x2dpvscan.slice
system-sshd\x2dkeygen.slice          loaded active active   system-
sshd\x2dkeygen.slice
```

```

system-systemd\x2dhibernate\x2dresume.slice      loaded active active  system-
systemd\x2dhibernate\x2dresume>
system-user\x2druntime\x2ddir.slice              loaded active active  system-
user\x2druntime\x2ddir.slice
system.slice                                      loaded active active  System Slice
user-1000.slice                                  loaded active active  User Slice of UID 1000
user-42.slice                                    loaded active active  User Slice of UID 42
user.slice                                        loaded active active  User and Session Slice
...

```

UNIT

コントロールグループ階層内のユニットの位置も反映するユニットの名前です。リソース制御に関連するユニットは、**スライス**、**スコープ** および **サービス** です。

LOAD

ユニット設定ファイルが正しく読み込まれたかどうかを示します。ユニットファイルの読み込みに失敗した場合には、フィールドの状態が **loaded** ではなく **error** になります。ユニットの読み込みの状態は他に **stub**, **merged**, and **masked** があります。

ACTIVE

ユニットのアクティベーションの状態 (概要レベル)。こちらは **SUB** を一般化したものです。

SUB

ユニットのアクティベーションの状態 (詳細レベル)。許容値の範囲は、ユニットタイプによって異なります。

DESCRIPTION

ユニットのコンテンツおよび機能の説明。

- すべてのアクティブなユニットと非アクティブなユニットをリスト表示します。

```
# systemctl --all
```

- 出力の情報量を限定します。

```
# systemctl --type service,masked
```

--type オプションでは、**サービス** および **スライス** などのユニットタイプのコンマ区切りのリスト、または **読み込み済み**、**マスク済み** などのユニットの読み込み状態が必要です。

関連情報

- RHEL での [systemctl によるシステムサービス管理](#)
- [systemd.resource-control\(5\)](#)、[systemd.exec\(5\)](#) man ページ

25.9. SYSTEMD CGROUPS 階層の表示

コントロールグループ (**cgroups**) の階層と、特定の **cgroups** で実行しているプロセスを表示します。

手順

- **systemd-cgls** コマンドを使用して、システム上の **cgroups** 階層全体を表示します。

```

# systemd-cgls
Control group /:
-.slice
├─user.slice
│   └─user-42.slice
│       └─session-c1.scope
│           └─965 gdm-session-worker [pam/gdm-launch-environment]
│               └─1040 /usr/libexec/gdm-x-session gnome-session --autostart
│                   /usr/share/gdm/greeter/autostart
...
├─init.scope
│   └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
└─system.slice
...
├─example.service
│   └─6882 /bin/bash /home/jdoe/example.sh
│       └─6902 sleep 1
├─systemd-journald.service
│   └─629 /usr/lib/systemd/systemd-journald
...

```

この出力例では **cgroups** 階層全体を返します。この階層は、**slices** で形成される最も高いレベルです。

- **systemd-cgls <resource_controller>** コマンドを使用して、リソースコントローラーによってフィルター処理された **cgroups** 階層を表示します。

```

# systemd-cgls memory
Controller memory; Control group /:
├─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
├─user.slice
│   └─user-42.slice
│       └─session-c1.scope
│           └─965 gdm-session-worker [pam/gdm-launch-environment]
...
└─system.slice
    |
    ...
    └─chronyd.service
        └─844 /usr/sbin/chronyd
    └─example.service
        └─8914 /bin/bash /home/jdoe/example.sh
            └─8916 sleep 1
...

```

この出力例では、選択したコントローラーと対話するサービスのリストを表示します。

- **systemctl status <system_unit>** コマンドを使用して、特定のユニットと **cgroups** 階層のその部分に関する詳細情報を表示します。

```

# systemctl status example.service
● example.service - My example service
   Loaded: loaded (/usr/lib/systemd/system/example.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2019-04-16 12:12:39 CEST; 3s ago

```



```

Main PID: 17737 (bash)
Tasks: 2 (limit: 11522)
Memory: 496.0K (limit: 1.5M)
CGroup: /system.slice/example.service
├─17737 /bin/bash /home/jdoe/example.sh
└─17743 sleep 1
Apr 16 12:12:39 redhat systemd[1]: Started My example service.
Apr 16 12:12:39 redhat bash[17737]: The current time is Tue Apr 16 12:12:39 CEST 2019
Apr 16 12:12:40 redhat bash[17737]: The current time is Tue Apr 16 12:12:40 CEST 2019

```

関連情報

- [Linux カーネルリソースコントローラーとは](#)
- [systemd.resource-control\(5\)](#) および [cgroups\(7\)](#) man ページ

25.10. リソースコントローラーの表示

どのプロセスがどのリソースコントローラーを使用しているかを調べます。

手順

1. プロセスが対話するリソースコントローラーを表示し、**cat proc/<PID>/cgroup** コマンドを実行します。

```

# cat /proc/11269/cgroup
12:freezer:/
11:cpuset:/
10:devices:/system.slice
9:memory:/system.slice/example.service
8:pids:/system.slice/example.service
7:hugetlb:/
6:rdma:/
5:perf_event:/
4:cpu,cpuacct:/
3:net_cls,net_prio:/
2:blkio:/
1:name=systemd:/system.slice/example.service

```

この出力例は、対象のプロセスに関するものです。今回の例では、**example.service** ユニットに属する **PID 11269** で識別されるプロセスです。**systemd** ユニットファイルの仕様で定義されているように、適切なコントロールグループにプロセスが置かれているかどうかを判断できません。



注記

デフォルトでは、デフォルトのリソースコントローラーがすべて自動的にマウントされるため、リソースコントローラーのリストにある項目とその順序は、**systemd** が起動するすべてのユニットで同じになります。

関連情報

- [cgroups\(7\)](#) の man ページ

- `/usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups-v1/` ディレクトリーのドキュメント

25.11. リソース消費の監視

現在実行中のコントロールグループ (**cgroups**) とそのリソース消費のリストをリアルタイムで表示します。

手順

1. **systemd-cgtop** コマンドを使用して、現在実行中の **cgroups** の動的アカウントを表示します。

