openSUSE Leap 15.7

システム分析／チューニングガイド

システム分析／チューニングガイド
openSUSE Leap 15.7

このガイドでは、問題点の抽出や解決、そしてシステムの最適化方法を 説明しています。

発行日: 2026/11/05

SUSE LLC
1800 South Novell Place
Provo, UT 84606
USA
https://documentation.suse.com

Copyright © 2006– 2026 SUSE LLC and contributors. All rights reserved.

訳: SUSE LLC および貢献者が全権利を留保しています。

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or (at your option) version 1.3; with the Invariant Section being this copyright notice and license. A copy of the license version 1.2 is included in the section entitled 「GNU Free Documentation License」.

訳: この文書を、フリーソフトウェア財団発行の GNU フリー文書利用許諾契約書バージョン 1.2 または (希望すれば) 1.3 が定める条件の下で複製、頒布、あるいは改変することを許可します。ただし、この著作権とライセンス表記については変更不可部分とします。この利用許諾契約書の複製物は、「GNU フリー文書利用許諾契約書」という章に含まれています。

For SUSE trademarks, see https://www.suse.com/company/legal/ . All third-party trademarks are the property of their respective owners. Trademark symbols (®, ™ etc.) denote trademarks of SUSE and its affiliates. Asterisks (*) denote third-party trademarks.

訳: SUSE 社の商標については、https://www.suse.com/company/legal/ をご覧ください。その他の商標は各所有者の所有物です。商標シンボル (®, ™ など) は、それぞれ SUSE 社およびその関連会社の商標であることを示しています。また、アスタリスク (*) は第三者の商標を示しています。

All information found in this book has been compiled with utmost attention to detail. However, this does not guarantee complete accuracy. Neither SUSE LLC, its affiliates, the authors nor the translators shall be held liable for possible errors or the consequences thereof.

訳: この書籍内にある全ての情報は、細部に至るまで最大限の注意を払って制作されていますが、完全に正確であることを保証するものではありません。SUSE LLC やその関連会社, 著者, 翻訳者のいずれも、本書籍内の誤りとそこから生じる結果について、一切の保証はいたしません。



注記

なお、本文書は原文 (英語) の翻訳文書であり、公式な文書ではありません。あらかじめご了承ください。

目次

前書き xiii
1 利用可能なドキュメンテーション xiii
2 ドキュメンテーションの改善 xiii
3 文書規約 xiv

I 基本 1

1 システムチューニングにおける一般的な事項 2
1.1 解決すべき問題の特定 3
1.2 よくある問題の排除 3
1.3 ボトルネックの発見 4
1.4 順を追ったチューニング 4

II システム監視 5

2 システム監視ユーティリティ 6
2.1 多用途ツール 6
`vmstat` 6 • `dstat` 9 • システムの動作状況の監視: `sar` 11
2.2 システム情報 15
デバイスの負荷情報: `iostat` 15 • プロセッサ動作の監視: `mpstat` 17 • プロセッサ周波数の監視: `turbostat` 17 • タスクの監視: `pidstat` 18 • カーネルのリングバッファの表示: `dmesg` 18 • 開いているファイルの一覧: `lsof` 19 • カーネルと udev のイベントシーケンスの表示: `udevadm monitor` 20
2.3 プロセス 20
プロセス間通信の情報: `ipcs` 20 • プロセス一覧: `ps` 21 • プロセスのツリー表示: `pstree` 23 • プロセスの一覧表示: `top` 23 • top コマン

ドに似た I/O モニタ: iotop 24 • プロセスの優先順位設定: nice および renice 26

2.4 メモリ 26
メモリ使用率の表示: free 26 • 詳細なメモリ使用率情報の取得: /proc/meminfo 27 • プロセスのメモリ使用率状況の表示: smaps 31 • numaTOP 32

2.5 ネットワーク 32
基本的なネットワーク設定: ip 32 • プロセスごとのネットワーク使用率の表示: nethogs 33 • 詳細なイーサネットカードの設定: ethtool 34 • ネットワーク状態の表示: ss 35

2.6 /proc ファイルシステム 36
procinfo 39 • システム制御パラメータ: /proc/sys/ 40

2.7 ハードウエア情報 41
PCI リソースの表示: lspci 41 • USB デバイスの一覧表示: lsusb 42 • サーマルサブシステムの監視とチューニング: tmon 42 • MCELog: マシンチェック例外 (MCE; Machine Check Exceptions) 43 • AMD64/Intel 64: dmidecode: DMI テーブルデコーダ 44

2.8 ファイルとファイルシステム 45
ファイルの種類判別: file 45 • ファイルシステムと使用率の表示: mount , df , du 46 • ELF 形式バイナリに対する追加情報 46 • ファイル属性の表示: stat 47

2.9 ユーザ情報 48
ユーザがアクセスしているファイルの表示: fuser 48 • ユーザの活動状況表示: w 48

2.10 日付と時刻 49
time による時間測定 49

2.11 データのグラフ化: RRDtool 50
RRDtool の仕組み 50 • 実際の例 51 • さらなる情報 55

3 システムログファイル 56

3.1 /var/log/ 内にあるシステムログファイル 56

3.2 ログファイルの表示と処理 58

3.3 `logrotate` によるログの管理 59

3.4 `logwatch` によるログの監視 60

3.5 `root` 宛のメールに対する転送設定 61

3.6 中央の syslog サーバへのログメッセージの転送 63
中央の syslog サーバの設定 63 • クライアント側の設定 65 • さらなる情報 65

3.7 `logger` コマンドによるシステムログへの記録 66

III カーネル監視 67

4 SystemTap: システムデータのフィルタリングと分析 68

4.1 考え方の概要 68
SystemTap スクリプト 68 • タップセット 69 • コマンドと権限 69 • 主要なファイルとディレクトリ 70

4.2 インストールと設定 71

4.3 スクリプトの文法 73
プローブの書式 73 • SystemTap イベント (プローブポイント) 74 • SystemTap ハンドラ (プローブボディ) 75

4.4 スクリプト例 80

4.5 ユーザスペースプローブ 81

4.6 さらなる情報 81

5 カーネルプローブ 83

5.1 対応するアーキテクチャ 83

5.2 カーネルプローブの種類 84
Kprobes 84 • Jprobes 84 • Return Probe 85

5.3 Kprobes API 85

5.4 debugfs インターフェイス 86
登録済みのカーネルプローブの一覧表示 86 • 指定したプローブの全体的な有効化／無効化 86
5.5 さらなる情報 87
6 Perf を利用したハードウエアベースの監視 88
6.1 ハードウエアベースの監視 88
6.2 サンプリングとカウンティング 88
6.3 Perf のインストール 89
6.4 Perf のサブコマンド 89
6.5 特定種類のイベントのカウント 90
6.6 特定のコマンド固有のイベント記録 91
6.7 さらなる情報 92
7 OProfile: システム全体に対するプロファイラ 93
7.1 考え方の概要 93
7.2 インストールと要件 93
7.3 利用可能な OProfile ユーティリティ 94
7.4 OProfile の使用 94
レポートの作成 95 • イベント設定の取得 96
7.5 レポートの生成 97
7.6 さらなる情報 98
8 ダイナミックデバッグ: カーネルのデバッグメッセージの調整 99
8.1 ダイナミックデバッグの利点 99
8.2 ダイナミックデバッグの状態確認 100
8.3 ダイナミックデバッグの使用 100
8.4 ダイナミックデバッグメッセージの表示 101

IV リソース管理 102

9 一般的なシステムリソースの管理 103

9.1 インストールの計画 103
パーティション設定 103 • インストール範囲 104 • 既定のターゲット 104

9.2 不要なサービスの無効化 105

9.3 ファイルシステムとディスクアクセス 106
ファイルシステム 106 • タイムスタンプの更新ポリシー 106 • `ionice` によるディスクアクセスの優先順位設定 107

10 カーネルコントロールグループ 109

10.1 概要 109
ハイブリッド型 cgroup 階層構造 109

10.2 リソースアカウンティング 110

10.3 リソース制限の設定 110

10.4 TasksMax を利用した fork ボムの防止 111
現時点での既定の TasksMax 値の検出 111 • DefaultTasksMax 値の設定 112 • ユーザに対する既定の TasksMax 制限 113

10.5 cgroups と I/O 制御の併用 114
事前要件 114 • 制御量の設定 115 • I/O 制御の動作説明 115 • ユーザセッション内での資源制御 117

10.6 さらなる情報 117

11 自動的な Non-Uniform Memory Access (NUMA) のバランス調整 118

11.1 実装 118

11.2 設定 119

11.3 監視 120

11.4 影響 121

12 電源管理 123
12.1 CPU レベルでの電源管理 123
C-ステート (プロセッサの待機状態) 123 • P-ステート (プロセッサの性能状態) 125 • Turbo 機能 125
12.2 カーネル内ガバナー 125
12.3 cpupower ツール 126
cpupower による現在設定の表示 127 • cpupower によるカーネル内のアイドル状態の表示 127 • cpupower によるカーネルとハードウエアの統計情報の表示 129 • cpupower による設定の変更 130
12.4 特殊なチューニングオプション 130
P-ステート向けのチューニングオプション 130
12.5 トラブルシューティング 131
12.6 さらなる情報 132
12.7 powerTOP による電力消費状況の監視 132

V カーネルのチューニング 135
13 I/O 性能のチューニング 136
13.1 I/O スケジューリングの切り替え 136
13.2 blk-mq I/O パスで利用可能な I/O エレベータ 137
MQ-DEADLINE 137 • NONE 138 • BFQ (Budget Fair Queueing) 138 • KYBER 140
13.3 I/O バリアのチューニング 140
14 タスクスケジューラのチューニング 142
14.1 概要 142
プリエンプション 142 • タイムスライス 143 • プロセスの優先順位 143
14.2 プロセスの分類 143
14.3 完全公平型スケジューラ (Completely Fair Scheduler) 144
CFS の仕組み 145 • プロセスのグループ化 145 • カーネルの設定オプション 145 • 用語 146 • chrt によるプロセスのリアルタイム属性の変

更 147 • sysctl による稼働中のチューニング 147 • デバッグ用インターフェイスとスケジューラの統計情報 152

14.4 さらなる情報 154

15 メモリ管理サブシステムのチューニング 155

15.1 メモリの用途 155
匿名メモリ 156 • ページキャッシュ 156 • バッファキャッシュ 156 • バッファヘッド 156 • ライトバック 157 • 先読み 157 • VFS キャッシュ 157

15.2 メモリ使用量の削減 158
malloc (匿名) 利用の削減 158 • カーネルのメモリオーバーヘッドの削減 158 • メモリコントローラ (メモリ cgroup) 158

15.3 仮想メモリマネージャ (VM) のチューニングパラメータ 159
回収比率に関するパラメータ 159 • ライトバック (書き戻し) 関連のパラメータ 160 • 先読み関連のパラメータ 162 • Transparent HugePage 関連のパラメータ 162 • khugepaged のパラメータ 163 • その他の VM パラメータ 164

15.4 VM の動作監視 164

16 ネットワークのチューニング 166

16.1 カーネルのソケットバッファの設定 166

16.2 ネットワーク内でのボトルネックの発見とネットワークトラフィックの分析 168

16.3 netfilter 168

16.4 Receive Packet Steering (RPS) によるネットワーク性能の改善 169

VI システムダンプの処理 171

17 トレーシングツール 172

17.1 strace によるシステムコールの追跡 172

17.2 ltrace によるライブラリコールの追跡 176

17.3 Valgrind によるデバッグとプロファイリング 177
一般的な情報 177 • 既定のオプション 178 • Valgrind の動作原理 179 • メッセージ 179 • エラーメッセージ 181

17.4 さらなる情報 181

18 Kexec と Kdump 182

18.1 概要 182

18.2 必要なパッケージ 183

18.3 Kexec の内部構造 183

18.4 crashkernel 割り当てサイズの計算 184

18.5 基本的な Kexec の使用方法 187

18.6 日々の再起動に対する Kexec の設定方法 188

18.7 基本的な Kexec の設定 188
手作業での Kdump の設定 189 • YaST での設定 191 • SSH 経由での Kdump 192

18.8 クラッシュダンプの解析 193
カーネルのバイナリ形式 195

18.9 高度な Kdump の設定 198

18.10 さらなる情報 199

19 アプリケーションクラッシュ時の systemd-coredump の使用 200

19.1 使用と設定 200

VII PRECISION TIME PROTOCOL による時刻同期 204

20 Precision Time Protocol 205

20.1 PTP の紹介 205
PTP の Linux 実装 205

20.2 PTP の使用 206
ネットワークドライバとハードウエアのサポート 206 • ptp4l の使用 207 • ptp4l 設定ファイル 208 • 遅延の測定 209 • PTP 管理クライアント: pmc 209

20.3 phc2sys による時刻同期 210
時刻同期の検証 211
20.4 設定例 212
20.5 PTP と NTP 213
NTP から PTP への同期 213 • PTP-NTP ブリッジの設定 214
A GNU ライセンス 215

前書き

改訂履歴

2023-02-03

1 利用可能なドキュメンテーション

オンラインドキュメンテーション

ドキュメンテーションは https://doc.opensuse.org で公開されています。ここから直接読むこともできますし、様々な形式でダウンロードを行うこともできます。



注記: 最新の更新について

ドキュメンテーションの最新版は通常、英語版が先に作成されます。

SUSE ナレッジベース

何らかの問題に直面した場合は、https://www.suse.com/support/kb/ からアクセスできる技術情報文書 (TID) をご確認ください。ここからお客様からの問い合わせで発生した、様々な SUSE 社の知識情報が検索できます。

お使いのシステム内

オフライン環境での利用を想定して、お使いのシステム内の `/usr/share/doc/release-notes` ディレクトリにはリリースノートが保存されているほか、各パッケージに対するドキュメンテーションが `/usr/share/doc` 内に存在しています。
また、多くのコマンドに対して、マニュアルページ も用意されています。マニュアルページは `man` コマンドで表示することができます。このコマンドの後ろに参照したいコマンド名を指定してください。なお、`man` コマンドがインストールされていない場合は、`sudo zypper install man` を実行してインストールしてください。

2 ドキュメンテーションの改善

このドキュメンテーションに対するご意見だけでなく、改善や追記などの貢献をいただければ幸いです。それぞれ下記のチャンネル経由でお送りいただくことができます:

バグ報告

ドキュメンテーション内に誤記などを見つけた場合は、https://bugzilla.opensuse.org/ で問題を報告してください。

なお、この文書の HTML 版の各章には、問題を報告するための［Report a bug］というアイコンが用意されていますので、こちらをお使いのうえ報告をお願いいたします。これにより、Bugzilla で対象の製品や分類、そしてリンク先などをそれぞれ自動で設定するようになっています。あとは問題点の説明を記述するだけです。
なお、報告には Bugzilla のアカウントが必要となるほか、英語でのやり取りが必要となりますので、あらかじめご了承ください。

貢献

このドキュメンテーションに対して追記もしくは更新すべき具体的な内容をお持ちの場合は、この文書の HTML 版の各章に用意されている［EDIT SOURCE］のリンクをお使いください。これにより、GitHub 内にある 英語版原文の ソースコードを表示し編集することができますので、作業が終わり次第 pull request を送信してください。
なお、GitHub のアカウントが必要となります。



注記: ［Edit Source］は英語版のみに提供される件について

［EDIT SOURCE］のリンクは各文書の英語版原文を編集する目的でのみご利用いただけます。その他の言語については［Report a bug］でお知らせください。

なお、本文書で使用しているドキュメンテーション環境の情報については、リポジトリ内の README をお読みください。

電子メール

本製品のドキュメンテーションに対するフィードバックは、`doc-team@suse.com` でも受け付けております。なお、文書のタイトルと製品のバージョン、およびドキュメンテーションの発行日付をそれぞれご記入ください。また、問題点の報告や記述の追加に関するご提案は、それぞれ概要と対応するセクション番号、およびページ (もしくは URL) をご記入ください。

ヘルプ

openSUSE Leap に対するさらなる支援をご希望の場合は、https://ja.opensuse.org/Portal:Support をご覧ください。

3 文書規約

この文書内では、下記のような記述ルールを使用しています:

- `/etc/passwd` : ディレクトリ名やファイル名を示しています
- `PLACEHOLDER` : `PLACEHOLDER` の箇所は、実際の値に置き換えるべきものであることを示しています

- `PATH` : 環境変数であることを示しています
- `ls` , `--help` : コマンドやオプション、パラメータであることを示しています
- `user` : ユーザ名やグループ名であることを示しています
- **`パッケージ名`** : ソフトウエアのパッケージ名を示しています
- `Alt` , `Alt`–`F1` : キー入力や組み合わせキー入力を示しています; キーはキーボードに書かれているとおりに大文字で示されます
- [ファイル] , [ファイル] › [名前を付けて保存] : メニュー項目やボタンなどを示しています
- 第1章「章のタイトル」: 本ガイド内の他の箇所への参照を示しています。
- 下記は `root` ユーザの権限で実行しなければならないコマンドを示しています。一般ユーザから実行する場合は、これらのコマンドの前に `sudo` を付けることで、`root` で実行できるようになります:

```
# コマンド
> sudo コマンド
```

- 下記は一般ユーザで実行できるコマンドを示しています:

```
> コマンド
```

- また、行末にバックスラッシュ文字 (`\`) を付けることで、コマンドを複数行に分けて記述している場合もあります。バックスラッシュ文字は、シェルに対して、これ以降にもコマンドが続くことを示す文字になります:

```
> echo a b \
c d
```

- このほか、行頭にプロンプトが書かれたコマンド行に続いて、そのコマンドを実行した場合の出力例を示す場合もあります:

```
> コマンド
出力
```

- 各種の情報について



警告: 警告

実際に実施したりする前に、注意しておかなければならない、きわめて重要な情報を記述しています。セキュリティ面の問題のほか、データを失ってしまう可能性への告知、ハードウエアの損傷の可能性や物理的な障害が発生する可能性を示しています。



重要: 重要な情報

実際に実施する前に注意すべき点を説明しています。



注記: 一般的な情報

一般的な補足情報を示しています。たとえばソフトウエアバージョン間での違いなどを説明しています。



ヒント: その他のヒント

ガイドラインや実践的なアドバイスなど、ヒントとなる情報を示しています。

- 簡潔な補足情報



一般的な補足情報を示しています。たとえばソフトウエアバージョン間での違いなどを説明しています。



ガイドラインや実践的なアドバイスなど、ヒントとなる情報を示しています。

I 基本

1 システムチューニングにおける一般的な事項 2

1 システムチューニングにおける一般的な事項

改訂履歴

2023-08-08

このマニュアルには、性能面での問題が発生する理由を見つけるための方法と、それらの問題を解決するための方法について説明しています。お使いのシステムを実際にチューニングする前に、まずはよくある問題を排除してから問題を洗い出すことをお勧めします。またシステムのチューニングにあたっては、手当たり次第に部分的なチューニングを行ってしまうと、解決に至らないばかりか、状況を悪化させることもあることから、綿密な計画をもって望むようにしてください。

手順 1.1: システムをチューニングする際の一般的なアプローチ

1. 解決すべき問題をはっきりさせます。
2. 性能劣化が最近発生したものである場合は、まずシステムに対する直近の変更点を確認します。
3. その問題が性能に影響する理由を特定します。
4. 性能を分析するために使用する尺度を決定します。尺度にはたとえば、遅延時間やスループット、ユーザの最大同時ログイン数や活動中の最大ユーザ数などがあります。
5. 決定した尺度を利用して、現時点での性能を計測します。
6. アプリケーションが最も時間を消費しているサブシステムを特定します。
7.
    a. システムやアプリケーションを監視します。
    b. データを分析して、分類ごとの時間消費をまとめていきます。
8. 特定されたサブシステムをチューニングします。
9. 監視をいったん停止し、以前と同じ尺度で改善後の性能を計測します。
10. 性能がまだ不十分であるとお考えの場合は、ステップ 3 からやり直します。

1.1 解決すべき問題の特定

システムのチューニングを始める前に、まずは問題点をできるだけ正確に定義してください。単純に「システムが遅い」と言うだけでは、問題の定義とは言えません。たとえばシステムの速度を全体的に改善する必要があるのか、それともピーク時間帯にのみ改善する必要があるのかによっても、改善の方向性が変わってきます。

それに加えて、問題点を定量的に測定する環境も必要となります。測定環境が存在しないと、チューニングが成功したのかどうかがわからなくなってしまうためです。常に「前」と「後」を比較するようにしてください。どのような尺度を使用するのかは、環境や用途、目的などによって異なります。たとえば Web サーバの尺度という観点では、下記のような尺度が存在します:

遅延時間
: ページを配信するのにかかる時間

スループット
: 毎秒提供ページ数、もしくは毎秒転送量 (速度)

アクティブユーザ数
: 許容可能な遅延時間内でページを配信することのできる、最大のユーザ数

1.2 よくある問題の排除

よくある性能問題の原因としては、ネットワークやハードウエアの問題のほか、バグや設定の問題によるものがあります。お使いのシステムのチューニングを実施する場合は、まずは下記の作業を実施して、よくある問題を排除してください:

- `systemd` のジャーナルの出力 (詳しくは『リファレンス』、第11章「`journalctl`：systemd ジャーナルへの問い合わせコマンド」を参照) を確認し、通常とは異なる出力が出ていないかどうかを確認してください。
- `top` や `ps` などのコマンドを使用して、不必要に CPU 時間を浪費していたり、メモリを占有していたりするプロセスがないかどうかを確認してください。
- `/proc/net/dev` を調査して、ネットワーク側に原因がないかどうかを確認してください。
- ハードウエア側の問題 (ディスクの場合は `smartmontools` などで確認することができます) 、もしくはディスク容量の不足によって、物理ディスクの I/O 障害が発生していないかどうかを確認してください。

- サーバの負荷が低い時間帯に、裏で動作するジョブを実行していることを確認してください。また、これらのジョブは優先度を落として (`nice` コマンドで設定することができます) 実行すべきものでもあります。
- 同一のリソースを共有して複数のサービスを動作させている場合は、サービスを他のサーバに移設できないかどうかを検討してください。
- 最後に、お使いのソフトウエアが最新の状態になっていることを確認してください。

1.3 ボトルネックの発見

システムをチューニングするにあたって、もっとも難しいのがボトルネックの発見です。openSUSE Leap では、様々なツールを提供することで、この解決を支援しています。一般的なシステム監視アプリケーションやログファイル分析ツールについては、パートII「システム監視」に詳細な説明があります。また、問題が長期にわたる深い分析を必要とするようなものである場合は、Linux カーネル側で提供する機能を使用することをお勧めします。詳しくは パートIII「カーネル監視」をお読みください。

データを収集した場合は、それらを分析する必要があります。まずはサーバのハードウエア (メモリ, CPU, バス) を調査して、I/O 性能 (ディスク, ネットワーク) が十分に発揮されていることを確認してください。これらの基本要件を満たしていれば、システムをチューニングする段階へと移行してください。

1.4 順を追ったチューニング

チューニングは注意深く計画してください。一度に複数のチューニングを実施したりせず、1 つだけチューニングを施すのがきわめて重要です。これにより、チューニングが改善をもたらしたのか、それとも悪化させてしまったのかを正確に判断できるためです。また、それぞれのチューニング作業後は十分な時間を確保して測定を行い、次の段階に移行するための足がかりを確立してください。もしもチューニングによって性能を改善できている確証が得られない場合は、その変更を恒久的に保存したりはしないでください。チューニングの内容によっては、しばらく時間をおいてから悪化するようなものもあります。

II システム監視

2 システム監視ユーティリティ 6

3 システムログファイル 56

2 システム監視ユーティリティ

改訂履歴

2025-06-12

お使いのシステムの状態を調査する際、提供されているプログラムやツール、ユーティリティは多数存在します。本章では、最も重要なものの紹介のほか、それらでよく使用されるパラメータについて説明しています。

以降のコマンドの説明では、コマンドの出力例も含めて示されています。たとえば下記の例では、最初の行 (`tux >` や `root #` 以降) がコマンド自身で、それ以降がコマンドの出力になっています。出力が長い場合は大括弧とピリオド (`[...]`) で省略される場合があるほか、1 行が長い場合は必要に応じて折り返して示しています。なお、行を折り返す場合、行末にはバックスラッシュ (`\`) が書かれます。

```
> command -x -y
1 行目の出力 1
2 行目の出力
3 行目の出力 ... 非常に長い出力を行っているため、\
    ここで改行しています
4 行目の出力
[...]
98 行目の出力
99 行目の出力
```

また、できる限り多くのユーティリティを紹介する目的から、説明は短くまとめられています。それぞれのコマンドの詳細については、対応するマニュアルページをお読みください。それ以外にも、ほとんどのコマンドには `--help` というオプションが用意されていて、ここから指定可能なパラメーター一覧を取得することができます。

2.1 多用途ツール

ほとんどの Linux におけるシステム監視ツールは、システム内の 1 つの要素しか参照しない仕組みになっていますが、ツールによっては幅広く監視を行ってくれるものもあります。システムの問題点を絞り込みたい場合は、まずこれらのツールをお使いのうえ、原因を探ってみてください:

2.1.1 `vmstat`

vmstat はプロセスやメモリ、I/O や割り込み、CPU に関する情報を収集します:

```
vmstat [オプション] [待ち時間 [回数]]
```

待ち時間と回数のパラメータを指定しないで実行すると、直近の再起動以降の平均値のみを出力します。待ち時間 (秒単位) を指定して実行すると、指定した時間内 (下記の例では 2 秒) での値を表示します。回数を指定した場合は、vmstat で情報を表示する回数を指定することができます。回数を指定しない場合は、停止するまで何度も情報を採取し続けます。

例 2.1: 負荷の軽いマシンにおける `vmstat` の出力

```
> vmstat 2
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0  44264  81520    424 935736    0    0    12    25   27   34  1  0 98  0  0
 0  0  44264  81552    424 935736    0    0     0     0   38   25  0  0 100  0  0
 0  0  44264  81520    424 935732    0    0     0     0   23   15  0  0 100  0  0
 0  0  44264  81520    424 935732    0    0     0     0   36   24  0  0 100  0  0
 0  0  44264  81552    424 935732    0    0     0     0   51   38  0  0 100  0  0
```

例 2.2: CPU 負荷の重いマシンにおける `vmstat` の出力

```
> vmstat 2
procs -----------memory---------- ---swap-- -----io---- -system-- -----cpu------
 r  b   swpd   free   buff   cache   si   so    bi    bo   in   cs us sy id wa st
32  1  26236 459640 110240 6312648    0    0  9944     2 4552 6597 95  5  0  0  0
23  1  26236 396728 110336 6136224    0    0  9588     0 4468 6273 94  6  0  0  0
35  0  26236 554920 110508 6166508    0    0  7684 27992 4474 4700 95  5  0  0  0
28  0  26236 518184 110516 6039996    0    0 10830     4 4446 4670 94  6  0  0  0
21  5  26236 716468 110684 6074872    0    0  8734 20534 4512 4061 96  4  0  0  0
```



ヒント: 出力の最初の行について

vmstat の出力のうち最初の行は、直近の再起動からの平均値を示しています。

それぞれの列の意味は下記のとおりです:

[r]

実行可能 (Runnable) 状態にあるプロセスの数を表します。これらのプロセスは実行中でするか、もしくは CPU スロットの空きを待っている状態であることになります。この列に表示されている値が、利用可能な CPU の数よりも定常的に高い場合は、CPU の能力が不足している可能性を示しています。

[b]

CPU 以外のリソースを待機しているプロセスの数を表します。この列の値が高い場合、ネットワークやディスクなど I/O の問題が発生している可能性を示しています。

[swpd]
: 現在使用中の状態にあるスワップ領域 (キロバイト単位) を表します。

[free]
: 未使用のメモリ (キロバイト単位) を表します。

[inact]
: 回収可能な未使用のメモリ量を表しています。この列は、`vmstat` に `-a` オプション (推奨) を指定した場合にのみ表示されます。

[active]
: 通常は回収することのできない、使用中のメモリ量を表しています。この列は、`vmstat` に `-a` オプション (推奨) を指定した場合にのみ表示されます。

[buff]
: ファイルシステムのメタデータを含む、メモリ内のファイルバッファキャッシュ量 (キロバイト単位) を表しています。こちらの列は、`vmstat` に `-a` オプションを指定した場合、表示されません。

[cache]
: ファイル内の実際の内容を含む、メモリ内のページキャッシュ量 (キロバイト単位) を表しています。こちらの列は、`vmstat` に `-a` オプションを指定した場合、表示されません。

[si / so]
: スワップ領域からメモリ (`si`) 、もしくはメモリからスワップ領域 (`so`) に移動されたメモリ量を、キロバイト毎秒で表した値です。`so` の値が長時間にわたって高いままである場合、アプリケーション側でメモリリーク (漏洩) が発生している可能性があり、そのメモリリークによってメモリがスワップ領域内に書き込まれている可能性があります。また `si` の値が長時間にわたって高いままである場合は、長い時間アプリケーションが休眠状態にあって、後から活動状態に戻ったことを表しています。`si` と `so` の値が両方とも長時間にわたって高いままである場合は、スワップ領域が多用されていることになりますので、アプリケーションの作業に必要なメモリのほうが実際のメモリ量よりも大きいことを示しています。

[bi]
: ブロックデバイスから受信された (たとえばディスクから読み込んだ) ブロック数を毎秒単位で表した値です。この値は、スワップ領域を使用した場合にも上昇します。ただし、ブロックサイズはファイルシステムによっても異なりますが、stat ユーティリティを使用することで知ることができます。スループットのデータをご希望の場合は、iostat をお使いください。

[bo]
: ブロックデバイスに送信した (たとえばディスクに書き込んだ) ブロック数を毎秒単位で表した値です。この値は、スワップ領域を使用した場合にも上昇します。

[in]
: 毎秒の割り込み数を表します。この値が高い場合、I/O (ネットワークもしくはディスク) の負荷が高いことを表していますが、他の処理でプロセッサ間割り込みを発生させているなど、他の理由によるものかもしれません。割り込みの詳細については、`/proc/interrupts` ファイルをご覧ください。

[cs]
: 毎秒のコンテキスト切り替え数を表しています。この値は、カーネルが一方のプロセスから他方のプロセスに処理を移した回数を表しています。

[us]
: アプリケーションコードを実行するのに消費した CPU 時間の割合を示しています。

[sy]
: カーネルコードを実行するのに消費した CPU 時間の割合を示しています。

[id]
: 待機状態にあった CPU 時間の割合を示しています。この値がずっと 0 であり続けているような場合、CPU に空きがないことを示しています。ただし、この値が 0 であったからといって、それが悪い知らせであるとは限りません。十分な CPU 性能があるものとお考えの場合は、[r] や [b] の列の値を読んで原因を探ってみることをお勧めします。

[wa]
: "wa" の値が 0 以上になっている場合、入出力処理の待ちによってスループットが落ちていることを表しています。ただし、ファイルを最初に読み込んだ場合や、裏での書き込みが間に合っていないような場合など、どうしても避けられない場合もあります。また、ハードウエアのボトルネック (ネットワークもしくはハードディスク) を示している可能性もあります。このほか、仮想メモリマネージャ (詳しくは 第15章「メモリ管理サブシステムのチューニング」をお読みください) でチューニングを行う必要性を示していることもあります。

[st]
: 仮想マシンであることにより、CPU 時間が確保できなかった割合を示しています。

その他のオプションについては、`vmstat --help` をお読みください。

2.1.2 `dstat`

`dstat` は `vmstat`, `iostat`, `netstat`, `ifstat` などに対する代替として提供されているツールです。`dstat` はシステムリソースの情報をリアルタイムに表示します。たとえばディスクの使用率と IDE コントローラの割り込みを比較したり、ネットワークの帯域とディスクの帯域を同じタイミングで計測したりすることができます。

既定では読みやすい表形式で出力しますが、表計算プログラムなどに取り込む目的で、CSV などの出力形式を指定することもできます。

また、このプログラムは Python 言語で記述され、プラグインで機能を拡張することができます。

一般的な書式は下記のとおりです:

```
dstat [-afv] [オプション..] [間隔 [回数]]
```

オプションやパラメータは全て必要に応じて設定します。何もパラメータを指定しない場合、dstat は CPU に関する統計情報 (`-c` , `--cpu`), ディスクに関する統計情報 (`-d` , `--disk`), ネットワークに関する統計情報 (`-n` , `--net`), ページング (`-g` , `--page`), システムの割り込みおよびコンテキストスイッチに関する統計情報 (`-y` , `--sys`) をそれぞれ表示します。また、1 秒間隔で情報を採取するほか、停止するまで半永久的に動作し続けるようになります:

```
# dstat
You did not select any stats, using -cdngy by default.
----total-cpu-usage---- -dsk/total- -net/total- ---paging-- ---system--
usr sys idl wai hiq siq| read  writ| recv  send|  in   out | int   csw
  0   0 100   0   0   0|  15k   44k|   0     0 |   0    82B| 148   194
  0   0 100   0   0   0|   0     0 |5430B  170B|   0     0 | 163   187
  0   0 100   0   0   0|   0     0 |6363B  842B|   0     0 | 196   185
```

`-a` , `--all`

: `-cdngy` (既定値) と同じ意味を持ちます

`-f` , `--full`

: `-C` , `-D` , `-I` , `-N` , `-S` にそれぞれ検出されたデバイスを指定したものとして処理されます

`-v` , `--vmstat`

: `-pmgdsc` , `-D total` と同じ意味を持ちます

間隔

: データを出力する間隔を秒単位で指定します

回数

: 指定した回数だけ情報を出力して終了するようにします

既定では 1 秒間隔で回数は無制限になります。

詳しくは `dstat` のマニュアルページ、もしくは http://dag.wieers.com/home-made/dstat/ にある Web ページをお読みください。

2.1.3 システムの動作状況の監視: sar

`sar` コマンドを使用することで、CPU やメモリ、IRQ の使用状況や I/O 、ネットワーキングに至るまで、ほとんど全ての主なシステム動作状況に対するレポートを生成することができます。必要であれば、リアルタイムなレポートを生成することもできます。また、`sar` は元となるデータを `/proc` ファイルシステムを利用して収集します。



注記: sysstat パッケージについて

`sar` コマンドは `sysstat` パッケージ内に含まれています。YaST を利用してインストールするか、`zypper in sysstat` と入力して実行し、インストールを行ってください。なお、`sysstat.service` は既定では動作しないようになっていますので、下記のようにして有効化し、起動しておく必要があります:

```
> sudo systemctl enable --now sysstat
```

2.1.3.1 sar によるレポートの生成

リアルタイムなレポートを生成したい場合は、`sar` コマンドに間隔 (秒単位) と回数を指定してください。また、指定したファイルからレポートを生成したい場合は、間隔と回数の代わりに `-f` でファイル名を指定してください。ファイル名も間隔と回数も指定しない場合、`sar` は `/var/log/sa/saDD` からレポートを生成しようとします。ここで、`DD` は今日の月内日を表します。これは `sadc` (System Activity Data Collector) コマンドがデータを書き込む際の既定の場所として決められているものです。複数のファイルを使用したい場合は、`-f` オプションを複数回指定してください。

```
sar 2 10                          # リアルタイムなレポートを生成 (2 秒間隔で 10 回)
sar -f ~/reports/sar_2014_07_17   # sar_2014_07_17 ファイルからレポートを生成
sar                               # /var/log/sa/ 内にある本日分のデータからレポートを生成
cd /var/log/sa && \
sar -f sa01 -f sa02               # /var/log/sa/0[12] ファイルからレポートを生成
```

下記には、便利な `sar` コマンドのオプションと、その解釈が示されています。各列の詳しい意味については、`sar` のマニュアルページ `(1)` をお読みください。オプションやレポートに関する詳細についてもマニュアルページをお読みください。

注記: sysstat サービスが停止した際にレポート生成も停止されてしまう件について

システムの再起動やシャットダウンなどで `sysstat` サービスが停止されていた場合でも、ツールは `/usr/lib64/sa/sa1 -S ALL 1 1` コマンドを自動的に実行して直近の統計情報を採取し続けます。収集されたバイナリデータはシステムの動作データファイル内に保存されます。

2.1.3.1.1 CPU 使用率レポート: `sar`

何もオプションを指定しないで `sar` を実行すると、CPU 使用率に関する基本的なレポートを生成します。マルチプロセッサ環境では、全ての CPU の結果がまとめて出力されます。個別の CPU について出力を行いたい場合は、`-P ALL` オプションを指定してください。

```
# sar 10 5
Linux 6.4.0-150600.9-default (jupiter)         2024年11月03日  _x86_64_        (2 CPU)

17時51分29秒     CPU     %user     %nice   %system   %iowait    %steal     %idle
17時51分39秒     all     57.93      0.00      9.58      1.01      0.00     31.47
17時51分49秒     all     32.71      0.00      3.79      0.05      0.00     63.45
17時51分59秒     all     47.23      0.00      3.66      0.00      0.00     49.11
17時52分09秒     all     53.33      0.00      4.88      0.05      0.00     41.74
17時52分19秒     all     56.98      0.00      5.65      0.10      0.00     37.27
平均値:          all     49.62      0.00      5.51      0.24      0.00     44.62
```

[%iowait] は CPU が I/O リクエストに対する応答を待っていて、待機状態になった時間の割合を示しています。この値が長時間にわたって 0 より高いままである場合は、I/O システム (ネットワークまたはハードディスク) 内にボトルネックが存在するものと考えられます。また、[%idle] が長い時間にわたって 0 になっている場合、お使いの CPU には余裕が無くなっていることを表しています。

2.1.3.1.2 メモリ使用率レポート: `sar -r`

システムに搭載されたメモリ (RAM) に関する概要を知りたい場合は、オプション `-r` を使用します:

```
# sar -r 10 5
Linux 6.4.0-150600.9-default (jupiter)         2024年03月11日  _x86_64_        (2 CPU)

17時55分27秒 kbmemfree kbmemused %memused kbbuffers kbcached kbcommit %commit kbactive kbinact kbdirty
17時55分37秒    104232   1834624     94.62        20    627340   2677656    66.24    802052   828024    1744
17時55分47秒     98584   1840272     94.92        20    624536   2693936    66.65    808872   826932    2012
17時55分57秒     87088   1851768     95.51        20    605288   2706392    66.95    827260   821304    1588
17時56分07秒     86268   1852588     95.55        20    599240   2739224    67.77    829764   820888    3036
17時56分17秒    104260   1834596     94.62        20    599864   2730688    67.56    811284   821584    3164
平均値:          96086   1842770     95.04        20    611254   2709579    67.03    815846   823746    2309
```

[kbcommit] と [%commit] の列は、それぞれ現在の処理内容に応じた最大メモリ量 (物理メモリとスワップ領域) の概算を表しています。[kbcommit] はキロバイト単位で、[%commit] は割合でそれぞれ表現されています。

2.1.3.1.3 ページング統計レポート: `sar -B`

カーネルのページングに関する統計情報を表示したい場合は、`-B` オプションを使用します。

```
# sar -B 10 5
Linux 6.4.0-150600.9-default (jupiter)         2024年11月03日  _x86_64_        (2 CPU)

18時23分01秒 pgpgin/s pgpgout/s fault/s majflt/s pgfree/s pgscank/s pgscand/s pgsteal/s %vmeff
18時23分11秒   366.80     11.60  542.50     1.10  4354.80      0.00      0.00      0.00   0.00
18時23分21秒     0.00    333.30 1522.40     0.00 18132.40      0.00      0.00      0.00   0.00
18時23分31秒    47.20    127.40 1048.30     0.10 11887.30      0.00      0.00      0.00   0.00
18時23分41秒    46.40      2.50  336.10     0.10  7945.00      0.00      0.00      0.00   0.00
18時23分51秒     0.00    583.70 2037.20     0.00 17731.90      0.00      0.00      0.00   0.00
平均値:         92.08    211.70 1097.30     0.26 12010.28      0.00      0.00      0.00   0.00
```

[majflt/s] (メジャーフォルト毎秒) の列には、ディスクからメモリに読み込まれたページ数が表示されています。フォルトを発生させる要因としては、ファイルへのアクセスのほか、アプリケーションのバグによるものである場合もあります。ただし、メジャーフォルトが大きいからといって、それがそのまま何らかの問題を表しているとは限りません。たとえばアプリケーションの起動時には、一般的に様々なファイルを読み込むことから、メジャーフォルトも大きくなりがちです。メジャーフォルトがアプリケーションの動作中ずっと高いままで、特にダイレクトスキャンも同時に大きくなっているような場合は、メインメモリが不足していることを表している場合があります。

[%vmeff] の列には、メインメモリやスワップ領域のキャッシュ ([pgsteal/s]) から再利用されたページに対する、スキャン済みページ数 ([pgscand/s]) の割合を示しています。これはページの回収処理の効率性を表している値で、性能に問題のないシステムであれば、100 (スワップアウトした全ての非活動ページが再利用された) もしくは 0 (ページをスキャンしていない) に近い値になるはずです。この値は通常、30 を下回るべきではありません。



注記: 最近の Linux カーネルでは [%vmeff] の値が 100% を超える可能性がある件について

[%vmeff] の値が 100% 以上になっている場合、メモリの回収処理でスキャンしたページ数 ([pgscank]) よりも回収されたページ数 ([pgsteal]) のほうが大きいことを表していることになります。カーネルはスキャンしたページから回収の可否を判断するため、通常の処理としては発生しえない値となります。

ですが、最近のカーネル (バージョン 5.x およびそれ以降) ではメモリ管理とページ回収の方式が変更され、必ずしも [%vmeff] の値が仮想メモリの効率を表すわけではなくなっています。これらの変更では回収処理を表側の処理と裏側の処理に分割し、メモリ操作をより詳細に追

跡するようになっているほか、遅延再利用と呼ばれる仕組みが追加され、NUMA に対応し、cgroup ベースのメモリ管理を行うようになっています。このような仕組みにより、`sar` のようなツールは［pgsteal］と［pgscank］を謝って解釈してしまうようになりますので、これによって特定状況下で［%vmeff］が 100% を超える場合があります。

なお、［%vmeff］が 100% を超えるような場合は、この値を性能指標として使用するのは避けてください。その代わり、［pgpgin/s］と［pgpgout/s］でページ処理を、［majflt/s］と［fault/s］でページフォルトを、`/proc/meminfo` でメモリ使用率の詳細 (例:［Active］,［Inactive］,［Dirty］,［Writeback］) をそれぞれ監視するようにしてください。これに加えて、これらの測定値をアプリケーションレベルの性能と I/O パターンに紐付けて、潜在的なメモリボトルネックや非効率を特定するようにしてください。

2.1.3.1.4 ブロックデバイス統計レポート: `sar -d`

`-d` オプションを使用することで、ブロックデバイス (ハードディスク、光学ドライブ、USB ストレージデバイスなど) の統計情報を表示することができます。［DEV］列を読みやすくしたい場合は、追加オプション `-p` (pretty-print, わかりやすい表示) を指定してください。

```
# sar -d -p 10 5
 Linux 6.4.0-150600.9-default (jupiter)         2024年11月03日  _x86_64_        (2 CPU)

18時46分09秒 DEV   tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
18時46分19秒 sda  1.70    33.60      0.00     19.76      0.00      0.47      0.47      0.08
18時46分19秒 sr0  0.00     0.00      0.00      0.00      0.00      0.00      0.00      0.00

18時46分19秒 DEV   tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
18時46分29秒 sda  8.60   114.40    518.10     73.55      0.06      7.12      0.93      0.80
18時46分29秒 sr0  0.00     0.00      0.00      0.00      0.00      0.00      0.00      0.00

18時46分29秒 DEV   tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
18時46分39秒 sda 40.50  3800.80    454.90    105.08      0.36      8.86      0.69      2.80
18時46分39秒 sr0  0.00     0.00      0.00      0.00      0.00      0.00      0.00      0.00

18時46分39秒 DEV   tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
18時46分49秒 sda  1.40     0.00    204.90    146.36      0.00      0.29      0.29      0.04
18時46分49秒 sr0  0.00     0.00      0.00      0.00      0.00      0.00      0.00      0.00

18時46分49秒 DEV   tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
18時46分59秒 sda  3.30     0.00    503.80    152.67      0.03      8.12      1.70      0.56
18時46分59秒 sr0  0.00     0.00      0.00      0.00      0.00      0.00      0.00      0.00

平均値: DEV   tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
平均値: sda 11.10   789.76    336.34    101.45      0.09      8.07      0.77      0.86
平均値: sr0  0.00     0.00      0.00      0.00      0.00      0.00      0.00      0.00
```