```
# systemd-cgtop
Control Group          Tasks %CPU  Memory Input/s Output/s
/                      607 29.8  1.5G   -      -
/system.slice         125  -    428.7M   -      -
/system.slice/ModemManager.service    3  -    8.6M   -      -
/system.slice/NetworkManager.service  3  -   12.8M   -      -
/system.slice/accounts-daemon.service  3  -    1.8M   -      -
/system.slice/boot.mount                -  -    48.0K   -      -
/system.slice/chronyd.service           1  -    2.0M   -      -
/system.slice/cockpit.socket            -  -    1.3M   -      -
/system.slice/colord.service             3  -    3.5M   -      -
/system.slice/crond.service              1  -    1.8M   -      -
/system.slice/cups.service               1  -    3.1M   -      -
/system.slice/dev-hugepages.mount        -  -   244.0K   -      -
/system.slice/dev-mapper-rhelx2dswap.swap -  -   912.0K   -      -
/system.slice/dev-mqueue.mount           -  -    48.0K   -      -
/system.slice/example.service            2  -    2.0M   -      -
/system.slice/firewalld.service          2  -   28.8M   -      -
...
```

この出力例では、現在実行中の **cgroups** が、リソースの使用状況 (CPU、メモリー、ディスク I/O 負荷) 別に表示されています。デフォルトでは1秒ごとにリストが更新されます。そのため、コントロールグループごとに、実際のリソースの使用状況について動的な見解が得られるようになります。

関連情報

- **systemd-cgtop(1)** man ページ

第26章 CGROUPS-V2 と SYSTEMD を使用したリソース管理の設定

systemd スイートの主な機能は、サービスの管理と監視です。そのため、**systemd** は、起動プロセス中に適切なサービスが適切なタイミングおよび正しい順序で起動するようにします。サービスの実行時には、スムーズに実行して基盤のハードウェアプラットフォームを最適に使用する必要があります。したがって、**systemd** はリソース管理ポリシーを定義する機能や、さまざまなオプションを調整する機能も備えており、これらによってサービスのパフォーマンスを改善します。

26.1. 前提条件

- [Linux cgroup](#) サブシステムの基本知識。

26.2. リソース配分モデルの概要

リソース管理では、**systemd** はコントロールグループバージョン 2 (**cgroups-v2**) インターフェイスを使用します。



注記

デフォルトでは、RHEL 8 は **cgroups-v1** を使用します。したがって、**cgroups-v2** を有効にして、**systemd** がリソース管理に **cgroups-v2** インターフェイスを使用できるようにする必要があります。**cgroups-v2** を有効にする方法の詳細については、[cgroups-v2 のマウント](#) を参照してください。

システムリソースの配分を変更するには、以下のリソース配分モデルの1つまたは複数を使用できます。

重み

リソースは、すべてのサブグループの重みを合計し、すべての重みの合計と比較した各サブグループの重みに基づいて、各サブグループにリソースの一部を与えることによって配分されます。たとえば、**cgroups** が 10 個あり、それぞれの重みが 100 の場合には、合計は 1000 で、各 **cgroup** はリソースの 10 分の 1 を受け取ります。

重みは通常、ステートレスリソースの配分に使用されます。コントロールグループの重みを調整するには、**CPUWeight=** オプションを使用します。

制限

cgroup は、設定したリソースの量だけ使用できますが、リソースをオーバーコミットすることもできます。そのため、サブグループ制限の合計は、親 **cgroup** の制限を超える可能性があります。コントロールグループの制限を調整するには、**MemoryMax=** オプションを使用します。

保護

cgroup に、保護するリソース量を設定できます。リソースの使用量が保護するリソース量を下回る場合でも、カーネルは、この **cgroup** にペナルティーを課さず、同じリソースを取得しようとしている他の **cgroup** を優先しません。オーバーコミットも可能です。

コントロールグループの保護リソース量を調整するには、**MemoryLow=** オプションを使用します。

割り当て

リアルタイムの予算など、上限のあるリソースの特定の量を排他的に割り当てます。オーバーコミットが可能です。

関連情報

- [Managing CPU with systemd](#)
- [systemd を使用したメモリーリソースの割り当て](#)
- [systemd を使用した I/O 帯域幅の設定](#)

26.3. SYSTEMD を使用した CPU リソースの割り当て

systemd が管理するシステムでは、各システムサービスは対象の **cgroup** で起動します。CPU **cgroup** コントローラーのサポートを有効にすると、システムはプロセスごとのディストリビューションではなく、CPU リソースのサービス対応ディストリビューションを使用します。サービス対応ディストリビューションでは、サービスを設定するプロセスの数にかかわらず、システムで実行中の他の全サービスと比較して、ほぼ同じ CPU 時間を受け取ります。

特定のサービスで多くの CPU リソースが必要な場合は、サービスの CPU 時間割り当てポリシーを変更することでリソースを付与できます。

手順

systemd の使用時に CPU 時間割り当てポリシーオプションを設定するには、以下を実行します。

1. 選択したサービスで、CPU 時間割り当てポリシーオプションに割り当てた値を確認します。

```
$ systemctl show --property <CPU time allocation policy option> <service name>
```

2. root として、CPU 時間割り当てポリシーのオプションに必要な値を設定します。

```
# systemctl set-property <service name> <CPU time allocation policy option>=<value>
```

cgroup プロパティは、プロパティの設定直後に適用されます。したがって、このサービスを再起動する必要はありません。

検証

- サービスに必要な CPU 時間割り当てポリシーオプションの値が正常に変更されたかどうかを確認するには、以下のコマンドを実行します。

```
$ systemctl show --property <CPU time allocation policy option> <service name>
```

関連情報

- [systemd の CPU 時間割り当てポリシーオプション](#)
- [リソース配分モデルの概要](#)

26.4. SYSTEMD の CPU 時間割り当てポリシーオプション

最も頻繁に使用される CPU 時間割り当てポリシーオプションには、以下が含まれます。

CPUWeight=

特定のサービスに対して、他の全サービスよりも **優先度を高く** 割り当てます。間隔 1-10,000 の値から選択できます。デフォルト値は 100 です。

たとえば、他の全サービスと比べて2倍のCPUを **httpd.service** に割り当てるには、値を **CPUWeight=200** に設定します。

CPUWeight= は、利用可能なCPUリソースが枯渇した場合にのみ適用されることに注意してください。

CPUQuota=

絶対CPU時間のクォータをサービスに割り当てます。このオプションの値は、利用可能なCPU時間合計に対してサービスが受け取るCPU時間の最大比率(例: **CPUQuota=30%**)を指定します。

CPUQuota= は、重みや制限など、特定のリソース配分モデルの制限値を表すことに注意してください。

CPUQuota= の詳細は、**systemd.resource-control(5)** の man ページを参照してください。

関連情報

- [リソース配分モデルの概要](#)
- [systemd を使用した CPU リソースの割り当て](#)
- [リソース配分モデルの概要](#)

26.5. SYSTEMD を使用したメモリーリソースの割り当て

systemd を使用してメモリーリソースを割り当てるには、次のいずれかのメモリー設定オプションを使用します。

- **MemoryMin**
- **MemoryLow**
- **MemoryHigh**
- **MemoryMax**
- **MemorySwapMax**

手順

systemd の使用時にメモリー割り当て設定オプションを指定するには、以下を実行します。

1. 選択したサービスで、メモリー割り当て設定オプションに割り当てた値を確認します。

```
$ systemctl show --property <memory allocation configuration option> <service name>
```

2. **root** として、メモリー割り当ての設定オプションに必要な値を設定します。

```
# systemctl set-property <service name> <memory allocation configuration option>=<value>
```



注記

cgroup プロパティは、プロパティの設定直後に適用されます。したがって、このサービスを再起動する必要はありません。

検証

- サービスのメモリー割り当て設定オプションの必要な値が正常に変更されたかどうかを確認するには、次のように入力します。

```
$ systemctl show --property <memory allocation configuration option> <service name>
```

関連情報

- [systemd のメモリー割り当て設定オプション](#)
- [リソース配分モデルの概要](#)

26.6. SYSTEMD のメモリー割り当て設定オプション

systemd を使用してシステムメモリー割り当てを設定する場合には、以下のオプションを使用できません。

MemoryMin

ハードメモリーの保護。メモリー使用量が指定された制限を下回る場合には、cgroup メモリーは解放されません。

MemoryLow

ソフトメモリーの保護。メモリー使用量が指定された制限を下回り、保護されていない cgroup からメモリーが解放されなかった場合のみ、cgroup のメモリーを解放できます。

MemoryHigh

メモリースロットルの制限。メモリー使用量が指定された制限を超えると、cgroup のプロセスにスロットリングが適用され、解放への圧力が高まります。

MemoryMax

メモリー使用量の絶対上限。キロバイト (K)、メガバイト (M)、ギガバイト (G)、テラバイト (T) の接尾辞 (例: **MemoryMax=1G**) を使用できます。

MemorySwapMax

swap の使用量のハード制限。



注記

メモリーの上限を超えると、OOM (Out-of-memory) killer は実行中のサービスを停止します。これを回避するには、**OOMScoreAdjust=** の値を減らして、メモリーの耐性を高めます。

関連情報

- [systemd を使用したメモリーリソースの割り当て](#)
- [リソース配分モデルの概要](#)

26.7. SYSTEMD を使用した I/O 帯域幅の設定

RHEL 8 で特定のサービスのパフォーマンスを改善するには、**systemd** を使用してそのサービスに I/O 帯域幅リソースを確保することができます。

これには、以下の I/O 設定オプションを使用できます。

- IOWeight
- IODeviceWeight
- IOReadBandwidthMax
- IOWriteBandwidthMax
- IOReadIOPSMax
- IOWriteIOPSMax

手順

systemd を使用して I/O 帯域幅設定オプションを設定するには、以下を実行します。

1. 選択したサービスで、I/O 帯域幅設定オプションに割り当てた値を確認します。