［平均値］内にある［tps］,［rd_sec/s］,［wr_sec/s］の各値を、全てのディスクに対して比較してください。［svctm］列や［%util］列の値が定常的に高い場合は、I/O サブシステムがボトルネックなっている可能性があります。

お使いのマシンに複数のディスクが接続されている場合、それらのディスクの速度と性能が等しい場合は、I/O を分散させることが最適です。また、ストレージが複数階層で構成されているかどうかも考慮する必要があります。これに加えて、ストレージへのアクセス方式が複数存在するような場合は、その使用方法を均等にする場合のリンク飽和度についても検討してください。

2.1.3.1.5 ネットワーク統計レポート: `sar -n` キーワード

`-n` オプションを使用することで、様々なネットワーク関連のレポートを生成することができます。`-n` の後ろに指定することのできるキーワードには、下記のものがあります:

- DEV : 全てのネットワークデバイスに対する統計レポートを生成
- EDEV : 全てのネットワークデバイスに対するエラー統計レポートを生成
- NFS : NFS クライアントに対する統計レポートを生成
- NFSD : NFS サーバに対する統計レポートを生成
- SOCK : ソケットに対する統計レポートを生成
- ALL : 全てのネットワーク統計レポートを生成

2.1.3.2 `sar` データの可視化

`sar` が生成するレポートは、必ずしも読みやすいものであるとは言い切れません。その代わり、kSar と呼ばれる `sar` データを可視化するための Java アプリケーションを使用することで、読みやすいグラフを生成することができます。このソフトウエアは PDF 形式のレポートにも対応しているほか、リアルタイムなデータとファイルに保存された過去データの両方に対応しています。kSar は BSD ライセンスで提供されています。詳しくは https://sourceforge.net/projects/ksar/ をお読みください。

2.2 システム情報

2.2.1 デバイスの負荷情報: `iostat`

システムのデバイス負荷を監視したい場合は、`iostat` をお使いください。このソフトウエアは、お使いのシステムに接続された複数のディスクの間で、負荷を分散させたいような場合に有用なレポートを生成します。

`iostat` を使用するには、`sysstat` パッケージをインストールしてください。

最初の `iostat` レポートには、システムの起動以降の収集データが表示されます。2 つ目以降のレポートには、前回のレポート生成以降の収集データが表示されます。

```
> iostat
Linux 6.4.0-150600.9-default (jupiter)         2024年11月03日  _x86_64_        (4 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          17.68    4.49    4.24    0.29    0.00   73.31

Device:             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sdb                2.02        36.74        45.73    3544894    4412392
sda                1.05         5.12        13.47     493753    1300276
sdc                0.02         0.14         0.00      13641         37
```

この方法で `iostat` を動作させることで、まずはスループットが期待通りの値になっているかどうかを確認することができます。ただし、原因については示されません。原因を調べたい場合は、`iostat -x` のように入力して実行し、さらに詳しい拡張レポートを生成してください。拡張レポートには、平均キューサイズや平均待機時間などの追加情報が含まれます。なお、動作していないブロックデバイスを除外して表示したい場合は、`-z` オプションを追加してください。それぞれの列の意味について、詳しくは `iostat` のマニュアルページ (`man 1 iostat`) をお読みください。

指定したデバイスを指定した間隔で監視することもできます。たとえば 3 秒間隔で合計 5 回、`sda` デバイスを監視したい場合は、下記のように実行します:

```
> iostat -p sda 3 5
```

ネットワークファイルシステム (NFS) の統計情報を表示したい場合は、下記に示す 2 種類の類似ユーティリティをお使いいただくことができます:

- `nfsiostat-sysstat`: `sysstat` パッケージ内に含まれています。
- `nfsiostat`: `nfs-client` パッケージ内に含まれています。



注記: マルチパス環境での `iostat` の使用について

`iostat` を利用しても、`nvme list-subsys` で一覧表示されるコントローラを全て表示するわけではありません。既定では `iostat` は、I/O の発生していないブロックデバイスを表示しないためです。`iostat` で 全て のデバイスを表示するようにしたい場合は、下記のようなコマンドを実行してください:

```
>  iostat -p ALL
```

2.2.2 プロセッサ動作の監視: mpstat

`mpstat` ユーティリティは、利用可能なプロセッサに対して、動作状況を調べることができます。お使いのシステムにプロセッサが 1 つしか接続されていない場合、グローバルな平均統計情報が出力されます。

間隔の指定は `iostat` コマンドと同じです。たとえば `mpstat 2 5` のように入力して実行すると、2 秒間隔で合計 5 回のレポートを生成します。

```
# mpstat 2 5
Linux 6.4.0-150600.9-default (jupiter)        2024年11月03日  _x86_64_        (2 CPU)

13時51分10秒  CPU   %usr  %nice  %sys  %iowait  %irq  %soft  %steal  %guest  %gnice   %idle
13時51分12秒  all   8,27   0,00  0,50     0,00  0,00   0,00    0,00    0,00    0,00   91,23
13時51分14秒  all  46,62   0,00  3,01     0,00  0,00   0,25    0,00    0,00    0,00   50,13
13時51分16秒  all  54,71   0,00  3,82     0,00  0,00   0,51    0,00    0,00    0,00   40,97
13時51分18秒  all  78,77   0,00  5,12     0,00  0,00   0,77    0,00    0,00    0,00   15,35
13時51分20秒  all  51,65   0,00  4,30     0,00  0,00   0,51    0,00    0,00    0,00   43,54
平均値:       all  47,85   0,00  3,34     0,00  0,00   0,40    0,00    0,00    0,00   48,41
```

`mpstat` のデータは、下記のようにして判断します:

- まずは［%usr］と［%sys］の比率を調べます。たとえば 10:1 のように、［%usr］のほうがずっと高い場合、負荷はアプリケーション側のコードによって生み出されているもので、アプリケーション側の調査を行う必要があることを示しています。逆に 1:10 のように、［%sys］のほうがずっと高い場合、負荷はカーネル側が生み出しているもので、カーネルのチューニングを検討する必要があることになります。このほか、アプリケーションがなぜカーネルを頼りがちなのかや、それを軽減できないかを調べる手もあります。
- システム全体では負荷が軽いのに、特定の CPU に負荷が集中しているような事象が発生していないかどうかを調べます。特定の CPU に負荷が集中する場合、それは負荷が適切に並列化されていないことを表しているため、より高速で少ない数のプロセッサの環境に構成し直したほうが良いことを示していることもあります。

2.2.3 プロセッサ周波数の監視: turbostat

`turbostat` は AMD64/Intel 64 プロセッサに対して、周波数や負荷、温度や速度を表示するためのコマンドです。このコマンドには 2 つのモードが存在しています。`turbostat` の後ろに実行したいコマンドラインを指定した場合は、指定したコマンドラインを実行し、完了時に統計情報を表示します。コマンドラインを指定しない場合、5 秒間隔で統計情報を更新して表示します。なお `turbostat` を実行するには、`msr` カーネルモジュールを読み込んでおく必要があります。

```
> sudo turbostat find /etc -type d -exec true {} \;
0.546880 sec
     CPU Avg_MHz   Busy% Bzy_MHz TSC_MHz
       -     416   28.43    1465    3215
```

```
       0     631   37.29    1691    3215
       1     416   27.14    1534    3215
       2     270   24.30    1113    3215
       3     406   26.57    1530    3214
       4     505   32.46    1556    3214
       5     270   22.79    1184    3214
```

出力される内容は CPU によって異なります。温度や消費電力などの追加情報を表示したい場合は、`--debug` オプションを指定してください。コマンドラインオプションの説明や、各項目の意味について、詳しくは `man 8 turbostat` をお読みください。

2.2.4 タスクの監視: `pidstat`

お使いのシステム内で、どの処理 (プロセス) が重いのかを調べたい場合は、`pidstat` コマンドをお使いください。選択したプロセスのみの動作状況を表示することができるほか、Linux カーネルが管理する全てのプロセスを表示することもできます。また、生成するレポート数や生成間隔なども指定することができます。

たとえば `pidstat -C` firefox 2 3 のように入力して実行すると、コマンド名に「firefox」という文字列を含むプロセスに関する情報を出力します。レポートは 2 秒間隔で計 3 回出力します。

```
# pidstat -C firefox 2 3
Linux 6.4.0-150600.9-default (jupiter)         2024年11月03日  _x86_64_        (2 CPU)

14時09分11秒     UID       PID    %usr %system  %guest    %CPU   CPU  Command
14時09分13秒    1000       387   22,77    0,99    0,00   23,76     1  firefox

14時09分13秒     UID       PID    %usr %system  %guest    %CPU   CPU  Command
14時09分15秒    1000       387   46,50    3,00    0,00   49,50     1  firefox

14時09分15秒     UID       PID    %usr %system  %guest    %CPU   CPU  Command
14時09分17秒    1000       387   60,50    7,00    0,00   67,50     1  firefox

平均値:         UID       PID    %usr %system  %guest    %CPU   CPU  Command
平均値:        1000       387   43,19    3,65    0,00   46,84     -  firefox
```

同様に、`pidstat -d` と入力して実行することで、プロセスがどれだけ I/O を実行しているのか、その I/O の間にプロセスが休眠状態にあるのかどうか、そしてどれだけの時間タスクが停止していたのかを調べることができます。

2.2.5 カーネルのリングバッファの表示: `dmesg`

Linux カーネルは出力したメッセージをリングバッファ内に保持しています。保持しているメッセージを表示したい場合は、`dmesg -T` のように入力して実行します。

ここに保持されていない古いメッセージは、`systemd` のジャーナル内に保存されています。ジャーナルに関する詳細は、『リファレンス』、第11章「`journalctl` : systemd ジャーナルへの問い合わせコマンド」をお読みください。

2.2.6 開いているファイルの一覧: `lsof`

プロセス ID を指定して開いているファイルを表示したい場合は、`lsof -p プロセス_ID` のように入力して実行します。たとえば現在のシェルが開いているファイルを表示したい場合は、下記のようにします:

```
# lsof -p $$
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF  NODE NAME
bash    8842 root  cwd    DIR   0,32      222  6772 /root
bash    8842 root  rtd    DIR   0,32      166   256 /
bash    8842 root  txt    REG   0,32   656584 31066 /bin/bash
bash    8842 root  mem    REG   0,32  1978832 22993 /lib64/libc-2.19.so
[...]
bash    8842 root    2u   CHR  136,2      0t0     5 /dev/pts/2
bash    8842 root  255u   CHR  136,2      0t0     5 /dev/pts/2
```

ここで、`$$` は特殊なシェル変数で、現在使用しているシェルのプロセス ID を表します。

`-i` オプションを指定すると、`lsof` はネットワーク接続を表示することができます:

```
# lsof -i
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
wickedd-d  917 root    8u  IPv4  16627      0t0  UDP *:bootpc
wickedd-d  918 root    8u  IPv6  20752      0t0  UDP [fe80::5054:ff:fe72:5ead]:dhcpv6-client
sshd      3152 root    3u  IPv4  18618      0t0  TCP *:ssh (LISTEN)
sshd      3152 root    4u  IPv6  18620      0t0  TCP *:ssh (LISTEN)
master    4746 root   13u  IPv4  20588      0t0  TCP localhost:smtp (LISTEN)
master    4746 root   14u  IPv6  20589      0t0  TCP localhost:smtp (LISTEN)
sshd      8837 root    5u  IPv4 293709      0t0  TCP jupiter.suse.de:ssh->venus.suse.de:33619 (ESTABLISHED)
sshd      8837 root    9u  IPv6 294830      0t0  TCP localhost:x11 (LISTEN)
sshd      8837 root   10u  IPv4 294831      0t0  TCP localhost:x11 (LISTEN)
```

2.2.7 カーネルと udev のイベントシーケンスの表示: `udevadm monitor`

`udevadm monitor` のように入力して実行すると、カーネルが生成する udev のイベントや udev ルールによって送信されたイベントのほか、そのイベントのデバイスパス (DEVPATH) をコンソールに出力することができます。下記は USB メモリを接続した場合のイベント例です:



注記: udev イベントの監視について

`udevadm` コマンドを実行して udev イベントを監視するには、root 権限が必要となります。

```
UEVENT[1138806687] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2
UEVENT[1138806687] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2/4-2.2
UEVENT[1138806687] add@/class/scsi_host/host4
UEVENT[1138806687] add@/class/usb_device/usbdev4.10
UDEV  [1138806687] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2
UDEV  [1138806687] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2/4-2.2
UDEV  [1138806687] add@/class/scsi_host/host4
UDEV  [1138806687] add@/class/usb_device/usbdev4.10
UEVENT[1138806692] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2/4-2.2
UEVENT[1138806692] add@/block/sdb
UEVENT[1138806692] add@/class/scsi_generic/sg1
UEVENT[1138806692] add@/class/scsi_device/4:0:0:0
UDEV  [1138806693] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2/4-2.2
UDEV  [1138806693] add@/class/scsi_generic/sg1
UDEV  [1138806693] add@/class/scsi_device/4:0:0:0
UDEV  [1138806693] add@/block/sdb
UEVENT[1138806694] add@/block/sdb/sdb1
UDEV  [1138806694] add@/block/sdb/sdb1
UEVENT[1138806694] mount@/block/sdb/sdb1
UEVENT[1138806697] umount@/block/sdb/sdb1
```

2.3 プロセス

2.3.1 プロセス間通信の情報: `ipcs`

`ipcs` コマンドは、現在使用されている IPC リソースの一覧を表示します:

```
# ipcs
------ メッセージキュー --------
```

```
キー        msqid      所有者      権限         使用済みバイト数   メッセージ

------ 共有メモリセグメント --------
キー        shmid      所有者      権限         バイト      nattch     状態
0x00000000 65536      tux         600          524288     2          dest
0x00000000 98305      tux         600          4194304    2          dest
0x00000000 884738     root        600          524288     2          dest
0x00000000 786435     tux         600          4194304    2          dest
0x00000000 12058628   tux         600          524288     2          dest
0x00000000 917509     root        600          524288     2          dest
0x00000000 12353542   tux         600          196608     2          dest
0x00000000 12451847   tux         600          524288     2          dest
0x00000000 11567114   root        600          262144     1          dest
0x00000000 10911763   tux         600          2097152    2          dest
0x00000000 11665429   root        600          2336768    2          dest
0x00000000 11698198   root        600          196608     2          dest
0x00000000 11730967   root        600          524288     2          dest

------ セマフォ配列 --------
キー        semid      所有者      権限        nsems
0xa12e0919 32768      tux         666         2
```

2.3.2 プロセス一覧: `ps`

`ps` コマンドはプロセスの一覧を表示します。このコマンドでは、ほとんどのパラメータをハイフン無しで指定しなければなりません。簡潔なヘルプを読みたい場合は `ps --help` と入力して実行し、より詳細な説明を読みたい場合はマニュアルページを参照してください。

ユーザとコマンドライン情報を含めて全てのプロセスを一覧表示するには、`ps axu` のように入力して実行します:

```
> ps axu
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.3  34376  4608 ?        Ss   Jul24   0:02 /usr/lib/systemd/systemd
root         2  0.0  0.0      0     0 ?        S    Jul24   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    Jul24   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   Jul24   0:00 [kworker/0:0H]
root         6  0.0  0.0      0     0 ?        S    Jul24   0:00 [kworker/u2:0]
root         7  0.0  0.0      0     0 ?        S    Jul24   0:00 [migration/0]
[...]
tux      12583  0.0  0.1 185980  2720 ?        Sl   10:12   0:00 /usr/lib/gvfs/gvfs-mtp-volume-monitor
tux      12587  0.0  0.1 198132  3044 ?        Sl   10:12   0:00 /usr/lib/gvfs/gvfs-gphoto2-volume-monitor
tux      12591  0.0  0.1 181940  2700 ?        Sl   10:12   0:00 /usr/lib/gvfs/gvfs-goa-volume-monitor
tux      12594  8.1 10.6 1418216 163564 ?      Sl   10:12   0:03 /usr/bin/gnome-shell
tux      12600  0.0  0.3 393448  5972 ?        Sl   10:12   0:00 /usr/lib/gnome-settings-daemon-3.0/gsd-
printer
tux      12625  0.0  0.6 227776 10112 ?        Sl   10:12   0:00 /usr/lib/gnome-control-center-search-
provider
tux      12626  0.5  1.5 890972 23540 ?        Sl   10:12   0:00 /usr/bin/nautilus --no-default-window
[...]
```

どれだけの数の `sshd` プロセスが動作しているのかを調べたい場合は、`-p` オプションとともに `pidof` コマンドを併用してください。`pidof` コマンドは、指定した名前のプロセス ID を列挙するためのコマンドです。

```
> ps -p $(pidof sshd)
  PID TTY      STAT   TIME COMMAND
 1545 ?        Ss     0:00 /usr/sbin/sshd -D
 4608 ?        Ss     0:00 sshd: root@pts/0
```

プロセス一覧の表示は、必要に応じて書式を変更することができます。`L` オプションを使用すると、全てのキーワードの一覧を表示することができます。たとえばメモリの使用率順に全てのプロセスを一覧表示したい場合は、下記のように入力して実行します:

```
> ps ax --format pid,rss,cmd --sort rss
  PID   RSS CMD
  PID   RSS CMD
    2     0 [kthreadd]
    3     0 [ksoftirqd/0]
    4     0 [kworker/0:0]
    5     0 [kworker/0:0H]
    6     0 [kworker/u2:0]
    7     0 [migration/0]
    8     0 [rcu_bh]
[...]
12518 22996 /usr/lib/gnome-settings-daemon-3.0/gnome-settings-daemon
12626 23540 /usr/bin/nautilus --no-default-window
12305 32188 /usr/bin/Xorg :0 -background none -verbose
12594 164900 /usr/bin/gnome-shell
```

便利な `ps` のオプション

`ps aux --sort 列名`
: `列名` で指定した項目の値で並べ替えを行います。`列名` には下記の名前を指定することができます:

  `pmem`: 物理メモリの比率
  `pcpu`: CPU 使用率
  `rss`: 常駐セットサイズ (スワップ領域に移動していない物理メモリ)

`ps axo pid,%cpu,rss,vsz,args,wchan`
: 各プロセスに対して、プロセス ID, CPU 使用率, メモリサイズ (常駐および仮想), 名前, システムコールをそれぞれ表示します。

`ps axfo pid,args`
: プロセスをツリー形式で表示します。

2.3.3 プロセスのツリー表示: `pstree`

`pstree` コマンドは、プロセスの一覧をツリー形式で出力します:

```
> pstree
systemd---accounts-daemon---{gdbus}
        |                 |-{gmain}
        |-at-spi-bus-laun---dbus-daemon
        |                 |-{dconf worker}
        |                 |-{gdbus}
        |                 |-{gmain}
        |-at-spi2-registr---{gdbus}
        |-cron
        |-2*[dbus-daemon]
        |-dbus-launch
        |-dconf-service---{gdbus}
        |               |-{gmain}
        |-gconfd-2
        |-gdm---gdm-simple-slav---Xorg
        |     |                 |-gdm-session-wor---gnome-session---gnome-setti+
        |     |                 |                 |               |-gnome-shell+++
        |     |                 |                 |               |-{dconf work+
        |     |                 |                 |               |-{gdbus}
        |     |                 |                 |               |-{gmain}
        |     |                 |                 |-{gdbus}
        |     |                 |                 |-{gmain}
        |     |                 |-{gdbus}
        |     |                 |-{gmain}
        |     |-{gdbus}
        |     |-{gmain}
[...]
```

`-p` オプションを指定すると、プロセス名に加えてプロセス ID が表示されるようになります。コマンドラインについても表示したい場合は、`-a` を指定してください。

2.3.4 プロセスの一覧表示: `top`

`top` (「Table Of Processes」(プロセス表) の略) コマンドは、2 秒間隔で更新されるプロセス表を表示します。プログラムを終了するには `Q` を押します。`-n 1` とパラメータを指定すると、プロセス表を一度だけ表示して終了します。下記は `top -n 1` コマンドの出力例です:

```
> top -n 1
Tasks: 128 total,   1 running, 127 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2.4 us,  1.2 sy,  0.0 ni, 96.3 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   1535508 total,   699948 used,   835560 free,      880 buffers
KiB Swap:  1541116 total,        0 used,  1541116 free.   377000 cached Mem
```

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    1 root      20   0  116292   4660   2028 S 0.000 0.303   0:04.45 systemd
    2 root      20   0       0      0      0 S 0.000 0.000   0:00.00 kthreadd
    3 root      20   0       0      0      0 S 0.000 0.000   0:00.07 ksoftirqd+
    5 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 kworker/0+
    6 root      20   0       0      0      0 S 0.000 0.000   0:00.00 kworker/u+
    7 root      rt   0       0      0      0 S 0.000 0.000   0:00.00 migration+
    8 root      20   0       0      0      0 S 0.000 0.000   0:00.00 rcu_bh
    9 root      20   0       0      0      0 S 0.000 0.000   0:00.24 rcu_sched
   10 root      rt   0       0      0      0 S 0.000 0.000   0:00.01 watchdog/0
   11 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 khelper
   12 root      20   0       0      0      0 S 0.000 0.000   0:00.00 kdevtmpfs
   13 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 netns
   14 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 writeback
   15 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 kintegrit+
   16 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 bioset
   17 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 crypto
   18 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 kblockd
```

既定では、出力は CPU の使用率 ([%CPU] 列) の降順に並んでいます。下記のキー入力を行うことで、並べ替えに使用する項目を変更することができます (CPU の使用率に戻したい場合は、`Shift`–`P` を押します) :

`Shift`–`M` : 常駐メモリ ([RES])
`Shift`–`N` : プロセス ID ([PID])
`Shift`–`T` : CPU 時間 ([TIME+])

その他の項目で並べ替えを行いたい場合は、`F` を押し表示された項目から選んでください。並べ替えの順序を逆にしたい場合は、`Shift`–`R` を押してください。

また、`-U UID` のようにオプションを指定することで、特定のユーザに結びついているプロセスのみを監視することができます。なお、`UID` はユーザの ID を表します。現在使用しているユーザのプロセスを表示したい場合は、`top -U $(id -u)` と入力して実行してください。

2.3.5 top コマンドに似た I/O モニタ: `iotop`

`iotop` ユーティリティは、プロセスやスレッドごとの I/O 使用率を表示します。



注記: `iotop` のインストールについて

`iotop` は既定ではインストールされません。`root` で `zypper in iotop` と入力して実行し、インストールを行ってください。

`iotop` は一定のサンプリング期間内に行われた読み込みおよび書き込みの処理を、プロセスごとにまとめて表示します。スワップ領域の読み込みや I/O の完了待ちなどで消費されたプロセス時間の割合も表示することができます。また、各プロセスの I/O 優先度 (クラス／レベル) も表示されます。これに加えて、システム全体での読み込みおよび書き込みの処理を、インターフェイスの上部に表示します。

- `←` や `→` のキーを押すことで、並べ替える項目を選択することができます。
- `R` を押すと並び順を逆にすることができます。
- `O` を押すと、全てのプロセスおよびスレッドを表示するビュー (既定の表示) と、I/O が実際に行われているもののみを表示するビューとの間を切り替えることができます (この機能は、コマンドラインに `--only` オプションを追加した場合に似た表示になります) 。
- `P` を押すと、スレッド表示 (既定の表示) とプロセス表示との間を切り替えることができます (この機能は、コマンドラインに `--only` オプションを追加した場合に似た表示になります) 。
- `A` を押すと、現在の I/O 帯域 (既定の表示) と `iotop` 起動時からの累積値表示との間を切り替えることができます (この機能は、コマンドラインに `--only` オプションを追加した場合に似た表示になります) 。
- `I` を押すと、スレッドやプロセス内のスレッドに対して、優先順位を変更することができます。
- `Q` を押すと `iotop` を終了することができます。
- 上記以外のキーを押すと、強制的な更新処理を行います。

下記は `iotop --only` の出力例で、ここでは `find` と `emacs` が動作しています:

```
# iotop --only
Total DISK READ: 50.61 K/s | Total DISK WRITE: 11.68 K/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 3416 be/4 tux         50.61 K/s    0.00 B/s  0.00 %  4.05 % find /
  275 be/3 root        0.00 B/s    3.89 K/s  0.00 %  2.34 % [jbd2/sda2-8]
 5055 be/4 tux          0.00 B/s    3.89 K/s  0.00 %  0.04 % emacs
```

`iotop` コマンドはバッチモード (`-b`) でも動作させることができます。この場合、ファイルに出力して後から分析するような使い方をすることもできます。オプションの一覧について、詳しくは (`man 8 iotop`) で表示されるマニュアルページをお読みください。

2.3.6 プロセスの優先順位設定: `nice` および `renice`

カーネルはプロセスの優先順位 (nice レベル, niceness) を利用して、他のプロセスとの間の CPU 時間の調整を行います。プロセスの「nice」レベルが高いほど、割り当てられる CPU 時間は少なくなります。nice レベルは -20 (最も低い「nice」レベル) から 19 までの間で指定します。また、負の値の nice レベルは、`root` のみが設定可能です。

niceness レベルの調整は、長時間 CPU 時間を占有し続けるような処理で、特に時間的な制約のない処理を動作させるような場合に有用です。たとえば他の処理も同時に動作しているシステム内で、カーネルのコンパイル処理を行ったりするような場合が該当します。このような処理の「nice」レベルを上げることで、たとえば Web サーバの処理をより優先して動作させることができるようになります。

`nice` コマンドに何もパラメータを指定しないで実行すると、現在の niceness を表示することができます:

```
> nice
0
```

`nice コマンド` のようにしてコマンドを実行すると、そのコマンドを起動する際に nice レベルを 10 足して実行します。また、`nice -n レベル コマンド` のように入力して実行すると、現在の nice レベルに指定したレベルを足して、コマンドを実行することができます。

動作中のプロセスに対して nice レベルを変更したい場合は、`renice レベル -p プロセス_ID` のように入力して実行します。たとえば下記のようになります:

```
> renice +5 3266
```

特定のユーザが所有する全てのプロセスの nice レベルを変更したい場合は、`-u ユーザ` のように指定して実行します。プロセスグループを指定して nice レベルを変更する場合は、`-g プロセスグループ_ID` のように入力して実行します。

2.4 メモリ

2.4.1 メモリ使用率の表示: `free`

`free` コマンドはメモリとスワップ領域の使用量を表示します。それぞれ空き／使用中のメモリとスワップ領域の詳細が出力されます:

```
> free
              total        used        free      shared     buffers      cached
```

```
Mem:      32900500   32703448     197052          0     255668    5787364
-/+ buffers/cache:   26660416    6240084
Swap:      2046972     304680    1742292
```

表示の単位を変更したい場合は、`-b` (バイト単位), `-k` (キロバイト単位), `-m` (メガバイト単位), `-g` (ギガバイト単位) のように指定します。`-s 遅延時間` のようにオプションを指定すると、`遅延時間` で指定した時間ごとに更新を行って表示します。たとえば `free -s 1.5` のように入力して実行すると、1.5 秒間隔で更新して表示します。

2.4.2 詳細なメモリ使用率情報の取得: /proc/meminfo

`free` で出力されるメモリ使用率よりも詳しい情報を取得したい場合は、`/proc/meminfo` ファイルをお使いください。このファイルは `free` コマンドも使用しているファイルです。64 ビット環境での出力例を下記に示します。なお、32 ビット環境ではメモリの管理方式が異なるため、出力が少し異なることに注意してください:

```
MemTotal:        1942636 kB
MemFree:         1294352 kB
MemAvailable:    1458744 kB
Buffers:             876 kB
Cached:           278476 kB
SwapCached:            0 kB
Active:           368328 kB
Inactive:         199368 kB
Active(anon):     288968 kB
Inactive(anon):    10568 kB
Active(file):      79360 kB
Inactive(file):   188800 kB
Unevictable:          80 kB
Mlocked:              80 kB
SwapTotal:       2103292 kB
SwapFree:        2103292 kB
Dirty:                44 kB
Writeback:             0 kB
AnonPages:        288592 kB
Mapped:            70444 kB
Shmem:             11192 kB
Slab:              40916 kB
SReclaimable:      17712 kB
SUnreclaim:        23204 kB
KernelStack:        2000 kB
PageTables:        10996 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     3074608 kB
```

```
Committed_AS:    1407208 kB
VmallocTotal:   34359738367 kB
VmallocUsed:       145996 kB
VmallocChunk:   34359588844 kB
HardwareCorrupted:     0 kB
AnonHugePages:     86016 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:       79744 kB
DirectMap2M:     2017280 kB
```

項目の意味は下記のとおりです:

[MemTotal]
: 搭載されているメモリ量です。

[MemFree]
: 未使用のメモリ量です。

[MemAvailable]
: 新しく起動するアプリケーションで、スワップ処理を行わずに確保できるメモリ量の見積値です。

[Buffers]
: ファイルシステムのメタデータを含むメモリ内のバッファキャッシュ量です。

[Cached]
: メモリ内のページキャッシュ量です。この量にはバッファキャッシュとスワップキャッシュが含まれていませんが、[Shmem] のメモリ量は含まれています。

[SwapCached]
: スワップアウトされたメモリに対するページキャッシュ量です。

[Active], [Active(anon)], [Active(file)]
: どうしても必要な場合や明示的に指定されたりした場合以外には回収されることのない、使用中のメモリ量。[Active] は [Active(anon)] と [Active(file)] の合計値です:

  - [Active(anon)] はスワップ領域に結びつけられているメモリ量です。この量には、コピーオンライト後のプライベート, 共有匿名マッピング, プライベートファイルページが含まれています。
  - [Active(file)] はファイルシステムに結びつけられているメモリ量です。

[Inactive], [Inactive(anon)], [Inactive(file)]
: 通常は真っ先に回収される、直近では未使用のメモリ量です。[Inactive] は [Inactive(anon)] と [Inactive(file)] の合計値です:

  - [Inactive(anon)] はスワップ領域に結びつけられているメモリ量です。この量には、コピーオンライト後のプライベート, 共有匿名マッピング, プライベートファイルページが含まれています。
  - [Inactive(file)] はファイルシステムに結びつけられているメモリ量です。

[Unevictable]
: 回収することのできない (たとえば [Mlocked] されているものや RAM ディスクとして使用されているものなど) メモリ量です。

[Mlocked]
: `mlock` システムコールに結びつけられているメモリ量です。`mlock` はプロセスに対して、仮想メモリをどの物理メモリに割り当てるのかを指定できる仕組みです。しかしながら、`mlock` はその配置を保証できるものではありません。

[SwapTotal]
: スワップ領域の量です。

[SwapFree]
: 未使用のスワップ領域の量です。

[Dirty]
: ディスクに書き込むべきデータを保持していて、ディスクへの書き込みを待っているメモリの量です。[Dirty] に指定されたデータはアプリケーション側から明示的に同期を行うか、しばらくしてカーネルが同期を行うことになります。この [Dirty] 領域が大きい場合、ディスクへの書き込みに時間を要することが考えられます。また、任意の時点での [Dirty] 領域のサイズは、`sysctl` パラメータの `vm.dirty_ratio` もしくは `vm.dirty_bytes` で設定することができます (詳しくは 15.1.5項「ライトバック」をお読みください) 。

[Writeback]
: 現在ディスクに書き込もうとしているメモリの量です。

[Mapped]
: `mmap` システムコールで確保されたメモリ量です。

[Shmem]
: IPC データや `tmpfs` 、および共有匿名メモリなどのプロセス間共有メモリの量です。

[Slab]
: カーネルの内部データ構造向けの割り当て済みメモリ量です。

[SReclaimable]
: キャッシュ (inode, dentry ほか) などの回収可能な Slab セクションの量です。

[SUnreclaim]
: 回収することのできない Slab セクションの量です。

[KernelStack]
: システムコールを介して、アプリケーションが使用しているカーネル領域メモリの量です。

[PageTables]
: 全てのプロセスのページテーブル専用メモリ量です。

[NFS_Unstable]
: 既にサーバ宛には送信されたものの、まだコミットされていない NFS ページの量です。

[Bounce]
: ブロックデバイスのバウンスバッファ向けに使用されているメモリ量です。

[WritebackTmp]
: 一時的な書き戻しバッファのために FUSE が使用するメモリ量です。

[CommitLimit]
: オーバーコミット比率設定をベースにした、システムで利用できるメモリ量です。この項目は、厳密なオーバーコミットアカウンティングが有効な場合にのみ強制されます。

[Committed_AS]
: 現在の負荷量で、最悪の場合に必要となるメモリ量 (物理メモリとスワップ) の合計見積値です。

[VmallocTotal]
: 割り当て済みのカーネル仮想アドレス領域の量です。

[VmallocUsed]
: カーネルの仮想アドレス領域のうち、使用中の量です。

[VmallocChunk]
: 利用可能なカーネルの仮想アドレス領域のうち、最も大きく連続したブロックの量です。

[HardwareCorrupted]
: 障害の発生しているメモリの量です (ECC メモリを使用している場合にのみ検出可能です) 。

[AnonHugePages]
: ユーザスペースのページテーブルにマップされている、匿名の huge page の量です。これらは特にプロセス側から要求されることなく透過的に割り当てられるため、透過型 HugePages としても知られています。

[HugePages_Total]
: `SHM_HUGETLB` と `MAP_HUGETLB` 、もしくは `hugetlbfs` ファイルシステムを介して使用することのできる、事前割り当て型の huge page の量です。`/proc/sys/vm/nr_hugepages` で定義されています。

[HugePages_Free]
: 利用可能な huge page の量です。

[HugePages_Rsvd]
: コミット済みの huge page の量です。

[HugePages_Surp]
: [HugePages_Total] を越えて使用することのできる huge page (「surplus」) の量です。`/proc/sys/vm/nr_overcommit_hugepages` で定義されています。

[Hugepagesize]
: huge page のサイズです。AMD64/Intel 64 の既定値は 2048 KB です。

[DirectMap4k] など
: 指定のサイズ (例: 4 kB) でページにマップされたカーネルメモリの量です。

2.4.3 プロセスのメモリ使用率状況の表示: smaps

`top` や `ps` コマンドのような標準ツールを使用しても、特定のプロセスがどれだけのメモリを使用しているのかを正確に知るのは困難です。正確なデータを必要とする場合は、カーネルバージョン 2.6.14 で導入された smaps サブシステムをお使いください。smaps サブシステムでは、プロセスごとに `/proc/プロセス_ID/smaps` ファイルが提供されます。ここには、その時点で特定のプロセスが使用していたメモリ量のうち、クリーンなものとダーティなものの両方を表示することができます。また、共有メモリとプライベートメモリを区別して表示しますので、他のプロセスとは共有していないメモリ量を正確に知ることができます。詳しくは `/usr/src/linux/Documentation/filesystems/proc.txt` ファイル (英語, `kernel-source` パッケージをインストールする必要があります) をお読みください。

smaps は読みにくい出力形態になってしまっています。そのため、定期的な監視としては使用せず、プロセスを特定してからお読みになることをお勧めします。

2.4.4 numaTOP

numaTOP は NUMA (Non-uniform Memory Access) システム向けのツールです。このツールは、NUMA システムのリアルタイム分析機能を提供して、NUMA に関する性能ボトルネックを発見することができるツールです。

一般的には、numaTOP はローカリティの低い (ローカルメモリよりもリモートメモリを多く使用している) プロセスやスレッドを識別したり、調査したりすることができます。これは、それぞれ Remote Memory Accesses (RMA) と Local Memory Accesses (LMA) の数値と、RMA/LMA の比率を分析することで行います。

numaTOP は PowerPC プロセッサのほか、Intel Xeon プロセッサのうち 5500 シリーズ, 6500/7500 シリーズ, 5600 シリーズ, E7-x8xx シリーズ, E5-16xx/24xx/26xx/46xx シリーズでそれぞれ動作します。

numaTOP は公式のソフトウエアリポジトリ内に用意されています。`sudo zypper in numatop` コマンドを実行してインストールしてください。numaTOP を起動するには `numatop` コマンドを実行してください。numaTOP の機能や使用方法に関する概要を知りたい場合は、`man numatop` コマンドを実行してください。

2.5 ネットワーク



ヒント: トラフィックシェイピングについて

ネットワーク帯域が期待よりも狭いと感じた場合、まずはお使いのネットワークセグメント内で、トラフィックシェイピングルールが有効化されているかどうかをご確認ください。

2.5.1 基本的なネットワーク設定: `ip`

`ip` はシステム内のネットワークインターフェイスを設定したり制御したりすることのできるパワフルなツールです。ネットワークインターフェイスの基本的な統計情報なども取得することができます。たとえばインターフェイスが動作しているかどうかや、エラーの数、廃棄されたパケットやコリジョン (衝突) などの数を表示することができます。

何もパラメータを指定せずに `ip` コマンドを実行すると、ヘルプメッセージが表示されます。ネットワークインターフェイスの一覧を表示したい場合は、`ip addr show` (もしくは省略形で `ip a`) と入力して実行してください。`ip addr show up` と入力して実行すると、動作中のネットワークインターフェイスのみを表示することができます。また、`ip -s link show デバイス名` のように入力して実行すると、指定したインターフェイスの統計情報のみを表示します:

```
# ip -s link show br0
6: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
    link/ether 00:19:d1:72:d4:30 brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    6346104756 9265517  0       10860   0       0
    TX: bytes  packets  errors  dropped carrier collsns
    3996204683 3655523  0       0       0       0
```

`ip` コマンドはインターフェイスの表示 (`link`) のほか、ルーティングテーブルの表示 (`route`) など、多数の機能が用意されています。詳しくは `man 8 ip` をお読みください。

```
# ip route
default via 192.168.2.1 dev eth1
192.168.2.0/24 dev eth0  proto kernel  scope link  src 192.168.2.100
192.168.2.0/24 dev eth1  proto kernel  scope link  src 192.168.2.101
192.168.2.0/24 dev eth2  proto kernel  scope link  src 192.168.2.102
```

```
# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen
 1000
    link/ether 52:54:00:44:30:51 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen
 1000
    link/ether 52:54:00:a3:c1:fb brd ff:ff:ff:ff:ff:ff
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen
 1000
    link/ether 52:54:00:32:a4:09 brd ff:ff:ff:ff:ff:ff
```

2.5.2 プロセスごとのネットワーク使用率の表示: `nethogs`

たとえばネットワークトラフィックが急激に増加したような場合、その原因となるアプリケーションを素早く見つけることができたほうが都合のよいことがあります。`nethogs` コマンドは `top` コマンドに似た設計で、それぞれのプロセスが処理している送受信トラフィックを表示することができます:

```
PID   USER  PROGRAM                                  DEV   SENT   RECEIVED
27145 root   zypper                                  eth0  5.719  391.749 KB/sec
?     root   ..0:113:80c0:8080:10:160:0:100:30015          0.102    2.326 KB/sec
26635 tux    /usr/lib64/firefox/firefox              eth0  0.026    0.026 KB/sec
```

```
?     root   ..0:113:80c0:8080:10:160:0:100:30045        0.000    0.021 KB/sec
?     root   ..0:113:80c0:8080:10:160:0:100:30045        0.000    0.018 KB/sec
?     root   ..0:113:80c0:8080:10:160:0:100:30015        0.000    0.018 KB/sec
?     root   ..0:113:80c0:8080:10:160:0:100:30045        0.000    0.017 KB/sec
?     root   ..0:113:80c0:8080:10:160:0:100:30045        0.000    0.017 KB/sec
?     root   ..0:113:80c0:8080:10:160:0:100:30045        0.069    0.000 KB/sec
?     root   unknown TCP                                 0.000    0.000 KB/sec

TOTAL                                                  5.916  394.192 KB/sec
```

`top` コマンドと同様に、`nethogs` コマンドも対話的にコマンドを入力することができます:

`M` : 表示モードの切り替え (kb/s, kb, b, mb)
`R` : [RECEIVED] (受信) で並べ替え
`S` : [SENT] (送信) で並べ替え
`Q` : 終了

2.5.3 詳細なイーサネットカードの設定: ethtool

`ethtool` はお使いのイーサネットデバイスの詳細な要素を表示したり、変更したりすることのできるツールです。既定では、指定したデバイスに対する現在の設定値を表示します。

```
# ethtool eth0
Settings for eth0:
 Supported ports: [ TP ]
 Supported link modes:   10baseT/Half 10baseT/Full
                         100baseT/Half 100baseT/Full
                         1000baseT/Full
 Supports auto-negotiation: Yes
 Advertised link modes:  10baseT/Half 10baseT/Full
                         100baseT/Half 100baseT/Full
                         1000baseT/Full
 Advertised pause frame use: No
[...]
 Link detected: yes
```

下記の表には、様々な情報を取得するための `ethtool` オプションを示しています:

表 2.1: `ethtool` の問い合わせオプションの一覧

`ethtool` のオプション	意味
-a	Pause パラメータの情報の表示
-c	Interrupt Coalescing の情報の表示

`ethtool` のオプション	意味
-g	Rx/Tx (送信／受信) の Ring パラメータ情報の表示
-i	対応するドライバに関する情報の表示
-k	オフロード設定の表示
-S	NIC およびドライバ固有の統計情報の表示

2.5.4 ネットワーク状態の表示: `ss`

`ss` はソケットの統計情報を表示するコマンドで、以前に用いられていた `netstat` コマンドを置き換えるものです。全ての接続を表示したい場合は、何もパラメータを指定せずに `ss` を実行してください:

```
# ss
Netid  State      Recv-Q Send-Q   Local Address:Port       Peer Address:Port
u_str  ESTAB      0      0                    * 14082                 * 14083
u_str  ESTAB      0      0                    * 18582                 * 18583
u_str  ESTAB      0      0                    * 19449                 * 19450
u_str  ESTAB      0      0      @/tmp/dbus-gmUUwXABPV 18784           * 18783
u_str  ESTAB      0      0      /var/run/dbus/system_bus_socket 19383 * 19382
u_str  ESTAB      0      0      @/tmp/dbus-gmUUwXABPV 18617           * 18616
u_str  ESTAB      0      0      @/tmp/dbus-58TPPDv8qv 19352           * 19351
u_str  ESTAB      0      0                    * 17658                 * 17657
u_str  ESTAB      0      0                    * 17693                 * 17694
[..]
```

現在開いている全てのネットワークポートを表示したい場合は、下記のように実行してください:

```
# ss -l
Netid  State      Recv-Q Send-Q      Local Address:Port  Peer Address:Port
nl     UNCONN     0      0                 rtnl:4195117                  *
nl     UNCONN     0      0       rtnl:wickedd-auto4/811                  *
nl     UNCONN     0      0       rtnl:wickedd-dhcp4/813                  *
nl     UNCONN     0      0                 rtnl:4195121                  *
nl     UNCONN     0      0                 rtnl:4195115                  *
nl     UNCONN     0      0       rtnl:wickedd-dhcp6/814                  *
nl     UNCONN     0      0                  rtnl:kernel                  *
nl     UNCONN     0      0             rtnl:wickedd/817                  *
nl     UNCONN     0      0                 rtnl:4195118                  *
nl     UNCONN     0      0               rtnl:nscd/706                   *
nl     UNCONN     4352   0              tcpdiag:ss/2381                  *
[...]
```

ネットワーク接続を表示する際、特定のソケットタイプに限定することもできます。それぞれ TCP (`-t`) もしくは UDP (`-u`) などを指定してください。また、`-p` オプションを指定すると、それぞれのソケットが属するプロセス ID と、その名前を表示することができます。

下記の例では、全ての TCP 接続とそれに結びつけられたプログラムを表示しています。`-a` オプションは、確立済みの全ての接続 (待ち受け中のものを含む) を表示するためのオプションです。`-p` オプションを指定すると、それぞれのソケットが属するプロセス ID と、その名前を表示することができます。

```
# ss -t -a -p
State    Recv-Q Send-Q  Local Address:Port   Peer Address:Port
LISTEN   0      128                 *:ssh               *:*  users:(("sshd",1551,3))
LISTEN   0      100         127.0.0.1:smtp              *:*  users:(("master",1704,13))
ESTAB    0      132      10.120.65.198:ssh  10.120.4.150:55715  users:(("sshd",2103,5))
LISTEN   0      128                 :::ssh              :::*  users:(("sshd",1551,4))
LISTEN   0      100               ::1:smtp              :::*  users:(("master",1704,14))
```

2.6 /proc ファイルシステム

`/proc` ファイルシステムは擬似的なファイルシステムで、仮想的なファイルの形式で、カーネル内の様々な情報にアクセスすることができます。たとえば CPU の種類を表示したい場合は、下記のように実行します:

```
> cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 30
model name      : Intel(R) Core(TM) i5 CPU         750  @ 2.67GHz
stepping        : 5
microcode       : 0x6
cpu MHz         : 1197.000
cache size      : 8192 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 4
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 11
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
 pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc
 arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf pni dtes64 monitor
 ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm sse4_1 sse4_2 popcnt lahf_lm ida dtherm
 tpr_shadow vnmi flexpriority ept vpid
```

```
bogomips        : 5333.85
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
[...]
```



ヒント: 詳細なプロセッサ情報の取得について

AMD64/Intel 64 アーキテクチャでプロセッサの詳細情報を取得したい場合は、`x86info` コマンドをお使いいただいてもかまいません。

割り込みの割り当てと使用状況を確認するには、下記のように実行します:

```
> cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  0:        121          0          0          0   IO-APIC-edge      timer
  8:          0          0          0          1   IO-APIC-edge      rtc0
  9:          0          0          0          0   IO-APIC-fasteoi   acpi
 16:          0      11933          0          0   IO-APIC-fasteoi   ehci_hcd:+
 18:          0          0          0          0   IO-APIC-fasteoi   i801_smbus
 19:          0     117978          0          0   IO-APIC-fasteoi   ata_piix,+
 22:          0          0    3275185          0   IO-APIC-fasteoi   enp5s1
 23:     417927          0          0          0   IO-APIC-fasteoi   ehci_hcd:+
 40:    2727916          0          0          0  HPET_MSI-edge      hpet2
 41:          0    2749134          0          0  HPET_MSI-edge      hpet3
 42:          0          0    2759148          0  HPET_MSI-edge      hpet4
 43:          0          0          0    2678206  HPET_MSI-edge      hpet5
 45:          0          0          0          0   PCI-MSI-edge      aerdrv, P+
 46:          0          0          0          0   PCI-MSI-edge      PCIe PME,+
 47:          0          0          0          0   PCI-MSI-edge      PCIe PME,+
 48:          0          0          0          0   PCI-MSI-edge      PCIe PME,+
 49:          0          0          0        387   PCI-MSI-edge      snd_hda_i+
 50:     933117          0          0          0   PCI-MSI-edge      nvidia
NMI:       2102       2023       2031       1920   Non-maskable interrupts
LOC:         92         71         57         41   Local timer interrupts
SPU:          0          0          0          0   Spurious interrupts
PMI:       2102       2023       2031       1920   Performance monitoring int+
IWI:      47331      45725      52464      46775   IRQ work interrupts
RTR:          2          0          0          0   APIC ICR read retries
RES:     472911     396463     339792     323820   Rescheduling interrupts
CAL:      48389      47345      54113      50478   Function call interrupts
TLB:      28410      26804      24389      26157   TLB shootdowns
TRM:          0          0          0          0   Thermal event interrupts
THR:          0          0          0          0   Threshold APIC interrupts
MCE:          0          0          0          0   Machine check exceptions
MCP:         40         40         40         40   Machine check polls
ERR:          0
```

```
MIS:          0
```

実行ファイルとライブラリのアドレス割り当てに関する情報は、`maps` ファイル内に含まれています:

```
> cat /proc/self/maps
08048000-0804c000 r-xp 00000000 03:03 17753      /bin/cat
0804c000-0804d000 rw-p 00004000 03:03 17753      /bin/cat
0804d000-0806e000 rw-p 0804d000 00:00 0          [heap]
b7d27000-b7d5a000 r--p 00000000 03:03 11867      /usr/lib/locale/en_GB.utf8/
b7d5a000-b7e32000 r--p 00000000 03:03 11868      /usr/lib/locale/en_GB.utf8/
b7e32000-b7e33000 rw-p b7e32000 00:00 0
b7e33000-b7f45000 r-xp 00000000 03:03 8837       /lib/libc-2.3.6.so
b7f45000-b7f46000 r--p 00112000 03:03 8837       /lib/libc-2.3.6.so
b7f46000-b7f48000 rw-p 00113000 03:03 8837       /lib/libc-2.3.6.so
b7f48000-b7f4c000 rw-p b7f48000 00:00 0
b7f52000-b7f53000 r--p 00000000 03:03 11842      /usr/lib/locale/en_GB.utf8/
[...]
b7f5b000-b7f61000 r--s 00000000 03:03 9109       /usr/lib/gconv/gconv-module
b7f61000-b7f62000 r--p 00000000 03:03 9720       /usr/lib/locale/en_GB.utf8/
b7f62000-b7f76000 r-xp 00000000 03:03 8828       /lib/ld-2.3.6.so
b7f76000-b7f78000 rw-p 00013000 03:03 8828       /lib/ld-2.3.6.so
bfd61000-bfd76000 rw-p bfd61000 00:00 0          [stack]
ffffe000-fffff000 ---p 00000000 00:00 0          [vdso]
```

`/proc` ファイルシステムを利用することで、数多くの情報を取得することができます。主なファイルとその内容を下記に示します:

`/proc/devices`
: 利用可能なデバイスに関する情報

`/proc/modules`
: 読み込み済みのカーネルモジュール

`/proc/cmdline`
: カーネルのコマンドライン

`/proc/meminfo`
: メモリの使用率に関する詳細情報

`/proc/config.gz`
: `gzip` 形式で圧縮された、現在動作中のカーネルの設定ファイル

/proc/`PID`/
: 現在動作中のプロセスに関する情報が含まれています。ここで `PID` には、プロセスのプロセス ID を代入します。自分自身のプロセスを参照したい場合は、`/proc/self/` で参照することができます。

さらに詳しい情報をご希望の場合は、`/usr/src/linux/Documentation/filesystems/proc.txt` ファイル (英語, `kernel-source` パッケージをインストールする必要があります) をお読みください。

2.6.1 `procinfo`

`/proc` ファイルシステム内で取得できる情報から主なものを抜き出すためのコマンドとして、`procinfo` が用意されています:

```
> procinfo
Linux 3.11.10-17-desktop (geeko@buildhost) (gcc 4.8.1 20130909) #1 4CPU
 [jupiter.example.com]

Memory:      Total        Used        Free      Shared     Buffers      Cached
Mem:       8181908     8000632      181276           0       85472     2850872
Swap:     10481660        1576    10480084

Bootup: Mon Jul 28 09:54:13 2014    Load average: 1.61 0.85 0.74 2/904 25949

user   :       1:54:41.84  12.7%  page in :    2107312  disk 1:    52212r   20199w
nice   :       0:00:00.46   0.0%  page out:    1714461  disk 2:    19387r   10928w
system:       0:25:38.00   2.8%  page act:     466673  disk 3:      548r      10w
IOwait:       0:04:16.45   0.4%  page dea:     272297
hw irq:       0:00:00.42   0.0%  page flt:  105754526
sw irq:       0:01:26.48   0.1%  swap in :          0
idle   :      12:14:43.65  81.5%  swap out:        394
guest :       0:02:18.59   0.2%
uptime:       3:45:22.24         context :   99809844

irq  0:       121 timer                 irq 41:   3238224 hpet3
irq  8:         1 rtc0                  irq 42:   3251898 hpet4
irq  9:         0 acpi                  irq 43:   3156368 hpet5
irq 16:     14589 ehci_hcd:usb1         irq 45:         0 aerdrv, PCIe PME
irq 18:         0 i801_smbus            irq 46:         0 PCIe PME, pciehp
irq 19:    124861 ata_piix, ata_piix, f irq 47:         0 PCIe PME, pciehp
irq 22:   3742817 enp5s1                irq 48:         0 PCIe PME, pciehp
irq 23:    479248 ehci_hcd:usb2         irq 49:       387 snd_hda_intel
irq 40:   3216894 hpet2                 irq 50:   1088673 nvidia
```

全ての情報を表示するには `-a` オプションを指定してください。また、`-n N` とオプションを指定すると、`N` 秒間隔で情報を更新し続けることができます。この場合、`Q` を押すと終了することができます。既定では累積値を表示しますが、`-d` を指定すると差分値を表示することもできます。たとえば `procinfo -dn5` とオプションを指定すると、直近の 5 秒間に変化した値が表示されます。

2.6.2 システム制御パラメータ: /proc/sys/

システム制御パラメータを使用することで、Linux カーネルのパラメータを動作中に変更することができます。これらのファイルは `/proc/sys/` ディレクトリ内に存在し、`sysctl` コマンドで表示および変更を行うことができます。全てのパラメータを一覧表示するには、`sysctl -a` と入力して実行してください。また、特定の値のみを表示したい場合は、`sysctl` パラメータ名 と入力して実行します。

パラメータは分野ごとに分類されていて、`sysctl` 分類名 のように入力して実行するか、もしくは対応するディレクトリ内の一覧を表示することで、確認することができます。主な分類の一覧を下記に示します。なお、詳細情報を説明しているファイルは、いずれも英語のみの提供で、`kernel-source` パッケージをインストールしておく必要があります。

`sysctl dev` (`/proc/sys/dev/`)
: デバイス固有の情報が含まれています。

`sysctl fs` (`/proc/sys/fs/`)
: 使用済みのファイルハンドルやクォータ、その他のファイルシステム指向のパラメータが含まれています。詳しくは `/usr/src/linux/Documentation/sysctl/fs.txt` ファイルをお読みください。

`sysctl kernel` (`/proc/sys/kernel/`)
: タスクスケジューラやシステムの共有メモリ、そしてその他のカーネル関連のパラメータが含まれています。詳しくは `/usr/src/linux/Documentation/sysctl/kernel.txt` ファイルをお読みください。

`sysctl net` (`/proc/sys/net/`)
: ネットワークブリッジのほか、一般的なネットワークパラメータが含まれています (主に `ipv4/` サブディレクトリ内にあります) 。詳しくは `/usr/src/linux/Documentation/sysctl/net.txt` ファイルをお読みください。

`sysctl vm` (`/proc/sys/vm/`)
: このパス内に存在するファイルには、仮想メモリやスワップ処理、キャッシュ処理に関するパラメータが含まれています。詳しくは `/usr/src/linux/Documentation/sysctl/vm.txt` ファイルをお読みください。

システムを再起動するまでの間、パラメータを設定したい場合は、`sysctl -w` パラメータ名 = 設定値 のように入力して実行してください。また、パラメータ設定を恒久的に行いたい場合は、`/etc/sysctl.conf` ファイル内に パラメータ名 = 設定値 の形式で行を追加してください。

2.7 ハードウエア情報

2.7.1 PCI リソースの表示: `lspci`



注記: PCI 情報へのアクセスについて

ほとんどの環境では、コンピュータの PCI 設定にアクセスするにあたって root 権限が必要となります。

`lspci` コマンドは、PCI リソースの一覧を表示します:

```
# lspci
00:00.0 Host bridge: Intel Corporation 82845G/GL[Brookdale-G]/GE/PE \
    DRAM Controller/Host-Hub Interface (rev 01)
00:01.0 PCI bridge: Intel Corporation 82845G/GL[Brookdale-G]/GE/PE \
    Host-to-AGP Bridge (rev 01)
00:1d.0 USB Controller: Intel Corporation 82801DB/DBL/DBM \
    (ICH4/ICH4-L/ICH4-M) USB UHCI Controller #1 (rev 01)
00:1d.1 USB Controller: Intel Corporation 82801DB/DBL/DBM \
    (ICH4/ICH4-L/ICH4-M) USB UHCI Controller #2 (rev 01)
00:1d.2 USB Controller: Intel Corporation 82801DB/DBL/DBM \
    (ICH4/ICH4-L/ICH4-M) USB UHCI Controller #3 (rev 01)
00:1d.7 USB Controller: Intel Corporation 82801DB/DBM \
    (ICH4/ICH4-M) USB2 EHCI Controller (rev 01)
00:1e.0 PCI bridge: Intel Corporation 82801 PCI Bridge (rev 81)
00:1f.0 ISA bridge: Intel Corporation 82801DB/DBL (ICH4/ICH4-L) \
    LPC Interface Bridge (rev 01)
00:1f.1 IDE interface: Intel Corporation 82801DB (ICH4) IDE \
    Controller (rev 01)
00:1f.3 SMBus: Intel Corporation 82801DB/DBL/DBM (ICH4/ICH4-L/ICH4-M) \
    SMBus Controller (rev 01)
00:1f.5 Multimedia audio controller: Intel Corporation 82801DB/DBL/DBM \
    (ICH4/ICH4-L/ICH4-M) AC'97 Audio Controller (rev 01)
01:00.0 VGA compatible controller: Matrox Graphics, Inc. G400/G450 (rev 85)
02:08.0 Ethernet controller: Intel Corporation 82801DB PRO/100 VE (LOM) \
    Ethernet Controller (rev 81)
```

`-v` オプションを指定すると、さらに詳しい情報を表示することができます:

```
# lspci -v
[...]
00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet \
Controller (rev 02)
  Subsystem: Intel Corporation PRO/1000 MT Desktop Adapter
  Flags: bus master, 66MHz, medium devsel, latency 64, IRQ 19
```

```
  Memory at f0000000 (32-bit, non-prefetchable) [size=128K]
  I/O ports at d010 [size=8]
  Capabilities: [dc] Power Management version 2
  Capabilities: [e4] PCI-X non-bridge device
  Kernel driver in use: e1000
  Kernel modules: e1000
```

デバイス名の情報は `/usr/share/pci.ids` ファイルから取得しています。そのため、このファイル内に書かれていないデバイスについては、「Unknown device」(未知のデバイス) として表示されます。`-vv` パラメータを指定すると、プログラムから問い合わせることのできる全ての情報を出力することができます。また、文字ではなく数値で表示したい場合は、`-n` オプションをお使いください。

2.7.2 USB デバイスの一覧表示: `lsusb`

`lsusb` コマンドは USB デバイスの一覧を表示します。`-v` オプションを指定すると、さらに詳しい情報を表示することができます。詳しい情報は `/proc/bus/usb/` 内から取得します。下記は USB バスにハブと USB メモリ、ハードディスクとマウスがそれぞれ接続された環境での、`lsusb` の出力例です。

```
# lsusb
Bus 004 Device 007: ID 0ea0:2168 Ours Technology, Inc. Transcend JetFlash \
    2.0 / Astone USB Drive
Bus 004 Device 006: ID 04b4:6830 Cypress Semiconductor Corp. USB-2.0 IDE \
    Adapter
Bus 004 Device 005: ID 05e3:0605 Genesys Logic, Inc.
Bus 004 Device 001: ID 0000:0000
Bus 003 Device 001: ID 0000:0000
Bus 002 Device 001: ID 0000:0000
Bus 001 Device 005: ID 046d:c012 Logitech, Inc. Optical Mouse
Bus 001 Device 001: ID 0000:0000
```

2.7.3 サーマルサブシステムの監視とチューニング: `tmon`

`tmon` は複雑なサーマル (温度管理) サブシステムを可視化したり、チューニングしたりテストしたりするためのツールです。何もパラメータを指定しないで起動すると、`tmon` は監視モードで動作します:

```
┌──────THERMAL ZONES(SENSORS)──────────────────────────────┐
│Thermal Zones:                acpitz00                    │
│Trip Points:                  PC                          │
└──────────────────────────────────────────────────────────┘

┌─────────── COOLING DEVICES ──────────────────────────────┐
│ID  Cooling Dev    Cur     Max    Thermal Zone Binding     │
```

```
|00    Processor     0      3   ||||||||||||                |
|01    Processor     0      3   ||||||||||||                |
|02    Processor     0      3   ||||||||||||                |
|03    Processor     0      3   ||||||||||||                |
|04 intel_powerc    -1     50   ||||||||||||                |
└──────────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────────┐
|                          10        20        30        40 |
|acpitz 0:[  8][>>>>>>>>>P9                     C31         |
└──────────────────────────────────────────────────────────┘

┌────────────────── CONTROLS ──────────────────────────────┐
|PID gain: kp=0.36 ki=5.00 kd=0.19 Output 0.00             |
|Target Temp: 65.0C, Zone: 0, Control Device: None         |
└──────────────────────────────────────────────────────────┘


Ctrl-c - Quit   TAB - Tuning
```

データの解釈方法に関して、および温度データの記録方法に関して、そして `tmon` で冷却デバイスやセンサーをテストしたりチューニングしたりする方法について、詳しくは `man 8 tmon` にあるマニュアルページをお読みください。なお、`tmon` パッケージは既定ではインストールされません。

2.7.4 MCELog: マシンチェック例外 (MCE; Machine Check Exceptions)



注記: 対応アーキテクチャについて

このツールは AMD64/Intel 64 システム向けにのみ提供されています。

`mcelog` パッケージは、I/O や CPU 、メモリエラーなどのハードウエアエラーが発生した際に発せられるマシンチェック例外 (Machine Check Exceptions (MCE)) を記録し、解釈／翻訳するためのパッケージです。これに加えて mcelog は、不正なページのオフライン化や、キャッシュエラーが発生した際に自動的にコアをオフライン化するなどの処理も行います。以前のバージョンでは cron ジョブとして 1 時間に 1 回動作していましたが、現在は mcelog デーモンの形で、ハードウエアエラーを即時に処理するようになっています。



注記: AMD Scalable MCA のサポートについて

openSUSE Leap では AMD 社の Scalable Machine Check Architecture (Scalable MCA) に対応するようになっています。Scalable MCA は AMD 社の Zen プロセッサでハードウエアエラー報告の機能を改善するための仕組みで、エラー報告やその後の解析を支援するために MCA バンク内に保存される情報を記録しています。

`mcelog` は MCA メッセージを捕捉 (`rasdaemon` や `dmesg` でも MCA メッセージを捕捉することができます) 。詳しくは Processor ProgrammingReference (PPR) for AMD Family 17h Model 01h, Revision B1 Processors (https://developer.amd.com/wordpress/media/2017/11/54945_PPR_Family_17h_Models_00h-0Fh.pdf) 内にある第 3.1 章:「Machine Check Architecture」(英語) をお読みください。

mcelog は `/etc/mcelog/mcelog.conf` ファイルで設定を行います。設定オプションは `man mcelog` 内や https://mcelog.org/ 内で説明されています。なお下記の例は、既定のファイルに対する変更箇所のみを示しています:

```
daemon = yes
filter = yes
filter-memory-errors = yes
no-syslog = yes
logfile = /var/log/mcelog
run-credentials-user = root
run-credentials-group = nobody
client-group = root
socket-path = /var/run/mcelog-client
```

mcelog サービスは既定では有効化されていません。YaST システムサービスエディタを使用するか、下記のコマンドラインを実行することで、有効化することができます:

```
# systemctl enable mcelog
# systemctl start mcelog
```

2.7.5 AMD64/Intel 64: dmidecode: DMI テーブルデコーダ

`dmidecode` はハードウエアのシリアル番号や BIOS のリビジョンなどを含む、マシンの DMI テーブルを表示することのできるツールです。

```
# dmidecode
# dmidecode 2.12
SMBIOS 2.5 present.
27 structures occupying 1298 bytes.
Table at 0x000EB250.

Handle 0x0000, DMI type 4, 35 bytes
Processor Information
        Socket Designation: J1PR
        Type: Central Processor
        Family: Other
        Manufacturer: Intel(R) Corporation
```

```
        ID: E5 06 01 00 FF FB EB BF
        Version: Intel(R) Core(TM) i5 CPU         750  @ 2.67GHz
        Voltage: 1.1 V
        External Clock: 133 MHz
        Max Speed: 4000 MHz
        Current Speed: 2667 MHz
        Status: Populated, Enabled
        Upgrade: Other
        L1 Cache Handle: 0x0004
        L2 Cache Handle: 0x0003
        L3 Cache Handle: 0x0001
        Serial Number: Not Specified
        Asset Tag: Not Specified
        Part Number: Not Specified
[..]
```

2.8 ファイルとファイルシステム

2.8.1 ファイルの種類判別: `file`

`file` は指定したファイル (複数可) の種類を判別します。その際、`/usr/share/misc/magic` ファイル内に書かれた情報を使用します。

```
> file /usr/bin/file
/usr/bin/file: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), \
    for GNU/Linux 2.6.4, dynamically linked (uses shared libs), stripped
```

`-f 一覧ファイル` オプションを指定すると、`一覧ファイル` に指定したファイル内を読み込み、読み込んだ内容をファイル名として解釈し、ファイルの種類を判別します。また `-z` オプションを指定すると、`file` が圧縮ファイル内を調査するようにすることができます:

```
> file /usr/share/man/man1/file.1.gz
/usr/share/man/man1/file.1.gz: gzip compressed data, from Unix, max compression
> file -z /usr/share/man/man1/file.1.gz
/usr/share/man/man1/file.1.gz: troff or preprocessor input text \
    (gzip compressed data, from Unix, max compression)
```

`-i` オプションを指定すると、ファイルの種類に対する詳しい説明ではなく、MIME タイプを表示するようになります。

```
> file -i /usr/share/misc/magic
/usr/share/misc/magic: text/plain charset=utf-8
```

2.8.2 ファイルシステムと使用率の表示: `mount` , `df` , `du`

`mount` コマンドは、どのファイルシステム (デバイスおよび種類) がどこにマウントされているのかを表示します:

```
# mount
/dev/sda2 on / type ext4 (rw,acl,user_xattr)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
debugfs on /sys/kernel/debug type debugfs (rw)
devtmpfs on /dev type devtmpfs (rw,mode=0755)
tmpfs on /dev/shm type tmpfs (rw,mode=1777)
devpts on /dev/pts type devpts (rw,mode=0620,gid=5)
/dev/sda3 on /home type ext3 (rw)
securityfs on /sys/kernel/security type securityfs (rw)
fusectl on /sys/fs/fuse/connections type fusectl (rw)
gvfs-fuse-daemon on /home/tux/.gvfs type fuse.gvfs-fuse-daemon \
(rw,nosuid,nodev,user=tux)
```

それぞれのファイルシステム内での使用率を調べたい場合は、`df` コマンドを使用します。このコマンドに対して `-h` (もしくは `--human-readable`) オプションを指定すると、人間にとってわかりやすい、単位付きの表示に切り替えることができます。

```
> df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda2              20G  5,9G   13G  32% /
devtmpfs              1,6G  236K  1,6G   1% /dev
tmpfs                 1,6G  668K  1,6G   1% /dev/shm
/dev/sda3             208G   40G  159G  20% /home
```

指定したディレクトリ内、およびそのサブディレクトリ内にある全ファイルのサイズ合計を知りたい場合は、`du` コマンドを使用します。`-s` オプションを指定すると、詳細な出力を省略して、パラメータで指定したディレクトリの合計値のみを表示します。また、このコマンドでも `-h` を指定することで、人間にとってわかりやすい、単位付きの表示に切り替えることができます。

```
> du -sh /opt
192M    /opt
```

2.8.3 ELF 形式バイナリに対する追加情報

バイナリファイルの内容を表示したい場合は、`readelf` ユーティリティを使用します。このプログラムは、ELF 形式であれば、どのアーキテクチャ向けのファイルであっても読み込むことができます:

```
> readelf --file-header /bin/ls
```

```
ELF Header:
  マジック:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  クラス:                            ELF64
  データ:                            2's complement, little endian
  バージョン:                        1 (current)
  OS/ABI:                            UNIX - System V
  ABI バージョン:                    0
  型:                                EXEC (Executable file)
  マシン:                            Advanced Micro Devices X86-64
  バージョン:                        0x1
  エントリポイントアドレス:          0x402540
  プログラムヘッダ始点:              64 (bytes into file)
  セクションヘッダ始点:              95720 (bytes into file)
  フラグ:                            0x0
  このヘッダのサイズ:                64 (bytes)
  プログラムヘッダサイズ:            56 (bytes)
  プログラムヘッダ数:                9
  セクションヘッダサイズ:            64 (bytes)
  セクションヘッダ数:                32
  セクションヘッダ文字列表索引:      31
```

2.8.4 ファイル属性の表示: `stat`

`stat` コマンドはファイルの属性を表示します:

```
> stat /etc/profile
  File: `/etc/profile'
  Size: 9662            Blocks: 24         IO Block: 4096   regular file
Device: 802h/2050d      Inode: 132349      Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2009-03-20 07:51:17.000000000 +0100
Modify: 2009-01-08 19:21:14.000000000 +0100
Change: 2009-03-18 12:55:31.000000000 +0100
```

`--file-system` パラメータを指定すると、指定したファイルが存在するファイルシステム内での情報を表示することができます:

```
> stat /etc/profile --file-system
  File: "/etc/profile"
    ID: d4fb76e70b4d1746 Namelen: 255     Type: ext2/ext3
Block size: 4096       Fundamental block size: 4096
Blocks: Total: 2581445    Free: 1717327    Available: 1586197
Inodes: Total: 655776     Free: 490312
```

2.9 ユーザ情報

2.9.1 ユーザがアクセスしているファイルの表示: `fuser`

どのプロセスやユーザが、どのファイルにアクセスしているのかを知りたいような場合があります。たとえば `/mnt` をマウント解除したいような場合がそれにあたります。`umount` コマンドが "device is busy" と出力してマウント解除に失敗した場合、`fuser` コマンドを使用することで、どのプロセスがデバイスにアクセスしているのかがわかります:

```
> fuser -v /mnt/*

                     USER        PID ACCESS COMMAND
/mnt/notes.txt       tux     26597 f....  less
```

あとは `less` プロセスが終了するのを待つか、もしくは強制終了させることで、ファイルシステムのマウントを解除できるようになります。なお、`fuser` コマンドに `-k` オプションを指定すると、ファイルにアクセスしているプロセスを終了させることができます。

2.9.2 ユーザの活動状況表示: `w`

`w` コマンドを使用すると、現在システムに誰がログインしているのかと、そのユーザが今何をしているのかを知ることができます。たとえば下記のようになります:

```
> w
 16:00:59 up 1 day,  2:41,  3 users,  load average: 0.00, 0.01, 0.05
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
tux      :0       console          Wed13   ?xdm?   8:15   0.03s /usr/lib/gdm/gd
tux      console  :0               Wed13   26:41m  0.00s  0.03s /usr/lib/gdm/gd
tux      pts/0    :0               Wed13   20:11   0.10s  2.89s /usr/lib/gnome-
```

`-f` オプションを指定すると、リモートからアクセスしている場合のアクセス元の表示を切り替えることもできます。

2.10 日付と時刻

2.10.1 `time` による時間測定

コマンドで使用した時間を計測したい場合は、`time` ユーティリティをお使いください。このユーティリティは、bash の内蔵コマンドとしても用意されているほか、純粋なプログラム (`/usr/bin/time`) としても提供されています。

```
> time find . > /dev/null

real    0m4.051s❶
user    0m0.042s❷
sys     0m0.205s❸
```

❶ コマンドが起動して終了するまでに経過した実時間を表しています。

❷ `times` システムコールで報告された、ユーザ部分の CPU 時間を表しています。

❸ `times` システムコールで報告された、システム部分の CPU 時間を表しています。

`/usr/bin/time` コマンドの場合は、出力をより詳しく行うことができます。読みやすい表示をご希望の場合は、`-v` オプションを指定してください。

```
/usr/bin/time -v find . > /dev/null
        Command being timed: "find ."
        User time (seconds): 0.24
        System time (seconds): 2.08
        Percent of CPU this job got: 25%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:09.03
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 2516
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 1564
        Voluntary context switches: 36660
        Involuntary context switches: 496
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

2.11 データのグラフ化: RRDtool

コンピュータシステムを扱っていると、様々なデータを採取できることがわかります。たとえばコンピュータの温度の変化やネットワークインターフェイスが送信もしくは受信したデータ量などがあります。RRDtool では、そのようなデータを詳細に保存し、カスタマイズ可能なグラフとして可視化する機能を提供しています。

RRDtool はほとんどの Unix プラットフォームや Linux ディストリビューション向けに提供されています。openSUSE® Leap でも RRDtool をご利用いただけます。YaST からインストールを行うか、もしくは `root` で下記のように入力して実行してください:

```
zypper install rrdtool
```



ヒント: バインディングについて

RRDtool には Perl, Python, Ruby, PHP などのバインディングが用意されています。そのため、お使いのスクリプト言語を利用して、独自の監視スクリプトを作成することができます。

2.11.1 RRDtool の仕組み

RRDtool は Round Robin Database tool の略で、「ラウンドロビン型のデータベースツール」という意味です。ラウンドロビン とは「持ち回り」のような意味で、一定量のデータを保持し続け、古いものから順に破棄していく仕組みを意味します。そのため、データには始まりも終わりもありません。RRDtool では、このようなラウンドロビン型のデータベースを利用してデータを保存し、読み込みを行います。

上述のとおり、RRDtool は時間が経過するごとに変化していくようなデータを扱うよう設計されています。よくある使用例としては、測定データを繰り返し読み取ることのできるセンサー (温度や速度など) からデータを採取して、特定の形式でグラフ化するなどの例が考えられます。このようなデータであれば、RRDtool は完璧に動作しますし、データの処理や必要な出力の作成も容易に行うことができます。

データを自動的に採取できないようなデータや、定期的に採取できないようなデータに対しては、RRDtool にデータを与える際、特定の書式を使用する必要があります。また、このようなデータの場合、RRDtool を手作業で動作させる必要があるかもしれません。

下記は RRDtool の基本的な使い方を説明するための例です。ここでは RRDtool における 3 つのフェーズ (データベースの作成, 測定データの更新, 測定データの可視化) を示しています。

2.11.2 実際の例

たとえば Linux システム内において随時変化する、メモリの使用率について情報を収集し、可視化したいと考えたとします。わかりやすいように、今回は 4 秒間隔で 40 秒間にわたって、空きメモリ量を計測するものとします。また、メモリを大きく使用するアプリケーションを 3 種類 (Firefox Web ブラウザ, Evolution 電子メールクライアント, Eclipse 開発フレームワーク) 動作させて終了し、メモリの使用率を変化させてみます。

2.11.2.1 データの収集

RRDtool はネットワークトラフィックを測定し、可視化するためによく使用されます。このような場合、Simple Network Management Protocol (SNMP) を使用して行います。このプロトコルはネットワークデバイスに対して、内部カウンタの情報からさまざまな値を生成させて取得することができます。SNMP に関する詳細は、http://www.net-snmp.org/ をお読みください。

今回の例は上記と少し異なります。データを手作業で取得する必要があります。下記のようなヘルパースクリプト `free_mem.sh` を使用して、現在の空きメモリ量を調べて、標準出力に書き込むことを行います。

```
> cat free_mem.sh
INTERVAL=4
for steps in {1..10}
do
    DATE=`date +%s`
    FREEMEM=`free -b | grep "Mem" | awk '{ print $4 }'`
    sleep $INTERVAL
    echo "rrdtool update free_mem.rrd $DATE:$FREEMEM"
done
```

- 時間間隔は 4 秒で、間隔を空ける際に `sleep` コマンドを使用します。
- RRDtool は日時形式を Unix 時間 と呼ばれる特殊な形式で受け付けます。これは 1970 年 1 月 1 日深夜からの経過秒数で日時を表すもので、たとえば 1272907114 は 2010 年 5 月 3 日 17:18:34 を意味することになります。
- 空きメモリの情報は `free -b` コマンドで取得します。つまり、「キロ」などの接頭辞のない純粋なバイト単位の値を取得しています。
- `echo` で始まる行には、次の手順で使用するデータベースのファイル名 (`free_mem.rrd`) と、RRDtool で値を更新する際のコマンドラインが書かれています。

`free_mem.sh` を実行して、下記のような結果が得られたものとします:

```
> sh free_mem.sh
rrdtool update free_mem.rrd 1272974835:1182994432
```

```
rrdtool update free_mem.rrd 1272974839:1162817536
rrdtool update free_mem.rrd 1272974843:1096269824
rrdtool update free_mem.rrd 1272974847:1034219520
rrdtool update free_mem.rrd 1272974851:909438976
rrdtool update free_mem.rrd 1272974855:832454656
rrdtool update free_mem.rrd 1272974859:829120512
rrdtool update free_mem.rrd 1272974863:1180377088
rrdtool update free_mem.rrd 1272974867:1179369472
rrdtool update free_mem.rrd 1272974871:1181806592
```

なお、このままでは端末に結果を書き出しているだけで、次の手順で使用することができませんので、下記のように実行してファイルに保存しておきます:

`sh free_mem.sh > free_mem_updates.log`

"¥n ¥n"

2.11.2.2 データベースの作成

まずは下記のようにコマンドを入力して実行し、ラウンドロビン型のデータベースを作成します:

```
> rrdtool create free_mem.rrd --start 1272974834 --step=4 \
DS:memory:GAUGE:600:U:U RRA:AVERAGE:0.5:1:24
```

注目すべき点

- このコマンドは `free_mem.rrd` というファイルを作成します。このファイル内に測定したデータを書き込むことになります。
- `--start` オプションは、開始日時を Unix 時間で表したものを指定しています。この値は、データベースに追加される最初の値となります。この例では、`free_mem.sh` の出力に現れた Unix 時間 (1272974835) よりも前の日時を指定しています。
- `--step` オプションは測定したデータの時間間隔を指定するためのものです。
- `DS:memory:GAUGE:600:U:U` の部分は、データベースに対して新しいデータソース (DS) を設定している箇所です。memory という名称で gauge という種類を設定し、データとデータの間の間隔は最大で 600 秒、値の 最小値 と 最大値 は未知 (U; Unknown) としています。
- `RRA:AVERAGE:0.5:1:24` はラウンドロビンアーカイブ (RRA) と呼ばれるもので、データ点の 平均値 を計算する 統合関数 を利用して処理を行い、データを格納することを表しています。このとき、行末には 3 種類の統合関数向けのパラメータが指定されています。

何もエラーが発生することなく終了すれば、`free_mem.rrd` がカレント (現在居る) ディレクトリ内に作成されているはずです:

```
> ls -l free_mem.rrd
```

```
-rw-r--r-- 1 tux users 776 May  5 12:50 free_mem.rrd
```

2.11.2.3 データベースの値の更新

データベースを作成したら、あとは測定したデータをデータベースに投入していきます。2.11.2.1項「データの収集」で行ったとおり、既に `free_mem_updates.log` というファイルを作成してあり、この中には `rrdtool update` で始まる更新コマンドの羅列が記載されていますので、あとはそのままそれらを実行するだけです。

```
> sh free_mem_updates.log; ls -l free_mem.rrd
-rw-r--r--  1 tux users  776 May  5 13:29 free_mem.rrd
```

上記のとおり、データを更新しても `free_mem.rrd` のサイズは変わりません。

2.11.2.4 記録された値の表示

データを測定し、データベースを作成し、測定されたデータをデータベースに投入したら、あとはデータベースを利用するだけです。ここでは値を読んでみることにします。

データベース内に保存された全てのデータを取得するには、下記のように入力して実行します:

```
> rrdtool fetch free_mem.rrd AVERAGE --start 1272974830 \
--end 1272974871
           memory
1272974832: nan
1272974836: 1.1729059840e+09
1272974840: 1.1461806080e+09
1272974844: 1.0807572480e+09
1272974848: 1.0030243840e+09
1272974852: 8.9019289600e+08
1272974856: 8.3162112000e+08
1272974860: 9.1693465600e+08
1272974864: 1.1801251840e+09
1272974868: 1.1799787520e+09
1272974872: nan
```

注目すべき点

- `AVERAGE` はデータベース内のデータ点の平均値を算出するためのもので、データソースを指定する際に設定したものです (2.11.2.2項「データベースの作成」を参照) 。それ以外の関数は指定していません。
- 出力の最初の行には、2.11.2.2項「データベースの作成」で行ったデータソースの設定時に指定した名前が現れています。

- 左側の列は時刻を、右側の列には指数表記での測定値の平均が示されています。
- `nan` は「not a number」(数値ではない) の意味で、最初と最後では平均値を算出できないため、このような表示になっています。

あとはデータベース内に保存されたデータをグラフ化するだけです:

```
> rrdtool graph free_mem.png \
--start 1272974830 \
--end 1272974871 \
--step=4 \
DEF:free_memory=free_mem.rrd:memory:AVERAGE \
LINE2:free_memory#FF0000 \
--vertical-label "GB" \
--title "Free System Memory in Time" \
--zoom 1.5 \
--x-grid SECOND:1:SECOND:4:SECOND:10:0:%X
```

注目すべき点

- `free_mem.png` は、作成するグラフのファイル名です。
- `--start` と `--end` は、グラフを作成する際の日時範囲の指定です。
- `--step` はグラフの時間間隔を秒単位で指定しているものです。
- `DEF:...` の部分は free_memory という名前のデータを定義している箇所です。データは `free_mem.rrd` データベースから読み込み、データソース名は memory 、データポイントは average という名前であることを定義しています (2.11.2.2項「データベースの作成」ではそれ以外のデータポイントは定義していないためです) 。
- `LINE...` の部分はグラフに描画する際の線を定義しているものです。2 ピクセルの太さで free_memory という名前のデータを、赤色で描画する旨を表しています。
- `--vertical-label` は y 軸に表示するラベルを、`--title` にはグラフ全体のタイトルをそれぞれ指定しています。
- `--zoom` ではグラフの拡大率を指定しています。この値は 0 より大きい値でなければなりません。
- `--x-grid` では、グラフ内のグリッド線の描画方法とラベルを指定しています。この例では 1 秒間隔でグリッド線を描画し、4 秒間隔で赤点線を、10 秒間隔でラベルをそれぞれ描画しています。

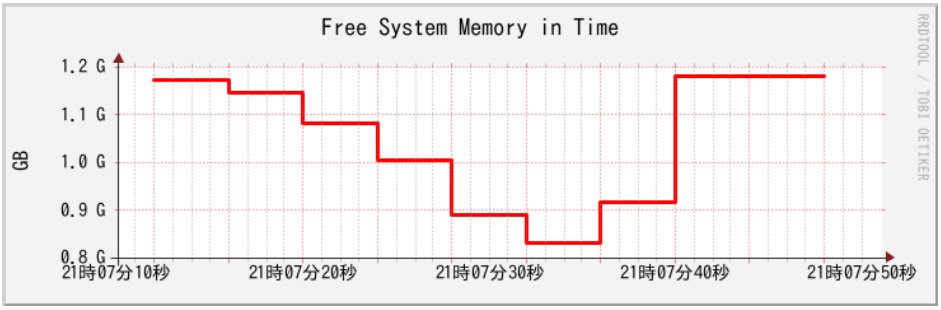

図 2.1: RRDTOOL で作成したグラフの例

2.11.3 さらなる情報

RRDtool は複雑なツールであり、多数のサブコマンドやコマンドラインオプションが用意されています。それらの中には分かりやすいものもある一方、期待するような出力を行おうとすると、いろいろ試行錯誤を行って調整する必要に迫られることもあります。

基本的な情報のみを提供している RRDtool のマニュアルページ (`man 1 rrdtool`) とは別に、RRDtool home page (https://oss.oetiker.ch/rrdtool/) (英語のみ) をご覧になることをお勧めします。ここには `rrdtool` コマンドやそのサブコマンドに対する詳細な ドキュメンテーション (https://oss.oetiker.ch/rrdtool/doc/index.en.html) (日本語訳が RRDtool マニュアル訳 (http://agj.at/~fors/sysworks/data/rrdtool/) などにあります) が用意されています。また チュートリアル (https://oss.oetiker.ch/rrdtool/tut/index.en.html) (英語) には、一般的な RRDtool の作業手順が示されています。

ネットワークトラフィックの監視を行いたい場合は、MRTG (Multi Router Traffic Grapher) (https://oss.oetiker.ch/mrtg/) (英語) プロジェクトをご覧になるとよいでしょう。MRTG は、様々なネットワークデバイスから収集した情報をグラフ化することができます。この中でも RRDtool を使用しています。

3 システムログファイル

改訂履歴

2023-08-11

システムログファイルの分析作業は、システムを分析する際に最も重要な作業です。事実、システムのメンテナンスやトラブルシューティングを行う場合、一般的には何よりも先にシステムログファイルを確認します。openSUSE Leap では、システム内で発生した事象をほぼ全て詳細に記録します。また `systemd` への移行により、カーネルメッセージとシステムサービスが発するメッセージは、`systemd` 内に保管されるようになっています (詳しくは『リファレンス』、第11章「`journalctl` : systemd ジャーナルへの問い合わせコマンド」をお読みください) 。その他のログファイル (主にシステムアプリケーションが記録するもの) については、従来通り純粋なテキスト形式で記録され、エディタやページャなどを利用すれば簡単に読むことができます。このほか、スクリプトを利用してそれらを処理することもできますので、これにより特定の内容だけを抜き出すこともできます。

3.1 `/var/log/` 内にあるシステムログファイル

システム関係のログファイルは、必ず `/var/log` ディレクトリ内に存在しています。下記の一覧には、openSUSE Leap 既定のインストール後の環境に現れる、全てのシステムログファイルを示しています。もちろん追加のソフトウエアをインストールすることで、`/var/log` には下記に示していない様々なログファイルが作成されます。また、ファイルやディレクトリによっては「プレースホルダ」として用意されるだけで、実際に対応するアプリケーションをインストールしないと、使用されないものもあります。さらに、ほとんどのログファイルは `root` でないと読むことができません。

`apparmor/`

AppArmor のログファイルが配置されます。AppArmor に関する詳細は、『セキュリティ強化ガイド』をお読みください。

`audit/`

監査フレームワークのログファイルが配置されます。詳しくは『セキュリティ強化ガイド』をお読みください。

`ConsoleKit/`

`ConsoleKit` デーモン (どのユーザがログインしてどのような作業を行ったのかを追跡するデーモン) のログファイルが配置されます。

`cups/`

汎用 Unix 印刷システム (Common Unix Printing System (`cups`)) のアクセスログファイルとエラーログファイルが配置されます。

`firewall`
: ファイアウオール機能のログファイルが配置されます。

`gdm/`
: GNOME ディスプレイマネージャのログファイルが配置されます。

`krb5/`
: Kerberos ネットワーク認証システムのログファイルが配置されます。

`lastlog`
: 各ユーザの最終ログイン日時に関する情報を保持するデータベースです。`lastlog` コマンドを使用すると、内容を読むことができます。詳しくは `man 8 lastlog` をお読みください。

`localmessages`
: いくつかの起動スクリプトのログメッセージが含まれています。たとえば DHCP クライアントのログなどがあります。

`mail*`
: メールサーバ (`postfix` , `sendmail` など) のログファイルが配置されます。

`messages`
: 全てのカーネルメッセージやシステムのログメッセージが記録されるファイルで、何らかの問題が発生した場合は、`/var/log/warn` ファイルとともに、最初に確認しておくべきファイルでもあります。

`NetworkManager`
: NetworkManager のログファイルです。

`news/`
: ニュースサーバからのログファイルが配置されるディレクトリです。

`chrony/`
: Network Time Protocol デーモン (`chrony`) のログファイルが配置されるディレクトリです。

`pk_backend_zypp*`
: PackageKit (with `libzypp` バックエンド) のログファイルです。

`samba/`
: Windows SMB/CIFS ファイルサーバである Samba のログファイルが配置されるディレクトリです。

`warn`

全てのシステム内の警告およびエラーが記録されるファイルです。何らかの問題が発生した場合は、`systemd` ジャーナルの出力とともに、最初に確認しておくべきファイルでもあります。

`wtmp`

全てのログイン／ログアウト処理 とリモート接続のデータベースです。`last` コマンドを使用すると、内容を読むことができます。詳しくは `man 1 last` をお読みください。

`Xorg.数字.log`

X.Org の起動時のログファイルです。X.Org の起動に何らかの問題が発生した場合は、このファイルを参照してください。
なお、ファイル名の `数字` の箇所には、ディスプレイ番号が入ります。たとえば `Xorg.0.log` であれば既定のディスプレイ番号である `0` のログファイルになりますし、`Xorg.1.log` であればディスプレイ番号 `1` のログファイルになります。また、古い X.Org のログファイルは `Xorg.数字.log.old` の形式で保存されます。



注記

なお、X.Org を `root` で起動した場合にのみ、`/var/log/` ディレクトリ内にログが記録されることに注意してください。その他の (一般) ユーザで X.Org を起動した場合は、`~/.local/share/xorg/` ディレクトリ内にログが保存されます。