```
$ systemctl show --property <I/O bandwidth configuration option> <service name>
```

2. **root** として、I/O 帯域幅の設定オプションで必要な値を設定します。

```
# systemctl set-property <service name> <I/O bandwidth configuration option>=  
<value>
```

cgroup プロパティは、プロパティの設定直後に適用されます。したがって、このサービスを再起動する必要はありません。

検証

- サービスの I/O 帯域幅設定オプションの必要な値が正常に変更されたかどうかを確認するには、次のように入力します。

```
$ systemctl show --property <I/O bandwidth configuration option> <service name>
```

関連情報

- [systemd の I/O 帯域幅設定オプション](#)
- [リソース配分モデルの概要](#)

26.8. SYSTEMD の I/O 帯域幅設定オプション

systemd を使用してブロックレイヤー I/O ポリシーを管理するには、次の設定オプションを使用できます。

IOWeight

デフォルトの I/O 加重を設定します。加重の値をベースとして使用し、他のサービスと比べ、実際に受け取る I/O 帯域幅を計算します。

IODeviceWeight

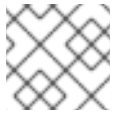
特定のブロックデバイスの I/O 加重を設定します。

たとえば、`IODeviceWeight=/dev/disk/by-id/dm-name-rhel-root 200` などです。

IOReadBandwidthMax、IOWriteBandwidthMax

デバイスまたはマウントポイントごとの絶対帯域幅を設定します。

たとえば、`IOWriteBandwidth=/var/log 5M` などです。



注記

`systemd` は、ファイルシステムからデバイスへの変換を自動的に処理します。

IOReadIOPSMax、IOWriteIOPSMax

1 秒あたりの I/O 処理数 (IOPS) の絶対帯域幅を設定します。



注記

加重ベースのオプションは、ブロックデバイスが CFQ I/O スケジューラーを使用している場合にのみサポートされます。デバイスが Multi-Queue Block I/O キューメカニズムを使用する場合は、オプションのサポートはありません。

関連情報

- [systemd を使用した I/O 帯域幅の設定](#)
- [リソース配分モデルの概要](#)

26.9. SYSTEMD を使用した CPuset コントローラーの設定

`systemd` リソース管理 API で、サービスが使用できる CPU および NUMA ノードのセットに制限を設定できます。この制限により、プロセスによって使用されるシステムリソースへのアクセスを制限します。要求された設定は、`cpuset.cpus` および `cpuset.mems` ファイルに書き込まれます。

ただし、親 `cgroup` は `cpus` または `mems` のいずれかを制限するため、要求された設定が使用されない場合があります。現在の設定にアクセスするために、`cpuset.cpus.effective` および `cpuset.mems.effective` ファイルがユーザーにエクスポートされます。

手順

- `AllowedCPUs` を設定するには、以下を実行します。

```
# systemctl set-property <service name>.service AllowedCPUs=<value>
```

以下に例を示します。

```
# systemctl set-property <service name>.service AllowedCPUs=0-5
```

- `AllowedMemoryNodes` を設定するには、以下を実行します。


```
# systemctl set-property <service name>.service AllowedMemoryNodes=<value>
```

以下に例を示します。

```
# systemctl set-property <service name>.service AllowedMemoryNodes=0
```

第27章 SYSTEMD を使用した CPU のアフィニティーおよび NUMA ポリシーの設定

CPU 管理、メモリー管理、および I/O 帯域幅オプションで、利用可能なリソースを分割します。

27.1. SYSTEMD を使用した CPU アフィニティーの設定

CPU アフィニティーの設定は、特定のプロセスにアクセスできる CPU を一部だけに制限する場合に役立ちます。実際には、CPU スケジューラーでは、プロセスのアフィニティーマスク上にない CPU で実行するプロセスはスケジューリングされません。

デフォルトの CPU アフィニティーマスクは、systemd が管理するすべてのサービスに適用されます。

特定の systemd サービスの CPU アフィニティーマスクを設定するには、systemd の `/etc/systemd/system.conf` でユニットファイルオプションとマネージャー設定オプション両方として `CPUAffinity=` を指定します。

`CPUAffinity=` ユニットファイルのオプションでは、CPU または CPU 範囲のリストを設定し、アフィニティーマスクとしてマージして使用されます。`/etc/systemd/system.conf` ファイルの `CPUAffinity` オプションは、プロセス ID 番号 (PID) 1 と、PID1 でフォークされた全プロセスにアフィニティーマスクを定義します。これにより、各サービスで `CPUAffinity` を上書きできます。



注記

特定の systemd サービスの CPU アフィニティーマスクを設定したら、システムを再起動して変更を適用する必要があります。

手順

`CPUAffinity` ユニットファイル オプションを使用して特定の systemd サービスの CPU アフィニティーマスクを設定するには以下を実行します。

1. 選択したサービスで `CPUAffinity` ユニットファイルオプションの値を確認します。

```
$ systemctl show --property <CPU affinity configuration option> <service name>
```

2. root として、アフィニティーマスクとして使用する CPU 範囲の `CPUAffinity` ユニットファイルで必要な値を設定します。

```
# systemctl set-property <service name> CPUAffinity=<value>
```

3. サービスを再起動して変更を適用します。

```
# systemctl restart <service name>
```

`manager configuration` オプションを使用して特定の systemd サービスの CPU アフィニティーマスクを設定するには、以下を実行します。

1. `/etc/systemd/system.conf` ファイルを編集します。

```
# vi /etc/systemd/system.conf
```

2. `CPUAffinity=` オプションを検索して、CPU 数を設定します。

3. 編集したファイルを保存し、サーバーを再起動して変更を適用します。

27.2. SYSTEMD を使用した NUMA ポリシーの設定

Non-Uniform Memory Access (NUMA) は、コンピューターメモリーのサブシステム設計で、この設計ではメモリーのアクセス時間は、プロセッサからの物理メモリーの場所により異なります。

CPU に近いメモリーは、別の CPU のローカルにあるメモリーや、一連の CPU 間で共有されているメモリーと比べ、レイテンシーが低くなっています (外部メモリー)。

Linux カーネルでは、NUMA ポリシーを使用して、カーネルがプロセス用に物理メモリーを割り当てる場所 (例: NUMA ノード) を制御します。

systemd は、サービスのメモリー割り当てポリシーを制御するためのユニットファイルオプション **NUMAPolicy** および **NUMAMask** を提供します。

手順

NUMAPolicy ユニットファイルオプションで NUMA メモリーポリシーを設定するには以下を実行します。

1. 選択したサービスで **NUMAPolicy** ユニットファイルオプションの値を確認します。

```
$ systemctl show --property <NUMA policy configuration option> <service name>
```

2. root として、**NUMAPolicy** ユニットファイルオプションに必要なポリシータイプを設定します。

```
# systemctl set-property <service name> NUMAPolicy=<value>
```

3. サービスを再起動して変更を適用します。