`YaST2/`

全ての YaST ログファイルが含まれるディレクトリです。

`zypp/`

`libzypp` のログファイルが含まれるディレクトリです。パッケージのインストール履歴を読みたい場合は、これらのファイルを参照してください。

`zypper.log`

コマンドラインインストーラである `zypper` のログファイルです。

3.2 ログファイルの表示と処理

ログファイルを表示したい場合は、任意のテキストエディタをお使いいただくことができます。それ以外にも、YaST コントロールセンターから［その他］›［システムログ］と選択していくことで、ログを閲覧するためのシンプルな YaST モジュールを起動することもできます。

テキストコンソールでログファイルを閲覧したい場合は、`less` や `more` のコマンドを使用します。それ以外にも、`head` や `tail` コマンドを使用することで、ログの冒頭や末尾だけを表示することもできます。ログファイルに追記される内容をリアルタイムに読みたい場合は、`tail -f` コマンドをお使いください。これらのツールの使用方法について、詳しくはそれぞれのマニュアルページをお読みください。

特定の文字列や正規表現を指定してログ内を検索したい場合は、`grep` ツールをお使いください。また、`awk` を使用することで、ログファイルの処理や書き換えなどを行うこともできます。

3.3 `logrotate` によるログの管理

`/var/log` 以下にあるログファイルは日々増えていくもので、容易に巨大化してしまいます。`logrotate` はそのようなログファイルを管理することができるツールです。ログファイルを自動的にローテーション (新しいログファイルへの切り替え) したり、削除や圧縮、メール送信などを行ったりすることができます。ログファイルは定期的 (毎日, 毎週, 毎月など) に処理することができるほか、特定のサイズに達した際に実行することもできます。

`logrotate` は `systemd` によって毎日起動されるようになっているため、ログファイルに対する処理は毎日 1 回実行されます。しかしながら、一定のサイズを超過した場合に発動するように設定している場合や、`--force` オプションを指定したりしたような場合は、それ以上に実行することもできます。それぞれのファイルをどのタイミングで直近にローテーションしたいのかを確認するには、`/var/lib/misc/logrotate.status` ファイルをご覧ください。

`logrotate` におけるメインとなる設定ファイルは、`/etc/logrotate.conf` です。システムパッケージやシステムプログラムによっては、独自のログファイルを生成するものがあります (たとえば `apache2`) が、その場合はそれらのログファイルに対応する設定を `/etc/logrotate.d/` ディレクトリ内に配置します。`/etc/logrotate.d/` 内の設定は、`/etc/logrotate.conf` から取り込むように設定します。

例 3.1: `/etc/logrotate.conf` の例

```
# 詳しくは "man logrotate" で表示されるマニュアルページをお読みください
# 毎週ログファイルをローテーションする指定
weekly

# 4 週分までを保管する設定
rotate 4

# 古いログファイルをローテーションした場合、新しい（空の）ログファイルを作成する指定

# ローテーションしたファイルに対して、ファイル名の末尾に日付を付与する指定
dateext

# ログファイルを圧縮したい場合は、下記の行のコメント文字を削除してください
```

```
# gzip など、他の圧縮形式を使用したい場合は、下記 2 行をコメントアウトしてください
compresscmd /usr/bin/bzip2
uncompresscmd /usr/bin/bunzip2

# RPM パッケージによっては、下記のディレクトリ内に設定ファイルを配置するものがあります
include /etc/logrotate.d
```

重要: パーミッションの矛盾回避について

`create` オプションを使用する場合は、`/etc/permissions*` で指定することのできるファイルのモードと所有者に注意する必要があります。これらの設定を変更している場合は、矛盾が起こらないように設定してください。

3.4 `logwatch` によるログの監視

`logwatch` はカスタマイズ可能でプラグインにも対応した、ログ監視スクリプトです。システムログを分析して重要な情報を抽出し、必要であれば人間にとってよりやすい形式で報告します。`logwatch` コマンドを使用するには、`logwatch` パッケージをインストールしてください。

`logwatch` はコマンドラインで直接実行することで、その場でレポートを生成することができるほか、`cron` 等を利用することで、定期的に独自のレポートを生成することもできます。レポートは画面に表示することができるほか、ファイルに保存したり特定のアドレスにメールを送信したりすることもできます。特にメール送信については、`cron` などでレポートを自動生成しているような場合に便利な仕組みです。

コマンドラインを使用することで、`logwatch` がレポートを生成すべきサービスの種類や時間範囲のほか、情報の細かさを指定することもできます:

```
# 昨日からの全てのカーネルメッセージを含む詳細レポートの生成
logwatch --service kernel --detail High --range Yesterday --print

# アーカイブされたものを含め、全ての sshd のイベントを含む詳細レポートを生成
logwatch --service sshd --detail Low --range All --archives --print

# 2005 年 5 月 5 日から 5 月 7 日までの全ての smartd のメッセージのレポートを、
# root@localhost 宛にメールで送信
logwatch --service smartd --range 'between 5/5/2005 and 5/7/2005' \
--mailto root@localhost --print
```

`--range` オプションは複雑な書式になっています。詳しくは `logwatch --range help` で表示されるヘルプをお読みください。また、問い合わせに使用できる全てのサービスを一覧で表示するには、下記のように入力して実行してください:

```
> ls /usr/share/logwatch/default.conf/services/ | sed 's/\.conf//g'
```

`logwatch` は非常に詳しくカスタマイズすることができます。しかしながら、通常は既定値の設定のままで十分です。既定の設定ファイルは `/usr/share/logwatch/default.conf/` 以下に書かれています。ただし、このディレクトリ内に存在するファイルは、パッケージの更新の際に上書きされてしまう設定になっていますので、変更は行わないでください。カスタマイズを行いたい場合は、`/etc/logwatch/conf/` 内に設定を書き込んでください (既定で配置される設定ファイルを雛形としてお使いください) 。`logwatch` のカスタマイズ方法について、詳しくは `/usr/share/doc/packages/logwatch/HOWTO-Customize-LogWatch` (英語のみ) をお読みください。具体的には、下記の設定ファイルが存在しています:

`logwatch.conf`
: メインの設定ファイルです。既定で配置されるファイルには、様々なコメント (英語のみ) が記載されています。また、それぞれの設定オプションは、コマンドラインを指定することで上書きすることができます。

`ignore.conf`
: `logwatch` で全体的に無視すべき行のフィルタを記述します。

`services/*.conf`
: レポートを作成することのできる各サービスに対して、個別の設定ファイルを保存しておくディレクトリです。

`logfiles/*.conf`
: 各サービスでどのログファイルを処理すべきかを示しているファイルです。

3.5 root 宛のメールに対する転送設定

システムデーモンや `cron` ジョブ、`systemd` タイマーなどのアプリケーションは、システム内の `root` に対してメールを送信することがあります。また、既定ではそれぞれのユーザに対してローカルのメールボックスが作成され、ログイン時にメールの到着を知らせます。

これらのアプリケーションが送信するメッセージには、システム管理者が即時に対応しなければならないセキュリティレポートやインシデント報告などが含まれる場合があります。このような緊急性の高いメールに素早く気付くことができるよう、これらのメールを各担当者のメールアドレスに転送しておくことを強くお勧めします。

手順 3.1: root ユーザに対するメール転送の設定

`root` ユーザに対するメール転送を設定するには、下記の手順を行います:

1. まずは `yast2-mail` パッケージをインストールします:

```
# zypper in yast2-mail
```

2. 対話的な YaST メール設定モジュールを起動します:

```
# yast mail
```

3. [接続の種類] では [常時接続] を選択して、[次へ] を押します。

4. [外部送信用メールサーバ] に、メールサーバのアドレスを入力します。必要であれば [認証] も設定してください。また、ネットワークを介して機密情報が送信されてしまわないよう、[TLS 暗号化] を [強制する] にしておくことを強くお勧めします。あとは [次へ] を押します。

5. 最後に [root 宛のメールの転送先] に転送先のメールアドレスを入力して [完了] を押します。これで設定は完了です。



重要: リモートからの SMTP 接続を受け付けてはならない問題について

通常は [リモートの SMTP 接続を許可] を選択しては なりません 。選択してしまうと、メールを受信して中継するように動作してしまうためです。

6. あとはメールの転送設定が正しく動作していることを確認します:

```
> mail root
subject: test
test
.
```

7. メールの転送が完了したかどうかは `mailq` コマンドで確認することができます。転送の設定に問題がなければ、[Mail queue is empty] のように出力されるはずです。あとは転送先のメールアカウントにアクセスして、メールが正しく届いていることを確認しておいてください。

なお、管理対象のマシン数とシステムイベントの通知先となる担当者数によって、様々な配信形態が考えられます:

- 様々なシステムから配信されるメールメッセージを、1 人の担当者のみがアクセスするメールアカウントに転送する。
- 別名やメーリングリストの機能を利用して、様々なシステムから配信されるメールメッセージを担当者全員に転送する。
- 各システムに対して別々のメールアカウントを準備する。

ただし、それぞれの担当者が定期的にメールをチェックしていることが何よりも重要です。このような取り組みを促して重要なイベントを見落とさないようにするため、不必要な情報は送信しないように設定しておくことも重要です。また、アプリケーション側の設定で、関連する情報のみを送信するようにもしておいてください。

3.6 中央の syslog サーバへのログメッセージの転送

システムログに記録されるメッセージを、個別のシステムからネットワーク内の中央に位置するサーバに転送するように設定することができます。これにより、全てのホストが出力したログメッセージを一括で閲覧および管理できるようになるほか、不正攻撃によって特定のホストが乗っ取られてしまったような場合でも、ログを削除して足跡を隠蔽するようなことができなくなります。

中央の syslog サーバにログを集約する設定を行う場合、中央の syslog サーバの設定を行ったあと、各システム (クライアント) の設定を行うことになります。

3.6.1 中央の syslog サーバの設定

手順 3.2: 中央の rsyslog サーバの設定

中央に配置する syslog サーバを設定するには、下記の手順を実施します:

1. 設定ファイル `/etc/rsyslog.d/remote.conf` をエディタなどで開きます。
2. 設定ファイル内にある `UDP Syslog Server` もしくは `TCP Syslog Server` のセクションを探すと、下記のような行があるはずです。これらの行のコメント文字 (#) を外します。また、`rsyslogd` に割り当てる IP アドレスとポートをそれぞれ設定します。
   TCP の場合の例:

```
$ModLoad imtcp.so
$UDPServerAddress IP❶
$InputTCPServerRun ポート❷
```

UDP の場合の例:

```
$ModLoad imudp.so
$UDPServerAddress IP ❶
$UDPServerRun ポート ❷
```

❶ `rsyslogd` がメッセージを受信する際に使用するインターフェイスの IP アドレスを指定します。アドレスを指定しない場合、デーモンは全てのインターフェイスで待ち受けを行います。

❷ `rsyslogd` がメッセージを受け付けるポートを指定します。特権ポートとして規定されている 1024 未満の値を指定してください。既定値は 514 です。

重要: TCP と UDP の違いについて

従来型の syslog では、UDP プロトコルを利用してネットワーク内にメッセージを転送していました。これにより負荷を減らす効果があるものの、その分だけ信頼性が失われることになります。つまり、高負荷な環境ではログメッセージが記録されない場合があり得ることになります。

そのため、UDP プロトコルではなく、信頼性の高い TCP プロトコルの使用をお勧めします。

注記: TCP での UDPServerAddress について

TCP の例でも `$UDPServerAddress` という設定パラメータを指定していますが、こちらは間違いではありません。名称とは異なり、TCP と UDP の両方に効果があります。

3. ファイルを保存してエディタを終了します。
4. `rsyslog` サービスを再起動します:

```
> sudo systemctl restart rsyslog.service
```

5. ファイアウオールで対象のポートを開きます。`firewalld` を TCP ポート 514 で動作させる場合は、下記のように入力して実行します:

```
> sudo firewall-cmd --add-port 514/tcp --permanent
> sudo firewall-cmd --reload
```

これで中央側の syslog サーバの設定は完了です。あとはログを送信する側、つまり個別のシステム側の設定を行います。

3.6.2 クライアント側の設定

手順 3.3: リモートログ記録のための［RSYSLOG］インスタンスの設定

中央の syslog サーバにログを送信するようマシンを設定したい場合は、下記の手順を実施します:

1. 設定ファイル `/etc/rsyslog.d/remote.conf` をエディタなどで開きます。
2. 設定ファイル内に下記のような行 (TCP または UDP) がありますので、これらの行のコメント文字 (#) を外したあと、`remote-host` の箇所を 3.6.1項「中央の syslog サーバの設定」で設定した中央の syslog サーバのアドレスに置き換えます。
   TCP の場合の例:

   ```
   # Remote Logging using TCP for reliable delivery
   # remote host is: name/ip:port, e.g. 192.168.0.1:514, port optional
   *.* @@remote-host
   ```

   UDP の場合の例:

   ```
   # Remote Logging using UDP
   # remote host is: name/ip:port, e.g. 192.168.0.1:514, port optional
   *.* @remote-host
   ```

3. ファイルを保存してエディタを終了します。
4. `rsyslog` サービスを再起動します:

   ```
   > sudo systemctl restart rsyslog.service
   ```

5. あとは syslog のメッセージが正しく転送できていることを確認します:

   ```
   > logger "hello world"
   ```

   上記を実行すると、中央の syslog サーバに `hello world` というメッセージが記録されているはずです。

中央の syslog サーバに送信するマシンがほかにもあれば、上記の手順をそれぞれで実施してください。これにより、中央の syslog サーバに情報を集約できるようになります。

3.6.3 さらなる情報

ここで説明した基本的な手順は暗号化を設定していないため、信頼のできる内部ネットワーク内でのみ適用可能な手順になっています。信頼のできないネットワークを経由する場合は TLS による暗号化が必須となりますが、これには証明書のインフラを構築する必要があります。

また、ここまでの説明では、遠隔からの全てのメッセージは全て同じに扱われる設定になります。ログの送信元が増えれば増えるほどメッセージも増えていくことになりますので、送信元ごとに別々のファイルに記録したり、メッセージの分類を行ったりなどの作業も必要となることでしょう。

暗号化やフィルタリング、もしくはその他の高度な設定について、詳しくは https://www.rsyslog.com/doc/master/index.html#manual (英語) にある RSyslog のドキュメンテーションをお読みください。

3.7 `logger` コマンドによるシステムログへの記録

`logger` はシステムログ内に指定した内容を出力するためのツールです。特にシェルスクリプトなどから、rsyslogd のシステムログモジュールにメッセージを書き込みたい場合に使用します。たとえば下記のように入力して実行すると、メッセージが `/var/log/messages` もしくは `systemd` のジャーナル (ログ記録用のサービスを何も動作させていなければ) に書き込まれます:

```
> logger -t Test "This message comes from $USER"
```

ログインしているユーザとホスト名によって異なりますが、ログには下記のような内容が書き込まれます:

```
Sep 28 13:09:31 venus Test: This message comes from tux
```

III カーネル監視

4 SystemTap: システムデータのフィルタリングと分析 68

5 カーネルプローブ 83

6 Perf を利用したハードウエアベースの監視 88

7 OProfile: システム全体に対するプロファイラ 93

8 ダイナミックデバッグ: カーネルのデバッグメッセージの調整 99

4 SystemTap: システムデータのフィルタリングと分析

改訂履歴

2024-06-26

SystemTap は、稼働中の Linux システムで動作状況を調査するためのコマンドラインインターフェイスとスクリプト言語を提供する仕組みです。特にカーネルの動作状況を詳細に収集することができます。SystemTap のスクリプトは SystemTap スクリプト言語で記述され、これを C 言語のカーネルモジュールにコンパイルして、カーネル内に挿入します。スクリプトはデータを抽出するだけでなく、フィルタや要約を行うこともできるため、複雑な性能問題や機能面の問題を分析することができるようになっています。また、SystemTap では `netstat` , `ps` , `top` , `iostat` などのツールの出力に似た情報を提供しますが、収集した情報に対してフィルタリングや分析などのオプションを適用することができます。

4.1 考え方の概要

SystemTap スクリプトを動作させると、SystemTap セッションが起動します。また、スクリプトが実行されるまでの間に、いくつかのパスが実行されます。その後、スクリプトはカーネルモジュールとしてコンパイルされ、読み込まれます。スクリプトを以前に実行していて、システムコンポーネント (コンパイラやカーネルのバージョン、ライブラリパスやスクリプトの内容) に変更がない場合、SystemTap は再度スクリプトをコンパイルするようなことはありません。その代わり、SystemTap のキャッシュ (`~/.systemtap`) 内に保存されている `*.c` と `*.ko` のデータを使用します。

タップの動作が完了すると、モジュールの読み込みも解除されます。たとえば 4.2項「インストールと設定」にあるテストスクリプトと、対応する説明をお読みください。

4.1.1 SystemTap スクリプト

SystemTap は SystemTap スクリプト (`*.stp`) を利用して行う仕組みです。スクリプト内では収集する情報の種類のほか、収集後に何をすべきかを指定しています。スクリプトは SystemTap スクリプト言語と呼ばれる、AWK や C 言語に似た言語で作成します。言語の詳しい定義については https://sourceware.org/systemtap/langref.pdf (英語) をお読みください。また、スクリプト例については https://www.sourceware.org/systemtap/examples/ をご覧ください。

SystemTap スクリプトの考え方のベースとして、**イベント** とそれに対応した **ハンドラ** があります。SystemTap がスクリプトを実行すると、SystemTap はまず特定のイベントを待機します。そのイベントが発生すると、Linux カーネルはサブルーチンとして指定されたハンドラを実行し、元の処理に戻り

ます。このような仕組みであることから、イベントはスクリプトを実行する際のトリガー (起動条件) と考えることができます。また、ハンドラは指定したデータをそのまま記録することができるほか、特定の書式でそれを出力することもできます。

SystemTap 言語にはいくつかのデータ型 (整数, 文字列, およびそれらの連想配列) と、完全な制御構造 (ブロック, 条件分岐, ループ, 関数) が用意されています。また、文法は軽量な構造で、文末のセミコロンは省略することができますし、詳細なデータ型指定を行う必要もありません (データ型は自動的に推測およびチェックされます) 。

SystemTap スクリプトとその文法に関する詳細については、4.3項「スクリプトの文法」のほか、`stapprobes` と `stapfuncs` の各マニュアルページをお読みください。マニュアルページは `systemtap-docs` パッケージ内に含まれています。

4.1.2 タップセット

タップセットは、あらかじめ記述されたプローブや関数を含むライブラリで、SystemTap のスクリプト内から使用することができます。SystemTap のスクリプトを実行すると、SystemTap はスクリプト内のプローブイベントとハンドラが、タップセットライブラリ内に存在していないかどうかを確認します。SystemTap はスクリプトを C 言語に翻訳する前に、対応するプローブや関数を読み込みます。SystemTap スクリプト自身のように、タップセットにも同じファイル拡張子 `*.stp` が設定されています。

ただし、SystemTap スクリプトとは異なり、タップセットは直接実行するためのものではありません。タップセットは他のスクリプトに定義を与える存在であり、タップセットライブラリはユーザに対して、イベントや関数の定義をやりやすくするために設計されているものです。タップセットは関数に対して別名を提供し、ユーザ側からイベントとして指定することができるようになります。そのため、カーネルのバージョンによって異なるようなカーネルの関数を、より簡単に指定することができるようになります。

4.1.3 コマンドと権限

SystemTap に付属するメインのコマンドは、`stap` と `staprun` の 2 種類です。これらを実行するには `root` の権限を使用するか、もしくは `stapdev` または `stapusr` のメンバーである必要があります。

`stap`

SystemTap のフロントエンドです。SystemTap スクリプトのファイルを指定するか、もしくは標準入力から読み込むことで実行することができます。スクリプトは C 言語のコードに翻訳され、コンパイルされて動作中の Linux カーネル内でモジュールとして読み込みが行われます。これにより、必要なシステムトレースやプローブ関数を実行することができるようになります。

`staprun`

SystemTap のバックエンドです。SystemTap のフロントエンドが作成したカーネルモジュールを、読み込んだり読み込みを解除したりすることができます。

それぞれのコマンドに対するオプションを知りたい場合は、`--help` オプションを指定して実行してください。詳しくは `stap` と `staprun` の各マニュアルページをお読みください。

`root` の権限を与えずに一般ユーザの権限で SystemTap を実行させたい場合は、対象のユーザを下記の SystemTap グループのいずれかに所属するように設定してください。これらのグループは既定の openSUSE Leap では作成されていませんので、必要に応じて作成し、アクセス権を調整してください。また、お使いの環境でセキュリティへの影響に問題がなければ、`staprun` コマンドのパーミッションを調整してもかまいません。

`stapdev`

このグループのメンバーであれば、`stap` コマンドで SystemTap スクリプトを実行することができるほか、`staprun` コマンドで計装モジュールを実行することもできます。なお、`stap` コマンドの実行には、スクリプトをカーネルモジュールにコンパイルする処理と、コンパイルしたカーネルモジュールをカーネルに読み込む処理が含まれますので、実質的には `root` のアクセスと同等の権限を持つことになります。

`stapusr`

このグループのメンバーは、`staprun` コマンドで計装モジュールを実行する権限のみを持ちます。これに加えて、`/lib/modules/カーネルバージョン/systemtap/` 内にあるモジュールのみを実行することができます。このディレクトリは `root` が所有しなければならず、かつ `root` ユーザのみが書き込むことができるようにしなければなりません。

4.1.4 主要なファイルとディレクトリ

下記では、SystemTap で使用される主なファイルとディレクトリについて、概要を説明しています。

`/lib/modules/カーネルバージョン/systemtap/`

SystemTap の計装モジュールが含まれるディレクトリです。

`/usr/share/systemtap/tapset/`

タップセットの標準ライブラリが含まれるディレクトリです。

`/usr/share/doc/packages/systemtap/examples`

様々な用途に対応した SystemTap スクリプトの例を配置しているディレクトリです。なお、このディレクトリは `systemtap-docs` パッケージをインストールした場合にのみ作成されます。

`~/.systemtap/cache`

キャッシュされた SystemTap ファイルを含むデータディレクトリです。

`/tmp/stap*`

SystemTap ファイルの一時ディレクトリです。C 言語に変換されたコードのほか、カーネルオブジェクトなどが含まれます。

4.2 インストールと設定

SystemTap を動作させるにはカーネルに関する情報が必要となることから、カーネル関連の追加パッケージをインストールしなければなりません。また、SystemTap でプローブを設定したいカーネルごとに、それぞれ下記のパッケージをインストールする必要があります。さらに、下記のパッケージをインストールする場合は、プローブを設定したいカーネルのフレーバー (下記では `*` と記してあります) とバージョンが厳密に一致したパッケージをインストールする必要があります。

重要: デバッグ情報 (debuginfo) パッケージのリポジトリについて

お使いのシステムでオンライン更新を有効にしていれば、openSUSE Leap 15.7 に対応する `*-Debuginfo-Updates` という名前のオンラインリポジトリ内に、「debuginfo」パッケージが存在しています。このリポジトリを有効にするには、YaST をお使いください。

従来型の SystemTap 設定を行う場合は、下記のパッケージをインストールしてください (YaST もしくは `zypper` を利用してインストールします。

- `systemtap`
- `systemtap-server`
- `systemtap-docs` (任意)
- `kernel-*-base`
- `kernel-*-debuginfo`
- `kernel-*-devel`
- `kernel-source-*`
- `gcc`

マニュアルページや SystemTap のスクリプト例など、様々な目的でのヘルプ情報を必要とする場合は、これらに加えて `systemtap-docs` パッケージもインストールしてください。

マシン内に必要なパッケージを全てインストールできていて、SystemTap を使用するための準備が整っているかどうかを確認するには、下記のコマンドを `root` で実行します:

```
# stap -v -e 'probe vfs.read {printf("read performed
"); exit()}'
```

上記のコマンドを実行すると、スクリプトを実行して現在動作しているカーネルに対してプローブを設定し、出力を返します。出力が下記のような内容になっていれば、SystemTap は問題なく配置できたことになります:

```
Pass ❶: parsed user script and 59 library script(s) in 80usr/0sys/214real ms.
Pass ❷: analyzed script: 1 probe(s), 11 function(s), 2 embed(s), 1 global(s) in
 140usr/20sys/412real ms.
Pass ❸: translated to C into
 "/tmp/stapDwEk76/stap_1856e21ea1c246da85ad8c66b4338349_4970.c" in 160usr/0sys/408real ms.
Pass ❹: compiled C into "stap_1856e21ea1c246da85ad8c66b4338349_4970.ko" in
 2030usr/360sys/10182real ms.
Pass ❺: starting run.
 read performed
Pass ❺: run completed in 10usr/20sys/257real ms.
```

❶ スクリプトと `/usr/share/systemtap/tapset/` 内にある既存のタップセットライブラリをチェックしています。タップセットとはあらかじめ作成されたプローブや関数を集めたライブラリで、SystemTap スクリプト内から使用できる仕組みです。

❷ スクリプトのコンポーネントを調べています。

❸ スクリプトを C 言語に変換しています。その後、作成された C 言語のソースコードを C コンパイラに通して、カーネルモジュールを作成しています。変換後の C 言語ソースコード (`*.c`) とカーネルモジュール (`*.ko`) が、`~/.systemtap` にある SystemTap キャッシュ内に保管されます。

❹ モジュールを読み込んだあと、スクリプト内に指定されている全てのプローブ (イベントおよびハンドラ) をカーネル内のフックとして設置し、有効化しています。この例で使用しているプローブは、仮想ファイルシステム (Virtual File System; VFS) の読み込み処理になります。任意のプロセッサ内で対象のイベントが発生すると、対応するハンドラが実行 (この例では、`read performed` と表示しています) され、エラー無しに終了しています。

❺ SystemTap のセッションが終了すると、プローブは無効化され、カーネルモジュールの読み込みも解除されます。

テスト時に何らかのエラーメッセージが表示された場合、まずは出力された内容をご確認ください。出力された内容に必要なパッケージに関する情報が書かれている場合は、まず必要なパッケージが正しくインストールされていることを確認してください。また、カーネルをインストールした場合は、システムを再起動して新しいカーネルを読み込む必要があります。

4.3 スクリプトの文法

SystemTap のスクリプトは、下記に示す 2 つのコンポーネントから構成されています:

SystemTap イベント (プローブポイント)
: カーネル内のどのイベントでスクリプトを実行するのかを表している箇所です。特定の関数内に入ったタイミングや特定の関数を出たタイミング、もしくはタイマーの経過後やセッションの開始および停止などに設定することができます。

SystemTap ハンドラ (プローブボディ)
: 特定のイベントが発生した際に実行すべきスクリプトを表している箇所です。これは通常、イベント内の情報を抽出したり、それらを内部の変数に格納したり、結果を出力したりする処理を行います。

イベントとそれに対応するハンドラをまとめて、**プローブ** と呼びます。SystemTap のイベントは **プローブポイント** 、対応するハンドラは **プローブボディ** とも呼びます。

コメントは SystemTap スクリプト内の任意の箇所に配置することができます。コメントの書式は、`#`, `/* */` , `//` のいずれかです。

4.3.1 プローブの書式

SystemTap のスクリプトには複数のプローブを設定することができます。プローブは下記の書式で作成します:

```
probe イベント名 {ステートメント}
```

それぞれのプローブには対応するステートメントブロックを設定します。ステートメントブロックは `{ }` で括らなければならず、かつイベントごとに実行すべきステートメントを含むものとします。

例 4.1: シンプルな SYSTEMTAP スクリプト

下記はシンプルな SystemTap スクリプトの例です。

```
probe❶ begin❷
{❸
   printf❹ ("hello world
")❺
   exit ()❻
}❼
```

❶ プローブの開始位置です。

❷ イベント `begin` (SystemTap セッションの開始) の指定です。

❸ `{` で示されているとおり、ハンドラ定義の開始位置です。

❹ ハンドラ内での最初の関数です。この例では `printf` 関数を使用しています。

❺ `printf` 関数で出力すべき文字列を指定しています。末尾には改行 (`/n`) が書かれています。

❻ ハンドラ内での 2 つめの関数です。この例では `exit()` 関数を使用しています。なお、SystemTap スクリプトは `exit()` 関数を実行するまでは、動作し続けることに注意してください。この関数を実行する前にスクリプトの実行を停止したい場合は、 `Ctrl` – `C` を押します。

❼ `}` で示されているとおり、ハンドラ定義の終了位置です。

イベント `begin` ❷ (SystemTap セッションの開始) が発生すると、`{ }` で括られたハンドラを実行します。ここでは、`printf` 関数 ❹ になります。この場合、`hello world` という文字列に加えて改行 ❺ を出力しています。出力が終わると、スクリプトは終了します。

ステートメント内に複数の記述が存在する場合、SystemTap ではそれらを順次実行します。言い換えると、複数のステートメントを記述するにあたって、特殊な区切り文字や終端などを入れる必要はありません。SystemTap 内では一般に、ステートメントブロックは C プログラミング言語と同じ書式になっています。

4.3.2 SystemTap イベント (プローブポイント)

SystemTap にはいくつかの内蔵イベントが用意されています。

一般的なイベント名はドットで区切った DNS のような命名体系になっています。この仕組みにより、名前をツリー構造で管理できるようになっています。また、ドット区切り内の各コンポーネントには、文字列や数値によるパラメータを設定できるようになっていて、見た目では関数のような表現になっています。また、コンポーネントに `*` を指定した場合、複数のプローブポイントにマッチするようなイベント名にすることができます。また、プローブポイントの後には `?` が付けられることもあります。この場合、それが任意指定のものであることを示し、展開に失敗した場合もエラーが無視されるようになります。また、プローブポイントの後ろに `!` を付けた場合、任意指定でかつ十分な存在であることを示します。

SystemTap では 1 つのプローブに対して複数のイベントを設定することができます。複数のイベントを指定する場合、イベント間は (`,`) で区切ります。また、1 つのプローブに対して複数のイベントが指定された場合、SystemTap はいずれかのイベントが発生した場合に、指定のハンドラを実行することになります。

イベントは下記のような分類に分けることができます:

- 同期型イベント: 任意のプロセスが、カーネルコード内の特定の箇所に達した場合に発生するイベントです。これは他のイベントに対して参照ポイント (インストラクションアドレス) を与え、ここから多くのコンテキスト情報を取得できるようになります。

同期型イベントの例として、`vfs.ファイル操作` があります。これは仮想ファイルシステム (Virtual File System; VFS) 内の `ファイル操作` イベントを表すもので、たとえば 4.2項「インストールと設定」では、`read` が `ファイル操作` の箇所にあたります。

- 非同期型イベント: 特定のインストラクションやカーネルコード内の位置に結びつかないイベントです。このようなプローブポイントとしては、カウンターやタイマーなどの構造があります。
  非同期型イベントの例としては、`begin` (SystemTap セッションの開始) や `end` (SystemTap セッションの終了) のほか、タイマーイベントがあります。タイマーイベントは定期的に実行すべきハンドラを指定するためのもので、たとえば `example timer.s(秒)` や `timer.ms(ミリ秒)` のように指定します。
  タイマーイベントは、他のプローブと併用することで、定期的なデータ更新を行うことができるようになるほか、時間が経過するたびに変化していくような情報を追跡することができるようになります。

例 4.2: タイマーイベントのプローブ

> たとえば下記のようなプローブを設定すると、「hello world」を 4 秒間隔で出力します:
>
> ```
> probe timer.s(4)
> {
>    printf("hello world
> ")
> }
> ```

対応するイベントの詳細について、詳しくは `stapprobes` のマニュアルページをお読みください。マニュアルページ内の See Also セクションには、特定のサブシステムやコンポーネントに対するイベントを含む、他のマニュアルページへのリンクが書かれています。

4.3.3 SystemTap ハンドラ (プローブボディ)

SystemTap イベントには対応するハンドラが指定されます。ハンドラ内では、ステートメントブロックから構成される処理を記述します。

4.3.3.1 関数

複数のプローブ内で同じステートメントを実行する必要がある場合は、それらの共通部分を関数として取り出して、再利用しやすくすることをお勧めします。関数はキーワード `function` に続いて関数名を指定することで、定義を行うことができます。また、任意の数の文字列や数値 (いずれも値渡し) をパラメータとして取ることができ、返り値は単一の文字列もしくは数値を設定することができます。

```
function 関数名(パラメータ) {ステートメント}
```

```
probe イベント {関数名(パラメータ)}
```

上記の例では、イベント のハンドラが実行されると、関数名 の関数が実行されます。パラメータ は任意指定のもので、関数に渡すパラメータを指定します。

関数はスクリプト内の任意の場所に記述することができます。

よく使用する関数の例として、既に 例4.1「シンプルな SystemTap スクリプト」で紹介を行っている printf 関数があります。この関数は書式を指定してデータを出力するための関数で、書式ではどれだけの数のパラメータを出力するのか、およびどのように整形して出力するのかを指定します。書式文字列は二重引用符で括り、書式指定は % 文字で書き始めます。

どのような書式指定文字列を使用するのかは、そのパラメータに依存して決まります。書式指定文字列は複数個指定することができ、その際にはパラメータをカンマ区切りで指定します。

例 4.3: 書式指定を伴う printf 関数

```
printf ("❶%s❷(%d❸) open
❹", execname(), pid())
```

❶ " で示されているとおり、書式指定文字列の開始位置です。

❷ 文字列の書式指定です。

❸ 整数の書式指定です。

❹ " で示されているとおり、書式指定文字列の終了位置です。

上記の例では、実行ファイル名 (execname()) を文字列で、プロセス ID (pid()) を括弧内に整数で、それぞれ出力する指定になっています。その後ろは固定でスペースを入れ、open と出力して改行を行います:

```
[...]
vmware-guestd(2206) open
held(2360) open
[...]
```

例4.3「書式指定を伴う printf 関数」で使用している execname() や pid() のほかにも、printf のパラメータとしては様々な関数を指定することができます。

SystemTap で最もよく使用される関数は下記のとおりです:

tid()
: 現在のスレッドの ID を返します。

pid()
: 現在のスレッドのプロセス ID を返します。

uid()
: 現在のユーザのユーザ ID を返します。

cpu()
: 現在の CPU 番号を返します。

execname()
: 現在のプロセスの名前を返します。

gettimeofday_s()
: Unix エポック (1970 年 1 月 1 日) からの経過秒数を返します。

ctime()
: 時刻を文字列に変換します。

pp()
: 現在処理しているプローブポイントの文字列表現を返します。

thread_indent()
: 出力結果を読みやすくするための便利な関数です。これは各スレッド (`tid()`) に対して (内部的な) インデントカウンタを保存するものです。この関数には 1 つだけパラメータを指定しますが、このパラメータはインデント差 (スレッドのインデントカウンタの増減値) を指定します。返り値は一般的なトレースデータと、インデントカウンタの値に対応した数のスペースを含む文字列になります。一般的なトレースデータには、タイムスタンプ (スレッドの初期インデントからの経過マイクロ秒) のほか、プロセス名とスレッド ID それ自身が含まれます。これにより、どの関数を誰が呼び出したのか、およびそれにどれだけの時間を要したのかがわかるようになっています。
  システムコールの開始と終了は、見た目ではわかりにくくなってしまうことがあります。たとえば一方のシステムコールが他方のシステムコールを呼び出していて、そのいずれにもプローブを設定しているような場合、開始と終了が入れ子になってしまい、出力がわかりにくくなってしまいます。このインデントカウンタの仕組みを使用することで、入れ子の形で呼び出されているシステムコールの出力を字下げ (インデント) して、読みやすくする効果を生み出します。

提供されている SystemTap 関数に関する詳細な情報については、`stapfuncs` のマニュアルページをお読みください。

4.3.3.2 その他の基本構造

関数のほかに、SystemTap ハンドラ内ではその他の一般的な構造を作成することができます。具体的には変数や条件分岐 (`if` / `else`) 、`while` ループや `for` ループ、そして配列やコマンドラインパラメータなどがあります。

4.3.3.2.1 変数

変数はスクリプト内の任意の場所に定義することができます。変数を定義したい場合は、単純に名前を選んで関数や表現を代入するだけです:

```
foo = gettimeofday( )
```

代入した後は、任意の箇所で変数を使用するだけです。SystemTap では、変数の代入元のデータ型 (文字列もしくは数値) に従って、変数のデータ型を自動継承します。何らかの矛盾が発生した場合、それらはエラーとして報告されます。上記の例では、`foo` は代入元に従って数値型として設定され、`printf()` 関数の整数書式指定子 (`%d`) で出力できる変数になります。

ただし、既定では変数は全てプローブ内ローカルになります。プローブ内ローカルであることから、変数はハンドラの実行のたびに初期化され、使用され、廃棄されることになります。プローブ間で変数を共有したい場合は、グローバル変数としてスクリプト内に定義してください。グローバル変数を定義するには、プローブの外で `global` キーワードを指定して宣言します:

例 4.4: グローバル変数の使用

```
global count_jiffies, count_ms
probe timer.jiffies(100) { count_jiffies ++ }
probe timer.ms(100) { count_ms ++ }
probe timer.ms(12345)
{
  hz=(1000*count_jiffies) / count_ms
  printf ("jiffies:ms ratio %d:%d => CONFIG_HZ=%d
",
    count_jiffies, count_ms, hz)
  exit ()
  }
```

上記の例では、jiffies とミリ秒によるタイマーを利用して、カーネルの CONFIG_HZ の設定を計算しています。jiffy/jiffies はシステムのタイマー割り込みの回数を数えるもので、プロセッサのクロック周波数によって間隔が変化するものであることから、これを実時間で割ることで、クロック周波数を計算しています。ここでの `global` 定義は `count_jiffies` と `count_ms` ですが、これらはそれぞれのタイマー値を保存しておくためのものです。また、`++` という演算子は、変数の値を `1` だけ足す意味です。

4.3.3.2.2 条件分岐

SystemTap スクリプト内で使用できる条件分岐には、いくつかのものがあります。最もよく使用されるものを下記に示します:

If/Else ステートメント

下記のように記述します:

```
if (条件)❶ステートメント_1❷
else❸ステートメント_2❹
```

`if` ステートメントは、整数表現がゼロであるかどうかを調べます。❶ の結果がゼロ以外であった場合、最初のステートメント ❷ が実行されます。逆に、ゼロであった場合は、2 番目のステートメント ❹ が実行されます。else 句 (❸ および ❹) は任意指定で、指定しても指定しなくてもかまいません。また、❷ と ❹ には、ステートメントブロックを指定することもできます。

While ループ

下記のように記述します:

```
while (条件)❶ステートメント❷
```

条件 がゼロ以外であった場合、❷ を実行します。❷ はステートメントブロックでもかまいません。最終的に 条件 がゼロになるまで、繰り返し実行します。

For ループ

`while` ループのショートカットとして規定されているもので、下記のように記述します:

```
for (初期化❶; 条件❷; 増加処理❸) ステートメント
```

❶ に記述したステートメントはカウンタの初期化用に使用すべき箇所で、ループの開始前に実行されます。ループは ❷ のステートメントの結果が false になるまで繰り返されます (判断処理はループの処理前に行われます) 。また、❸ はループカウンタの増加処理で使用すべき箇所です。こちらはループ処理後に実行されます。

比較演算子

条件の指定では、下記の演算子を使用することができます:

==: 等しいかどうか

!=: 等しくないかどうか

>=: 前のほうが後より大きいかどうか、もしくは等しいかどうか

<=: 前のほうが後より小さいかどうか、もしくは等しいかどうか

4.4 スクリプト例

`systemtap-docs` パッケージをインストールしていれば、`/usr/share/doc/packages/systemtap/examples` 内に SystemTap のスクリプト例がいくつか用意されています。

本章では、`/usr/share/doc/packages/systemtap/examples/network/tcp_connections.stp` にあるシンプルなスクリプト例を詳しく説明します:

例 4.5: `tcp_connections.stp` での着信 TCP 接続の監視

```
#! /usr/bin/env stap

probe begin {
  printf("%6s %16s %6s %6s %16s
",
         "UID", "CMD", "PID", "PORT", "IP_SOURCE")
}

probe kernel.function("tcp_accept").return?,
      kernel.function("inet_csk_accept").return? {
  sock = $return
  if (sock != 0)
    printf("%6d %16s %6d %6d %16s
", uid(), execname(), pid(),
           inet_get_local_port(sock), inet_get_ip_source(sock))
}
```

この SystemTap スクリプトは着信側の TCP 接続を監視して、リアルタイムに望まない接続を識別する仕組みを提供するものです。それぞれコンピュータが受け入れた着信側 TCP 接続に対して、下記のような情報を表示します:

- ユーザ ID (`UID`)
- 接続を受け付けたコマンド (`CMD`)
- コマンドのプロセス ID (`PID`)
- 接続に使用しているポート (`PORT`)
- TCP 接続の発信元 (`IP_SOURCE`)

スクリプトを実行するには、下記のように入力して実行します:

```
stap /usr/share/doc/packages/systemtap/examples/network/tcp_connections.stp
```

すると、画面内に出力が表示されるようになります。スクリプトを手作業で停止するには、`Ctrl`–`C` を押します。

4.5 ユーザスペースプローブ

DTrace のようにユーザスペースで動作するアプリケーションをデバッグする目的で、openSUSE Leap 15.7 では、SystemTap でユーザスペースのプローブを行うことができます。ユーザスペースのプローブでは、アプリケーション内の任意の場所にプローブポイントを設置することができます。これにより、SystemTap はカーネルスペースとユーザスペースの両方を調査できることになり、結果としてシステム全体を調査できるようになっています。

ユーザスペースのプローブを設置する際に必要となる utrace インフラストラクチャと uprobe カーネルモジュールを取得するには、4.2項「インストールと設定」に示されているパッケージに加え、`kernel-trace` パッケージをインストールする必要があります。

`utrace` はユーザスペースの処理を制御するためのフレームワークを実装するものです。様々なトレーシング「エンジン」で使用することのできるインターフェイスを提供し、これ自身は読み込み可能なカーネルモジュールとして提供されます。エンジンは特定のイベントに対してコールバック関数を登録し、トレース対象のスレッドに結びつけられます。コールバックはカーネル内の「安全な」場所から行われるため、関数が実行できる処理の種類に大きな余裕ができることになります。utrace では、たとえばシステムコールの開始と終了、fork() やタスクに送信されるシグナルなど、様々なイベントを監視することができます。utrace のインフラストラクチャについて、詳しくは https://sourceware.org/systemtap/wiki/utrace (英語) をお読みください。

SystemTap では、ユーザスペースプロセス内での関数の開始および終了のプローブに対応しているほか、コード内にマーカーを設定してプローブしたり、特定のプロセス内イベントを監視したりすることもできます。

現在実行しているカーネルが utrace に対応しているかどうかを調べるには、下記のコマンドを実行します:

```
> sudo grep CONFIG_UTRACE /boot/config-`uname -r`
```

ユーザスペースのプローブについて、詳しくは https://sourceware.org/systemtap/SystemTap_Beginners_Guide/userspace-probing.html (英語) をお読みください。

4.6 さらなる情報

本章では、SystemTap の概要部までしか言及できていません。SystemTap に関する詳細な情報を得たい場合は、それぞれ下記を参照してください (英語のみの提供である場合もあります):

https://sourceware.org/systemtap/
: SystemTap プロジェクトの Web ページです。

https://sourceware.org/systemtap/wiki/

SystemTap に関する便利な情報を数多く集めているサイトです。ユーザに対する詳しい説明から開発者向けの資料のほか、他のツールとの比較やレビュー、よくある質問とその回答 (FAQ) やヒントなどが書かれています。また、SystemTap スクリプトの例もあり、使用例や SystemTap に関する直近の対話、ペーパーなども用意されています。

https://sourceware.org/systemtap/documentation.html

PDF および HTML 形式で、SystemTap Tutorial (SystemTap チュートリアル), SystemTap Beginner's Guide (SystemTap 初心者ガイド), Tapset Developer's Guide (タップセット開発者ガイド), SystemTap Language Reference (SystemTap 言語リファレンス) などが用意されています。

なお、SystemTap の言語リファレンスとチュートリアルについては、`/usr/share/doc/packages/systemtap` ディレクトリ内にも用意されています。また、スクリプトの例については、上記の `example` サブディレクトリをご覧ください。

5 カーネルプローブ

改訂履歴

2023-08-08

カーネルプローブは Linux カーネルのデバッグや性能に関する情報を収集するためのツール集です。開発者やシステム管理者は、これらを利用してカーネルのデバッグを行ったり、システムにおける性能のボトルネックを発見したりすることができます。また、収集したデータをシステムの性能改善のために使用することもできます。