```
# systemctl restart <service name>
```

[Manager] 設定オプションを使用してグローバルな **NUMAPolicy** を設定するには、以下を実行します。

1. **/etc/systemd/system.conf** ファイルで [Manager] セクションにある **NUMAPolicy** オプションを検索します。
2. ポリシータイプを編集してファイルを保存します。
3. **systemd** 設定をリロードします。

```
# systemctl daemon-reload
```

4. サーバーを再起動します。



重要

bind などの厳密な NUMA ポリシーを設定する場合は、**CPUAffinity=** ユニットファイルオプションも適切に設定されていることを確認してください。

関連情報

- [systemd の NUMA ポリシー設定オプション](#)
- **systemd.resource-control(5)**、**systemd.exec(5)**、および **set_mempolicy(2)** の man ページ

27.3. SYSTEMD の NUMA ポリシー設定オプション

Systemd で以下のオプションを指定して、NUMA ポリシーを設定します。

NUMAPolicy

実行したプロセスの NUMA メモリーポリシーを制御します。次のポリシータイプを使用できます。

- default
- preferred
- bind
- interleave
- local

NUMAMask

選択した NUMA ポリシーに関連付けられた NUMA ノードリストを制御します。
次のポリシーには **NUMAMask** オプションを指定する必要がないことに注意してください。

- default
- local

優先ポリシーの場合、このリストで指定できるのは単一の NUMA ノードのみです。

関連情報

- **systemd.resource-control(5)**、**systemd.exec(5)**、および **set_mempolicy(2)** の man ページ
- [systemd を使用した NUMA の設定](#)

第28章 BPF コンパイラコレクションでシステムパフォーマンスの分析

システム管理者として BPF コンパイラコレクション (BCC) ライブラリーで Linux オペレーティングシステムのパフォーマンスを分析するツールを作成します。ただし、他のインターフェイス経由での取得は困難な場合があります。

28.1. BCC-TOOLS パッケージのインストール

bcc-tools パッケージをインストールします。これにより、依存関係として BPF Compiler Collection (BCC) ライブラリーもインストールされます。

手順

1. **bcc-tools** をインストールします。

```
# yum install bcc-tools
```

BCC ツールは、`/usr/share/bcc/tools/` ディレクトリーにインストールされます。

2. 必要に応じて、ツールを検証します。

```
# ll /usr/share/bcc/tools/  
...  
-rwxr-xr-x. 1 root root 4198 Dec 14 17:53 dcsnoop  
-rwxr-xr-x. 1 root root 3931 Dec 14 17:53 dcstat  
-rwxr-xr-x. 1 root root 20040 Dec 14 17:53 deadlock_detector  
-rw-r--r--. 1 root root 7105 Dec 14 17:53 deadlock_detector.c  
drwxr-xr-x. 3 root root 8192 Mar 11 10:28 doc  
-rwxr-xr-x. 1 root root 7588 Dec 14 17:53 execsnoop  
-rwxr-xr-x. 1 root root 6373 Dec 14 17:53 ext4dist  
-rwxr-xr-x. 1 root root 10401 Dec 14 17:53 ext4slower  
...
```

上記のリストにある **doc** ディレクトリーには、各ツールのドキュメントが含まれます。

28.2. BCC-TOOLS でパフォーマンスの分析

BPF Compiler Collection (BCC) ライブラリーから事前に作成された特定のプログラムを使用して、システムパフォーマンスをイベントごとに効率的かつセキュアに分析します。BCC ライブラリーで事前作成されたプログラムセットは、追加プログラム作成の例として使用できます。

前提条件

- [bcc-tools](#) パッケージがインストールされている
- root 権限がある。

execsnoop を使用したシステムプロセスの検証

1. 1つの端末で **execsnoop** プログラムを実行します。

```
# /usr/share/bcc/tools/execsnoop
```

- たとえば、別のターミナルで次のように実行します。

```
$ ls /usr/share/bcc/tools/doc/
```

これにより、**ls** コマンドの短命プロセスが作成されます。

- execsnoop** を実行している端末は、以下のような出力を表示します。

```
PCOMM PID  PPID  RET  ARGS
ls  8382  8287  0    /usr/bin/ls --color=auto /usr/share/bcc/tools/doc/
...
```

execsnoop プログラムは、新しいプロセスごとに出力行を出力するため、システムリソースを消費します。また、**ls** などの非常に短期間に実行されるプログラムのプロセスを検出します。なお、ほとんどの監視ツールはそれらを登録しません。

execsnoop 出力には以下のフィールドが表示されます。

PCOMM

親プロセス名。(ls)

PID

プロセス ID。(8382)

PPID

親プロセス ID。(8287)

RET

exec() システムコールの戻り値 (0)。プログラムコードを新規プロセスに読み込みます。

ARGS

引数を使用して起動したプログラムの場所。

execsnoop の詳細、例、およびオプションを確認するには、`/usr/share/bcc/tools/doc/execsnoop_example.txt` ファイルを参照してください。

exec() の詳細は、**exec(3)** man ページを参照してください。

opensnoop を使用した、コマンドにより開かれるファイルの追跡

- 1つのターミナルで **opensnoop** プログラムを実行します。

```
# /usr/share/bcc/tools/opensnoop -n uname
```

上記の出力では、**uname** コマンドのプロセスによってのみ開かれるファイルの内容が出力されます。

- 別のターミナルで、次のように実行します。

```
$ uname
```

上記のコマンドは、特定のファイルを開きます。このファイルは次のステップでキャプチャーされます。

- opensnoop** を実行している端末は、以下のような出力を表示します。

```
PID COMM FD ERR PATH
8596 uname 3 0 /etc/ld.so.cache
8596 uname 3 0 /lib64/libc.so.6
8596 uname 3 0 /usr/lib/locale/locale-archive
...
```

opensnoop プログラムは、システム全体で **open()** システム呼び出しを監視し、**uname** が開こうとしたファイルごとに出力行を出力します。

opensnoop 出力には、以下のフィールドが表示されます。

PID

プロセス ID。(8596)

COMM

プロセス名。(uname)

FD

ファイルの記述子。開いたファイルを参照するために **open()** が返す値です。(3)

ERR

すべてのエラー。

PATH

open() で開こうとしたファイルの場所。

コマンドが、存在しないファイルを読み込もうとすると、**FD** コラムは **-1** を返し、**ERR** コラムは関連するエラーに対応する値を出力します。その結果、**opensnoop** は、適切に動作しないアプリケーションの特定に役立ちます。

opensnoop の詳細、例、およびオプションを確認するには、`/usr/share/bcc/tools/doc/opensnoop_example.txt` ファイルを参照してください。

open() の詳細は、**open(2)** man ページを参照してください。

ディスク上の I/O 操作を調べるための **biotop** の使用

- 1つのターミナルで **biotop** プログラムを実行します。

```
# /usr/share/bcc/tools/biotop 30
```

このコマンドにより、ディスク上で I/O 操作を実行する上位のプロセスを監視できます。この引数は、コマンドが 30 秒の概要を生成するようにします。



注記

引数を指定しないと、デフォルトでは1秒ごとに出力画面が更新されます。

2. 別の端末で、たとえば次のように実行します。

```
# dd if=/dev/vda of=/dev/zero
```

上記のコマンドは、ローカルのハードディスクデバイスからコンテンツを読み込み、出力を `/dev/zero` ファイルに書き込みます。この手順では、**biotop** を示す特定の I/O トラフィックを生成します。

3. **biotop** を実行している端末は、以下のような出力を表示します。