カーネルプローブは任意のカーネルルーチン内に設定し、カーネル処理内の特定の箇所 (ブレークポイント) に到達した後に、指定したハンドラを実行することができるようになります。カーネルプローブの主な利点としては、カーネルの再構築が不要であることと、プローブの変更後にシステムの再起動が不要になっている点です。

カーネルプローブを使用するには、まずは必要なカーネルモジュールを作成する必要があります。このモジュール内には 初期化 と 終了 の各関数を用意する必要があります。初期化の関数 (たとえば `register_kprobe()`) では 1 つ以上のプローブを設定し、終了の関数ではそれらの登録を解除します。登録処理ではカーネル内の どこ にプローブを挿入するかと、そのプローブに到達した際に どのようなハンドラ を実行するのかを定義します。複数のプローブを一括で登録もしくは登録解除するには、対応する `register_<プローブの種類>probes()` や `unregister_<プローブの種類>probes()` を使用する必要があります。

デバッグや状態を表すメッセージは一般に、`printk` カーネルルーチンを利用して行います。`printk` はユーザスペースでの `printf` と等価なカーネルルーチンです。`printk` に関する詳細については、Logging kernel messages (https://www.win.tue.nl/~aeb/linux/lk/lk-2.html#ss2.8) (英語) をお読みください。通常、このカーネルルーチンで出力されたメッセージは、`systemd` ジャーナルで読むことができます (詳しくは **『リファレンス』、第11章「`journalctl` : systemd ジャーナルへの問い合わせコマンド」** をお読みください) 。なお、ログファイルに関する詳細については、第3章「システムログファイル」をお読みください。

5.1 対応するアーキテクチャ

カーネルプローブは、下記のアーキテクチャ向けに 完全 実装されています:

- x86
- AMD64/Intel 64
- Arm
- POWER

カーネルプローブは、下記のアーキテクチャ向けに 部分 実装されています:

- IA64 (インストラクション `slot1` に対するプローブには対応していません)
- sparc64 (return probe が実装されていません)

5.2 カーネルプローブの種類

カーネルプローブには 3 種類のものがあります。具体的には Kprobes , Jprobes , Kretprobes の 3 種類です。Kretprobes は return probes (返りプローブ) とも呼ばれます。3 種類のプローブの例については、Linux カーネルのソースコードをお読みください。`/usr/src/linux/samples/kprobes/` ディレクトリ内 (`kernel-source` パッケージ内) にあります。

5.2.1 Kprobes

Kprobes は Linux カーネル内の任意のインストラクション (命令) に接続することができる仕組みです。Kprobes を登録すると、対象となるインストラクションの最初のバイト位置にブレークポイントが設定されます。プロセッサがそのブレークポイントに到達すると、プロセッサレジスタを保存して、処理を Kprobes に渡します。その後 pre-handler (事前ハンドラ) を実行し、対象のインストラクションをステップ実行したのち、最後に post-handler (事後ハンドラ) を実行します。あとはプローブポイント後の位置に制御が移り、元の動作に戻ることになります。

5.2.2 Jprobes

Jprobes は Kprobes の仕組みを利用して実装されているものです。関数の開始位置に挿入され、プローブ対象の関数のパラメータに対して、直接アクセスできるようになります。ハンドラルーチンにはプローブ対象の関数と全く同じパラメータリストを指定しなければならず、かつ返り値も同じ型でなければなりません。これを終了するには、`jprobe_return()` 関数を呼び出します。

JProbes が設定された関数を到達すると、プロセッサレジスタを保存したあと、インストラクションポインタは JProbes のハンドラルーチンに転送されます。その後、制御は ハンドラ内に移り、プローブ対象の関数と同じレジスタ状態になります。最後に、ハンドラは `jprobe_return()` を呼び出し、制御が元の関数に戻るようになります。

一般的には、1 つの関数に対して複数のプローブを設定することができますが、Jprobes については 1 つの関数に対して 1 つのインスタンスのみに制限されます。

5.2.3 Return Probe

Return Probe も Kprobes を介して実装されています。`register_kretprobe()` を呼び出すと、Kprobes はプローブ対象の関数の開始位置に設定されます。プローブ位置に到達すると、カーネルはプローブ対象の関数の返りアドレスを保存して、ユーザが指定した返りハンドラを呼び出します。あとは制御が元の関数に戻ります。

`register_kretprobe()` を呼び出す場合は、`maxactive` パラメータを指定する必要があります。これはこの関数を同時にいくつまでプローブするのかを指定するパラメータで、値が小さすぎるとプローブが失敗することがあります。

5.3 Kprobes API

Kprobes のプログラミングインターフェイスは、使用するカーネルプローブと対応するプローブハンドラの登録と登録解除に使用する関数から構成されています。これらの関数とパラメータに対する詳しい説明については、5.5項「さらなる情報」にある情報源を参照してください。

`register_kprobe()`
: 指定したアドレスにブレークポイントを挿入します。ブレークポイントに到達すると、`pre_handler` (事前ハンドラ) と `post_handler` (事後ハンドラ) がそれぞれ呼び出されます。

`register_jprobe()`
: 指定したアドレスにブレークポイントを挿入します。アドレスは、プローブ対象となる関数内で最初のインストラクションのアドレスである必要があります。ブレークポイントに到達すると、指定したハンドラを実行します。ハンドラはプローブ対象の関数と同じパラメータリストでなければならず、かつ返り値も同じデータ型でなければなりません。

`register_kretprobe()`
: 指定した関数に return probe を挿入します。プローブ対象の関数が終了して値を返す際、指定したハンドラが呼び出されます。この関数は 0 を返すと成功の意味に、負の数のエラー番号である場合は失敗の意味になります。

`unregister_kprobe()`, `unregister_jprobe()`, `unregister_kretprobe()`
: 指定したプローブを削除します。これらの関数は、プローブの登録後であれば任意のタイミングで使用することができます。

`register_kprobes()`, `register_jprobes()`, `register_kretprobes()`
: 指定した配列内にあるそれぞれのプローブを挿入します。

`unregister_kprobes()`, `unregister_jprobes()`, `unregister_kretprobes()`
: 指定した配列内にあるそれぞれのプローブを削除します。

`disable_kprobe()`, `disable_jprobe()`, `disable_kretprobe()`
: 指定したプローブを一時的に無効化します。

`enable_kprobe()`, `enable_jprobe()`, `enable_kretprobe()`
: 指定したプローブを一時的に有効化します。

5.4 debugfs インターフェイス

新しい Linux カーネルでは、Kprobes はカーネルの `debugfs` インターフェイスを利用して制御することができます。ここでは登録済みのプローブの一覧や、全てのプローブの一括有効化／無効化を行うことができます。

5.4.1 登録済みのカーネルプローブの一覧表示

現時点で登録されている全てのプローブの一覧は、`/sys/kernel/debug/kprobes/list` 内に書かれています。

```
saturn.example.com:~ # cat /sys/kernel/debug/kprobes/list
c015d71a  k  vfs_read+0x0   [DISABLED]
c011a316  j  do_fork+0x0
c03dedc5  r  tcp_v4_rcv+0x0
```

左から最初の列には、プローブを挿入したカーネル内のアドレスが書かれています。2 番目の列にはプローブの種類が書かれています (`k` (kprobe), `j` (jprobe), `r` (return probe)) 。3 番目の列はシンボルとオフセット、そしてプローブのモジュール名 (もしあれば) が書かれています。また、それ以降の列は必要に応じて現れる箇所で、プローブの状態が書かれています。挿入済みのプローブが無効なアドレスを指している場合は `[GONE]` 、対象のプローブが一時的に無効化されて居る場合は `[DISABLED]` と書かれます。

5.4.2 指定したプローブの全体的な有効化／無効化

`/sys/kernel/debug/kprobes/enabled` ファイルは、登録済みの全てのカーネルプローブに対して、一括でかつ強制的に有効化および無効化を行うスイッチになっています。全てのカーネルプローブを無効化したい場合は、`root` で下記のように入力して実行します:

```
# echo "0" > /sys/kernel/debug/kprobes/enabled
```

再度有効化したい場合は、`root` で下記のように入力して実行します:

```
# echo "1" > /sys/kernel/debug/kprobes/enabled
```

なお、この方法ではプローブの状態変更を行うことはできません。特定のプローブが一時的に無効化されている場合、後者のコマンドを入力して実行しても、`[DISABLED]` (無効) のままになります。

5.5 さらなる情報

カーネルプローブに関する詳細については、下記の情報源をご覧ください:

- カーネルプローブに関する技術寄りの情報については、`/usr/src/linux/Documentation/trace/kprobes.txt` ファイル (`kernel-source` パッケージ内に含まれています) をお読みください。
- 3 種類のプローブに関する例 (および対応する `Makefile`) については、`/usr/src/linux/samples/kprobes/` ディレクトリ内 (`kernel-source` パッケージ内に含まれています) をお読みください。
- Linux のカーネルモジュールや `printk` カーネルルーチンに関する詳細については、The Linux Kernel Module Programming Guide (https://tldp.org/LDP/lkmpg/2.6/html/lkmpg.html) (英語) 内に説明が書かれています。

6 Perf を利用したハードウエアベースの監視

改訂履歴

2023-12-22

Perf はプロセッサに搭載された性能監視ユニット (PMU) にアクセスするためのインターフェイスで、ページフォルトなどのソフトウエア全てを記録し、表示することができます。また、システム全体やスレッド単位、そして KVM の仮想化ゲストの監視にも対応しています。

収集された情報はレポート内に保存することができます。このレポートにはインストラクションポインタに関する情報のほか、その時点でスレッドが実行していたコードの内容などが含まれています。

Perf は下記の 2 種類の部品から構成されています:

- ハードウエアに対して指示を与える、Linux カーネル内に統合されたコード。
- カーネルコードを使用することができ、収集したデータの分析を支援するための `perf` ユーザスペースユーティリティ。

6.1 ハードウエアベースの監視

性能監視とは、アプリケーションやシステムがどのような処理を行ったのかについて、関連する情報を収集することを意味します。この情報はソフトウエアベースの手段のほか、CPU やチップセットなどを介して取得することができます。Perf では、これらの方式の両方を統合する仕組みです。

多くの新しいプロセッサには、性能監視ユニット (Performance Monitoring Unit (PMU)) が搭載されています。PMU の設計と機能は CPU ごとに異なります。たとえばレジスタの数や対応するカウンタ、機能などは CPU 側の実装によって異なります。

それぞれの PMU のレジスタを分類すると、性能監視設定 (Performance Monitor Configuration; PMC) と性能監視データ (Performance Monitor Data (PMD)) に分けられます。どちらのレジスタとも読み込みは可能ですが、書き込みに関しては PMC にのみ行うことができます。これらのレジスタには、設定情報とデータが記憶されています。

6.2 サンプリングとカウンティング

Perf にはいくつかのプロファイルモードが用意されています:

- **カウンティング:** イベントの発生回数を数えるモードです。
- **イベントベースのサンプリング:** あまり正確ではない計数方法です。ある一定回数のイベント発生が起こるたびにサンプルを記録するモードです。

- **時間ベースのサンプリング：** あまり正確ではない計数方法です。一定の時間間隔でサンプルを記録するモードです。
- **インストラクションベースのサンプリング (AMD64 のみ)：** プロセッサは、指定された間隔で現れるインストラクションに従い、生成されるイベントを採取します。これにより、個別のインストラクションをたどることができますので、性能面で何が問題なのかを確認することができるようになります。

6.3 Perf のインストール

Perf のカーネルコードは既定のカーネル内に含まれています。ユーザスペース側のユーティリティを使用するには、`perf` パッケージをインストールしてください。

6.4 Perf のサブコマンド

必要な情報を収集する目的で、`perf` ツールにはいくつかのサブコマンドが用意されています。本章では、最もよく使用するサブコマンドについて、概要を説明しています。

サブコマンドに対するヘルプを表示したい場合は、`perf help サブコマンド` もしくは `man perf-サブコマンド` のように入力して実行してください。

`perf stat`
: プログラムを起動し、終了時に統計情報の概要を生成します。`perf stat` はイベントの発生回数を数えるために使用します。

`perf record`
: プログラムを起動し、パフォーマンスカウンタ情報のレポートを作成します。レポートはカレントディレクトリ内の `perf.data` ファイルに保存されます。`perf record` はイベントのサンプリングで使用します。

`perf report`
: `perf record` で作成したレポートを表示します。

`perf annotate`
: レポートファイルを表示するとともに、実行されていたコードの注釈付きバージョンを表示します。デバッグシンボルがインストールされていれば、ソースコードも表示されます。

`perf list`
: 現在のカーネルと CPU で Perf がレポートすることのできる、イベントの種類を一覧表示します。イベントの種類は分類でフィルタリングすることができます。たとえばハードウエアイベントのみを表示したい場合は、`perf list hw` のように入力して実行します。

`perf_event_open` に対するマニュアルページには、最も重要なイベントに関する短い説明が書かれています。たとえば `branch-misses` というイベントに対する説明を読むには、`BRANCH_MISSES` を検索してください (大文字で、ハイフンをアンダースコアにしていることに注意してください):

```
> man perf_event_open | grep -A5 BRANCH_MISSES
```

イベントによっては名前がわかりにくいものもあります。ハードウエアイベント名のうち、小文字のものはハードウエアイベントを直接指し示すものではなく、Perf が独自に別名として作成したものであることに注意してください。このような別名定義により、対応する様々なプロセッサで名前の異なる似通ったイベントを統一的に収集できるようになっています。
たとえば `cpu-cycles` というイベントは、Intel プロセッサでは `UNHALTED_CORE_CYCLES` というハードウエアイベントにマップされていますが、AMD プロセッサでは `CPU_CLK_UNHALTED` というハードウエアイベントにマップされています。
Perf では、ハードウエア固有のイベントの計数にも対応しています。それらの説明を読みたい場合は、CPU の製造元が提供するアーキテクチャソフトウエア開発者マニュアル (Architecture Software Developer's Manual) をお読みください。AMD64/Intel 64 プロセッサでの対応文書については、6.7項「さらなる情報」にリンクがあります。

`perf top`
: 発生したシステムの動作状況を表示します。

`perf trace`
: このコマンドは `strace` に似た動作をする仕組みで、特定のスレッドやプロセスでどのシステムコールが実行されているのかと、シグナルの受信状況を表示することができます。

6.5 特定種類のイベントのカウント

イベント (`perf list` で表示されるイベント) の発生回数を数えるには、下記のように入力して実行します:

```
# perf stat -e イベント名 -a
```

複数のイベントを一括で数えたい場合は、それらをカンマ区切りで指定します。たとえば `cpu-cycles` と `instructions` をまとめて数えるには、下記のように入力して実行します:

```
# perf stat -e cpu-cycles,instructions -a
```

セッションを停止するには、`Ctrl – C` を押します。

一定時間内でのイベント発生回数を数えることもできます:

```
# perf stat -e イベント名 -a -- sleep 時間
```

`時間` の箇所は秒単位で指定してください。

6.6 特定のコマンド固有のイベント記録

特定のコマンドに固有のイベントを採取する方法には、いくつかのものがあります:

- 新しく起動したコマンドに対してレポートを作成するには、下記のようにします:

  ```
  # perf record コマンド
  ```

  起動したコマンドは通常通りに処理を行うだけです。コマンドのプロセスが終了すると、Perf のセッションも終了します。

- 新しく起動したコマンドが実行されている間、システム全体で発生したイベントのレポートを作成するには、下記のようにします:

  ```
  # perf record -a コマンド
  ```

  起動したコマンドは通常通りに処理を行うだけです。コマンドのプロセスが終了すると、Perf のセッションも終了します。

- 動作中のプロセスに対するレポートを作成するには、下記のようにします:

  ```
  # perf record -p PID
  ```

  ここで、`PID` にはプロセス ID を指定します。セッションを停止するには、 `Ctrl` – `C` を押します。

収集されたデータ (`perf.data`) を表示するには、下記のように入力して実行します:

```
> perf report
```

すると、擬似的なグラフィカルインターフェイスが起動されます。ヘルプを表示したい場合は `H` を、終了したい場合は `Q` をそれぞれ押してください。

グラフィカルインターフェイスを使用したい場合は、Perf の GTK+ インターフェイスをお試しください:

```
> perf report --gtk
```

ただし、GTK+ インターフェイスの機能は限られたものであることに注意してください。

6.7 さらなる情報

本章では非常に短い説明しか記載していません。下記に示すリンク先で様々な情報が提供されています (英語のみの提供となります):

https://perf.wiki.kernel.org/index.php/Main_Page
: プロジェクトの Web ページです。`perf` を使用する際のチュートリアルも用意されています。

https://www.brendangregg.com/perf.html
: `perf` の様々な使用例を含む、非公式ページです。

https://web.eece.maine.edu/~vweaver/projects/perf_events/
: 様々なりソースを含む非公式ページです。主に Perf の Linux カーネルコードと、その API に関連した情報が書かれています。たとえば CPU の互換性テーブルやプログラミングガイドなどがあります。

https://www-ssl.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf
: Intel Architectures Software Developer's Manual, Volume 3B です。

https://support.amd.com/TechDocs/24593.pdf
: AMD Architecture Programmer's Manual, Volume 2 です。

第7章「OProfile: システム全体に対するプロファイラ」
: 性能を最適化したい場合は、この章もお読みください。

7 OProfile: システム全体に対するプロファイラ

改訂履歴

2023-12-22

OProfile は動的なプログラム分析のためのプロファイラです。動作中のプログラムに対する振る舞いを調査して、情報を収集します。この情報は表示することができるため、さらなる最適化のためのヒントとすることができます。

OProfile を使用するにあたって、再コンパイルやラッパーライブラリの使用は必要ありません。カーネルに適用すべきパッチもありません。負荷とサンプリング周期にもよりますが、アプリケーションをプロファイルしている間、少しだけオーバーヘッドが現れます。

7.1 考え方の概要

OProfile はカーネルドライバとデーモンから構成され、データを収集することができます。また、多くのプロセッサに搭載されている、ハードウエア側のパフォーマンスカウンタを利用して測定を行います。OProfile はカーネルやカーネルモジュール、カーネルの割り込みハンドラやシステムの共有ライブラリ、その他のアプリケーションなど、全てのコードをプロファイルすることができます。

新しいプロセッサであれば、パフォーマンスカウンタと呼ばれるハードウエアを介してプロファイルを行うことができます。プロセッサ側の使用にも依存しますが、様々なカウンタが用意され、それらはイベントの発生回数をカウントするようプログラムすることができます。また、それぞれのカウンタには、どれくらいの間隔でサンプルを採取したのかを示す値が用意されています。値が低いほど多く使用されたことになります。

なお、事後の作業で全ての情報を収集し、インストラクションアドレスは関数名に変換されます。

7.2 インストールと要件

OProfile を使用するには `oprofile` パッケージをインストールします。

なお、プロファイル対象のアプリケーションに対応する `*-debuginfo` パッケージをインストールしておくことをお勧めします。カーネルをプロファイルする場合は、カーネルの `debuginfo` パッケージもインストールしてください。

7.3 利用可能な OProfile ユーティリティ

OProfile にはプロファイルを行ったりそのデータを収集したりする目的で、いくつかのユーティリティが提供されています。下記の一覧には、本章で使用するプログラムの概要を説明しています:

`opannotate`
: 注釈付きソースやアセンブリリストとプロファイル情報を出力するためのユーティリティです。注釈付きのレポートは `addr2line` と組み合わせて使用され、ホットスポットが存在しうるソースコードのファイルとその行を識別することができます。詳しくは `man addr2line` をお読みください。

`operf`
: プロファイラツールです。プロファイリングを停止すると、既定では `カレントディレクトリ/oprofile_data/samples/current` にデータを保存しますので、ここから `opreport` コマンドなどで処理できるようになります。

`ophelp`
: 利用可能なイベントと、その短い説明を一覧表示します。

`opimport`
: 異なるバイナリ形式で書かれたサンプルデータベースファイルを、ネイティブ形式に変換します。

`opreport`
: プロファイルしたデータからレポートを生成します。

7.4 OProfile の使用

OProfile を使用することで、カーネルとアプリケーションの両方をプロファイルすることができます。カーネルをプロファイルする場合は、OProfile に対して `vmlinuz*` ファイルの検索先を指定するため、`--vmlinux` オプションで場所を指定します (一般に `/boot` です) 。カーネルモジュールをプロファイルする必要がある場合も、OProfile は自動的にそれを行います。ただし、念のため https://oprofile.sourceforge.net/doc/kernel-profiling.html (英語) をお読みになることをお勧めします。

アプリケーションをプロファイルする際、ほとんどの場合においてカーネルのプロファイルは不要になります。そのため、`--no-vmlinux` オプションを指定して情報量を減らしておくことをお勧めします。

7.4.1 レポートの作成

アプリケーション `COMMAND` に対するプロファイルを行うため、デーモンを起動してデータを収集し、デーモンを停止させたあとレポートを作成するまでの手順を説明します。

1. シェルを開いて `root` になります。
2. Linux カーネルを含めてプロファイルするかどうかを判断し、それぞれを行います:

   a. **Linux カーネルを利用したプロファイル:** `operf` は無圧縮のイメージのみを扱うことができますので、下記のようにコマンドを入力して実行します:

   ```
   > cp /boot/vmlinux-`uname -r`.gz /tmp
   > gunzip /tmp/vmlinux*.gz
   > operf--vmlinux=/tmp/vmlinux* COMMAND
   ```

   b. **Linux カーネルを利用しないプロファイル:** 下記のようにコマンドを入力して実行します:

   ```
   # operf --no-vmlinux COMMAND
   ```

   出力内に一方の関数から他方の関数を呼び出している様子を表示させたい場合は、`--callgraph` を指定して最大の `深さ` を設定してください:

   ```
   # operf --no-vmlinux --callgraph
   深さ COMMAND
   ```

3. `operf` はデータを `カレントディレクトリ/oprofile_data/samples/current` に書き込みます。`operf` コマンドが終了したら (もしくは `Ctrl`–`C` で停止させたら) 、`oreport` を利用して解析を行うことができます:

```
# opreport
Overflow stats not available
CPU: CPU with timer interrupt, speed 0 MHz (estimated)
Profiling through timer interrupt
          TIMER:0|
  samples|      %|
------------------
    84877 98.3226 no-vmlinux
...
```

7.4.2 イベント設定の取得

イベントを設定するための一般的な手順は下記のとおりです:

1. まずは `CPU-CLK_UNHALTED` と `INST_RETIRED` の各イベントを利用して、最適化の可能性を探ります。
2. 特定のイベントを利用してボトルネックを探ります。イベントの一覧を表示するには、`perf list` と入力して実行してください。

特定のイベントをプロファイルする必要がある場合は、まずお使いのプロセッサ側がそのイベントに対応しているかどうかを調べる必要があります。具体的には、`ophelp` コマンドを実行します (下記は Intel Core i5 CPU での出力例です):

```
# ophelp
oprofile: available events for CPU type "Intel Architectural Perfmon"

See Intel 64 and IA-32 Architectures Software Developer's Manual
Volume 3B (Document 253669) Chapter 18 for architectural perfmon events
This is a limited set of fallback events because oprofile does not know your CPU
CPU_CLK_UNHALTED: (counter: all))
        Clock cycles when not halted (min count: 6000)
INST_RETIRED: (counter: all))
        number of instructions retired (min count: 6000)
LLC_MISSES: (counter: all))
        Last level cache demand requests from this core that missed the LLC (min count:
 6000)
        Unit masks (default 0x41)
        ----------
        0x41: No unit mask
LLC_REFS: (counter: all))
        Last level cache demand requests from this core (min count: 6000)
        Unit masks (default 0x4f)
        ----------
        0x4f: No unit mask
BR_MISS_PRED_RETIRED: (counter: all))
        number of mispredicted branches retired (precise) (min count: 500)
```

`--event` オプションでパフォーマンスカウンタイベントを指定します。複数のオプションを指定してもかまいません。このオプションはイベント名 (`ophelp` で表示される名前) とサンプリングレートを指定します。たとえば下記のようになります:

```
# operf --events CPU_CLK_UNHALTED:100000
```

警告: CPU_CLK_UNHALTED に対するサンプリングレート設定について

サンプリングレートを低く設定しすぎてしまうと、システムの性能を著しく低下させてしまう危険性がありますし、逆にサンプリングレートを高く設定してしまうと、データの意味が無くなってしまうほどシステムの負荷が重くなります。そのため、OProfile を入れることで性能を落とすことなく、かつ負荷が高くなりすぎないような値を探り、設定するようにしてください。

7.5 レポートの生成

レポートを生成する前に、まずは `operf` が停止していることを確認してください。また、`--session-dir` でデータのディレクトリを指定していない場合、`operf` はデータを カレントディレクトリ/oprofile_data/samples/current に書き込みます。なお、`opreport` や `opannotate` のようなレポートツールは、既定で上述のディレクトリを使用します。

何もパラメータを指定しないで `opreport` を実行すると、詳しい説明を表示することができます。オプションとして実行ファイル名を指定すると、その実行ファイルからのプロファイルデータのみを取得します。また C++ 言語で書かれたアプリケーションを解析したい場合は、`--demangle smart` オプションを指定してください。

`opannotate` はソースコードからの注釈情報を生成します。下記のようなオプションを指定して実行してください:

```
# opannotate --source \
   --base-dirs=ベースディレクトリ \
   --search-dirs=検索ディレクトリ \
   --output-dir=annotated/ \
   /lib/libfoo.so
```

`--base-dir` には、デバッグソースファイルから削除されたパス情報をカンマ区切りで指定します。ここで指定したパスは `--search-dirs` で指定したパスよりも優先して検索されます。また、`--search-dirs` には、ソースファイルを検索するパスをカンマ区切りで指定します。

注記: 注釈付きソースの不正確性について

コンパイラの最適化の仕組みによってコードが削除されることがあるほか、別の場所に現れたりすることがあります。詳しい仕組みについては、https://oprofile.sourceforge.net/doc/debug-info.html (英語) をお読みください。

7.6 さらなる情報

本章では短い概要しか説明していません。詳しい情報を得るには、下記のリンク先を参照してください (いずれも英語のみの提供です):

https://oprofile.sourceforge.net
: プロジェクトの Web ページです。

マニュアルページ
: 様々なツールのオプションに関する詳しい説明が書かれています。

`/usr/share/doc/packages/oprofile/oprofile.html`
: OProfile のマニュアルが配置されています。

https://developer.intel.com/
: Intel プロセッサ向けのアーキテクチャリファレンスです。

8 ダイナミックデバッグ: カーネルのデバッグメッセージの調整

改訂履歴

2024-06-25

ダイナミックデバッグは Linux カーネル内に用意されているパワフルなデバッグ機能で、カーネルの再コンパイルやシステムの再起動を実施することなく、動的にデバッグメッセージの有効化や無効化を実施できる仕組みです。

ダイナミックデバッグは下記のような状況で使用します:

- カーネル内部の問題の調査
- 新しいハードウエアに対応するためのドライバ開発
- セキュリティイベントの追跡や監査

8.1 ダイナミックデバッグの利点

ダイナミックデバッグを使用することで、下記のような利点をもたらします:

リアルタイムなデバッグ
: ダイナミックデバッグはシステムの再起動を行わずにデバッグメッセージを有効化できます。そのため、本番環境下で問題を調査する場合など、性能を損なってはならないような場合に有用です。

選択的なデバッグ
: デバッグメッセージの有効化をカーネル内のパーツ単位や個別のモジュール単位で設定することができます。これにより、必要な情報のみに絞って出力することができます。

性能のチューニング
: 現在の解析要件に合わせてデバッグメッセージの有効／無効を切り替えることで、ダイナミックデバッグをカーネルの性能監視や最適化に使用することもできます。

8.2 ダイナミックデバッグの状態確認

既定でダイナミックデバッグ機能の搭載されたカーネルがインストールされているはずですが、念のためダイナミックデバッグ機能が搭載されているかどうかを確認するには、下記のようなコマンドを root で実行します:

```
# zcat /proc/config.gz | grep CONFIG_DYNAMIC_DEBUG
```

ダイナミックデバッグ機能が組み込まれていれば、下記のような出力が現れるはずです:

```
CONFIG_DYNAMIC_DEBUG=y
CONFIG_DYNAMIC_DEBUG_CORE=y
```

8.3 ダイナミックデバッグの使用

動作中のカーネルに対して、特定のデバッグメッセージやログを有効化したい場合は、`echo` コマンドで `/sys/kernel/debug/dynamic_debug/control` ファイルに書き込みを行います。

ダイナミックデバッグの簡単な使用手順は、下記のようになります:



注記

ダイナミックデバッグの機能は、カーネルのソースコード内に組み込まれた `pr_debug` 等の特定のデバッグマクロに依存しています。これらのマクロはカーネルの開発者に対して、コード内でのデバッグメッセージ出力に使用されています。

本章内での例は、動作中のカーネルでダイナミックデバッグが既に有効化されているはずであることから、`pr_debug` マクロが正しく動作していることを前提にしています。

特定のカーネルソースコードファイルに対するデバッグメッセージの有効化

たとえば特定のカーネルのソースコードファイル内にあるデバッグメッセージを有効化するには、下記のように入力して実行します:

```
# echo "file ファイル名.c +p" > /sys/kernel/debug/dynamic_debug/control
```

特定のカーネルモジュールに対するデバッグメッセージの有効化

特定のカーネルモジュール内にあるデバッグメッセージを有効化するには、下記のように入力して実行します:

```
# echo "module モジュール名 +p" > /sys/kernel/debug/dynamic_debug/control
```

デバッグメッセージの無効化

特定のカーネルのソースコードやカーネルモジュールに対して有効化したデバッグメッセージを無効化したい場合は、`echo` コマンドで `-p` オプションを指定します。たとえば下記のようになります:

```
# echo "file ファイル名.c -p" > /sys/kernel/debug/dynamic_debug/control
```

```
# echo "module モジュール名 -p" > /sys/kernel/debug/dynamic_debug/control
```

ダイナミックデバッグに関する詳細な情報や使用例などについては、公式文書 (英語) (https://www.kernel.org/doc/html/latest/admin-guide/dynamic-debug-howto.html) をお読みください。

8.4 ダイナミックデバッグメッセージの表示

上記までの手順で有効化し、その後に出力されたデバッグメッセージを表示するには、`dmesg` コマンドを使用します。なお、`grep` コマンドで出力をフィルタすると、より読みやすくなります:

```
# dmesg | grep -i "ファイル名.c"
```

また、生成されたメッセージを継続的に追跡したい場合は、`tail` コマンドに `-f` オプションを指定して実行します:

```
# tail -f /var/log/messages
```

IV リソース管理

9 一般的なシステムリソースの管理 103

10 カーネルコントロールグループ 109

11 自動的な Non-Uniform Memory Access (NUMA) のバランス調整 118

12 電源管理 123

9 一般的なシステムリソースの管理

改訂履歴

2023-12-22

システムのチューニングはカーネルの最適化やアプリケーションの活用だけでなく、無駄のない高速なシステムを構築するところから始まります。たとえばパーティションやファイルシステムの設定によっても、サーバの速度は速くも遅くもなります。また動作させるサービスや定期的に実行されるスケジュールタスクなどについても、性能への影響があります。

9.1 インストールの計画

よく注意して計画し、インストールを行うことで、特定の目的に正確に適合したシステムを構築することができるようになります。そこからシステムのチューニングを行う場合であっても、かかる時間を削減することに繋がります。本章で示している全ての変更点は、インストール時の［インストール設定］の手順内で実施することができます。詳しくは**『スタートアップ』、第3章「インストール手順」、3.11項「インストール設定」**をお読みください。

9.1.1 パーティション設定

サーバの用途とハードウエアの配置にもよりますが、パーティション方式がマシンの性能そのものに影響を与えることがあります。用途やハードウエアの配置によるパーティション方式の選択は、本章の範疇ではありませんので詳しく説明はしていません。ただし、一般的に下記のようなルールでパーティション方式を選択すると、よりよい性能を得られます。なお、外付けのストレージシステムを使用する場合は、下記のルールには当てはまりません。

- ディスク領域を最大限に使用した場合であっても、少しだけ空き容量が残るように設計してください。ディスク領域が完全に埋まってしまうと、明らかな性能劣化が発生します。
- 複数のディスクに対して、読み込みや書き込みの処理が分散されるように設計してください。たとえば下記のようなことが考えられます:
    - オペレーティングシステムとデータ、ログファイルのディスクをそれぞれ別々のものにする
    - メールサーバのスプールディレクトリを別のディスクに分ける
    - 複数のディスクに対して、ユーザのホームディレクトリのディスクを分ける

9.1.2 インストール範囲

インストールするパッケージの選択は、マシンの性能に直接影響するものではありませんが、注意深くパッケージを選択することで、いくつかの利点が発生します。また、サーバを構築するにあたっては、必要な最小限のパッケージのみをインストールするようにしてください。パッケージの数を減らしておけばサーバの管理の手間を省くことができるだけでなく、潜在的なセキュリティ問題も減らすことができます。これに加えて、不要なパッケージをインストールしなければ、不要なサービスが動作してしまうようなこともありません。

openSUSE Leap では、インストール概要の画面でインストールするパッケージを選択することができます。既定では作業範囲からあらかじめ設定されたパターンを選択し、インストールを行うことができますが、YaST のソフトウエアマネージャを起動して、より細かくインストールするパッケージを選択することもできます。

よくある不要なパターンには、たとえば下記のようなものがあります:

[GNOME デスクトップ環境]
: サーバを動作させるだけであれば、完全なデスクトップ環境は不要です。どうしてもグラフィカル環境が必要な場合は、IceWM などのより経済的なソリューションをお使いになることをお勧めします。

[X Window System]
: サーバを単独で管理しなければならない場合で、アプリケーションをコマンドラインから制御できる場合は、このパターンをインストールする必要はありません。ただし、リモートのマシンから GUI アプリケーションを起動して管理するような場合は、このパターンが必要となることに注意してください。また、お使いのアプリケーションが GUI のみで管理できるような場合や、YaST の GUI 版が必要である場合は、このパターンをインストールしてください。

[印刷サーバ]
: このパターンは、そのマシンから印刷を行いたい場合にのみ選択すべきです。

9.1.3 既定のターゲット

X Window System を動作させてしまうと、サーバではあまり必要とされないにもかかわらず、多くのリソースを消費することになってしまいます。お使いのシステムをサーバとして使用する場合、`multi-user.target` ([マルチユーザシステム]) を `systemd` の既定のターゲットに指定しておくことをお勧めします。この場合でも、リモートからグラフィカルなアプリケーションを起動することができます。

9.2 不要なサービスの無効化

既定の手順でインストールを行うと、いくつかのサービスを開始するように設定してしまいます (具体的にどのようなサービスであるのかは、インストール時のパターン選択やパッケージ選択によって異なります) 。それぞれのサービスはリソースを消費してしまうものですので、不要なサービスについては止めておくことをお勧めします。YaST のサービス管理モジュールを起動するには、［YaST］›［システム］›［サービスマネージャ］を選択します。

YaST のグラフィカル版をお使いの場合は、列ヘッダの部分を押すことで、並べ替えを行うことができます。これを利用して、サービスの動作状況を確認してください。次に再起動を行うまでの間、サービスを無効化しておきたい場合は、サービスを選んで［停止］を押します。恒久的に無効化したい場合は、［開始モード］内の［手動］を選択します。

下記の一覧には、openSUSE Leap のインストール直後に既定で開始されるサービスの一覧を示しています。各サービスの説明をお読みのうえ、不要であれば無効化してください:

［alsasound］
: Advanced Linux Sound System (Linux でのサウンドシステム) を読み込みます。

［auditd］
: 監査システム (詳しくは『**セキュリティ強化ガイド**』をお読みください) 向けのデーモンです。監査を必要としない場合は、無効化してください。

［bluez-coldplug］
: Bluetooth ドングルのコールドプラグを処理します。

［cups］
: プリンタデーモンです。

［java.binfmt_misc］
: `*.class` や `*.jar` というファイル名の Java プログラムを実行できるようにします。

［nfs］
: NFS をマウントする際に必要なサービスです。

［smbfs］
: Windows* サーバ内の SMB/CIFS ファイルシステムをマウントする際に必要となります。

［splash / splash_early］
: 起動時にスプラッシュスクリーン (ロゴの表示) を行います。

9.3 ファイルシステムとディスクアクセス

ハードディスクはコンピュータシステム内で最も遅いコンポーネントであるため、性能面ではよくボトルネックになります。お使いの用途に適したファイルシステムを使用することで、性能を改善できる可能性があります。また、特殊なマウントオプションを設定したり、プロセスの I/O 優先順位を設定したりすることで、システムの速度をさらに向上させることができる場合があります。

9.3.1 ファイルシステム

openSUSE Leap には btrfs, ext4, ext3, ext2, xfs など、様々なファイルシステムが用意されています。それぞれのファイルシステムには利点と欠点がそれぞれ存在しています。

9.3.1.1 NFS

NFS (バージョン 3) のチューニングについては、https://nfs.sourceforge.net/nfs-howto/ にある NFS Howto (英語) に詳細が書かれています。なお、NFS で共有されているファイルシステムをマウントする場合、最初に試すべきことは、`wsize` と `rsize` の各マウントオプションで、読み込みと書き込みのブロックサイズを `32768` に増やしてみることです。

9.3.2 タイムスタンプの更新ポリシー

ファイルシステム内のファイルやディレクトリには、3 種類のタイムスタンプが設定されています。対象のファイルやディレクトリを最後に読み込んだ日時を表す アクセス日時, 最後に修正した日時を表す 更新日時, そしてメタデータを最後に修正した日時を表す 変更日時 です。このうちアクセス日時については、対象のファイルやディレクトリを読み込んだだけで書き込みを行う必要が生まれてしまうことから、著しい性能オーバーヘッドを与える結果になってしまいます。そのため既定では、その時点までのアクセス日時が 1 日以上古い場合か、もしくは更新日時や変更日時よりも古い場合にのみ、ファイルシステムのアクセス日時の更新を行うように設定されています。この機能は relative access time (相対アクセス日時) と呼ばれ、マウント時のオプションでは `relatime` という名称が設定されています。アクセス日時の更新を完全に無効化したい場合は、`noatime` というオプションを設定しますが、こちらはアプリケーション側に悪影響が出ないかどうかをご確認のうえ、設定してください。特にファイルサーバや Web サーバ、ネットワークストレージサーバなどに使用する場合は、影響が発生することがあります。また、既定で設定される相対アクセス日時の機能についても、アプリケーション側に影響があることがありますので、そのような場合は、`strictatime` オプションを指定してください。

ファイルシステムによっては (ext4 などがこれにあたります) 、lazy time stamp updates という機能に対応しているものがあります。この機能はマウントオプションで `lazytime` という名前が設定されていて、タイムスタンプの更新をメモリ内にのみ保持し、ディスクには書き込まないようにします。ディスクへの書き込みは、`fsync` や `sync` のシステムコールが発生した場合や、ファイルサイズの更新などでファイルの情報を書き込まなければならない場合、もしくはタイムスタンプが 24 時間以上古い場合や、ファイル情報をメモリから掃き出す必要がある場合にのみ、行われます。

ファイルシステムに対するマウントオプションを変更するには、`/etc/fstab` ファイルを直接編集するか、もしくは YaST のパーティション設定モジュール内のマウント設定で、［fstab オプション］の欄に指定を行ってください。

9.3.3 `ionice` によるディスクアクセスの優先順位設定

`ionice` コマンドは 1 つのプロセスに対してディスクアクセスの優先順位を設定するためのコマンドです。この仕組みにより、バックアップジョブなどのように、ディスクアクセスを頻繁に行うものの、時間的な制約がないようなプログラムに対して、他のプログラムへの影響を小さくすることができます。`ionice` ではその逆に、ディスクへのアクセスを即時に行わせるよう、プログラムの優先順位を上げることもできます。ただし一般的な書き込み処理の場合、ページキャッシュにのみ書き込まれ、ディスクへの書き込みは別途のカーネルプロセスが行うことに注意してください。この場合、I/O 優先順位の設定が適用されなくなってしまいます。また、I/O クラスと優先順位の設定は、blk-mq I/O パス (詳しくは 13.2 項「blk-mq I/O パスで利用可能な I/O エレベータ」をお読みください) 向けの BFQ I/O スケジューラの場合にのみ効果があることにも注意してください。スケジューリングクラスとしては、下記の 3 種類を設定することができます:

アイドル

このクラスが設定されたプロセスは、その他のプロセスがディスク I/O を行っていない場合にのみ、ディスクへのアクセスが許可されるようになります。

ベストエフォート

こちらが既定のスケジューリングクラスで、特段の I/O 優先順位を指定しない場合の優先順位になります。このクラスが設定されたプロセスに対しては、さらに細かく `0` から `7` までのレベル (`0` が最も高い優先順位になります) を設定することができます。同じベストエフォートの優先順位が設定されたプロセス同士では、ラウンドロビン形式で I/O が割り当てられることになります。なお、カーネルのバージョンによっては、ベストエフォートのクラスの扱いが異なる場合があります。詳しくは `ionice(1)` のマニュアルページをお読みください。

リアルタイム

このクラス内のプロセスの場合、ディスクへのアクセスは常に取得することができます。この場合も、`0` から `7` までのレベル (`0` が最も高い優先順位) を設定することができます。なお、他のプロセスの動作を妨害する可能性があり得ることから、注意してお使いください。

詳細や正確なコマンド書式については、`ionice(1)` のマニュアルページをお読みください。各アプリケーションに対して帯域を精密に制御する必要がある場合は、第10章「カーネルコントロールグループ」で説明しているカーネルコントロールグループを使用してください。

10 カーネルコントロールグループ

改訂履歴

2024-06-25

カーネルコントロールグループ (「cgroups」) は、プロセスに対してハードウエアやシステムの資源を割り当てたり、制限したりするための仕組みです。この機能を利用することで、プロセスをツリー構造で管理することができるようになります。

10.1 概要

それぞれのプロセスは正確に 1 つの管理用 cgroup に割り当てられます。cgroup は階層構造型のツリー (木構造) として管理するもので、その構造の任意の箇所 (枝) もしくは 1 つのプロセスに対して、CPU やメモリ、ディスクの I/O やネットワーク帯域などのリソース制限を割り当てます。

openSUSE Leap では、`systemd` が cgroup を利用してグループ内の全てのプロセスを管理しています。この場合、`systemd` はグループをスライスと呼んでいます。`systemd` には、cgroup の設定を行うためのインターフェイスも用意されています。

`systemd-cgls` コマンドでは、階層構造を表示することができます。

カーネルが提供する cgroup の API には v1 と v2 と呼ばれる 2 種類のものが存在しています。それに加えて、異なる API を提供する複数の cgroup 階層構造が存在しています。これらの組み合わせのうち、一般的に使用される組み合わせは下記の 2 種類になります:

- 統合型: コントローラを含めて v2 階層構造を使用する構成
- ハイブリッド型: コントローラ以外は v2, コントローラは v1 の階層構造を使用する構成 (廃止予定)

既定のモードは統合型です。アプリケーション側の要件に応じて後方互換性を提供するハイブリッド型もあります。

いずれかのモードのみを設定できます。

10.1.1 ハイブリッド型 cgroup 階層構造



注記: 廃止予定について

cgroup v1 は廃止予定になっています。将来のバージョンで削除される予定です。

ハイブリッド型のコントロールグループ階層構造を有効化したい場合は、GRUB 2 ブートローダの設定で、カーネルのコマンドラインパラメータに `systemd.unified_cgroup_hierarchy=0` を追加してください。GRUB 2 の設定方法に関する詳細は、『リファレンス』、第12章「ブートローダ GRUB 2」をお読みください。

10.2 リソースアカウンティング

複数の cgroup にプロセスをまとめることで、cgroup ごとの資源消費データを取得できるようになります。

このような機能をアカウンティングと呼びますが、この機能自身にも小さいながらオーバーヘッドが存在しています。このオーバーヘッドはその中での処理内容にも依存しますが、特定の 1 つのユニットに対してアカウンティングを有効にすると、同じスライスに含まれる全てのユニットだけでなく、親スライスやそこに直接含まれるユニットに対しても、この機能が有効化されることに注意してください。

ユニット単位でアカウンティングを有効化したい場合は `MemoryAccounting=` のようなディレクティブを使用することができます。全てのユニットに対して有効化したい場合は、`/etc/systemd/system.conf` ファイル内の `DefaultMemoryAccounting=` ディレクティブを設定してください。設定可能なディレクティブに関する詳細は、`man systemd.resource-control` で表示されるマニュアルページ (英語) をお読みください。

10.3 リソース制限の設定



注記: 暗黙のリソース消費について

暗黙のうちに消費され、実行環境によって異なるリソースが存在することに注意してください。これにはたとえば、ライブラリやカーネル内のデータ構造のほか、利用しているユーティリティの fork() 処理の振る舞い、計算の効率性などがあります。このようなことから、実行環境を変えた場合は、リソース制限を再計算する必要があります。

cgroup に対する制限は、`systemctl set-property` コマンドで設定します。書式は下記のとおりです:

```
# systemctl set-property [--runtime] 名前 プロパティ_1=値 [プロパティ_2=値]
```

設定値は即時に適用されます。なお、必要であれば `--runtime` オプションを指定することもできます。このオプションを指定すると、再起動後には指定した制限が適用されなくなります。

また、名前 には systemd のサービス名やスコープ名、もしくはスライス名を指定します。

プロパティの一覧と詳細については、man systemd.resource-control で表示されるマニュアルページをお読みください。

10.4 TasksMax を利用した fork ボムの防止

systemd では、ユニットごとやスライスごとにタスク数の制限を設定することができます。systemd の提供元では、ユニットごとのタスク数制限の既定値が設定されています (カーネル全体での制限 (詳しくは /usr/sbin/sysctl kernel.pid_max を参照) に対して 15% に設定しています) 。また、各ユーザのスライスはカーネル全体での制限の 33% になっています。ただし、openSUSE Leap では異なる設定になっています。

10.4.1 現時点での既定の TasksMax 値の検出

しかしながら、全ての用途に対して単一の制限を適用するのは現実的ではありません。openSUSE Leap では、システムユニットやユーザスライスに対する提供元の既定値を上書きするための独自設定ファイルが 2 つ用意され、いずれも infinity に設定されています。/usr/lib/systemd/system.conf.d/__20-defaults-SUSE.conf には、下記のような設定が書かれています:

```
[Manager]
DefaultTasksMax=infinity
```

もう 1 つの存在である /usr/lib/systemd/system/user-.slice.d/10-defaults.conf には、下記のような設定が書かれています:

```
[Slice]
TasksMax=infinity
```

DefaultTasksMax の値を確認するには、systemctl を下記のように入力して実行します:

```
> systemctl show --property DefaultTasksMax
DefaultTasksMax=infinity
```

infinity は無制限の意味です。特に要件がなければ既定値を変更する必要はありませんが、システムのクラッシュを防ぐために必要であれば設定を変更してください。

10.4.2 DefaultTasksMax 値の設定

グローバルな `DefaultTasksMax` の値を変更したい場合は、設定を上書きするための新しい設定ファイル `/etc/systemd/system.conf.d/90-system-tasksmax.conf` を作成して対応してください。この設定ファイルには、下記のような内容を記述します (下記の例では、systemd のユニットごとに最大 256 個までのタスク制限を設定します):

```
[Manager]
DefaultTasksMax=256
```

新しい設定を読み込んで、設定が反映されたことを確認します:

```
> sudo systemctl daemon-reload
> systemctl show --property DefaultTasksMax
DefaultTasksMax=256
```

設定値はお使いのシステムの要件に合わせて指定してください。また、特定のサービスに限定して制限を高くすることもできます。たとえば MariaDB で設定を変更したい場合、まずは現在の設定値を確認します:

```
> systemctl status mariadb.service
  • mariadb.service - MariaDB database server
   Loaded: loaded (/usr/lib/systemd/system/mariadb.service; disabled; vendor preset>
   Active: active (running) since Tue 2020-05-26 14:15:03 PDT; 27min ago
     Docs: man:mysqld(8)
           https://mariadb.com/kb/en/library/systemd/
 Main PID: 11845 (mysqld)
   Status: "Taking your SQL requests now..."
    Tasks: 30 (limit: 256)
   CGroup: /system.slice/mariadb.service
           └─11845 /usr/sbin/mysqld --defaults-file=/etc/my.cnf --user=mysql
```

Tasks 以下には現在動作中のタスク数 (30 個) と上限 (256 個) が示されています。負荷の高いデータベースシステムとしては不十分な値であることから、たとえば MariaDB のみを 8192 個までに拡大してみることにします。

```
> sudo systemctl set-property mariadb.service TasksMax=8192
> systemctl status mariadb.service
• mariadb.service - MariaDB database server
   Loaded: loaded (/usr/lib/systemd/system/mariadb.service; disabled; vendor preset:
 disab>
  Drop-In: /etc/systemd/system/mariadb.service.d
           └─50-TasksMax.conf
   Active: active (running) since Tue 2020-06-02 17:57:48 PDT; 7min ago
     Docs: man:mysqld(8)
           https://mariadb.com/kb/en/library/systemd/
  Process: 3446 ExecStartPre=/usr/lib/mysql/mysql-systemd-helper upgrade (code=exited,
 sta>
```

```
 Process: 3440 ExecStartPre=/usr/lib/mysql/mysql-systemd-helper install (code=exited,
sta>
Main PID: 3452 (mysqld)
  Status: "Taking your SQL requests now..."
   Tasks: 30 (limit: 8192)
  CGroup: /system.slice/mariadb.service
          └─3452 /usr/sbin/mysqld --defaults-file=/etc/my.cnf --user=mysql
```

`systemctl set-property` コマンドは、`/etc/systemd/system/mariadb.service.d/50-TasksMax.conf` という名前の上書き用設定ファイルを作成して、新しい制限を設定します。ここには既存のユニットファイルに対する上書き値のみを保存します。もちろん 8192 でなくてもかまいません。お使いのシステムの負荷状況に合わせて設定してください。

10.4.3 ユーザに対する既定の TasksMax 制限

ユーザに対する既定の制限値は高めに設定されています。これは、ユーザセッションではより多くのリソースを必要とするためです。独自の制限を設定したい場合は、`/etc/systemd/system/user-.slice.d/40-user-taskmask.conf` のような設定ファイルを作成し、その中に設定値を記述してください。下記の例では、タスクの最大値を 16284 に設定しています:

```
[Slice]
TasksMax=16284
```



注記: ファイル名の冒頭に付与する数値について

上書き用の設定ファイルを作成する場合、そのファイル名の冒頭には数値を指定する必要があります。その数値の設定方法に関する詳細は、『リファレンス』、第10章「systemd デーモン」、10.5.3項「手作業によるドロップイン・ファイルの作成」をお読みください。

あとは systemd に対して設定値の再読み込みを指示し、設定が変更されたことを確認します:

```
> sudo systemctl daemon-reload
> systemctl show --property TasksMax user-1000.slice
TasksMax=16284
```

具体的にどのような設定値にすべきかについては、システムの用途と搭載されているリソースのほか、他のリソース設定によっても異なります。`TasksMax` の値が少なすぎる場合は Failed to fork (Resources temporarily unavailable) (fork に失敗した (リソースが一時的に利用できなくなっている)) や Can't create thread to handle new connection (新しい接続を処理するためのスレッドが作成できない), Error: Function call 'fork' failed with error code 11, 'Resource temporarily unavailable' (エラーコード 11 (リソースが一時的に利用できなくなっている) で fork の関数呼び出しが失敗した) などのエラーが発生します。

systemd でのシステムリソースの制限の設定方法について、詳しくは `systemd.resource-control(5)` をお読みください。

10.5 cgroups と I/O 制御の併用

本章では、Linux カーネルのブロック I/O コントローラに対して、I/O 操作の優先順位を設定したり、負荷制限を行ったりするための方法を説明しています。この仕組みにより、systemd が提供する cgroup の仕組みをさらに効果的なものにすることができますが、I/O の制御によって陥りやすい落とし穴もまた存在しています。

10.5.1 事前要件

ここではまず、システムを設計したり設定したりする前の準備について説明しています。これらはいずれも、動作中に変更できるようなものではないためです。

10.5.1.1 ファイルシステム

まずは cgroup のライトバック機能に対応したファイルシステムを使用する必要があります (対応したファイルシステムを使用しないと、ライトバックチャージングと呼ばれる機能が使用できないためです) 。openSUSE Leap では下記のファイルシステムに対してサポートを提供しています:

- Btrfs (v4.3)
- Ext4 (v4.3)
- XFS (v5.3)

openSUSE Leap 15 .3 の時点では、上記のファイルシステムのいずれかを使用することができます。

10.5.1.2 ブロック I/O スケジューラ

負荷制御のポリシーはスタック内の高い箇所で実装されているため、それ以外の調整は特に行う必要はありません。また、I/O 制御のポリシーとしては、BFQ とコストベースモデルの 2 つの実装が提供されていますが、ここでは BFQ のみを説明しています。これは、特定のデバイスに対して負荷制御を行う場合、スケジューラとして BFQ を使用するのが最適であるためです。まずは現時点で使用しているスケジューラを判断します:

```
> cat /sys/class/block/sda/queue/scheduler
mq-deadline kyber bfq [none]
```

スケジューラを BFQ に変更します:

```
# echo bfq > /sys/class/block/sda/queue/scheduler
```

ここではパーティションではなく、ディスクデバイスを指定しなければなりません。また、この設定を適用するのに適切な方法は、udev のルール設定です。ただし openSUSE Leap の場合、回転型のディスクドライブに対しては既に BFQ が有効化されていることに注意してください。

10.5.1.3 cgroup の階層構造の仕組み

通常、全ての処理は階層構造内のルート (根幹) に位置し、互いに競合する関係になります。処理が cgroup の階層構造内に配布されると、兄弟姉妹関係の cgroup 間でのみ競合が発生します。これにより、負荷制御は全ての子孫が持つ帯域を集約することになりますので、これによって I/O の負荷制御が実現されます (下記の図を参照) 。

```
r
`-  a      IOWeight=100
    `- [c] IOWeight=300
    `-  d  IOWeight=100
`- [b]     IOWeight=200
```

I/O は cgroups の c と b からのみ発生するものとすると、c には高い優先順位が付けられているものの、b との比較では低いため、低い優先順位として扱われます。

10.5.2 制御量の設定

長期間動作するようなサービスに対しては、値を恒久的に適用することができます。

```
> sudo systemctl set-property fast.service IOWeight=400
> sudo systemctl set-property slow.service IOWeight=50
> sudo systemctl set-property throttled.service IOReadBandwidthMax="/dev/sda 1M"
```

それ以外にも、個別のコマンドに対して I/O 制御を適用することもできます。たとえば下記のようになります:

```
> sudo systemd-run --scope -p IOWeight=400 高い優先順位で実行するコマンド
> sudo systemd-run --scope -p IOWeight=50 低い優先順位で実行するコマンド
> sudo systemd-run --scope -p IOReadBandwidthMax="/dev/sda 1M" dd if=/dev/sda of=/dev/null
 bs=1M count=10
```

10.5.3 I/O 制御の動作説明

下記は I/O 制御の動作と、異なる状況下において何を期待すべきかについて説明しています。

- I/O 制御は直接的な I/O 操作 (ページキャッシュを迂回する操作) に対しては最適な動作を提供しますが、実際の I/O と呼び出し元が独立して動作するような場合 (一般にページキャッシュを介して書き戻す処理) は様々な振る舞いになることに注意してください。たとえば遅延型の I/O 制御を使用しているような場合や、全く I/O 制御を行わなかった場合などがそれに該当します。具体的にはごく僅かな帯域超過や、互いに完全に独立した負荷などの場合、I/O が全く同じタイミングで送信されることになりますので、帯域を飽和させる結果になってしまいます。これらの理由により、最終的に生成された I/O 帯域が完全には設定した重み付けにならないことがあります。

- systemd ではより厳密な BFQ 調整のため、設定した重み付けを規模に応じて処理します。そのため、最終的な帯域比も異なる場合があります。

- 書き戻しの処理は sysctl で設定したグローバル値 (`vm.dirty_background_ratio` および `vm.dirty_ratio`) のほか、dirty ページの量に依存して動作を行います。また、個別の cgroup に対して dirty 制限が分散されるような場合、それぞれの cgroup に対するメモリ制限にも影響を受けることがあります。これによって、それら cgroup の I/O 集中度に影響がある場合があります。

- 全てのストレージが等価でないことにも注意してください。I/O 制御は I/O スケジューラの階層で行われるため、その先にスケジューリングを行わないデバイスが存在しているような場合には異なる動作になります。たとえば複数の物理デバイスを利用した論理ボリュームのデバイスマッパーや MD RAID 、そして btrfs の RAID などがそれに該当します。これらのデバイスに対して I/O 制御を行う場合は、注意してお使いください。

- また、読み込みと書き込みのそれぞれに対して I/O 制御を行うような設定は提供されていません。

- I/O 制御の制限は相互に作用するポリシーのうちの 1 つであることに注意してください。ただし、適切なリソース設計をしていれば、問題なく動作するはずのものです。

- I/O デバイスの帯域は I/O パス内での共有資源というだけではないことに注意してください。I/O 制御が一定の帯域を保障することを目指している場合、グローバルなファイルシステム構造も考慮する必要があります。場合によっては優先順位が効果を発揮しない場合があるほか、たとえば優先順位の高い cgroup が遅い cgroup を待ち合わせる必要が生じてしまうことにより、優先順位が逆転する可能性すらあることに注意してください。

- ここまではファイルシステムデータに対する明示的な I/O 制御について説明してきましたが、スワップデバイスの入出力に対する制御も行うことができます。このような要件が発生した場合は、メモリの配置 (もしくはメモリ制限) を設定することを検討してください。

10.5.4 ユーザセッション内での資源制御

ユーザセッション内で cgroup の資源制御を適用したい場合は、`systemd` のユーザインスタンスに対してコントローラの委任を行う必要があります。なお、openSUSE Leap の既定の設定では、コントローラの委任は設定されて いません 。

コントローラの委任を実施したい場合は、ドロップイン・ファイルを利用して設定してください。たとえば `/etc/systemd/system/user@.service.d/60-delegate.conf` のようなファイルを作成すれば、全てのユーザに対してコントローラの委任を行うことができますし、`/etc/systemd/system/user@uid.service.d/60-delegate.conf` のようなファイルを作成すれば、特定のユーザに対してのみ委任を実施することができます。それぞれのファイルの内容は下記のように記述します:

```
[Service]
Delegate=pids memory
```

設定が終わったら、`systemd` のインスタンスとユーザインスタンスに対して、変更を通知して再読み込みを実施します。

```
> sudo systemctl daemon-reload
> systemctl --user daemon-reexec
```

2 行目を実行する以外にも、対象のユーザがいったんログアウトしてログインしなおし、ユーザインスタンスを再起動してもかまいません。

10.6 さらなる情報

- カーネルのドキュメンテーション (`kernel-source` パッケージ内): `/usr/src/linux/Documentation/admin-guide/cgroup-v1` および `/usr/src/linux/Documentation/admin-guide/cgroup-v2.rst` の各ファイル
- `man systemd.resource-control`
- https://lwn.net/Articles/604609/: Brown, Neil: Control Groups Series (2014 年, 7 部構成)
- https://lwn.net/Articles/243795/: Corbet, Jonathan: Controlling memory use in containers (2007 年)
- https://lwn.net/Articles/236038/: Corbet, Jonathan: Process containers (2007 年)

11 自動的な Non-Uniform Memory Access (NUMA) のバランス調整

改訂履歴

2023-08-08

多数の CPU と大容量のメモリを必要とする構成では、ハードウエアに対する物理的な制限にぶつかることがあります。本章では、CPU とメモリの間の通信帯域の制限が存在することによる制限事項について説明しています。また、このような問題に対応する目的で作られた、Non-Uniform Memory Access (NUMA) と呼ばれるアーキテクチャ変更についても、説明を行っています。

NUMA 構成のハードウエアには複数のノードが存在しています。それぞれのノード内には、CPU とメモリが存在しています。メモリへのアクセス速度は、CPU に近い場所にあるのかどうかによって変化します。つまり全体的な処理性能も、アクセスすべきメモリが近い場所にあるのかどうかによって異なることになります。自動的な NUMA バランス調整の仕組みは、CPU と CPU がアクセスすべきメモリを、できる限り CPU に近い位置に配置しようとする仕組みです。この仕組みを利用することで、NUMA ハードウエアを使用していれば、劇的に性能を改善することができるようになります。

11.1 実装

自動的な NUMA バランス調整は、下記の 3 つの手順で動作します:

1. タスクスキャナが定期的にタスクのアドレス領域の部分をスキャンし、次にアクセスする際に強制的にページフォルトを発生するようメモリにマークを設定します。

2. データへのアクセスが発生すると、NUMA ヒンティングフォルトが発生します。このフォルトをベースにして、そのタスクと同じノード内にメモリを移行させるようにします。

3. タスクを維持するため、タスクが処理されている CPU とアクセスしようとしているメモリをグループ化して管理するようにします。

このような仕組みから、データのマップ解除とページフォルトが発生することになりますので、性能面のオーバーヘッドが生じることになります。しかしながら、一般的には性能向上によって十分に相殺されるはずです。

11.2 設定

以前は NUMA ハードウエアで性能チューニングを行う場合、静的な設定を適用することが推奨されてきました。静的な設定を行う場合、メモリポリシーを `numactl` , `taskset` , `cpusets` のいずれかで設定することができます。また、NUMA に対応したアプリケーションの場合、API を使用することもあります。静的な設定が既に作成されている場合は、アクセスすべきデータが既にローカルに存在していますので、自動的な NUMA バランス調整を無効化すべきです。

`numactl --hardware` を実行すると、マシン内のメモリ設定を表示することができるほか、NUMA に対応しているかどうかも表示することができます。下記は 4 ノードのマシンでの出力例です。

```
> numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 4 8 12 16 20 24 28 32 36 40 44
node 0 size: 16068 MB
node 0 free: 15909 MB
node 1 cpus: 1 5 9 13 17 21 25 29 33 37 41 45
node 1 size: 16157 MB
node 1 free: 15948 MB
node 2 cpus: 2 6 10 14 18 22 26 30 34 38 42 46
node 2 size: 16157 MB
node 2 free: 15981 MB
node 3 cpus: 3 7 11 15 19 23 27 31 35 39 43 47
node 3 size: 16157 MB
node 3 free: 16028 MB
node distances:
node   0   1   2   3
  0:  10  20  20  20
  1:  20  10  20  20
  2:  20  20  10  20
  3:  20  20  20  10
```

自動的な NUMA バランス調整を再起動するまでの間だけ有効化もしくは無効化したい場合は、`/proc/sys/kernel/numa_balancing` に `1` (有効化) もしくは `0` (無効化) を書き込んでください。恒久的に有効化もしくは無効化を行いたい場合は、カーネルのコマンドラインオプションで `numa_balancing=[enable|disable]` [有効化|無効化] を指定してください。

自動的な NUMA バランス調整が有効化されている場合、タスクスキャナの動作を調整することができます。タスクスキャナは、自動的な NUMA バランス調整によるオーバーヘッドと、それにかかる時間を考慮して、最適なデータ配置を判別します。

`numa_balancing_scan_delay_ms`

データをスキャンするまでに、スレッドが消費しなければならない CPU 時間を指定します。これにより、起動してすぐ終了してしまうようなプロセスを調整から除外することができます。

`numa_balancing_scan_period_min_ms` および `numa_balancing_scan_period_max_ms`

タスクのデータをスキャンする頻度を制御します。フォルトの発生箇所によって、スキャン頻度は大きくも小さくもなります。これらの設定では、それぞれ最小値と最大値を設定します。

`numa_balancing_scan_size_mb`

タスクスキャナが有効化されている場合、どれだけの量のアドレス領域をスキャンするかを設定します。

11.3 監視

最も重要な作業は、自動的な NUMA バランス調整を有効化した場合と無効化した場合で、その性能差を比較するために、お使いのサーバに対する性能基準を策定することです。CPU 側に NUMA 関連の監視機能が付属していれば、メモリアクセスが近くに (ローカルに) あったものか、もしくは遠くに (リモートに) あったものかを監視するためのプロファイリングツールを使用することができます。自動的な NUMA バランス調整では、`/proc/vmstat` 内の下記のパラメータで監視を行うことができます:

`numa_pte_updates`

NUMA ヒンティングフォルトでマークされたベースページの量を表します。

`numa_huge_pte_updates`

NUMA ヒンティングフォルトでマークされた、透過型巨大ページの量を表します。`numa_pte_updates` の値と組み合わせることで、マークされた全体のアドレス領域を計算することができます。

`numa_hint_faults`

どれだけの数に NUMA ヒンティングフォルトを捉えることができたのかを記録しています。

`numa_hint_faults_local`

ヒンティングフォルトのうち、どれだけがローカルのノードによるものであったのかを表します。`numa_hint_faults` の値と組み合わせることで、ローカルとリモートの割合を計算することができます。ローカルのノードによるものが大きければ大きいほど、多くの処理が単一ノード内に収束できていることがわかります。

`numa_pages_migrated`

配置がリモートであったために移行されたページ数を記録しています。移行処理ではコピー操作が行われるため、NUMA バランス調整での最も大きなオーバーヘッドになります。

11.4 影響

下記の出力例は、メモリポリシーまわりに特にチューニングを行っていない JVM の単一インスタンスを利用して、SpecJBB 2005 を動作させた 4 ノードの NUMA マシンの様子を示しています。ただし、性能への影響は必ずしも良いものであるとは限りませんし、また下記の例は openSUSE Leap 12 のプレリリース版での例であることに注意してください。

```
           Balancing disabled      Balancing enabled
TPut 1      26629.00 (  0.00%)     26507.00 ( -0.46%)
TPut 2      55841.00 (  0.00%)     53592.00 ( -4.03%)
TPut 3      86078.00 (  0.00%)     86443.00 (  0.42%)
TPut 4     116764.00 (  0.00%)    113272.00 ( -2.99%)
TPut 5     143916.00 (  0.00%)    141581.00 ( -1.62%)
TPut 6     166854.00 (  0.00%)    166706.00 ( -0.09%)
TPut 7     195992.00 (  0.00%)    192481.00 ( -1.79%)
TPut 8     222045.00 (  0.00%)    227143.00 (  2.30%)
TPut 9     248872.00 (  0.00%)    250123.00 (  0.50%)
TPut 10    270934.00 (  0.00%)    279314.00 (  3.09%)
TPut 11    297217.00 (  0.00%)    301878.00 (  1.57%)
TPut 12    311021.00 (  0.00%)    326048.00 (  4.83%)
TPut 13    324145.00 (  0.00%)    346855.00 (  7.01%)
TPut 14    345973.00 (  0.00%)    378741.00 (  9.47%)
TPut 15    354199.00 (  0.00%)    394268.00 ( 11.31%)
TPut 16    378016.00 (  0.00%)    426782.00 ( 12.90%)
TPut 17    392553.00 (  0.00%)    437772.00 ( 11.52%)
TPut 18    396630.00 (  0.00%)    456715.00 ( 15.15%)
TPut 19    399114.00 (  0.00%)    484020.00 ( 21.27%)
TPut 20    413907.00 (  0.00%)    493618.00 ( 19.26%)
TPut 21    413173.00 (  0.00%)    510386.00 ( 23.53%)
TPut 22    420256.00 (  0.00%)    521016.00 ( 23.98%)
TPut 23    425581.00 (  0.00%)    536214.00 ( 26.00%)
TPut 24    429052.00 (  0.00%)    532469.00 ( 24.10%)
TPut 25    426127.00 (  0.00%)    526548.00 ( 23.57%)
TPut 26    422428.00 (  0.00%)    531994.00 ( 25.94%)
TPut 27    424378.00 (  0.00%)    488340.00 ( 15.07%)
TPut 28    419338.00 (  0.00%)    543016.00 ( 29.49%)
TPut 29    403347.00 (  0.00%)    529178.00 ( 31.20%)
TPut 30    408681.00 (  0.00%)    510621.00 ( 24.94%)
TPut 31    406496.00 (  0.00%)    499781.00 ( 22.95%)
TPut 32    404931.00 (  0.00%)    502313.00 ( 24.05%)
TPut 33    397353.00 (  0.00%)    522418.00 ( 31.47%)
TPut 34    382271.00 (  0.00%)    491989.00 ( 28.70%)
TPut 35    388965.00 (  0.00%)    493012.00 ( 26.75%)
TPut 36    374702.00 (  0.00%)    502677.00 ( 34.15%)
TPut 37    367578.00 (  0.00%)    500588.00 ( 36.19%)
TPut 38    367121.00 (  0.00%)    496977.00 ( 35.37%)
TPut 39    355956.00 (  0.00%)    489430.00 ( 37.50%)
TPut 40    350855.00 (  0.00%)    487802.00 ( 39.03%)
TPut 41    345001.00 (  0.00%)    468021.00 ( 35.66%)
```

```
TPut 42    336177.00 (  0.00%)    462260.00 ( 37.50%)
TPut 43    329169.00 (  0.00%)    467906.00 ( 42.15%)
TPut 44    329475.00 (  0.00%)    470784.00 ( 42.89%)
TPut 45    323845.00 (  0.00%)    450739.00 ( 39.18%)
TPut 46    323878.00 (  0.00%)    435457.00 ( 34.45%)
TPut 47    310524.00 (  0.00%)    403914.00 ( 30.07%)
TPut 48    311843.00 (  0.00%)    459017.00 ( 47.19%)

                        Balancing Disabled        Balancing Enabled
 Expctd Warehouse          48.00 (  0.00%)          48.00 (  0.00%)
 Expctd Peak Bops      310524.00 (  0.00%)      403914.00 ( 30.07%)
 Actual Warehouse          25.00 (  0.00%)          29.00 ( 16.00%)
 Actual Peak Bops      429052.00 (  0.00%)      543016.00 ( 26.56%)
 SpecJBB Bops            6364.00 (  0.00%)        9368.00 ( 47.20%)
 SpecJBB Bops/JVM        6364.00 (  0.00%)        9368.00 ( 47.20%)
```

自動的な NUMA バランス調整の仕組みにより、NUMA マシンで高性能なチューニングを行う際、それにかかる手間を減らすことができます。とはいえ、可能であれば各ノード内で負荷を分割できるような静的なチューニングを行っておく必要はあります。しかしながら、ほとんどの場合において NUMA バランス調整は性能改善の効果があります。

12 電源管理

改訂履歴

2024-06-25

電源管理機能は、電力や冷却システムに対する運用コストを下げるためのものだけでなく、それと同時に現在の要件に適合したレベルの性能を維持するために必要な機能です。そのため電源管理機能は、必要な性能の維持と省電力のバランスを常に取る必要があることになります。電源管理機能はシステム内の様々なレベルで実装され使用されています。デバイスやオペレーティングシステムの電源管理機能の仕様集は、Advanced Configuration and Power Interface (ACPI) として規定されています。サーバ環境での省電力は主にプロセッサレベルで実現されていますが、本章では主な考え方と分析に使用するツール類、そして影響するパラメータについて説明しています。

12.1 CPU レベルでの電源管理

CPU のレベルでは、様々な方法で電力使用量を制御することができます。たとえばアイドル (待機) 状態での電力使用量の調整 (C-ステート) や、CPU の動作周波数変更 (P-ステート) 、そして CPU の減速 (T-ステート) があります。下記のセクションには、それぞれのアプローチに対する簡潔な説明と、電力使用量にもたらす影響度合いを示しています。詳しい仕様をお読みになりたい場合は、https://uefi.org/sites/default/files/resources/ACPI_Spec_6_4_Jan22.pdf (英語) を参照してください。

12.1.1 C-ステート (プロセッサの待機状態)

新しいプロセッサであれば、`C-ステート` と呼ばれる省電力モードに対応しています。この設定は、使用していないコンポーネントの電源を落としておくことで、待機状態のプロセッサの消費電力を減らすための仕組みです。

プロセッサが `C0` 状態に設定されている場合、そのプロセッサは何らかの処理を実行していることになります。`C0` 以外の状態であれば、アイドル (待機) 状態にあることになります。C の後ろの番号が大きければ大きいほど、より CPU が深く休眠し、より多くのコンポーネントを休止させることで、電力の消費を抑えていることになります。ただし、C の後ろの番号が大きければ大きいほど、遅延を生み出す結果になります。つまり、CPU が `C0` に戻るまでに時間が必要になります。処理内容 (スレッドが起動して CPU の使用を開始し、再度休眠状態に入るまでの時間など) やハードウエア (たとえばネットワーク

デバイスの割り込み状況) にも依存しますが、深い休眠状態を無効化すれば、全体の性能は高くなります。C-ステートの設定方法について、詳しくは 12.3.2項「`cpupower` によるカーネル内のアイドル状態の表示」をお読みください。

また、ステートによってはサブモードが規定され、さらに細かい省電力レベルが用意されていることもあります。C-ステートやサブモードの対応状況は、プロセッサによって異なります。ただし、`C1` については必ず提供されています。

表12.1「C-ステート」には、主な C-ステートの概要を示しています。

表 12.1: C-ステート

モード	定義
C0	動作状態を表します。CPU の機能が全て有効化されています。
C1	最初のアイドル状態です。ソフトウエアを介して CPU のメイン内部クロックを停止させている状態です。バスインターフェイスユニットと APIC はフルスピードで動作し続けています。
C2	ハードウエアを介して CPU のメイン内部クロックを停止させている状態です。ソフトウエアから参照できる全ての状態を維持していますが、割り込みでの再動作にはしばらくの時間を要するようになります。
C3	全ての CPU 内部クロックを停止させている状態です。プロセッサ内のキャッシュの一貫性 (コヒーレンシー) についても維持を行う必要のない状態ですが、その他の状態については維持し続けています。また、プロセッサによっては C3 ステートのバリエーションが用意され、割り込みによる再動作にかかる時間が異なる設定が用意されているものがあります。

不要な電力消費を避けるため、深い休眠状態を有効化した場合と無効化した場合で、必要な負荷に対する処理速度を確認しておくことをお勧めします。詳しくは 12.3.2項「`cpupower` によるカーネル内のアイドル状態の表示」を参照するか、もしくは `cpupower-idle-set(1)` のマニュアルページをお読みください。

12.1.2 P-ステート (プロセッサの性能状態)

プロセッサが動作している状態 (C0 ステート) にある場合、CPU はいずれかの性能状態 (P-ステート) にあります。C0 を除く C-ステートがアイドル (待機) 状態を表すのに対して、P-ステート は CPU の動作周波数と電圧に関連する動作状態を表します。

P の後ろの数値が大きければ大きいほど、より低い動作周波数および電圧でプロセッサが動作していることになります。P-ステートの番号体系と実装には様々なものがありますが、P0 は (12.1.3項「Turbo 機能」を除く) 最高の性能状態を表す意味になっています。P の後ろの番号が大きいほど、プロセッサの速度が落ちていて、かつ電力消費も少ないモードであることになります。たとえば P3 の状態は、P1 の状態に比べて遅く、省電力で動作しています。また、P-ステートは C0 ステートで動作している (つまり、全ての機能を動作させ、休眠状態に無い) 場合にのみ使用されるものでもあります。CPU の P-ステートは ACPI 仕様でも定義されています。詳しくは https://uefi.org/sites/default/files/resources/ACPI_Spec_6_5_Aug29.pdf をお読みください。

なお、C-ステートと P-ステートは、互いに独立して提供される仕組みです。

12.1.3 Turbo 機能

Turbo 機能を利用することで、動作中の CPU コアの動作速度を動的に 引き上げ ながら、動作していない CPU コアを深い休眠状態にすることができます。これにより、熱設計電力 (Thermal Design Power (TDP)) の制限を守りながら、動作中のスレッドの性能を上げることができます。

しかしながら、CPU コアが Turbo 機能に対応できるかどうかは、アーキテクチャによって異なります。このような新しい機能の効率を調べたい場合は、12.3項「cpupower ツール」を参照してください。

12.2 カーネル内ガバナー

カーネル内のガバナー (統治する仕組み) は、Linux カーネルの CPUfreq インフラストラクチャに属するもので、動作中に動的にプロセッサの動作周波数を変更するために使用することができます。ガバナーは CPU 側であらかじめ設定された電力スキームを表すものとして考えることができます。CPUfreq は P-ステートを使用して周波数や消費電力を統治します。動的なガバナーの仕組みによって、CPU の周波数を使用されている状況に応じて切り替えることができます。これにより、性能への影響を少なくしたまま、省電力を実現できることになります。

CPUfreq サブシステムでは、下記のガバナーが提供されています:

Performance ガバナー
: 最大限の性能を発揮するため、CPU の動作周波数を設定可能な最大値に固定します。そのため、このガバナーでは電力消費量の削減は行わないことになります。

12.4.1項「P-ステート向けのチューニングオプション」もお読みください。

Powersave ガバナー

CPU の動作周波数を設定可能な最小値に固定します。これにより、プロセッサがどれだけ忙しい状態であっても、周波数を上げることがなくなりますので、性能には大きな影響が現れます。ただし、`intel_pstate` については例外で、既定値が `powersave` モードになっていますが、こちらはハードウエア固有の仕組みによって設定されているものであり、実際には `on-demand` ガバナーのように動作することに注意してください。
また、このガバナーを利用して最大限の省電力を実現しようとしても、実環境では C-ステートによる待機時の電力消費削減のほうが効果が大きくなってしまい、期待通りの省電力にはならないことがありますので、あらかじめご注意ください。さらに、Powersave のガバナーでは周波数が低くなることから、処理にかかる時間も長くなります。つまり、C-ステートの変化も遅くなります。
チューニングオプション: ガバナーで調整する最小限の周波数を設定することができます (`cpupower` コマンドラインツールなどで行うことができます) 。

On-demand ガバナー

動的な CPU 周波数制御を行うカーネル実装です。このガバナーではプロセッサの使用状況を確認して、ある特定の閾値を超過すると指定可能な最大の周波数に設定します。逆に使用状況が閾値を下回ると、指定可能な次の低い周波数に設定します。さらに使用されない状態が続くと、最も低い周波数になるまで少しずつ低く設定します。

重要: ドライバとカーネル内ガバナーの関係性について

全てのドライバがカーネル内のガバナーを利用して動的な周波数調整を行っているわけではありません。たとえば `intel_pstate` ドライバでは、独自に周波数を調整しています。お使いのドライバが何であるのかを調べたい場合は、`cpupower frequency-info` と入力して実行してください。

12.3 cpupower ツール

`cpupower` ツールは、Turbo 機能 (もしくはブースト機能) を含め、マシン内で利用できる 全ての CPU 消費電力関連のパラメータを取得したり設定したりすることのできるツールとして設計されています。このツールを使用することで、カーネル関連の CPUfreq や cpuidle システムのほか、周波数調整やアイドル状態とは直接関係のない様々な設定を表示したり、変更したりすることができます。また、統合されている監視フレームワークでは、カーネル関連のパラメータとハードウエアの統計情報の両方にアクセスする機能を提供します。そのため、性能のベンチマークを行う用途に適した仕組みです。このほか、Turbo やアイドル状態の依存関係の調査を行うこともできます。

`cpupower` パッケージをインストールしたら、`cpupower --help` を実行することで、利用可能な `cpupower` のサブコマンド一覧を表示することができます。また、全体のマニュアルページには `man cpupower` と入力して実行することで評議することができるほか、サブコマンド向けのマニュアルページを表示したい場合は、`man cpupower-サブコマンド` と入力して実行してください。

12.3.1 `cpupower` による現在設定の表示

`cpupower frequency-info` のように入力して実行すると、カーネル内で使用されている cpufreq ドライバの統計情報を表示することができます。これに加えて、BIOS 内で Turbo (ブースト) 機能が有効化されているかどうかも表示することができます。何もオプションを指定しないで実行すると、下記のような出力が現れます:

例 12.1: `cpupower frequency-info` の出力例

```
# cpupower frequency-info
analyzing CPU 0:
  driver: intel_pstate
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated by software: 0
  maximum transition latency: 0.97 ms.
  hardware limits: 1.20 GHz - 3.80 GHz
  available cpufreq governors: performance, powersave
  current policy: frequency should be within 1.20 GHz and 3.80 GHz.
                  The governor "powersave" may decide which speed to use
                  within this range.
  current CPU frequency is 3.40 GHz (asserted by call to hardware).
  boost state support:
    Supported: yes
    Active: yes
    3500 MHz max turbo 4 active cores
    3600 MHz max turbo 3 active cores
    3600 MHz max turbo 2 active cores
    3800 MHz max turbo 1 active cores
```

全ての CPU に対する現在の値を表示するには、`cpupower -c all frequency-info` コマンドを使用します。

12.3.2 `cpupower` によるカーネル内のアイドル状態の表示

`idle-info` サブコマンドは、カーネル内で使用されている cpuidle ドライバの統計情報を表示することができます。cpuidle カーネルフレームワークを使用していれば、どのようなアーキテクチャでも動作します。

例 12.2: `cpupower idle-info` の出力例

```
# cpupower idle-info
CPUidle driver: intel_idle
CPUidle governor: menu

Analyzing CPU 0:
Number of idle states: 6
Available idle states: POLL C1-SNB C1E-SNB C3-SNB C6-SNB C7-SNB
POLL:
Flags/Description: CPUIDLE CORE POLL IDLE
Latency: 0
Usage: 163128
Duration: 17585669
C1-SNB:
Flags/Description: MWAIT 0x00
Latency: 2
Usage: 16170005
Duration: 697658910
C1E-SNB:
Flags/Description: MWAIT 0x01
Latency: 10
Usage: 4421617
Duration: 757797385
C3-SNB:
Flags/Description: MWAIT 0x10
Latency: 80
Usage: 2135929
Duration: 735042875
C6-SNB:
Flags/Description: MWAIT 0x20
Latency: 104
Usage: 53268
Duration: 229366052
C7-SNB:
Flags/Description: MWAIT 0x30
Latency: 109
Usage: 62593595
Duration: 324631233978
```

プロセッサがどのアイドル状態に対応しているのかを `cpupower idle-info` で調べたあとは、`cpupower idle-set` コマンドを使用することで、個別のステートを無効化することができます。一般的には最も深い休眠状態を無効化することになるかと思いますが、このような場合は下記のように入力して実行します:

```
# cpupower idle-set -d 5
```

レイテンシが `80` と等しいか、それより大きいものを全て無効化したい場合は、下記のように入力して実行します:

```
# cpupower idle-set -D 80
```

12.3.3 `cpupower` によるカーネルとハードウエアの統計情報の表示

`monitor` サブコマンドを使用することで、プロセッサのトポロジ情報を表示することができるほか、特定の時間内における周波数やアイドル状態の統計を表示することができます。既定の間隔は `1` 秒ですが、`-i` オプションを使用することで自由に変更することができます。また、ツール内ではプロセッサのスリープ (休眠) 状態と周波数のカウンタは独立して実装されています。これらのうちのいくつかはカーネルの統計情報から、残りはハードウエアレジスタから取得しています。利用可能な監視機能は、お使いのハードウエアとシステムによって異なります。利用可能な監視機能を表示したい場合は、`cpupower monitor -l` と入力して実行してください。また、各モニタに関する詳細は、cpupower-monitor のマニュアルページをお読みください。

`monitor` サブコマンドでは、性能を測定するためのベンチマーク機能にも対応しています。特定の処理におけるハードウエアの統計情報を比較したい場合は、その処理のコマンドを末尾に付けて実行します。たとえば下記のようになります:

```
cpupower monitor db_test.sh
```

例 12.3: `cpupower monitor` の出力例

```
# cpupower monitor
|Mperf                  || Idle_Stats
 ❶                        ❷
CPU | C0   | Cx   | Freq || POLL | C1   | C2   | C3
   0|  3.71| 96.29|  2833||  0.00|  0.00|  0.02| 96.32
   1| 100.0| -0.00|  2833||  0.00|  0.00|  0.00|  0.00
   2|  9.06| 90.94|  1983||  0.00|  7.69|  6.98| 76.45
   3|  7.43| 92.57|  2039||  0.00|  2.60| 12.62| 77.52
```

❶ Mperf はブーストした周波数を含む、CPU の平均動作周波数を表示します。これに加えて、CPU が動作中 (`C0`) であった割合と、休眠中であった割合 (`Cx`) を、時間比で表示することができます。Turbo 機能は BIOS で管理されている機能であるため、その場での周波数を取得することができませんが、Mperf 監視を使用することで、過去にどのような動作状況であったのかを確認することができます。

❷ Idle_Stats は cpuidle カーネルサブシステムの統計情報を表示します。カーネルはアイドル状態に入ったり出たりするごとに、それらの値を更新していきます。なお、測定の開始と終了時にはアイドル状態の変化が発生するため、その分だけ値が不正確になることに注意してください。

> 上述の (一般的な) 監視機能とは別に、アーキテクチャ依存の監視機能も使用することができます。詳しい説明については、`cpupower-monitor` のマニュアルページをお読みください。

個別のモニタで出力された値を比較することで関係性や依存性を見つけることができるほか、特定の処理に対してどれだけ省電力機能が正しく働いたのかを確認することもできます。たとえば 例 12.3 では CPU `0` がほぼアイドル状態 (つまり `Cx` がほぼ 100%) になっていますが、それにも関わらず高い動作周波数になってしまっています。これは CPU `0` と `1` が同じ周波数しか設定できないことによるもので、これを依存性と称しています。

12.3.4 `cpupower` による設定の変更

`root` で `cpupower frequency-set` コマンドを実行することで、現在の設定を変更することができます。ガバナーが設定する最小／最大 CPU 動作周波数のほか、新しいガバナーを作成することもできます。また `-c` オプションを指定することで、どのプロセッサに対する設定を変更するのかを指定することもできます。これにより、個別のプロセッサに対して調整を行うことなく、全てのプロセッサに対して一貫したポリシーを適用できるようになっています。詳細と利用可能なオプションについて、詳しくは `cpupower-frequency-set` のマニュアルページをお読みになるか、`cpupower frequency-set --help` と入力して実行してください。

12.4 特殊なチューニングオプション

下記の章では、主な設定について説明しています。

12.4.1 P-ステート向けのチューニングオプション

CPUfreq サブシステムでは、P-ステート向けのチューニングオプションがいくつか提供されています。オプションにはガバナーの切り替えのほか、使用すべき最小／最大の CPU 周波数と、ガバナー別のパラメータの変更などがあります。

ガバナーを変更したい場合は、`cpupower frequency-set` コマンドに `-g` オプションを付けて実行します。たとえば `root` で下記のように入力して実行すると、Performance ガバナーに切り替えることができます:

```
# cpupower frequency-set -g performance
```

ガバナーが使用すべき最小および最大の CPU 周波数を指定したい場合は、それぞれ `-d` や `-u` のオプションで設定します。

12.5 トラブルシューティング

BIOS オプションが有効化されていますか?

C-ステートや P-ステートを使用するには、BIOS 側での確認が必要です:

- C-ステートを使用するには、まず `CPU C State` などのオプションを有効化して、アイドル時の省電力を有効化する必要があります。
- P-ステートや CPUfreq のガバナーを使用するには、`Processor Performance States` などのオプションを設定します。
- P-ステートや C-ステートが利用可能である場合でも、プラットフォーム側のファームウエアで CPU の動作周波数を管理していることがあります。これは準最適とも呼べる仕組みで、たとえば `pcc-cpufreq` を読み込むことで、OS 側からファームウエアにヒントを提示することができます。ファームウエア側ではその情報が無視される場合もあります。このような仕組みは、BIOS のセットアップで CPU の周波数を "OS Management" などの選択を行うことで、解決できることもあります。BIOS 側で設定を変更すれば、代替ドライバが使用されるようになりますので、そこから最適化をやり直すことができます。

CPU をアップグレードしているような場合は、BIOS 側についてもアップグレードを行う必要があることがあります。BIOS 側でも新しい CPU に対応させる必要があり、周波数の設定などもここから採取することがあるためです。

ログファイル内に何か現れていませんか?