```

PID  COMM          D MAJ MIN DISK   I/O Kbytes  AVGms
9568 dd            R 252 0 vda    16294 14440636.0 3.69
48   kswapd0       W 252 0 vda    1763 120696.0 1.65
7571 gnome-shell   R 252 0 vda    834 83612.0 0.33
1891 gnome-shell   R 252 0 vda    1379 19792.0 0.15
7515 Xorg          R 252 0 vda    280 9940.0 0.28
7579 llvmpipe-1    R 252 0 vda    228 6928.0 0.19
9515 gnome-control-c R 252 0 vda    62 6444.0 0.43
8112 gnome-terminal- R 252 0 vda    67 2572.0 1.54
7807 gnome-software R 252 0 vda    31 2336.0 0.73
9578 awk          R 252 0 vda    17 2228.0 0.66
7578 llvmpipe-0   R 252 0 vda    156 2204.0 0.07
9581 pgrep        R 252 0 vda    58 1748.0 0.42
7531 InputThread  R 252 0 vda    30 1200.0 0.48
7504 gdbus        R 252 0 vda    3 1164.0 0.30
1983 llvmpipe-1    R 252 0 vda    39 724.0 0.08
1982 llvmpipe-0   R 252 0 vda    36 652.0 0.06
...

```

biotop 出力には、以下のフィールドが表示されます。

PID

プロセス ID。(9568)

COMM

プロセス名。(dd)

DISK

読み取り操作を実行するディスク。(vda)

I/O

実行された読み取り操作の数。(16294)

Kbytes

読み取り操作によって使用したバイト数 (KB)。(14,440,636)

AVGms

読み取り操作の平均 I/O 時間。(3.69)

biotop の詳細、例、およびオプションを確認するには、`/usr/share/bcc/tools/doc/biotop_example.txt` ファイルを参照してください。

dd の詳細は、**dd(1)** man ページを参照してください。

xfsslower を使用した、予想外に遅いファイルシステム動作の明確化

- 1 つのターミナルで **xfsslower** プログラムを実行します。

```
# /usr/share/bcc/tools/xfsslower 1
```

上記のコマンドは、XFS ファイルシステムが、読み込み、書き込み、開く、または同期 (**fsync**) 操作を実行するのに費やした時間を測定します。1 引数を指定すると、1ms よりも遅い操作のみが表示されます。



注記

引数を指定しないと、**xfsslower** はデフォルトで 10 ms よりも低速な操作を表示します。

- 別のターミナルで、たとえば次のように入力します。

```
$ vim text
```

上記のコマンドは、**vim** エディターでテキストファイルを作成し、XFS ファイルシステムと特定の対話を開始します。

- xfsslower** を実行している端末は、前の手順でファイルを保存した場合と同様の内容を示しています。

```
TIME  COMM      PID  T BYTES  OFF_KB  LAT(ms)  FILENAME
13:07:14 b'bash'   4754  R 256   0       7.11 b'vim'
13:07:14 b'vim'    4754  R 832   0       4.03 b'libgpm.so.2.1.0'
13:07:14 b'vim'    4754  R 32    20      1.04 b'libgpm.so.2.1.0'
13:07:14 b'vim'    4754  R 1982  0       2.30 b'vimrc'
13:07:14 b'vim'    4754  R 1393  0       2.52 b'getscriptPlugin.vim'
13:07:45 b'vim'    4754  S 0     0       6.71 b'text'
13:07:45 b'pool'   2588  R 16    0       5.58 b'text'
...
```

上記の各行はファイルシステム内の操作を表し、特定のしきい値よりも時間がかかります。**xfsslower** は、操作に想定以上に時間がかかるなど、ファイルシステムで発生し得る問題を表面化するのに適しています。

xfsslower 出力には、以下のフィールドが表示されます。

COMM

プロセス名。(b'bash')

T

操作の種類。(R)

- Read
- Write
- Sync

OFF_KB

ファイルオフセット (KB)。(0)

FILENAME

読み取り、書き込み、または同期中のファイル。

xfsslower の詳細、例、およびオプションについては、`/usr/share/bcc/tools/doc/xfsslower_example.txt` ファイルを参照してください。

fsync の詳細は、**fsync(2)** の man ページを参照してください。

第29章 カーネル整合性サブシステムによるセキュリティの強化

カーネル整合性サブシステムのコンポーネントを使用して、システムの保護を強化できます。関連するコンポーネントとその設定の詳細をご覧ください。

29.1. カーネル整合性サブシステム

整合性サブシステムは、システムデータの全体的な整合性を維持するカーネルの一部です。このサブシステムは、システムの状態を構築時と同じ状態に保つのに役立ちます。このサブシステムを使用すると、特定のシステムファイルの望ましくない変更を防ぐことができます。

カーネル整合性サブシステムは、2つの主要なコンポーネントで設定されています。

Integrity Measurement Architecture (IMA)

- IMA は、ファイルが実行されるか開かれるたびに、暗号的にハッシュするか、暗号化キーで署名することにより、ファイルの内容を測定します。キーは、カーネルキーリングサブシステムに格納されます。
- IMA は、測定値をカーネルのメモリー空間内に格納します。これにより、システムのユーザーが測定値を変更できなくなります。
- IMA は、ローカルおよびリモートのユーザーが測定値を検証できるようにします。
- IMA は、カーネルメモリー内の測定リストに以前に格納された値に対して、ファイルの現在の内容をローカルで検証します。この拡張機能は、現在の測定と以前の測定が一致しない場合に、特定のファイルに対して操作を実行することを禁止します。

Extended Verification Module (EVM)

- EVM は、IMA 測定値や SELinux 属性など、システムセキュリティに関連するファイルの拡張属性 (`xattr` と呼ばれます) を保護します。EVM は、対応する値を暗号的にハッシュするか、暗号鍵で署名します。キーは、カーネルキーリングサブシステムに格納されます。

カーネル整合性サブシステムは、TPM (Trusted Platform Module) を使用して、システムセキュリティをさらに強化できます。

TPM は、暗号化鍵が統合されたハードウェア、ファームウェア、または仮想コンポーネントで、重要な暗号化機能のために Trusted Computing Group (TCG) による TPM 仕様に従って構築されています。TPM は通常、プラットフォームのマザーボードに接続された専用ハードウェアとして構築されます。ハードウェアチップの保護された改ざん防止領域から暗号化機能を提供することにより、TPM はソフトウェアベースの攻撃から保護されます。TPM は次の機能を提供します。

- 乱数ジェネレーター
- 暗号化キーのジェネレーターと安全なストレージ
- ハッシュジェネレーター
- リモート認証

関連情報

- [セキュリティの強化](#)

- SELinux (Security-Enhanced Linux) の基本設定および高度な設定

29.2. 信頼できる鍵および暗号化された鍵

信頼できる鍵 および 暗号化鍵 は、システムセキュリティを強化する上で重要な要素です。

信頼できる鍵と暗号化された鍵は、カーネルキーリングサービスを使用するカーネルが生成する可変長の対称鍵です。キーの整合性を検証できます。つまり、実行中のシステムの整合性を検証および確認するために、Extended Verification Module (EVM) などでキーを使用できます。ユーザーレベルのプログラムがアクセス可能なのは、暗号化された **プロブ** の形式での鍵のみです。

信頼できる鍵

信頼できる鍵は、鍵の作成と暗号化 (保護) の両方に使用される Trusted Platform Module (TPM) チップを必要とします。各 TPM には、ストレージルートキーと呼ばれるマスターラッピングキーがあります。これは TPM 自体に保存されます。



注記

Red Hat Enterprise Linux 8 は、TPM 1.2 と TPM 2.0 の両方をサポートしています。詳細は、[Red Hat 製品では Trusted Platform Module \(TPM\) がサポートされますか?](#) を参照してください。

次のコマンドを入力すると、TPM 2.0 チップが有効になっていることを確認できます。