まずは `systemd` のジャーナル (詳しくは『リファレンス』、第11章「`journalctl`：systemd ジャーナルへの問い合わせコマンド」をお読みください) をお読みのうえ、CPUfreq サブシステムに関する出力が現れていないかどうかを確認してください。ただし、ここには致命的なエラーのみが記録されます。

お使いのマシンで CPUfreq サブシステムの問題を調査したい場合は、デバッグ出力を有効化して調べる方法があります。デバッグ出力を有効化するには、カーネルのコマンドラインパラメータに `cpufreq.debug=7` を指定するか、もしくは下記のコマンドを `root` で実行します:

```
# echo 7 > /sys/module/cpufreq/parameters/debug
```

このコマンドを実行すると、CPUfreq での状態遷移など、さらに詳しい情報を `dmesg` に出力するようになります。これを元に調査を行ってください。ただし、かなり広範囲なログの出力を行いますので、明らかに問題があるような場合にのみお使いになることをお勧めします。

12.6 さらなる情報

Baseboard Management Controller (BMC) の搭載されたプラットフォームであれば、このサービスプロセッサを通じて様々な電力管理設定にアクセスすることができます。これらの設定は製造元ごとに異なるものであるため、本章では説明していません。詳しくは製造元が提供するマニュアルをお読みください。

12.7 powerTOP による電力消費状況の監視

powerTOP は電源消費が大きくなってしまっている理由を探すためのツールです。これは主に、電源消費が重要となるラップトップで有用なツールとなります。このツールは Intel および AMD の各プロセッサに対応しています。インストールを行うには、通常通り下記のようにして行います:

```
> sudo zypper in powertop
```

powerTOP は様々な情報源からのデータ (プログラムやデバイスドライバ、カーネルのオプションやスリープ状態の解除を行った割り込みの回数や生成源) を組み合わせて、それらをいくつかの方法で表示することができます。一般的には ncurses セッション内で、対話モードとして起動を行います (詳しくは 図12.1「対話モードでの powerTOP」 をご覧ください):

```
> sudo powertop
```

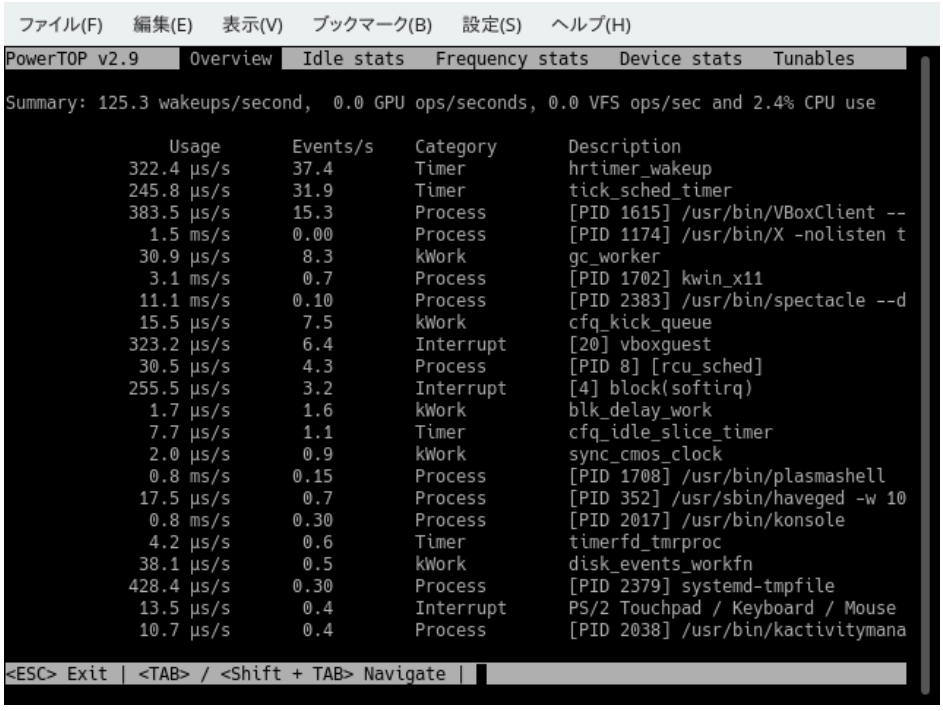

図 12.1: 対話モードでの POWERTOP

powerTOP では、レポートを HTML 形式や CSV 形式で出力することができます。下記の例では、240 秒間の統計情報を 1 つのレポートとして出力しています:

```
> sudo powertop --iteration=1 --time=240 --html=POWERREPORT.HTML
```

レポートを複数回に分けて出力させることもできます。下記の例では、20 秒おきに合計で 10 回レポートを出力しています。なお、それぞれのレポートは別々の HTML レポートとして出力します:

```
> sudo powertop --iteration=10 --time=20 --html=POWERREPORT.HTML
```

上記を実行すると、下記のように日時付きのファイル名でレポートが生成されます:

```
powerreport-20200108-104512.html
powerreport-20200108-104451.html
powerreport-20200108-104431.html
[...]
```

HTML レポートは 図12.2「HTML での powerTOP レポート」 のような出力になります:

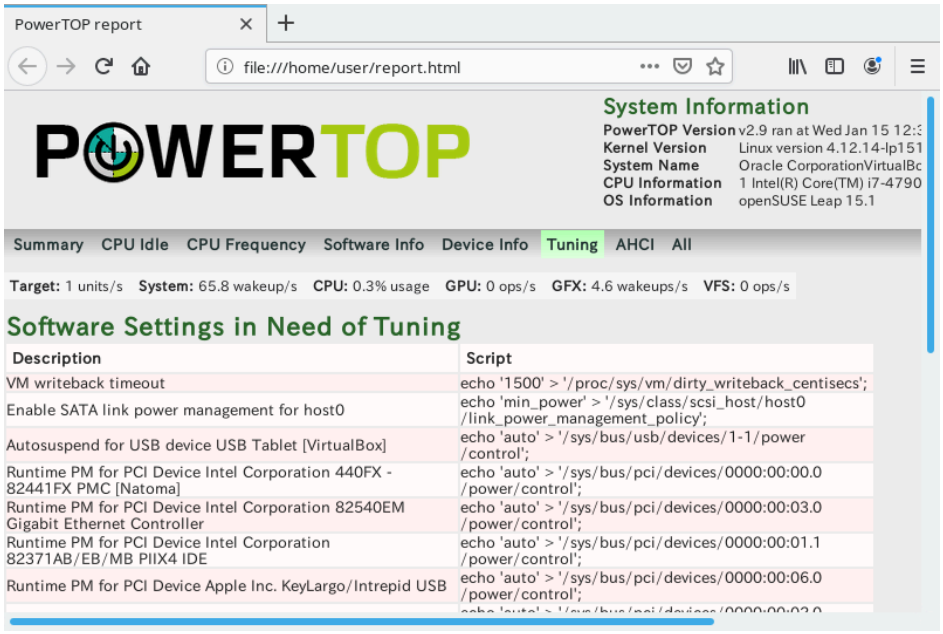

図 12.2: HTML での POWERTOP レポート

HTML レポート内の Tuning タブ、もしくは対話モードでの Tunables タブには、さまざまな電源設定を試すためのコマンドが提供されています。HTML レポートの場合にはコマンドライン (例: `echo '0' > '/proc/sys/kernel/nmi_watchdog'`) が書かれていますので、これを root のコマンドラインで実行すると、テストを行うことができます。ncurses モードの場合はコマンドラインではなく、`Good` (良い) もしくは `Bad` (悪い) が書かれた切り替えスイッチになっています。`Good` は省電力の観点で適切な設定になっていることを、`Bad` は省電力の観点で適切な設定になっていないことをそれぞれ示しています。powerTOP で全ての省電力設定を適用したい場合は、下記のコマンドを実行します:

```
> sudo powertop --auto-tune
```

なお、これらの設定はシステムを再起動すると失われてしまいます。これらの設定を常に適用しておきたい場合は、`sysctl` や `udev` を使用するか、もしくは `systemd` を利用して、起動時に必要なコマンドを実行するように設定してください。なお、powerTOP には `/usr/lib/systemd/system/powertop.service` というファイルが提供されていて、これを有効化することで、システムの起動時に powerTOP の `--auto-tune` を自動的に実行することができます:

```
ExecStart=/usr/sbin/powertop --auto-tune
```

なお、`systemd` のサービスを有効化する場合は、期待通りの結果になるのかをよくご確認ください。たとえば USB キーボードやマウスを接続している場合、これらは省電力モードに設定してもすぐに復帰してしまうため、省電力が適切に働かなくなってしまいます。また、テストや設定の編集を簡単に実施するため、`awk` を利用して HTML レポートからコマンドを抽出するとよいでしょう:

```
> awk -F '</?td ?>' '/tune/ { print $4 }' POWERREPORT.HTML
```

また、powerTOP にはキャリブレーション (較正) モードも用意されています。このモードはバックライトや CPU 、Wi-Fi や USB デバイス、ディスクなどに対して異なる設定を施して、バッテリー消費が最適な状態になるようにします:

```
> sudo powertop --calibrate
```

より正確な較正を行うには、ワークロード (負荷) ファイルを作成して呼び出してもかまいません:

```
> sudo powertop --calibrate --workload=FILENAME --html=POWERREPORT.HTML
```

さらに詳しい情報を読みたい場合は、それぞれ下記を参照してください:

- https://01.org/powertop にある powerTOP プロジェクトのページ (英語)
- 2.6.2項「システム制御パラメータ: `/proc/sys/`」
- 『リファレンス』、第10章「`systemd` デーモン」
- 『リファレンス』、第16章「`udev` による動的なカーネルデバイス管理」

V カーネルのチューニング

13 I/O 性能のチューニング 136

14 タスクスケジューラのチューニング 142

15 メモリ管理サブシステムのチューニング 155

16 ネットワークのチューニング 166

13 I/O 性能のチューニング

改訂履歴

2023-08-08

I/O スケジューリングの制御とは、ストレージに対して送信される入出力操作をどのような順序にするのかを決めるものです。openSUSE Leap では **エレベータ** と呼ばれる様々な I/O アルゴリズムに対応しています。これにより、様々な使用環境に対応することができるようになっています。エレベータはシーク操作を減らすためのものであるほか、I/O 要求に対する優先順位付けとしても使用されます。

最適な I/O エレベータを選択するにあたって、その根拠となるのは負荷だけではなく、ハードウエアの仕様も含みます。たとえば単一の ATA ディスクと SSD, RAID アレイやネットワークストレージシステムでは、それぞれ異なるチューニング戦略が必要となります。

13.1 I/O スケジューリングの切り替え

openSUSE Leap では起動時に既定の I/O スケジューラを選択しますが、I/O スケジューラはデバイス単位で即時に変更することができます。これにより、システムパーティションを含んでいるデバイスと、データベースを含んでいるデバイスで別々のアルゴリズムを設定したりすることができるようになっています。

既定の I/O スケジューラは、回転型のディスクであるかどうかによって異なります。回転型のディスクの場合は `BFQ` I/O スケジューラが使用されます。それ以外の場合は、`MQ-DEADLINE` か `NONE` のいずれかが使用されます。

動作中のシステム内で、特定のデバイスに対するエレベータを変更したい場合は、下記のコマンドを実行します:

```
> sudo echo スケジューラ > /sys/block/デバイス名/queue/scheduler
```

ここで、`スケジューラ` には `bfq`, `none`, `kyber`, `mq-deadline` のいずれかを指定します。また `デバイス名` には、ブロックデバイスのデバイス名 (例: `sda`) を指定します。ただし、この手順で変更を行った場合、システムを再起動すると既定値に戻ってしまうことに注意してください。特定のデバイスに対して I/O スケジューラを恒久的に変更したい場合は、`/usr/lib/udev/rules.d/60-io-scheduler.rules` ファイルを `/etc/udev/rules.d/60-io-scheduler.rules` ファイルにコピーしてから、コピー先のファイルを必要に応じて編集してください。

13.2 blk-mq I/O パスで利用可能な I/O エレベータ

下記の表には、openSUSE Leap で blk-mq I/O パスを使用する場合に、利用可能なエレベータの一覧を示しています。それぞれのエレベータにはさらに細かくチューニングを行うためのチューナブルパラメータが存在していますが、これらは下記のようなコマンドを実行することで、設定することができます:

```
echo 値 > /sys/block/デバイス名/queue/iosched/チューナブル名
```

ここで、`値` には `チューナブル名` に対応する値を、`デバイス名` にはブロックデバイス名をそれぞれ指定します。

また、デバイス (例: `sda`) に対して利用可能なエレベータを確認したい場合は、下記のようなコマンドを入力して実行します (なお、現在選択されているエレベータが [] で括られて出力されます):

```
> cat /sys/block/sda/queue/scheduler
[mq-deadline] kyber bfq none
```

13.2.1 MQ-DEADLINE

`MQ-DEADLINE` は遅延を重視した I/O スケジューラです。`MQ-DEADLINE` には下記のようなチューナブルパラメータが用意されています:

表 13.1: MQ-DEADLINE でのチューナブルパラメータ

ファイル	説明
`writes_starved`	書き込み回数に対する読み込み回数の推奨値を制御します。たとえば `3` という値であれば、同じ選択範囲のディスクに対して、書き込み 1 回に対して読み込みを 3 回までまとめて実施できるようになります。 既定値は `3` です。¥t
`read_expire`	ミリ秒単位で読み込み処理の期限 (現在時刻からの経過時間) を指定します。 既定値は `500` です。¥t
`write_expire`	ミリ秒単位で書き込み処理の期限 (現在時刻からの経過時間) を指定します。 既定値は `5000` です。¥t

ファイル	説明
`front_merges`	フロントマージ要求の試行をする (1) かしない (0) かを指定します。既定値は `1` です。
`fifo_batch`	一括 (バッチ) 処理時の最大要求数を設定します (期限の判断がその単位で行われます) 。このパラメータは遅延とスループットのバランスを取るために使用されるもので、たとえば `1` に設定する (一括処理で 1 つずつ処理する) と、"最初に到着したものを最初に処理する" 動作になり、最も遅延を少なくすることができます。逆により大きな数値にすると、一般にスループットを改善することができます。既定値は `16` です。¥t

13.2.2 NONE

blk-mq 向けの I/O エレベータとして `NONE` を選択した場合、I/O スケジューラでは何も行わず、デバイスに対してそのまま I/O 要求が渡されるようになります。

NVM Express デバイスに対しては、`NONE` が既定値となっています。他の I/O エレベータオプションとは異なり、オーバーヘッドが存在しないため、これらのデバイスに対して最も高速に I/O リクエストを送信できるようになっています。

`NONE` に対するチューナブルパラメータはありません。

13.2.3 BFQ (Budget Fair Queueing)

`BFQ` は公平性を重視したスケジューラです。大まかに説明すると、"CFQ のスライス単位のサービス方式をベースにした、比例分配型のストレージ I/O スケジューリングアルゴリズム" であると言えるものです。ただし、BFQ では CFQ のようなタイムスライスではなく、セクタ数をベースに割り当てを設定します (ソースコード: linux-4.12/block/bfq-iosched.c (https://github.com/torvalds/linux/blob/6f7da290413ba713f0cdd9ff1a2a9bb129ef4f6c/block/bfq-iosched.c#L31)) 。

`BFQ` では、スケジューリングの決定に対して影響を及ぼすことのできる、I/O 優先順位を設定することができます (詳しくは 9.3.3項「`ionice` によるディスクアクセスの優先順位設定」をお読みください) 。

BFQ には下記のようなチューナブルパラメータが用意されています:

表 13.2: BFQ でのチューナブルパラメータ

ファイル	説明
slice_idle	空のキューに対して次の要求が届くまでに待機する時間を、ミリ秒単位で指定します。 既定値は 8 です。
slice_idle_us	slice_idle と同じ意味ですが、こちらはマイクロ秒単位で指定します。 既定値は 8000 です。
low_latency	BFQ を低遅延モードで動作させる (1) か動作させない (0) かを設定します。低遅延モードで動作させると、特定のアプリケーション (たとえば対話的なアプリケーション) を低遅延で動作させることができるようになります。 既定値は 1 です。
back_seek_max	後方シーク処理に対する最大値 (キロバイト単位) を指定します。 既定値は 16384 です。
back_seek_penalty	後方シーク処理でのコスト値の算出に使用する値です。 既定値は 2 です。
fifo_expire_async	非同期要求の期限切れを設定する際に使用する値 (ミリ秒単位) です。 既定値は 250 です。
fifo_expire_sync	同期要求の期限切れを設定する際に使用する値 (ミリ秒単位) です。 既定値は 125 です。
timeout_sync	タスク (キュー) が選択されてから処理されるまでの最大時間をミリ秒単位で指定します。 既定値は 124 です。

ファイル	説明
`max_budget`	`timeout_sync` 内で処理する最大セクタ数を制限するための設定です。この値を `0` に設定すると、`BFQ` は `timeout_sync` の値と予測されるピークレートを元に内部計算を行うようになります。 既定値は `0` (自動チューニング) です。
`strict_guarantees`	`BFQ` において、特定の条件下で帯域の共有をより厳密に行うための処理を行う (1) か行わない (0) かを設定します。 既定値は `0` です。

13.2.4 KYBER

`KYBER` は遅延時間の削減を重視した I/O スケジューラです。読み込みや同期書き込みに対して遅延の目標値を設定し、この目標値に適合するように I/O 要求の流量を調整します。

表 13.3: KYBER でのチューナブルパラメータ

ファイル	説明
`read_lat_nsec`	読み込み処理での遅延目標値をナノ秒単位で指定します。 既定値は `2000000` です。
`write_lat_nsec`	書き込み処理での遅延目標値をナノ秒単位で指定します。 既定値は `10000000` です。

13.3 I/O バリアのチューニング

ext3, ext4 などのファイルシステムでは、fsync やトランザクションのコミット時に、ディスクに対して書き込みバリアを送信します。書き込みバリアは書き込みの順序を保証するための仕組みで、これによって揮発性のあるディスクの書き込みキャッシュを安全に使用できるようにしています (ただし、これによって少しの性能劣化があります) 。お使いのディスクに何らかの方式によるバッテリーが搭載されている場合、バリアを無効化しても、安全に性能を改善することができます。

重要: XFS での nobarrier の廃止予定について

XFS における `nobarrier` オプションは廃止される予定であり、openSUSE Leap 15.2 もしくはそれ以降のバージョンでは正しいマウントオプションではなくなっていることにご注意ください。XFS のファイルシステムをマウントする際、このオプションを指定してしまうと、マウントが失敗してしまいます。この問題を回避するには、スクリプトや fstab でマウントオプションを指定する際、`nobarrier` オプションを含むことがないようにしてください。

書き込みバリア送信の無効化は、`nobarrier` マウントオプションで行うことができます。

警告: バリアの無効化によるデータ損失の危険性について

電源障害時にキャッシュからディスクへの書き込みが正しく保証されない環境でバリアを無効化すると、ファイルシステムの破壊やデータ損失が発生することがあります。

14 タスクスケジューラのチューニング

改訂履歴

2026-09-03

カーネルソースに添付されているスケジューラ統計情報のファイル名は `sched-stats.rst` であったため修正。

2024-06-25

openSUSE® Leap のような現代型のオペレーティングシステムでは、通常の状態でも複数の処理を同時に動作させることができます。たとえばテキストファイル内を検索しながら、同時に電子メールを受信し、さらに外付けのハードディスクから巨大なファイルをコピーしたりすることができます。また、これらのようなシンプルな処理であっても、システム内では様々な追加のプロセスが動作します。それぞれの処理に対して必要なリソースを提供するため、Linux カーネルではツールを利用して、個別の処理にリソースを配分することができます。これが タスクスケジューラ の仕組みです。

下記の章では、プロセスのスケジューリングに関する最も重要な用語について説明しているほか、openSUSE Leap で使用されているタスクスケジューラのポリシーやスケジューリングのアルゴリズム、タスクスケジューラに関する説明なども提供しています。また、関連する情報源へのリンクも用意されています。

14.1 概要

Linux カーネルは、システム内で管理されているタスク (もしくはプロセス) の管理方法を制御します。タスクスケジューラは プロセススケジューラ と呼ばれることもありますが、これは次に動作させるタスクを決定するカーネル内の構成部品を指します。複数のタスクが同時に実行できることを保証するため、システムのリソースを最適に使用する責任を負っています。このような構造により、タスクスケジューラはマルチタスク型のオペレーティングシステムの中枢を成しています。

14.1.1 プリエンプション

タスクスケジューリングでの考え方はシンプルです。システム内に動作可能なプロセスが存在すれば、それらのうちの少なくとも 1 つを動作させます。また、プロセッサ数よりも多く動作可能なプロセスが存在した場合、同時にそれら全てを動作させることはできません。

そのため、プロセスによっては一時的に停止させる (サスペンドさせる) 必要があることになり、後で実行させる必要があることになります。タスクスケジューラは、キュー内にあるどのプロセスを次に実行させるのかを決定します。

上述のとおり、他の Unix 系オペレーティングシステムと同様に、Linux は マルチタスク型の (複数のタスクを同時に動作させることのできる) オペレーティングシステムです。Linux は プリエンプティブマルチタスクと呼ばれる構造で、スケジューラ側でプロセスの一時停止を決定します。このような強制的な一時停止を プリエンプション (横取り等の意味) と呼びます。全ての Unix 系オペレーティングシステムは、その登場当初からプリエンプティブマルチタスクを提供しています。

14.1.2 タイムスライス

それぞれのプロセスに対して与えられる時間枠で、その時間が経過するまではタスクスケジューラ側から一時停止を指示されない、最小単位の時間を指します。タイムスライスとは、各プロセスに対して提供されるプロセッサ時間の量を表す値でもあります。タイムスライスを割り当てておくことで、タスクスケジューラは動作中のシステムに対して大まかに決定を下しながら、かつプロセスがプロセッサを占有しないようにすることができます。

14.1.3 プロセスの優先順位

スケジューラは各プロセスに割り当てられた優先順位を元にして判断を行います。プロセスの現在の優先順位を計算するにあたって、タスクスケジューラは複雑なアルゴリズムを使用します。その結果、各プロセスにはプロセッサの能力に応じた実行の「許可」が与えられます。

14.2 プロセスの分類

プロセスは目的や振る舞いに応じて分類されます。明確な区切りがあるとは限りませんが、一般的には 2 種類の分類が行われます。これらの基準はそれぞれ個別に判断されるべきもので、お互いに排他関係にあるわけではありません。

1 つの考え方として、I/O バウンド と プロセッサバウンド に分類するやり方があります。

I/O バウンド
: I/O とは入出力の意味で、キーボードやマウス、光学ドライブやハードディスクなどを意味します。I/O バウンドプロセス とは、リクエストの送受信の待機に多くの時間を費やすプロセスの意味で、非常に頻繁に動作するものの、同様に I/O リクエストを待機している他のプロセスの動作を妨害することがないよう、その動作は短くなります。

プロセッサバウンド
: プロセッサバウンド のタスクでは、コードの実行に多くの時間を費やし、スケジューラによって一時停止させられるまで動き続けるようなものを指します。I/O 要求を待機しているようなプロセスの動作を妨害することはありませんが、より長い時間にわたって動作し続ける特徴があります。

もう 1 つの考え方として、プロセスを 対話型, バッチ, リアルタイム の 3 種類に分類するやり方もあります。

- 対話型 のプロセスでは、キーボードやマウス操作など、I/O のリクエスト待機に多くの時間を費やします。スケジューラはユーザ側からの要求が届き次第、すぐにこれらのプロセスを起床させなければなりません。そうでないと、ユーザ側にはプログラムの応答が無くなったかのように見えてしまうからです。一般的には 100 [ミリ秒] 程度の許容遅延が設定されます。対話型プロセスには、たとえばオフィスアプリケーションやテキストエディタ、イメージ操作プログラムなどが該当します。

- バッチ のプロセスは裏で動作するものであり、応答性については特に求められません。通常はスケジューラ側で低い優先順位が設定されます。バッチプロセスには、マルチメディアデータの変換処理やデータベースの検索エンジン、ログファイルの検索機能などが該当します。

- リアルタイム のプロセスは、低い優先順位のプロセスによって動作を妨害されてはならないプロセスで、スケジューラ側では短い応答時間となるように設定されます。リアルタイムプロセスには、マルチメディアコンテンツの編集アプリケーションなどが該当します。

14.3 完全公平型スケジューラ (Completely Fair Scheduler)

Linux カーネルバージョン 2.6.23 およびそれ以降では、プロセスの実行を制御するスケジューリングに対して新しいアプローチが追加され、完全公平型スケジューラ (Completely Fair Scheduler (CFS)) が既定の Linux カーネルのスケジューラとして設定されるようになりました。それ以降重要な変更や改善が行われていますが、本章での説明は openSUSE Leap のカーネルバージョン 2.6.32 およびそれ以降 (バージョン 3.x を含む) に対応しています。スケジューラ環境は複数のパーツから構成されていますが、主として 3 種類の機能が追加されています:

モジュール型のスケジューラコア
: スケジューラの中枢部分は スケジューリングクラス で拡張されています。これらのクラスはモジュール型で、それぞれがスケジューリングポリシーを表しています。

完全公平型スケジューラ (Completely Fair Scheduler)
: カーネルバージョン 2.6.23 で追加され、2.6.24 で拡張された CFS は、プロセッサ時間をプロセッサに対して「公平に」分け与える機能をもたらすものです。

グループスケジューリング
: たとえばプロセスをユーザごとに分けてグループ化することで、CFS はユーザ単位で公平なプロセッサ時間を提供するように設定することができます。

これらのことから、CFS はサーバ環境でもデスクトップ環境でも最適なスケジューリングを提供できるようになっています。

14.3.1 CFS の仕組み

CFS は実行可能なそれぞれのタスクに対して、公平な割り振りを行おうとします。タスクスケジューリングで最も公平なやり方として、CFS は 赤黒木 の考え方を採用しています。赤黒木は平衡二分探索木の一種で、項目の挿入や削除を合理的な方法で提供し、平衡性を保ったままにすることができるものです。

CFS がタスクをスケジュールする際、「仮想ランタイム」もしくは「vruntime」と呼ばれる値を足していきます。実行する次のタスクを選択する際には、それまでに足された最小の vruntime のタスクを使用します。タスクが 実行キュー (次に実行すべきプロセスの予定時刻表) 内に挿入された場合、赤黒木は平衡を保つように動作するため、最小の vruntime 値を持つタスクが赤黒木内では最初の項目となるように動きます。

タスクに対して足される vruntime の量は、その優先順位に対応して決められます。高い優先順位のタスクは、低い優先順位のタスクよりも遅い頻度で vruntime が足される仕組みであるため、より高い優先順位のタスクほど、より頻繁に処理が行われることになります。

14.3.2 プロセスのグループ化

Linux カーネルバージョン 2.6.24 では、CFS が個別のタスクよりもグループに対して公平となるよう調整されました。実行可能なタスクはエンティティ (タスクの集合) としてグループ化され、CFS では個別のタスクではなく、エンティティに対して公平になるように動作するようになっています。なおスケジューラ側では、エンティティ内の個別タスクに対しても、公平になるように処理を行います。

カーネルスケジューラは、コントロールグループを利用して実行可能なタスクをグループ化しています。詳しくは 第10章「カーネルコントロールグループ」をお読みください。

14.3.3 カーネルの設定オプション

タスクスケジューラの基本的な動作に関わる要素は、カーネルの設定オプションを介して設定することができます。これらのオプションの設定は、カーネルのコンパイル時のオプションとして用意されています。カーネルのコンパイル作業は複雑な作業となるため、ここでは説明していません。詳しくは別途の資料をお探しください。



警告: カーネルのコンパイルについて

openSUSE Leap に同梱されていないカーネル (たとえばご自身でコンパイルしたカーネル) をお使いの場合、サポートを全く受けられなくなることにご注意ください。

14.3.4 用語

タスクスケジューリングポリシーに関する文書では、いくつかの技術用語を用いる箇所があります。これらの用語の意味を知っておくことで、正しく文書を読み取ることができるようになります。下記にそれらのうちのいくつかを示します:

遅延 (レイテンシ)
: プロセスに対して実行するようにスケジュールされてから、実際にその実行が行われるまでの遅延時間を意味します。

粒度
: 粒度と遅延の関係は、下記の数式で表すことができます:

```
gran = ( lat / rtasks ) - ( lat / rtasks / rtasks )
```

gran は粒度を、lat は遅延を、rtasks は実行中のタスク数をそれぞれ表しています。

14.3.4.1 スケジューリングポリシー

Linux カーネルは下記のスケジューリングポリシーに対応しています:

SCHED_FIFO
: 特別なタイムクリティカルな (時間的に制約の厳しい) アプリケーション向けのスケジューリングポリシーです。先入れ先出し (First In-First Out; FIFO) のスケジューリングアルゴリズムを使用します。

SCHED_BATCH
: CPU に負荷が集中するタスク向けに設計されたスケジューリングポリシーです。

SCHED_IDLE
: 非常に 優先順位の低いタスク向けのスケジューリングポリシーです。

SCHED_OTHER
: ほとんどのプロセスに対して適用される、既定の Linux 時間共有スケジューリングポリシーです。

SCHED_RR

`SCHED_FIFO` に似ていますが、ラウンドロビン型のスケジューリングアルゴリズムを使用するポリシーです。

14.3.5 `chrt` によるプロセスのリアルタイム属性の変更

`chrt` コマンドは、実行中のプロセスに対するリアルタイムスケジューリング属性を設定したり取得したりすることができるほか、指定した属性で指定したコマンドを実行することもできます。プロセスのスケジューリングポリシーのほか、優先順位の取得や設定も行うことができます。

下記の例では、PID が 16244 のプロセスを対象にしています。

既存のタスクからリアルタイム属性を 取得 するには、下記のように入力して実行します:

```
# chrt -p 16244
pid 16244's current scheduling policy: SCHED_OTHER
pid 16244's current scheduling priority: 0
```

プロセスに対して新しいスケジューリングポリシーを設定する前に、まずは各スケジューリングアルゴリズムで設定可能な最小および最大の優先順位を確認しておきます:

```
# chrt -m
SCHED_SCHED_OTHER min/max priority : 0/0
SCHED_SCHED_FIFO min/max priority : 1/99
SCHED_SCHED_RR min/max priority : 1/99
SCHED_SCHED_BATCH min/max priority : 0/0
SCHED_SCHED_IDLE min/max priority : 0/0
```

上記の例では、SCHED_OTHER, SCHED_BATCH, SCHED_IDLE の各ポリシーに対しては優先順位 0 のみを設定することができ、SCHED_FIFO, SCHED_RR の各ポリシーに対しては 1 から 99 までを設定することができることを表しています。

SCHED_BATCH スケジューリングポリシーを設定するには、下記のようにします:

```
# chrt -b -p 0 16244
pid 16244's current scheduling policy: SCHED_BATCH
pid 16244's current scheduling priority: 0
```

`chrt` に関する詳細については、マニュアルページ (`man 1 chrt`) をお読みください。

14.3.6 `sysctl` による稼働中のチューニング

稼働中にカーネルのパラメータを調査したり変更したりすることのできる `sysctl` インターフェイスを利用して変数を変更することで、タスクスケジューラの既定の動作を変更することができます。`sysctl` の書式はシンプルですが、いずれのコマンドとも `root` で実行しなければなりません。

カーネル変数の値を表示するには、下記のように入力して実行します:

```
# sysctl 変数名
```

値を設定するには、下記のように入力して実行します:

```
# sysctl 変数名=値
```

スケジューラ関連の変数を全て表示したい場合は `sysctl` コマンドを利用し、`grep` で出力をフィルタリングします:

```
# sysctl -A | grep "sched" | grep -v "domain"
kernel.sched_cfs_bandwidth_slice_us = 5000
kernel.sched_child_runs_first = 0
kernel.sched_compat_yield = 0
kernel.sched_latency_ns = 24000000
kernel.sched_migration_cost_ns = 500000
kernel.sched_min_granularity_ns = 8000000
kernel.sched_nr_migrate = 32
kernel.sched_rr_timeslice_ms = 25
kernel.sched_rt_period_us = 1000000
kernel.sched_rt_runtime_us = 950000
kernel.sched_schedstats = 0
kernel.sched_shares_window_ns = 10000000
kernel.sched_time_avg_ms = 1000
kernel.sched_tunable_scaling = 1
kernel.sched_wakeup_granularity_ns = 10000000
```

なお、「_ns」や「_us」で終わる名前の変数は、値をナノ秒単位やマイクロ秒単位で指定するものであることを示しています。

最も重要なタスクスケジューラ関連の `sysctl` チューニング変数 (`/proc/sys/kernel/` からもアクセスすることができます) と、その簡潔な説明を下記に示します:

`sched_cfs_bandwidth_slice_us`

: CFS の帯域制御が使用されている場合、このパラメータはタスクのコントロールグループの帯域プールから実行キューに転送されるランタイムの量 (帯域幅) を制御することができます。小さい値であるほど、全体の帯域幅をタスク間できめ細かく共有できるようになりますし、大きい値であるほど、転送にかかるオーバーヘッドを減らすことができます。詳しくは https://www.kernel.org/doc/Documentation/scheduler/sched-bwc.txt (英語) をお読みください。

`sched_child_runs_first`

: 新しく fork された子プロセスは、親プロセスの実行が再開されるよりも早く実行が行われます。このパラメータを `1` にすると、子プロセスが fork 後すぐに処理を始めるような場合に有用となります。

`sched_compat_yield`

古い O(1) スケジューラに存在していた、放棄タスクを実行可能キューの末尾 (赤黒木で一番右側) に移動させる積極的な CPU の yielding 動作を有効にするかどうかを設定します。`sched_yield(2)` システムコールの動作に依存するアプリケーションでは、競合の激しいリソース (たとえばロックなど) が存在するような場合、他のプロセスに対して実行のチャンスを与えることになることから、性能面での改善をもたらす場合があります。ただし、このシステムコールはコンテキストスイッチによって発生するものであるため、誤って使用してしまうと逆に性能が落ちてしまう場合があります。そのため、性能面での改善が確実である場合にのみ設定してください。既定値は `0` です。

`sched_migration_cost_ns`

直近の実行が行われたのち、特定のタスクがマイグレーションの決定に際して「キャッシュホット」であると判断する時間量を設定します。「ホット」なタスクほど、他の CPU にマイグレーション (移行) することが少なくなりますので、この値を増やすことでタスクマイグレーションを少なくすることができます。既定値は `500000` [ナノ秒] です。

実行可能なプロセスが存在しているにもかかわらず、CPU のアイドル時間が予想よりも長い場合は、この値を小さくすることをお勧めします。逆に、CPU 間やノード間でのタスクのマイグレーションが過剰である場合、大きくすることをお勧めします。

`sched_latency_ns`

CPU バウンドのタスクに対して、ターゲットとするプリエンプション遅延を指定します。この値を増やすと、CPU バウンドのタスクのタイムスライスが長くなります。タスクのタイムスライスは、スケジューリングの間隔と重み付けの設定から、下記のように公平な共有が行われます:

`タイムスライス` = `スケジュール間隔` * (`タスクの重み` / `実行キュー内にあるタスクの重みの合計`)

タスクの重みはタスクの nice レベルとスケジューリングポリシーによって決まります。SCHED_OTHER の場合、最小のタスクの重みは 15 で、これは nice 値 19 に対応します。最大のタスクの重みは 88761 で、これは nice 値 -20 に対応します。

タイムスライスは負荷が増えるほど小さくなります。実行可能なタスクの数が `sched_latency_ns` / `sched_min_granularity_ns` を越えると、タイムスライスは `実行可能なタスクの数` * `sched_min_granularity_ns` になります。それより前に、タイムスライスは `sched_latency_ns` に等しくなります。

この値は、実行権情報の計算のために、休眠中のタスクが実行中であると見なされる最大の時間幅を指定することにもなります。また、この値を増やすことで、動作中のタスクが消費する可能性のある時間を増やすことになります。そのため、CPU バウンドのタスクに対しては、スケジューラによる遅延を増やすことになります。既定値は `6000000` [ナノ秒] です。

`sched_min_granularity_ns`

CPU バウンドのタスクに対するプリエンプションの最小粒度です。詳しくは `sched_latency_ns` をご覧ください。既定値は `750000 * (1 + ilog(ncpus))` ナノ秒です。ここで、`ilog(ncpus)` はオンラインの CPU 数に対して、2 を底とする整数対数です。たとえば 2 個の CPU が搭載されたシステムでは、既定値は `1500000` ナノ秒になります。そのため、`sysctl` で報告される実際の値はマシン固有の値となり、CPU 数と `sched_tunable_scaling` の値によって異なるものになります。

`sched_wakeup_granularity_ns`

起床プリエンプション粒度を表します。この変数の値を増やすと、起床時のプリエンプションを減らし、計算処理が主となるタスクの乱れを減らすことになります。この値を減らすと、遅延の回避が重要となるタスクにおいて起床時の遅延とスループットを改善することができます。特に短いデューティサイクルの負荷コンポーネントが、CPU バウンドのコンポーネントと確実に競合するような場合に有用です。既定値は `1000000 * (1 + ilog(ncpus))` ナノ秒です。ここで、`ilog(ncpus)` はオンラインの CPU 数に対して、2 を底とする整数対数です。たとえば 2 個の CPU が搭載されたシステムでは、既定値は `2000000` ナノ秒になります。そのため、`sysctl` で報告される実際の値はマシン固有の値となり、CPU 数と `sched_tunable_scaling` の値によって異なるものになります。



警告: 正しい起床粒度値の選択について

`sched_latency_ns` の半分よりも大きな値を設定してしまうと、起床時のプリエンプションが発生しなくなってしまいます。また、短いデューティサイクルのタスクが CPU を多く使用するタスクに打ち勝てなくなります。

`sched_rr_timeslice_ms`

SCHED_RR のタスクが、他のタスクに奪われたり、タスクリストの末尾に回されたりすることなく実行できる時間単位を表します。

`sched_rt_period_us`

リアルタイムタスクの帯域強制を測定する時間を表します。既定値は `1000000` [マイクロ秒] です。

`sched_rt_runtime_us`

sched_rt_period_us の間にリアルタイムタスクに割り当てられる時間単位です。-1 を設定すると、リアルタイムタスクの帯域強制を行わなくなります。既定では、リアルタイムタスクは 95 [%CPU/sec] を消費するため、5 [%CPU/sec] もしくは 0.05 [s] が SCHED_OTHER のタスクに残されることになります。既定値は `950000` [マイクロ秒] です。

`sched_nr_migrate`

負荷分散の目的で、プロセッサ間でタスクをマイグレーション (移行) できるタスクの数を制御します。負荷分散では割り込みを無効化 (ソフト IRQ) して実行キューを列挙するため、リアルタイムのタスクに対しては IRQ 遅延のペナルティを招く可能性があります。そのため、この値を増やすと、巨大な SCHED_OTHER のスレッドに対して性能を向上させる代わりに、リアルタイムタスクの IRQ 遅延が増えることになります。既定値は `32` です。

`sched_time_avg_ms`

このパラメータは、リアルタイムのタスクの実行に費やす時間の平均値を設定します。この平均値は CFS に対して、負荷分散の決定を支援する値となるほか、CPU が高い優先順位のリアルタイムタスクでどれだけ忙しくなっているのかを表す値にもなっています。
このパラメータに対する最適な設定値は、負荷の内容に大きく依存するものであるほか、リアルタイムのタスクをどれだけ頻繁に、どれだけ長く実行するのかにもよります。



警告: いくつかのスケジューラ関連のパラメータが debugfs に移行されている件について

お使いのシステムで、既定の Linux カーネルバージョンが 5.13 もしくはそれ以降の場合 (`rpm -q kernel-default` コマンドを実行すると確認できます) 、下記のようなメッセージがカーネルログ内に記録されていることに気がつくかもしれません:

```
[ 20.485624] The sched.sched_min_granularity_ns sysctl was moved to debugfs in
 kernel 5.13 for CPU scheduler debugging only. This sysctl will be removed in a
 future SLE release.
[ 20.485632] The sched.sched_wakeup_granularity_ns sysctl was moved to debugfs in
 kernel 5.13 for CPU scheduler debugging only. This sysctl will be removed in a
 future SLE release.
```

上記のメッセージは、6 種類のスケジューラ関連パラメータが `/proc/sys/kernel/sched_*` から `/sys/kernel/debug/sched/*` に移動している旨を表したメッセージです。6 種類のパラメータは下記のとおりです:

- `sched_latency_ns`
- `sched_migration_cost_ns`
- `sched_min_granularity_ns`
- `sched_nr_migrate`
- `sched_tunable_scaling`
- `sched_wakeup_granularity_n`

上述のメッセージのとおり、これらのスケジューラ関連パラメータは移行期間中として現行の openSUSE Leap の `sysctl` 内で提供され続けています。ですが、将来的には CPU スケジューラの実装が更新される予定であることから、将来バージョンの openSUSE Leap で上記パラメータが `sysctl` で提供され続ける保証はなく、`debugfs` 経由で利用する必要があるかもしれないことに注意してください。

もしもお使いのシステム内で、これら 6 種類のパラメータのうちいずれかを使用している場合は、目的に合わせて別の方式を使用するようにし、特に本番環境ではこれらに依存した作りにしないようにすることを強くお勧めします。

14.3.7 デバッグ用インターフェイスとスケジューラの統計情報

CFS では新しく改善されたデバッグインターフェイスが用意され、稼働中の統計情報を提供することができるようになっています。これらのファイルは `/proc` ファイルシステム内に用意されていますので、`cat` や `less` などのコマンドを使用することで、簡単に調べることができるようになっています。下記にはタスクスケジューラ関連の `/proc` ファイルと、その簡潔な説明を示しています:

`/proc/sched_debug`

タスクスケジューラの動作に関わる全てのチューニング可能な変数 (詳しくは 14.3.6項「`sysctl` による稼働中のチューニング」をご覧ください) のほか、CFS の統計情報や全てのプロセッサでの実行キュー (CFS, RT, deadline) に対する現在の値を含んでいるファイルです。各プロセッサで動作しているタスクの概要についても、タスク名と PID 、そしてスケジューラ固有の統計情報を表示することができます。最初の列には `tree-key` 列が表示されていますが、これはタスクの仮想ランタイムを示すもので、その名前は赤黒木内で全ての実行可能なタスクを並べ替える際のキーでもあります。また `switches` 列には、スイッチの合計数 (involuntary であるものを含む) を示しています。さらに、`prio` 列にはプロセスの優先順位が、`wait-time` にはタスクがスケジュールされるまでに待機した時間の合計値が、`sum-exec` と `sum-sleep` には、それぞれそのタスクをプロセッサ内で実行した時間の合計と、休眠 (スリープ) していた時間の合計がナノ秒単位で示されています。

```
# cat /proc/sched_debug
Sched Debug Version: v0.11, 6.4.0-150600.9-default #1
ktime                                   : 23533900.395978
sched_clk                               : 23543587.726648
cpu_clk                                 : 23533900.396165
jiffies                                 : 4300775771
sched_clock_stable                      : 0

sysctl_sched
  .sysctl_sched_latency                    : 6.000000
```

```
  .sysctl_sched_min_granularity            : 2.000000
  .sysctl_sched_wakeup_granularity         : 2.500000
  .sysctl_sched_child_runs_first           : 0
  .sysctl_sched_features                   : 154871
  .sysctl_sched_tunable_scaling            : 1 (logarithmic)

cpu#0, 2666.762 MHz
  .nr_running                    : 1
  .load                          : 1024
  .nr_switches                   : 1918946
[...]

cfs_rq[0]:/
  .exec_clock                    : 170176.383770
  .MIN_vruntime                  : 0.000001
  .min_vruntime                  : 347375.854324
  .max_vruntime                  : 0.000001
[...]

rt_rq[0]:/
  .rt_nr_running                 : 0
  .rt_throttled                  : 0
  .rt_time                       : 0.000000
  .rt_runtime                    : 950.000000

dl_rq[0]:
  .dl_nr_running                 : 0

  task   PID         tree-key  switches  prio     wait-time        [...]
-------------------------------------------------------------------------
R  cc1 63477     98876.717832       197   120      0.000000         ...
```

`/proc/schedstat`

現在の実行キューに関連する統計情報を表示します。SMP システムの場合、接続されている全てのプロセッサに対して、ドメイン固有の統計情報も表示することができます。出力形式はわかりにくい形式であるため、詳しくは `/usr/src/linux/Documentation/scheduler/sched-stats.rst` をお読みください。

`/proc/PID/sched`

`PID` で指定したプロセスのスケジューリング情報を表示することができます。

```
# cat /proc/$(pidof gdm)/sched
gdm (744, #threads: 3)
-------------------------------------------------------------------
se.exec_start                                :          8888.758381
se.vruntime                                  :          6062.853815
se.sum_exec_runtime                          :             7.836043
se.statistics.wait_start                     :             0.000000
```

```
se.statistics.sleep_start                       :          8888.758381
se.statistics.block_start                       :             0.000000
se.statistics.sleep_max                         :          1965.987638
[...]
se.avg.decay_count                              :                8477
policy                                          :                   0
prio                                            :                 120
clock-delta                                     :                 128
mm->numa_scan_seq                               :                   0
numa_migrations, 0
numa_faults_memory, 0, 0, 1, 0, -1
numa_faults_memory, 1, 0, 0, 0, -1
```