```
$ cat /sys/class/tpm/tpm0/tpm_version_major
2
```

TPM 2.0 チップを有効にし、マシンのファームウェアの設定を通じて TPM 2.0 デバイスを管理することもできます。

さらに、TPM の **platform configuration register (PCR)** 値の特定セットを使用して、信頼できる鍵を保護できます。PCR には、ファームウェア、ブートローダー、およびオペレーティングシステムを反映する整合性管理値のセットが含まれます。つまり、PCR で保護された鍵は、暗号化を行った同じシステム上にある TPM でしか復号できません。ただし、PCR で保護された信頼できる鍵が読み込まれると (キーリングに追加されると)、新しいカーネルなどを起動できるように、関連する PCR 値が検証され、新しい (または今後の) PCR 値に更新されます。単一のキーを、それぞれ異なる PCR 値を持つ複数のプロブとして保存することもできます。

暗号化鍵

暗号化鍵はカーネル Advanced Encryption Standard (AES) を使用するため、TPM を必要としません。これにより、暗号化鍵は信頼できる鍵よりも高速になります。暗号化鍵は、カーネルが生成した乱数を使用して作成され、ユーザー空間のプロブへのエクスポート時に **マスターキー** により暗号化されます。

マスターキーは信頼できる鍵か、ユーザーキーのいずれかです。マスターキーが信頼されていない場合には、暗号化鍵のセキュリティは、暗号化に使用されたユーザーキーと同じように保護されます。

29.3. 信頼できる鍵での作業

keyctl ユーティリティを使用して、信頼できる鍵を作成、エクスポート、ロード、更新することにより、システムのセキュリティを向上できます。

前提条件

- 64 ビット ARM アーキテクチャーおよび IBM Z の場合、**trusted** カーネルモジュールがロードされます。

```
# modprobe trusted
```

カーネルモジュールをロードする方法の詳細は、[システムランタイム時のカーネルモジュールの読み込み](#) を参照してください。

- Trusted Platform Module (TPM) が有効でアクティブである。[カーネル整合性サブシステム](#) および [信頼できる鍵および暗号化鍵](#) を参照してください。



注記

Red Hat Enterprise Linux 8 は、TPM 1.2 と TPM 2.0 の両方をサポートしています。TPM 1.2 を使用する場合は、手順 1 をスキップします。

手順

- 次のユーティリティーのいずれかを使用して、永続ハンドル (たとえば、81000001) を持つ SHA-256 プライマリストレージキーを使用して 2048 ビット RSA キーを作成します。

- tss2** パッケージを使用する場合:

```
# TPM_DEVICE=/dev/tpm0 tsscreateprimary -hi o -st
Handle 80000000
# TPM_DEVICE=/dev/tpm0 tssevictcontrol -hi o -ho 80000000 -hp 81000001
```

- tpm2-tools** パッケージを使用する場合:

```
# tpm2_createprimary --key-algorithm=rsa2048 --key-context=key.ctxt
name-alg:
  value: sha256
  raw: 0xb
...
sym-keybits: 128
rsa: xxxxxx...

# tpm2_evictcontrol -c key.ctxt 0x81000001
persistentHandle: 0x81000001
action: persisted
```

- 信頼できるキーを作成します。

- keyctl add trusted <NAME> "new <KEY_LENGTH> keyhandle=<PERSISTENT-HANDLE> [options]" <KEYRING>** の構文の TPM 2.0 を使用します。この例では、永続ハンドルは 81000001 です。

```
# keyctl add trusted kmk "new 32 keyhandle=0x81000001" @u
642500861
```

このコマンドは、**kmk** という名前の信頼できる鍵を **32** バイト (256 ビット) の長さで作成し、ユーザーキーリング (**@u**) に配置します。鍵の長さは 32 から 128 バイト (256 から 1024 ビット) です。

- b. `keyctl add trusted <NAME> "new <KEY_LENGTH>" <KEYRING>` の構文の TPM 1.2 を使用します。

```
# keyctl add trusted kmk "new 32" @u
```

3. カーネルキーリングの現在の構造を一覧表示します。

```
# keyctl show
Session Keyring
  -3 --alswrv 500 500 keyring: ses 97833714 --alswrv 500 -1 \ keyring: uid.1000
642500861 --alswrv 500 500 \ trusted: kmk
```

4. 信頼できる鍵のシリアル番号を使用して、鍵をユーザー空間のblobにエクスポートします。

```
# keyctl pipe 642500861 > kmk.blob
```

このコマンドは、`pipe` サブコマンドと `kmk` のシリアル番号を使用します。

5. ユーザー空間のblobから信頼できる鍵をロードします。

```
# keyctl add trusted kmk "load `cat kmk.blob`" @u
268728824
```

6. TPM で保護された信頼できる鍵 (`kmk`) を使用するセキュアな暗号化鍵を作成します。`keyctl add encrypted <NAME> "new FORMAT <KEY_TYPE>:<PRIMARY_KEY_NAME> <KEY_LENGTH>" <KEYRING>` という構文に従います。

```
# keyctl add encrypted encr-key "new trusted:kmk 32" @u
159771175
```

関連情報

- [keyctl\(1\) の man ページ](#)
- [信頼できる鍵および暗号化された鍵](#)
- [Kernel Key Retention Service](#)
- [カーネル整合性サブシステム](#)

29.4. 暗号化鍵での作業

暗号化鍵を管理することで、Trusted Platform Module (TPM) が使用できないシステムのシステムセキュリティを向上できます。

前提条件

- 64 ビット ARM アーキテクチャーおよび IBM Z の場合、`encrypted-keys` カーネルモジュールがロードされます。

```
# modprobe encrypted-keys
```

カーネルモジュールをロードする方法の詳細は、[システムランタイム時のカーネルモジュールの読み込み](#)を参照してください。

手順

1. 無作為な数列を使用してユーザーキーを生成します。

```
# keyctl add user kmk-user "$(dd if=/dev/urandom bs=1 count=32 2>/dev/null)" @u
427069434
```

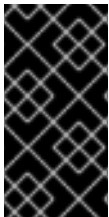
このコマンドは、**kmk-user** という名前のユーザーキーを生成し、**プライマリーキー** として動作させ、このユーザーキーを使用して実際の暗号化鍵を保護します。

2. 前の手順で取得したプライマリーキーを使用して、暗号化鍵を生成します。

```
# keyctl add encrypted encr-key "new user:kmk-user 32" @u
1012412758
```

3. 必要に応じて、指定したユーザーキーリングにあるすべての鍵をリスト表示します。

```
# keyctl list @u
2 keys in keyring:
427069434: --alswrv 1000 1000 user: kmk-user
1012412758: --alswrv 1000 1000 encrypted: encr-key
```



重要

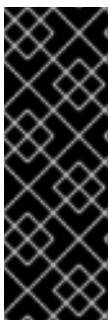
信頼できるプライマリーキーで保護されていない暗号化鍵では、暗号化に使用されたユーザーのプライマリーキー (乱数キー) と同程度のセキュリティーしか得られません。そのため、プライマリーユーザーキーはできるだけセキュアに、システムの起動プロセスの早い段階でロードしてください。

関連情報

- [keyctl\(1\) の man ページ](#)
- [Kernel Key Retention Service](#)

29.5. IMA と EVM の有効化

Integrity Measurement Architecture (IMA) と Extended Verification Module (EVM) を有効にして設定することで、オペレーティングシステムのセキュリティーを向上できます。



重要

必ず IMA と一緒に EVM を有効にしてください。

EVM を単独で有効にすることもできますが、EVM 評価は IMA 評価ルールによってのみトリガーされます。したがって、SELinux 属性などのファイルメタデータが EVM によって保護されません。ファイルメタデータがオフラインで改ざんされた場合、EVM はファイルメタデータの変更を防ぐことしかできません。ファイルの実行などのファイルアクセスは妨げません。

前提条件

- セキュアブートが一時的に無効になっている。



注記

セキュアブートが有効になっている場合、**ima_appraise=fix** カーネルコマンドラインパラメーターが機能しません。

- **securityfs** ファイルシステムが **/sys/kernel/security/** ディレクトリーにマウントされており、**/sys/kernel/security/integrity/ima/** ディレクトリーが存在している。**mount** コマンドを使用して、**securityfs** がマウントされている場所を確認できます。

```
# mount
...
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
...
```

- **systemd** サービスマネージャーに、ブート時に IMA と EVM をサポートするパッチが適用されている。次のコマンドを使用して確認します。

```
# grep <options> pattern <files>
```

以下に例を示します。

```
# dmesg | grep -i -e EVM -e IMA -w
[ 0.598533] ima: No TPM chip found, activating TPM-bypass!
[ 0.599435] ima: Allocated hash algorithm: sha256
[ 0.600266] ima: No architecture policies found
[ 0.600813] evm: Initialising EVM extended attributes:
[ 0.601581] evm: security.selinux
[ 0.601963] evm: security.ima
[ 0.602353] evm: security.capability
[ 0.602713] evm: HMAC attrs: 0x1
[ 1.455657] systemd[1]: systemd 239 (239-74.el8_8) running in system mode. (+PAM
+AUDIT +SELINUX +IMA -APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP
+GCRYPT +GNUTLS +ACL +XZ +LZ4 +SECCOMP +BLKID +ELFUTILS +KMOD +IDN2 -
IDN +PCRE2 default-hierarchy=legacy)
[ 2.532639] systemd[1]: systemd 239 (239-74.el8_8) running in system mode. (+PAM
+AUDIT +SELINUX +IMA -APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP
+GCRYPT +GNUTLS +ACL +XZ +LZ4 +SECCOMP +BLKID +ELFUTILS +KMOD +IDN2 -
IDN +PCRE2 default-hierarchy=legacy)
```

手順

1. 現在のブートエントリーの **fix** モードで IMA と EVM を有効にし、次のカーネルコマンドラインパラメーターを追加することで、ユーザーが IMA 測定値を収集および更新できるようにします。

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
args="ima_policy=appraise_tcb ima_appraise=fix evm=fix"
```

このコマンドは、現在のブートエントリーの **fix** モードで IMA および EVM を有効にしてユーザーが IMA 測定を収集し、更新できるようにします。

ima_policy=appraise_tcb カーネルコマンドラインパラメーターにより、カーネルは、デフォルトの TCB (Trusted Computing Base) 測定ポリシーと評価手順を使用するようになります。評価手順では、以前の測定と現在の測定が一致しないファイルへのアクセスを禁止します。

- 再起動して変更を適用します。
- オプション: 必要に応じて、パラメーターがカーネルコマンドラインに追加されていることを確認します。

```
# cat /proc/cmdline
BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-167.el8.x86_64 root=/dev/mapper/rhel-root
ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet ima_policy=appraise_tcb ima_appraise=fix evm=fix
```

- カーネルマスターキーを作成して、EVM 鍵を保護します。

```
# keyctl add user kmk "$(dd if=/dev/urandom bs=1 count=32 2> /dev/null)" @u
748544121
```

kmk は、すべてカーネル領域メモリー内に保持されます。**kmk** の 32 バイトの Long 値は、**/dev/urandom** ファイルの乱数バイト数から生成し、ユーザーの (@u) キーリングに配置します。鍵のシリアル番号は、前の出力の 1 行目にあります。

- kmk** に基づいて暗号化された EVM 鍵を作成します。

```
# keyctl add encrypted evm-key "new user:kmk 64" @u
641780271
```

kmk を使用して 64 バイトの long 型ユーザーキー (**evm-key**) を生成してユーザー (@u) のキーリングに配置します。鍵のシリアル番号は、前の出力の 1 行目にあります。



重要

ユーザーキーの名前は **evm-key** (EVM サブシステムが想定して使用している名前) にする必要があります。

- エクスポートする鍵のディレクトリーを作成します。

```
# mkdir -p /etc/keys/
```

- kmk** を検索し、その暗号化されていない値を新しいディレクトリーにエクスポートします。

```
# keyctl pipe $(keyctl search @u user kmk) > /etc/keys/kmk
```

- evm-key** を検索し、その暗号化された値を新しいディレクトリーにエクスポートします。

```
# keyctl pipe $(keyctl search @u encrypted evm-key) > /etc/keys/evm-key
```

evm-key は、すでにカーネルのマスターキーにより暗号化されています。

- オプション: 新しく作成されたキーを表示します。

■


```
# keyctl show
Session Keyring
974575405 --alswrv 0 0 keyring: ses 299489774 --alswrv 0 65534 \ keyring:
uid.0 748544121 --alswrv 0 0 \ user: kmk
641780271 --alswrv 0 0 \ _ encrypted: evm-key

# ls -l /etc/keys/
total 8
-rw-r--r--. 1 root root 246 Jun 24 12:44 evm-key
-rw-r--r--. 1 root root 32 Jun 24 12:43 kmk
```

10. オプション: システムの再起動後など、鍵がキーリングから削除されている場合は、新しい **kmk** と **evm-key** を作成せずに、すでにエクスポートされているものをインポートできます。

- a. **kmk** をインポートします。

```
# keyctl add user kmk "$(cat /etc/keys/kmk)" @u
451342217
```

- b. **evm-key** をインポートします。

```
# keyctl add encrypted evm-key "load $(cat /etc/keys/evm-key)" @u
924537557
```

11. EVM をアクティブ化します。

```
# echo 1 > /sys/kernel/security/evm
```

12. システム全体のラベルを付け直します。

```
# find / -fstype xfs -type f -uid 0 -exec head -n 1 '{}' >/dev/null \;
```



警告

システムのラベルを変更せずに IMA と EVM を有効にすると、システム上のファイルの大部分にアクセスできなくなる可能性があります。

検証

- EVM が初期化されていることを確認します。

```
# dmesg | tail -1
[...] evm: key initialized
```

関連情報

- [grep\(1\) manpage](#)

- [カーネル整合性サブシステム](#)
- [信頼できる鍵および暗号化された鍵](#)

29.6. INTEGRITY MEASUREMENT ARCHITECTURE によるファイルのハッシュの収集

測定 フェーズでは、ファイルハッシュを作成し、そのファイルの拡張属性 (xattrs) としてファイルハッシュを保存できます。ファイルハッシュを使用すると、RSA ベースのデジタル署名またはハッシュベースのメッセージ認証コード (HMAC-SHA1) を生成できるため、拡張属性に対するオフライン改ざん攻撃を防ぐことができます。

前提条件

- IMA と EVM が有効になっている。詳細は、[整合性測定アーキテクチャーと拡張検証モジュールの有効化](#) を参照してください。
- 有効な信頼できる鍵または暗号化鍵が、カーネルキーリングに保存されている。
- **ima-evm-utils**、**attr**、および **keyutils** パッケージがインストールされている。

手順

1. テストファイルを作成します。

```
# echo <Test_text> > test_file
```

IMA と EVM は、**test_file** サンプルファイルにハッシュ値が割り当てられ、その拡張属性として格納されていることを確認します。

2. ファイルの拡張属性を検査します。

```
# getfattr -m . -d test_file
# file: test_file
security.evm=0sAnDly4VPA0HArpPO/EquitnNyBql
security.ima=0sAQOEDeuUnWzwwKYk+n66h/vby3eD
```

出力例には、IMA および EVM ハッシュ値と SELinux コンテキストを含む拡張属性が示されています。EVM は、他の属性に関連する **security.evm** 拡張属性を追加します。この時点で、**security.evm** で **evmctl** ユーティリティーを使用して、RSA ベースのデジタル署名またはハッシュベースのメッセージ認証コード (HMAC-SHA1) を生成できます。

関連情報

- [セキュリティの強化](#)

第30章 RHEL システムロールを使用したカーネルパラメーターの永続的な設定

kernel_settings RHEL システムロールを使用すると、複数のクライアントにカーネルパラメーターを一度に設定できます。この解決策は以下のとおりです。

- 効率的な入力設定を持つ使いやすいインターフェイスを提供します。
- すべてのカーネルパラメーターを1か所で保持します。

コントロールマシンから **kernel_settings** ロールを実行すると、カーネルパラメーターはすぐに管理システムに適用され、再起動後も維持されます。



重要

RHEL チャンネルで提供される RHEL システムロールは、デフォルトの App Stream リポジトリ内の RPM パッケージとして RHEL のお客様に提供されることに注意してください。また、RHEL システムロールは、Ansible Automation Hub を介して Ansible サブスクリプションをご利用のお客様に、コレクションとして提供されます。

30.1. KERNEL_SETTINGS RHEL システムロールの概要

RHEL システムロールは、複数のシステムをリモートで管理する、一貫した設定インターフェイスを提供する一連のロールです。

RHEL システムロールは、**kernel_settings** RHEL システムロールを使用してカーネルを自動的に設定するために導入されました。**rhel-system-roles** パッケージには、このシステムロールと参考ドキュメントも含まれます。

カーネルパラメーターを自動的に1つ以上のシステムに適用するには、Playbook で選択したロール変数を1つ以上使用して、**kernel_settings** ロールを使用します。Playbook は人間が判読でき、YAML 形式で記述される1つ以上のプレイのリストです。

インベントリーファイルを使用して、Ansible が Playbook に従って設定するシステムセットを定義することができます。

kernel_settings ロールを使用して、以下を設定できます。

- **kernel_settings_sysctl** ロールを使用したカーネルパラメーター
- **kernel_settings_sysfs** ロールを使用したさまざまなカーネルサブシステム、ハードウェアデバイス、およびデバイスドライバー
- **systemd** サービスマネージャーの CPU アフィニティーを、**kernel_settings_systemd_cpu_affinity** ロール変数を使用してフォーク処理します。
- **kernel_settings_transparent_hugepages** および **kernel_settings_transparent_hugepages_defrag** のロール変数を使用したカーネルメモリーサブシステムの Transparent Huge Page

関連情報

- `/usr/share/ansible/roles/rhel-system-roles.kernel_settings/README.md` ファイル
- `/usr/share/doc/rhel-system-roles/kernel_settings/` ディレクトリー

- [Playbook の使用](#)
- [インベントリーの構築方法](#)

30.2. KERNEL_SETTINGS RHEL システムロールを使用して選択したカーネルパラメーターの適用

以下の手順に従って、Ansible Playbook を準備および適用し、複数の管理システムで永続化の影響でカーネルパラメーターをリモートに設定します。

前提条件

- [制御ノードと管理ノードを準備している](#)
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: `~/playbook.yml`) を作成します。

```
---
- name: Configure kernel settings
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.kernel_settings
  vars:
    kernel_settings_sysctl:
      - name: fs.file-max
        value: 400000
      - name: kernel.threads-max
        value: 65536
    kernel_settings_sysfs:
      - name: /sys/class/net/lo/mtu
        value: 65000
    kernel_settings_transparent_hugepages: madvise
```

- **name:** 任意の文字列をラベルとしてプレイに関連付け、プレイの対象を特定するオプションのキー。
- **hosts:** プレイを実行するホストを指定するプレイ内のキー。このキーの値または値は、マネージドホストの個別名または **inventory** ファイルで定義されているホストのグループとして指定できます。
- **vars:** 選択したカーネルパラメーター名と、それらに対して設定する必要がある値を含む変数のリストを表す Playbook のセクション。
- **role: vars** セクションで指定されたパラメーターと値を設定する RHEL システムロールを指定するキー。



注記

必要に応じて、Playbook のカーネルパラメーターとその値を変更することができます。

2. Playbook の構文を検証します。

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

```
$ ansible-playbook ~/playbook.yml
```

4. マネージドホストを再起動して、影響を受けるカーネルパラメーターをチェックし、変更が適用され、再起動後も維持されていることを確認します。

関連情報

- [/usr/share/ansible/roles/rhel-system-roles.kernel_settings/README.md](#) ファイル
- [/usr/share/doc/rhel-system-roles/kernel_settings/](#) ディレクトリー
- [Playbook の使用](#)
- [変数の使用](#)
- [ロール](#)

第31章 高度なエラー報告の使用

Advanced Error Reporting (AER) を使用すると、**Peripheral Component Interconnect Express (PCIe)** デバイスのエラーイベントの通知を受け取ります。RHEL はデフォルトでこのカーネル機能を有効にし、報告されたエラーをカーネルログに収集します。さらに、**rasdaemon** プログラムを使用すると、これらのエラーが解析され、データベースに保存されます。

31.1. AER の概要

Advanced Error Reporting (AER) は、**Peripheral Component Interconnect Express (PCIe)** デバイスの拡張エラーレポートを提供するカーネル機能です。**AER** カーネルドライバは、次の目的で **PCIe AER** 機能をサポートするルートポートを接続します。

- 包括的なエラー情報を収集する
- エラーをユーザーに報告する
- エラー回復アクションを実行する

AER がエラーをキャプチャすると、**error** メッセージがコンソールに送信されます。修復可能なエラーの場合、コンソール出力は **警告** です。

例31.1 AER 出力の例

```
Feb 5 15:41:33 hostname kernel: pcieport 10003:00:00.0: AER: Corrected error received:
id=ae00
Feb 5 15:41:33 hostname kernel: pcieport 10003:00:00.0: AER: Multiple Corrected error received:
id=ae00
Feb 5 15:41:33 hostname kernel: pcieport 10003:00:00.0: PCIe Bus Error: severity=Corrected,
type=Data Link Layer, id=0000(Receiver ID)
Feb 5 15:41:33 hostname kernel: pcieport 10003:00:00.0: device [8086:2030] error
status/mask=000000c0/00002000
Feb 5 15:41:33 hostname kernel: pcieport 10003:00:00.0: [ 6] Bad TLP
Feb 5 15:41:33 hostname kernel: pcieport 10003:00:00.0: [ 7] Bad DLLP
Feb 5 15:41:33 hostname kernel: pcieport 10003:00:00.0: AER: Multiple Corrected error received:
id=ae00
Feb 5 15:41:33 hostname kernel: pcieport 10003:00:00.0: PCIe Bus Error: severity=Corrected,
type=Data Link Layer, id=0000(Receiver ID)
Feb 5 15:41:33 hostname kernel: pcieport 10003:00:00.0: device [8086:2030] error
status/mask=00000040/00002000
```

31.2. AER メッセージの収集および表示

AER メッセージを収集して表示するには、**rasdaemon** プログラムを使用します。

手順

1. **rasdaemon** パッケージをインストールします。

```
# yum install rasdaemon
```

2. **rasdaemon** サービスを有効にして開始します。

```
# systemctl enable --now rasdaemon
```

```
Created symlink /etc/systemd/system/multi-user.target.wants/rasdaemon.service →  
/usr/lib/systemd/system/rasdaemon.service.
```

3. **ras-mc-ctl** コマンドを発行します。

```
# ras-mc-ctl --summary
```

```
# ras-mc-ctl --errors
```

このコマンドは、ログに記録されたエラーの概要を表示するか (**--summary** オプション)、エラーデータベースに保存されているエラーを表示します (**--errors** オプション)。

関連情報

- **rasdaemon(8)** man ページ
- **ras-mc-ctl(8)** man ページ