14.4 さらなる情報

Linux のタスクスケジューリングに関して、よりコンパクトな知識を得たい場合は、下記のような情報源をお読みになることをお勧めします:

- タスクスケジューラ関連のシステムコールの説明については、それぞれ対応するマニュアルページ (例: `man 2 sched_setaffinity`) をお読みください。
- Linux のスケジューラポリシーおよびアルゴリズムに関するわかりやすい説明については、https://www.inf.fu-berlin.de/lehre/SS01/OS/Lectures/Lecture08.pdf (英語) をお読みになることをお勧めします。
- Linux のプロセススケジューリングに関する概要については、Robert Love 氏による Linux Kernel Development (ISBN-10: 0-672-32512-8) (英語) がよいでしょう。詳しくは https://www.informit.com/articles/article.aspx?p=101760 をご覧ください。
- Linux カーネルの内部に関する広範囲な概要を知りたい場合は、Daniel P. Bovet 氏と Marco Cesati 氏による Understanding the Linux Kernel (ISBN 978-0-596-00565-8) (英語) がよいでしょう。
- タスクスケジューラに関する技術的な情報については、`/usr/src/linux/Documentation/scheduler` 内にあるファイルをお読みください。

15 メモリ管理サブシステムのチューニング

改訂履歴

2024-06-25

カーネルのメモリ管理動作を理解してチューニングするには、まず動作に関する概要と、他のサブシステムとの連携を理解するところから始めるのがよいでしょう。

メモリ管理サブシステムは仮想メモリマネージャとも呼ばれ、下記では「VM」と略しています。VM の役割は、カーネル全体とユーザプログラムに対する物理メモリ (RAM) の割り当て管理です。それだけではなく、ユーザプロセスに対して、仮想メモリ環境を提供する責任も負っています (Linux 拡張付きの POSIX API を介して管理します) 。最後に、VM はメモリが枯渇した場合に、キャッシュを解放したり「匿名」メモリをスワップアウトしたりすることで、メモリを空ける処理も行います。

VM の調査やチューニングに際して理解しておくべき最重要事項は、キャッシュの管理方法についてです。VM のキャッシュの基本的なゴールは、スワップやファイルシステムの操作(ネットワークファイルシステムを含みます) を行うことによって生成される I/O のコストを、最小化することにあります。これは I/O を排除するか、もしくはよりよいパターンで I/O を送信することによって達成します。

空きメモリは、必要に応じてキャッシュとして使用されます。キャッシュや匿名メモリとして使用できるメモリ領域が大きければ大きいほど、より効率的にキャッシュやスワップを制御できるようになります。しかしながら、いったんメモリが枯渇してしまった場合は、キャッシュを解放するか、メモリをスワップアウトする必要があります。

適切な負荷範囲であれば、性能を改善するにあたって最初にやるべきことは、メモリを増やしてキャッシュの効率を上げ、キャッシュの解放やスワップアウトの頻度を減らすのがよいでしょう。次にやるべきことは、カーネルのパラメータを変更して、キャッシュの管理方法を変えることです。

最後に、負荷それ自身についても調査してチューニングする必要があります。アプリケーション側で多くのプロセスやスレッドを動作させることができる場合、各プロセスがファイルシステム内の独自の領域を利用していると、VM の効率が落ちてしまいます。また、メモリによるオーバーヘッドも増えてしまいます。アプリケーション側で独自のバッファやキャッシュを持っている場合、そのキャッシュを大きくしてしまうと、VM 側に割り当てることのできるサイズが小さくなってしまいます。しかしながら、プロセスやスレッドの数を増やすことで、I/O の重複度合いやパイプライン処理の機会が増えることがありますので、これによってマルチコア環境で性能が向上することもあります。従って、最適な結果を得るためには、実験が必要ということになります。

15.1 メモリの用途

メモリの割り当ては「ピン済み」(「回収不可」と呼ばれることもあります),「回収可」,「スワップ可」に分類することができます。

15.1.1 匿名メモリ

匿名メモリはプログラムのヒープやスタックメモリ (例: `>malloc()`) として使用されるメモリで、`mlock` のような特殊ケースや、利用可能なスワップ領域が存在しないような場合を除いて、回収可能なメモリとして位置づけられます。また、匿名メモリは回収される前にスワップに書き込まなければなりません。また、スワップの I/O (ページのスワップインおよびスワップアウト) は、割り当てとアクセスのパターンにより、ページキャッシュの I/O よりも効率が落ちる傾向があります。

15.1.2 ページキャッシュ

ファイルデータのキャッシュです。ディスクやネットワークからファイルが読み込まれると、その内容がページキャッシュ内に保存されます。ページキャッシュ内の内容が最新のものであれば、ディスクやネットワークへのアクセスは不要となります。また、tmpfs や共有メモリセグメントについても、ページキャッシュとしてカウントします。

ファイルに対して書き込みが行われる場合、ディスクやネットワークに書き戻される (つまりライトバックキャッシュを作成する) 前に、ページキャッシュにも保存が行われます。まだディスクやネットワークに書き込まれていない新しいデータが存在する場合、そのページは「dirty」であるとされます。逆に dirty ではないページは、「clean」であるとされます。clean やページキャッシュのページは、メモリが枯渇した場合、単純に解放するだけで回収することができます。dirty なページの場合は、回収できるようにするためにまず clean にしなければなりません。

15.1.3 バッファキャッシュ

ブロックデバイス (例: /dev/sda) に対するページキャッシュの一種です。ファイルシステムでは、inode テーブルやアロケーションビットマップなど、ディスク内のメタデータにアクセスする際にバッファキャッシュを使用します。バッファキャッシュはページキャッシュと同様に回収することができます。

15.1.4 バッファヘッド

バッファヘッドは小さな補助構造で、一般的にページキャッシュへのアクセス時に割り当てられるものです。ページキャッシュやバッファキャッシュが clean であれば、これらも一般に容易に回収することができます。

15.1.5 ライトバック

アプリケーションがファイルに書き込みを行う場合、ページキャッシュが dirty となるほか、バッファキャッシュについても必要に応じて dirty となります。dirty なメモリの量が指定したページ数 (バイト単位で vm.dirty_background_bytes で指定します) を越えるか、メモリの全体量との比率が指定した値 (vm.dirty_background_ratio) を越えるか、もしくは指定した時間 (vm.dirty_expire_centisecs) より長く dirty であり続けた場合、カーネルは最初に dirty になったページのファイルからページの書き戻しを始めます。なお、bytes の設定と ratio のパラメータは相互に排他な仕組みであり、一方を変更すると他方を上書きすることになります。また、flusher スレッドは裏で書き戻しを行う仕組みであるため、アプリケーションは通常通り動作し続けることができます。ただし、I/O の速度よりもアプリケーションがページを dirty にする速度のほうが早い場合、dirty なデータが致命的な水準 (vm.dirty_bytes もしくは vm.dirty_ratio) を越えると、アプリケーションの速度が抑制され、dirty なデータが過剰に生み出されないようにします。

15.1.6 先読み

VM はファイルのアクセスパターンを監視し、必要に応じて先読みを行おうとします。先読みはその名前の通り、まだ要求されていない範囲のものをファイルシステムからページキャッシュにページを読み込むものです。これは、より少ない回数でより大きな I/O を送信できるようにする (これによって効率化を図る) ためのものです。また、I/O をパイプライン化する (アプリケーションの実行と同時に I/O を実施する) ためのものでもあります。

15.1.7 VFS キャッシュ

15.1.7.1 inode キャッシュ

これは各ファイルシステムに対する inode の構造をメモリ内にキャッシュしておくものです。この中にはファイルサイズやパーミッション、所有者情報やファイルデータに対するポインタなどを保持しています。

15.1.7.2 ディレクトリエントリキャッシュ

これはシステム内でのディレクトリエントリのメモリ内キャッシュです。この中には名前 (ファイル名) のほか、それが指し示す inode と、子エントリが含まれます。このキャッシュは、ディレクトリ構造をたどる場合と、名前でファイルにアクセスする場合に使用されるものです。

15.2 メモリ使用量の削減

15.2.1 malloc (匿名) 利用の削減

openSUSE Leap 15.7 で動作しているアプリケーションは、以前のバージョンに比べてより多くのメモリを割り当てることができます。これは `glibc` がユーザスペースメモリの割り当てに際して、既定の動作を変更したことによるものです。これらのパラメータについて、詳しくは https://www.gnu.org/s/libc/manual/html_node/Malloc-Tunable-Parameters.html (英語) をお読みください。

以前のバージョンの動作に戻したい場合、M_MMAP_THRESHOLD の値を 128*1024 に設定する必要があります。これはアプリケーション側から mallopt() を利用することによって実現できるほか、アプリケーションの起動前に `MALLOC_MMAP_THRESHOLD_` 環境変数を設定しても実現することができます。

15.2.2 カーネルのメモリオーバーヘッドの削減

回収可能なカーネルメモリ (上述のとおりキャッシュなど) については、メモリの枯渇時に自動的に解放が行われます。その他のカーネルメモリについては容易に縮小することができますが、カーネルに与えられた負荷の特性によって決まります。

ユーザスペースの負荷要件を減らす (プロセスを減らす、ファイルやソケットを減らすなど) ことで、カーネルのメモリ使用量も削減することができます。

15.2.3 メモリコントローラ (メモリ cgroup)

メモリ cgroup の機能が不要である場合は、カーネルのコマンドラインに cgroup_disable=memory を追加することで、無効化を行うことができます。これにより、カーネルが使用するメモリを少しだけ削減することができます。また、メモリ cgroup が利用できるにもかかわらず設定していない場合、少しだけメモリのオーバーヘッドが存在するため、少しだけ性能改善をはかることもできます。

15.3 仮想メモリマネージャ (VM) のチューニングパラメータ

VM をチューニングする場合、パラメータが実際の処理に反映されるまでには、しばらく時間がかかるものがあることに注意してください。また、負荷が 1 日を通して変化するような場合、時間帯によっては異なる振る舞いになることもあります。それだけでなく、特定の環境でスループットを改善できるパラメータが、別の環境ではスループットを悪化させてしまうようなこともあります。

15.3.1 回収比率に関するパラメータ

`/proc/sys/vm/swappiness`

この変数は、ページキャッシュやその他のキャッシュに比べて、カーネルがどれだけの比率で匿名メモリを積極的にスワップアウトさせるかを指定するものです。この値を増やすことで、スワップ量が増えることになります。既定値は `60` です。

スワップの I/O は一般に、その他の I/O よりもずっと効率が落ちるものです。しかしながら、ページキャッシュのページによっては、あまり使用されない匿名メモリよりも、頻繁にアクセスされるものがあります。ここでは適切な比率を設定してください。

速度が低下した際にスワップの動作が観測できた場合、このパラメータを減らしてみることをお勧めします。また、多くの I/O 動作が存在する環境で、システム内のページキャッシュの量が比較的少ない場合や、動作しているものの休眠中の巨大なアプリケーションが存在するような環境では、この値を増やすことで性能を改善できるかもしれません。

ただし、データのスワップアウト量が増えれば増えるほど、必要とされた際にスワップアウトされたデータを取り戻すのに時間がかかることに注意してください。

`/proc/sys/vm/vfs_cache_pressure`

この変数は、ページキャッシュやスワップに比べて、カーネルが VFS キャッシュに使用しているメモリの回収傾向を制御するためのものです。この値を増やすと、VFS キャッシュの回収比率を高めることができます。

ただし、試してみる以外の方法では、適切な値を推測することは難しい値でもあります。`slabtop` コマンド (`procps` パッケージ内に含まれています) を使用することで、カーネル側で使用しているメモリオブジェクトを多い順に並べることができます。このコマンド内で、vfs キャッシュは "dentry" と "*_inode_cache" として表示されます。ページキャッシュに比べてこれらのメモリが巨大になっている場合、この圧力値を増やしてみることをお勧めします。これにより、スワップ処理を減らすことにもなります。既定値は `100` です。

`/proc/sys/vm/min_free_kbytes`

この変数は、「アトミックな」(回収を待つことができない) 割り当てなど、特別に予約しておく必要のあるメモリ量を制御します。これは通常、メモリ使用率について注意深くチューニングしている場合を除いて、減らすべきではない値です (通常はサーバ用途ではなく、組み込み用途で設定すべき値です) 。もしもログ内に「ページ割り当て失敗」に関するメッセージとスタックトレースが頻繁に現れているような場合、min_free_kbytes をエラーが出なくなる程度まで増やしてみることをお勧めします。このようなメッセージが頻繁に現れたりしていなければ、特に気にする必要はありません。既定値は搭載されている RAM の量に従って決められます。

`/proc/sys/vm/watermark_scale_factor`

大まかに言うと、空きメモリには高／低／最小の水準が設定されています。低い水準に達した場合、`kswapd` が動作して裏でメモリの回収を行うようになります。この回収作業は、高いほうの水準に達するまで続けられます。最小の水準に達した場合、アプリケーションの動作は一時的に停止させられます。

`watermark_scale_factor` は、kswapd が動き出す際のノードもしくはシステム内のメモリ量を表す値と、そこから kswapd が休眠状態に戻るまでにどれだけの量のメモリを解放するのかを表す値です。単位は 10,000 の対する比率で、既定値の 10 は 水準間の長さがノード／システム内で利用できるメモリの 0.1% 分であることを示します。最大値は 1000 で、10% 分を表します。

直接的な回収処理で処理速度が遅くなってしまっている場合、`/proc/vmstat` 内の `allocstall` の値が増えますが、この場合はこのパラメータを変更することで、問題を回避できるかもしれません。同様に `kswapd` が早く休眠状態になってしまう場合、`kswapd_low_wmark_hit_quickly` の値が増えますが、この場合はアプリケーションの一時停止を回避するために空いているページ数が、少なすぎることを表しています。

15.3.2 ライトバック (書き戻し) 関連のパラメータ

以前の openSUSE Leap バージョンからのライトバック関連の主な変更点として、ファイルに結びつけられた mmap() メモリが、即時に dirty なメモリとして判断されるようになった (つまりライトバックの対象となる) ことがあげられます。以前のバージョンでは、munmap() でマップが解除された場合や msync() システムコールが呼び出された場合、もしくはメモリへの圧力が大きい場合にのみ書き戻しが行われていました。

アプリケーションによっては、このような書き戻し動作への変更を期待していないものもあり、場合によっては性能が低下してしまうことがあります。

`/proc/sys/vm/dirty_background_ratio`

この変数は空き領域や回収可能なメモリの全体量の割合を表すものです。この割合以上に dirty なページキャッシュが存在していると、ライトバックスレッドが dirty なメモリを書き込み始めるようになります。既定値は `10` (%) です。

`/proc/sys/vm/dirty_background_bytes`

この変数は、裏で動作するカーネルのライトバックスレッドが、その書き込みを始める割合を表すものです。`dirty_background_bytes` は `dirty_background_ratio` に対応するもので、一方を設定すると他方が自動的に `0` に設定されます。

`/proc/sys/vm/dirty_ratio`

`dirty_background_ratio` に似た意味を持つ割合値です。ここで指定した割合を超過すると、ページキャッシュに対して書き込みを行いたいアプリケーションの動作が一時的に止められ、カーネルのライトバックスレッドが dirty なメモリを書き込んで clean に戻すまで待機するようになります。既定値は `20` (%) です。

`/proc/sys/vm/dirty_bytes`

この変数は `dirty_ratio` と同じようなチューニングパラメータですが、ここでは割合ではなくバイト単位でサイズを指定します。`dirty_ratio` と `dirty_bytes` は同じチューニングパラメータであることから、一方を設定すると他方が自動的に `0` になります。`dirty_bytes` に設定可能な最小値は 2 ページ分 (ただしバイト単位) で、それより小さい値を設定しようとしても、それは無視されて元の値に戻ってしまいます。

`/proc/sys/vm/dirty_expire_centisecs`

この変数は、ここで設定した値よりも長い時間 dirty であり続けたメモリが存在した場合、次回のライトバックスレッドの起床時に書き込みが行われるようになるものです。ここでの期限設定はファイルの inode に設定された最終更新日時を基準にします。そのため、同じファイルに対して発生した複数の dirty ページが存在した場合、この期限を超過するとそれら全てが書き込まれるようになります。

`dirty_background_ratio` と `dirty_ratio` は、いずれもページキャッシュのライトバック動作を設定するためのものです。これらの値を増やした場合、システム内に dirty なメモリがより多く、かつ長く保持されることになります。システム内により多くの dirty なメモリを保持できる環境であれば、ライトバックの I/O を減らして、より最適な I/O パターンでの書き込みを増やすために、これらの値を活用できる場合があります。ただし dirty なメモリが多く存在していると、メモリを回収する必要がある場合や、ディスクに書き込む必要が発生した場合に一貫性を確保するタイミング (「同期ポイント」) で、遅延が発生する場合があります。

15.3.3 先読み関連のパラメータ

`/sys/block/ブロックデバイス/queue/read_ahead_kb`

1 つもしくは複数のプロセスがファイルを順次読み込みしている場合、カーネルはそれらのデータを前もって読み込み (先読みし) 、プロセスがデータを待つ時間を減らすように処理を行います。先読みのデータ量は、どれだけ順に I/O を行っているのかに従って、動的に計算されます。このパラメータは、カーネルが先読みを行う際、1 ファイルあたりの最大データ量を設定するためのものです。巨大なファイルの順次読み込みが十分に早いものであるとは思えない場合、この値を増やしてみることをお勧めします。ただし、大きくしすぎると、先読みスラッシングと呼ばれる現象が発生し、先読みで使用したページキャッシュが、使用される前に回収されてしまうことがあります。また、意味のない I/O が発生して速度が落ちてしまうこともあります。既定値は `512` [KB] です。

15.3.4 Transparent HugePage 関連のパラメータ

Transparent HugePages (THP) は動的な huge page の割り当て方法で、プロセス側から要求によって割り当てたり、後から `khugepaged` カーネルスレッドを介して遅延割り当てを行ったりするためのものです。この方式は `hugetlbfs` の使用とは区別されていて、こちらは割り当てと使用を手作業で管理する方式です。連続したメモリのアクセスパターンが存在する負荷の場合、THP によって大きく性能を改善できる可能性があります。連続したメモリのアクセスパターンで一括処理を行うと、ページフォルトを 1000 倍減少させることもできてしまいます。

逆に THP が望ましくない場合もあります。たとえばメモリ内のあちこちをアクセスするようなパターンの場合、メモリの使用量が増える結果になってしまうため、かえって性能が落ちてしまいます。たとえばフォルトごとに 4 KB ではなく、フォルト発生時点で 2 MB のメモリを使用してしまうことがあり、最終的にはページの回収を早める結果になってしまいます。

THP の動作は `transparent_hugepage=` のカーネルパラメータか、sysfs 経由で設定することができます。たとえばカーネルのパラメータに `transparent_hugepage=never` を追加して grub2 の設定を再構築し、システムを再起動します。すると、下記のように入力して実行することで、THP が無効化されていることを確認できます:

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
always madvise [never]
```

無効化されている場合、上記の例のように `never` が [] で囲まれて表示されます。`always` に設定すると THP を常に使用するようになりますが、割り当てが失敗した場合は `khugepaged` に従います。`madvise` に設定すると、アプリケーション側で明示的に指定されたアドレス空間に THP を割り当てるだけになります。

`/sys/kernel/mm/transparent_hugepage/defrag`

このパラメータは、THP を割り当てる際にアプリケーションがどれだけの努力を行うのかを制御するものです。`always` は openSUSE 42.1 およびそれ以前 の THP に対応するリリースでの既定値でした。もしも THP が利用できない場合、アプリケーションはメモリのデフラグを実施しようとします。つまり THP が利用できない場合、メモリがフラグメントされていると、アプリケーションの動作が潜在的に一時的ながら停止する可能性があることになります。

値に `madvise` を設定すると、THP の割り当てリクエストでは、アプリケーション側から明示的に要求された場合にのみ、デフラグを実施します。こちらが openSUSE 42.2 およびそれ以降のバージョンでの既定値となります。

`defer` は openSUSE 42.2 およびそれ以降で利用できるようになった値で、THP が利用できない場合には小さなページを使用して処理するように動作します。これにより `kswapd` と `kcompactd` の各カーネルスレッドを起床させ、裏でデフラグを行うことで、後から `khugepaged` が THP を割り当てる動作になります。

最後の値である `never` は、THP が利用できない場合には単純に小さなページを利用して処理を行うもので、それ以外の処理は行わない意味になります。

15.3.5 khugepaged のパラメータ

khugepaged は `transparent_hugepage` の値が `always` もしくは `madvise` である場合に自動的に開始され、`never` である場合には自動的に停止します。通常は低頻度で動作するものではありますが、動作をチューニングすることもできます。

`/sys/kernel/mm/transparent_hugepage/khugepaged/defrag`

値に 0 を設定すると、フォルトの時点で THP が使用されていても `khugepaged` を無効化します。これは遅延に敏感なアプリケーションにとっては重要な設定で、THP による利点を受けるものの、`khugepaged` がアプリケーションのメモリ使用を更新しようとする際に、一時的に動作が止まってしまう事象を防ぐことができるからです。

`/sys/kernel/mm/transparent_hugepage/khugepaged/pages_to_scan`

この変数は、`khugepaged` が 1 回の処理でスキャンするページ数を制御するためのものです。スキャン処理では小さいページを検出して、THP で再割り当てができないかどうかを確認します。この値を増やすことで、CPU の使用率が上がるものの、裏でより高速に THP を割り当てることができるようになります。

`/sys/kernel/mm/transparent_hugepage/khugepaged/scan_sleep_millisecs`

この変数は、`khugepaged` が 1 回の処理を完了した後に待機する時間を設定するためのもので、CPU の使用率が過剰に上がらないようにするためのものです。この値を小さくすると、CPU の使用率が上がる代わりに、裏でより高速に THP を割り当てることができるようになります。

`/sys/kernel/mm/transparent_hugepage/khugepaged/alloc_sleep_millisecs`

この変数は、`khugepaged` が `kswapd` と `kcompactd` が動作するのを待っている間、裏で THP が割り当てに失敗した際に休眠する時間を指定するためのものです。

`khugepaged` に対するその他のパラメータは、性能チューニングにあたってはほとんど用途のないものではありますが、`/usr/src/linux/Documentation/vm/transhuge.txt` ファイル (英語) で詳しく説明しています。

15.3.6 その他の VM パラメータ

VM に関連するチューニングパラメータに関する完全な一覧については、`/usr/src/linux/Documentation/sysctl/vm.txt` ファイル (英語, `kernel-source` パッケージ内に含まれています) をお読みください。

15.4 VM の動作監視

VM の動作の監視には、下記のようなシンプルなツールを使用することができます:

1. vmstat: このツールは、VM が現在何をしているのかをわかりやすく表示することができます。詳しくは 2.1.1項「`vmstat`」をお読みください。

2. `/proc/meminfo` : このファイルは、メモリの使途を分解して示しているファイルです。詳しくは 2.4.2項「詳細なメモリ使用率情報の取得: `/proc/meminfo`」をお読みください。

3. `slabtop` : このツールは、カーネルの slab メモリについて、その使用状況を詳細に表示することができます。buffer_head, dentry, inode_cache, ext3_inode_cache などの値が主なキャッシュです。このコマンドは `procps` パッケージ内に含まれています。

4. `/proc/vmstat` : このファイルには VM の内部動作を詳しく分解して表示したものが含まれています。含まれている情報は実装に依存して決まるものであり、環境によって表示される内容が異なります。いくつかの項目は `/proc/meminfo` でも表示されていますが、それ以外の項目は

ユーティリティを使用してわかりやすく表示するのがよいでしょう。また、情報を最大限に活用するため、このファイルは変化率を観察するために長時間監視しておくことをお勧めします。他の情報源からは導き出すのが難しい主な情報は下記のとおりです:

`pgscan_kswapd_*, pgsteal_kswapd_*`

これらのレポートには、システムが起動してからスキャンしたページ数の合計と、`kswapd` が回収したページ数の合計が書かれています。これらの値の比率は回収効率として解釈することができ、この値が低い場合は、システムがメモリを回収するのに苦労していて、おそらく使用する前に回収されてしまっていることを表しています。処理が軽い場合は、あまり気にする必要はありません。

`pgscan_direct_*, pgsteal_direct_*`

これらのレポートには、アプリケーションが直接スキャンしたページ数の合計と、回収したページ数の合計が書かれています。これは `allocstall` のカウンタ値と相関しています。これらのイベントが多く発生している場合、プロセスの動作が一時的に止まっていることを示すことから、`kswapd` での処理より深刻な問題となります。`kswapd` での処理が重く、`pgpgin` , `pgpout` の値が高いか、もしくは `pswapin` や `pswpout` の値が高い場合は、システムが過剰にスラッシングしている (実際に使用される前に回収されてしまっている) ことを表しています。
より詳しい情報を得たい場合は、トレースポイントを使用してください。

`thp_fault_alloc, thp_fault_fallback`

これらのカウンタは THP がアプリケーションから直接割り当てられた回数と、THP が利用できずに小さいページを使用した回数を表しています。thp_fault_fallback の値が大きい場合でも、アプリケーションが TLB の圧力に敏感でない限り、有害ではありません。

`thp_collapse_alloc, thp_collapse_alloc_failed`

これらのカウンタは、`khugepaged` が割り当てた THP の回数と、THP が利用できずに小さなページを使用した回数を示しています。failed の割合が高い場合、システムがフラグメント状態にあり、アプリケーション側が許可したメモリ使用量であるにもかかわらず、THP が使用されていないことを表しています。アプリケーションが TLB の圧力に敏感である場合にのみ、問題となる項目です。

`compact_*_scanned, compact_stall, compact_fail, compact_success`

これらのカウンタは THP が有効化され、システムがフラグメントしている場合に増えるものです。`compact_stall` は THP の割り当てに際してアプリケーションの動作が一時的に停止した場合に増加します。それ以外のカウンタは、スキャンしたページの数と、成功もしくは失敗したデフラグイベントの数を表しています。

16 ネットワークのチューニング

改訂履歴

2024-06-25

ネットワークサブシステムは複雑な構造であるため、チューニングはシステムの用途に大きく依存するほか、ソフトウエアクライアントやハードウエアコンポーネント (スイッチ、ルータ、ゲートウエイ) などの外部要素にも依存します。Linux カーネルでは、オーバーヘッドを低くしたり高いスループットを提供したりする代わりに、信頼性と遅延の少なさを主眼に置いてします。また、その他の設定はセキュリティを低下させますが、性能を改善することができるようになっています。

16.1 カーネルのソケットバッファの設定

昨今のネットワーク通信の多くは TCP/IP プロトコルをベースにして行われていて、実際の処理はソケットインターフェイスを使用するのが一般的です。TCP/IP に関する詳細については、**『リファレンス』、第13章「ネットワークの基礎」**をお読みください。Linux カーネルでは、ソケットインターフェイスを介して、バッファ内に受信したデータや送信すべきデータを蓄積し、必要な送受信を行います。これらのカーネルのソケットバッファについては、チューニングによる調整を行うことができます。

重要: TCP の自動チューニングについて

カーネルバージョン 2.6.17 もしくはそれ以降のバージョンでは、最大バッファサイズ 4 MB で自動チューニングを行うようになっています。この仕組みにより、手作業でチューニングを行ったとしても、通常はネットワーク性能を改善できないことを意味しています。また下記の変数についても、通常は変更せずにそのままにしておくことが最適です。変更を行う場合は、チューニング作業による影響をよく確認しておくことをお勧めします。

また、古いバージョンのカーネルから更新した場合は、手作業で行っていた TCP のチューニングを削除し、自動チューニングに任せることをお勧めします。

`/proc` ファイルシステム内にある特殊なファイルを使用することで、カーネルのソケットバッファのサイズや動作を変更することができます。`/proc` ファイルシステムに関する一般的な情報については、2.6項「/proc ファイルシステム」をお読みください。このうち、ネットワーク関連のファイルは下記の中に含まれています:

```
/proc/sys/net/core
/proc/sys/net/ipv4
/proc/sys/net/ipv6
```

一般的な `net` 関係の変数は、カーネルのドキュメンテーション (`linux/Documentation/sysctl/net.txt`) 内に説明があります。また、`ipv4` 以下の変数は、`linux/Documentation/networking/ip-sysctl.txt` と `linux/Documentation/networking/ipvs-sysctl.txt` 内に説明があります。

`/proc` ファイルシステム内では、たとえば全てのプロトコルに対する最大ソケット受信バッファサイズや最大ソケット送信バッファサイズを設定することができます。この場合、これらを TCP プロトコルにのみ適用することができる (`ipv4` 内) ほか、全てのプロトコルに対して上書きで設定することのできるもの (`core` 内) もあります。

`/proc/sys/net/ipv4/tcp_moderate_rcvbuf`

: `/proc/sys/net/ipv4/tcp_moderate_rcvbuf` を `1` に設定すると、自動チューニング機能が有効化され、バッファサイズが動的に調整されるようになります。

`/proc/sys/net/ipv4/tcp_rmem`

: 3 種類の値を設定する変数で、1 接続あたりのメモリ内受信バッファの最小値／初期値／最大値をそれぞれ設定します。ここでは TCP のウインドウサイズだけでなく、実際のメモリ使用量を調整することにもなります。

`/proc/sys/net/ipv4/tcp_wmem`

: `tcp_rmem` と同じような変数ですが、こちらは 1 接続あたりのメモリ内の送信バッファを設定します。

`/proc/sys/net/core/rmem_max`

: アプリケーション側から要求することのできる、最大の受信バッファサイズを制限するための変数です。

`/proc/sys/net/core/wmem_max`

: アプリケーション側から要求することのできる、最大の送信バッファサイズを制限するための変数です。

`/proc` を利用することで、不要な TCP 機能を無効化することができます (既定では全ての TCP 機能が有効化されています) 。たとえば下記のようなファイルがあります:

`/proc/sys/net/ipv4/tcp_timestamps`

: TCP のタイムスタンプ機能は、RFC1323 で規定されているものです。

`/proc/sys/net/ipv4/tcp_window_scaling`

: TCP のウインドウスケーリングについても、RFC1323 で規定されています。

`/proc/sys/net/ipv4/tcp_sack`

: 選択的確認応答 (SACK) の設定を行います。

`sysctl` コマンドを使用することで、`/proc` ファイルシステム内の変数を読み込んだり書き込んだりすることができます。`sysctl` コマンドは `/etc/sysctl.conf` ファイルから設定を読み込む仕組みであり、これによってシステムを再起動しても設定を再適用することができるため、`cat` (読み込み) や `echo` (書き込み) を利用して `/proc` ファイルシステムにアクセスするよりは、`sysctl` コマンドを使用することをお勧めします。`sysctl` コマンドでの変数の読み込みや書き込みは簡単で、たとえば下記のように入力して実行することで、TCP 関連の変数を全て表示することができます:

```
> sudo sysctl -a | grep tcp
```



注記: ネットワーク変数のチューニングによる副次的な影響について

ネットワーク変数のチューニングによって、CPU やメモリなどの他のシステムリソースに影響がある場合があります。

16.2 ネットワーク内でのボトルネックの発見とネットワークトラフィックの分析

ネットワークのチューニングを行う前に、あらかじめネットワーク内にボトルネックが存在していないかを確認し、ネットワークのトラフィックパターンについても調べておくことが重要です。これらを実施するために、いくつかのツールが提供されています。

ネットワークトラフィックを分析するには、`netstat` , `tcpdump` , `wireshark` などのツールをお使いいただくことができます。Wireshark はネットワークトラフィックアナライザです。

16.3 netfilter

Linux におけるファイアウオール機能やマスカレード機能は、netfilter と呼ばれるカーネルモジュールが提供する機能です。このモジュールは、ルールベースのフレームワークら介して、高度に設定することができます。あるパケットが指定したルールに該当した場合、netfilter ではパケットを受け付けるか拒否するかを選択することができるほか、特殊なアクション (「ターゲット」と呼びます) を指定して、アドレス変換などの処理を行うこともできます。

netfilter には多数の設定項目が存在しています。そのため、ルールを多く設定すればするほど、パケットの処理にも時間がかかることになります。また、高度な接続追跡機能を使用すると、より CPU 負荷がかかることになりますので、ネットワーク全体の処理の低下にも繋がります。

カーネル側のキューがいっぱいになると、新しく届いたパケットが廃棄されるようになります。これにより、既存の接続が正しく動作しなくなってしまいます。'故障時開' (fail-open) の機能を使用すると、ネットワークトラフィックが多すぎる場合、一時的にパケットの調査を無効化して接続を維持することができるようになります。詳しくは https://home.regit.org/netfilter-en/using-nfqueue-and-libnetfilter_queue/ (英語) をお読みください。

詳しくは netfilter/iptables プロジェクトの Web ページ https://www.netfilter.org (英語) をご覧ください。

16.4 Receive Packet Steering (RPS) によるネットワーク性能の改善

新しいネットワークインターフェイスを使用している場合、多数のパケットを取り扱うことになることから、ホスト側が性能面のボトルネックとなる場合があります。これらのパケットを問題なく扱うことができるようにするため、システム側では複数の CPU コアに分散させて処理を行わなければなりません。

新しいネットワークインターフェイスの場合、ハードウエア側に実装された複数の送受信キューを使用することで、複数の CPU コアに処理を分散させることができます。ハードウエア側にそのような仕組みが用意されておらず、単一のキューしか用意されていない場合、ドライバは単一の直列化されたストリームを介して、全ての到着パケットを処理しなければならなくなります。この問題に対応するには、オペレーティングシステムがストリームを「並列化」して、複数の CPU に処理を分散させなければなりません。このような分散処理の仕組みが Receive Packet Steering (RPS) です。RPS は仮想環境でも使用することができます。

RPS は各データストリームに対して、IP アドレスとポート番号をベースにしたユニークなハッシュを作成します。このハッシュを使用することで、同じデータストリームを同じ CPU で処理できるようにし、性能を向上させることができるようになっています。

RPS はネットワークデバイスの受信キューおよびインターフェイスごとに設定することができます。設定のファイル名は、下記のようになっています:

```
/sys/class/net/デバイス名/queues/受信キュー名/rps_cpus
```

デバイス名 にはネットワークデバイスのデバイス名が入ります。たとえば `eth0` , `eth1` のような名前になります。また、*受信キュー名* には、受信キューの名前が入ります。たとえば `rx-0` , `rx-1` のような名前になります。

ネットワークインターフェイスが単一の受信キューしか提供していない場合は、`rx-0` のみが存在することになります。複数の受信キューに対応している場合は、rx-`N` のディレクトリが複数存在していることになります。

下記の設定ファイルには、CPU をビットマップ形式で指定します。既定では全てのビットが `0` になっています。この設定では、RPS が無効化されているため、割り込みを処理した CPU がパケットキューの処理をも行うことになります。

RPS を有効化し、特定の CPU がインターフェイスの受信キューを処理するように設定するには、その CPU のビットを `1` に設定します。たとえば CPU 0 から 3 までを eth0 の最初の受信キューの処理に使用したい場合は、ビット 0 から 3 までを 1 にした値、つまり 2 進数で `00001111` を設定します。なお、実際の設定作業は、16 進数で指定します。この場合は `F` を設定することになります。このことから、設定作業は下記のようになります:

```
> sudo echo "f" > /sys/class/net/eth0/queues/rx-0/rps_cpus
```

CPU 8 から 15 までを有効化したい場合は、下記のように設定します:

```
1111 1111 0000 0000 (2 進数)
15     15    0    0 (10 進数)
F       F    0    0 (16 進数)
```

生成できた 16 進数値は `ff00` になりますので、下記のコマンドで設定します:

```
> sudo echo "ff00" > /sys/class/net/eth0/queues/rx-0/rps_cpus
```

NUMA マシンの場合、RPS を設定してインターフェイスの受信キュー割り込みの処理と同じ NUMA ノードの CPU で処理を行うようにすると、最適な性能を発揮できるようになります。

非 NUMA マシンの場合、全ての CPU を使用することができます。割り込みの割合が大きい場合、ネットワークインターフェスを処理している CPU を除外することで、性能を発揮できるようになります。ネットワークインターフェイスを処理している CPU は、`/proc/interrupts` ファイルから判断することができます。たとえば下記のようになります:

```
> sudo cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
...
  51:  113915241          0          0          0      Phys-fasteoi   eth0
...
```

この場合、`eth0` の割り込みを処理しているのは `CPU 0` になります。これは、`CPU0` のみが 0 より大きくなっているためです。

x86 および AMD64/Intel 64 プラットフォームの場合、`irqbalance` を使用することで、ハードウエア割り込みを CPU 間に分散させることができます。詳しくは `man 1 irqbalance` をお読みください。

VI システムダンプの処理

17 トレーシングツール 172

18 Kexec と Kdump 182

19 アプリケーションクラッシュ時の systemd-coredump の使用 200

17 トレーシングツール

改訂履歴

2023-12-22

openSUSE Leap には、お使いのシステムに関して情報の採取を行うためのツールがいくつか用意されています。この情報採取では、たとえばプログラム内での問題を解析 (デバッグ) したり検出したり、性能面での問題点となっている箇所の発見や、プログラムの動作中に使用しているシステムリソースを見つけたりすることができます。



注記: トレーシングによる性能面への影響について

動作中のプロセスに対してシステムコールやライブラリコールの監視を行うと、そのプロセスの性能が大幅に落ちることになります。そのため、トレーシングツールの使用は、データ収集の際にのみ実施することをお勧めします。

17.1 strace によるシステムコールの追跡

`strace` コマンドは、プロセスが使用したシステムコールと、プロセスが受信したシグナルを追跡するためのコマンドです。`strace` では新しくプロセスを起動してシステムコールを追跡することができるほか、既に実行されているコマンドに接続して、システムコールを追跡することもできます。コマンドの出力の各行にはシステムコールの名前のほか、括弧でくくられたパラメータの一覧と返り値も表示されます。

新しくコマンドを起動してシステムコールの追跡を開始したい場合は、通常のコマンドラインの前に `strace` を入れて実行します:

```
> strace ls
execve("/bin/ls", ["ls"], [/* 52 vars */]) = 0
brk(0)                                  = 0x618000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) \
        = 0x7f9848667000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) \
        = 0x7f9848666000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT \
(No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=200411, ...}) = 0
mmap(NULL, 200411, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f9848635000
close(3)                                = 0
open("/lib64/librt.so.1", O_RDONLY)     = 3
[...]
```

```
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) \
= 0x7fd780f79000
write(1, "Desktop
Documents
bin
inst-sys
", 31Desktop
Documents
bin
inst-sys
) = 31
close(1)                                = 0
munmap(0x7fd780f79000, 4096)            = 0
close(2)                                = 0
exit_group(0)                           = ?
```

既に起動しているプロセスに対して `strace` を接続するには、接続先のプロセスのプロセス ID (`PID`) を `-p` パラメータで指定します:

```
> strace -p `pidof cron`
 Process 1261 attached
 restart_syscall(<... resuming interrupted call ...>) = 0
  stat("/etc/localtime", {st_mode=S_IFREG|0644, st_size=2309, ...}) = 0
  select(5, [4], NULL, NULL, {0, 0})      = 0 (Timeout)
  socket(PF_LOCAL, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 5
  connect(5, {sa_family=AF_LOCAL, sun_path="/var/run/nscd/socket"}, 110) = 0
  sendto(5, "\2\0\0\0\0\0\0\5\0\0\0root\0", 17, MSG_NOSIGNAL, NULL, 0) = 17
  poll([{fd=5, events=POLLIN|POLLERR|POLLHUP}], 1, 5000) = 1 ([{fd=5, revents=POLLIN|
POLLHUP}])
  read(5, "\2\0\0\0\1\0\0\0\5\0\0\0\2\0\0\0\0\0\0\0\0\0\0\5\0\0\0\6\0\0\0"..., 36) = 36
  read(5, "root\0x\0root\0/root\0/bin/bash\0", 28) = 28
  close(5)                                = 0
  rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
  rt_sigaction(SIGCHLD, NULL, {0x7f772b9ea890, [], SA_RESTORER|SA_RESTART,
 0x7f772adf7880}, 8) = 0
  rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
  nanosleep({60, 0}, 0x7fff87d8c580)      = 0
  stat("/etc/localtime", {st_mode=S_IFREG|0644, st_size=2309, ...}) = 0
  select(5, [4], NULL, NULL, {0, 0})      = 0 (Timeout)
  socket(PF_LOCAL, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 5
  connect(5, {sa_family=AF_LOCAL, sun_path="/var/run/nscd/socket"}, 110) = 0
  sendto(5, "\2\0\0\0\0\0\0\5\0\0\0root\0", 17, MSG_NOSIGNAL, NULL, 0) = 17
  poll([{fd=5, events=POLLIN|POLLERR|POLLHUP}], 1, 5000) = 1 ([{fd=5, revents=POLLIN|
POLLHUP}])
  read(5, "\2\0\0\0\1\0\0\0\5\0\0\0\2\0\0\0\0\0\0\0\0\0\0\5\0\0\0\6\0\0\0"..., 36) = 36
  read(5, "root\0x\0root\0/root\0/bin/bash\0", 28) = 28
  close(5)
  [...]
```

`-e` オプションはいくつかのサブオプションを指定するための仕組みです。たとえばファイルに対する open や write だけを追跡したい場合は、下記のように入力して実行します:

```
> strace -e trace=open,write ls ~
open("/etc/ld.so.cache", O_RDONLY)       = 3
open("/lib64/librt.so.1", O_RDONLY)      = 3
open("/lib64/libselinux.so.1", O_RDONLY) = 3
open("/lib64/libacl.so.1", O_RDONLY)     = 3
open("/lib64/libc.so.6", O_RDONLY)       = 3
open("/lib64/libpthread.so.0", O_RDONLY) = 3
[...]
open("/usr/lib/locale/cs_CZ.utf8/LC_CTYPE", O_RDONLY) = 3
open(".", O_RDONLY|O_NONBLOCK|O_DIRECTORY|O_CLOEXEC) = 3
write(1, "addressbook.db.bak
bin
cxoffice
"..., 311) = 311
```

ネットワーク関連のシステムコールのみを追跡したい場合は、`-e trace=network` を指定します:

```
> strace -e trace=network -p 26520
Process 26520 attached - interrupt to quit
socket(PF_NETLINK, SOCK_RAW, 0)         = 50
bind(50, {sa_family=AF_NETLINK, pid=0, groups=00000000}, 12) = 0
getsockname(50, {sa_family=AF_NETLINK, pid=26520, groups=00000000}, \
[12]) = 0
sendto(50, "\24\0\0\0\26\0\1\3~p\315K\0\0\0\0\0\0\0\0", 20, 0,
{sa_family=AF_NETLINK, pid=0, groups=00000000}, 12) = 20
[...]
```

`-c` オプションを指定すると、それぞれのシステムコールでカーネルが費やした時間を計算することができます:

```
> strace -c find /etc -name xorg.conf
/etc/X11/xorg.conf
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 32.38    0.000181         181         1           execve
 22.00    0.000123           0       576           getdents64
 19.50    0.000109           0       917        31 open
 19.14    0.000107           0       888           close
  4.11    0.000023           2        10           mprotect
  0.00    0.000000           0         1           write
[...]
  0.00    0.000000           0         1           getrlimit
  0.00    0.000000           0         1           arch_prctl
  0.00    0.000000           0         3         1 futex
  0.00    0.000000           0         1           set_tid_address
  0.00    0.000000           0         4           fadvise64
```

```
  0.00    0.000000           0         1           set_robust_list
------ ----------- ----------- --------- --------- ----------------
100.00    0.000559                  3633        33 total
```

子プロセスについても追跡を行いたい場合は、`-f` を指定します:

```
> strace -f systemctl status apache2.service
execve("/usr/bin/systemctl", ["systemctl", "status", "apache2.service"], \
 0x7ffea44a3318 /* 56 vars */) = 0
brk(NULL)                               = 0x5560f664a000
[...]
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f98c58a5000
mmap(NULL, 4420544, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f98c524a000
mprotect(0x7f98c53f4000, 2097152, PROT_NONE) = 0
[...]
[pid  9130] read(0, "\342\227\217 apache2.service - The Apache"..., 8192) = 165
[pid  9130] read(0, "", 8027)           = 0
● apache2.service - The Apache Webserver227\217 apache2.service - Th"..., 193
   Loaded: loaded (/usr/lib/systemd/system/apache2.service; disabled; vendor preset:
 disabled)
   Active: inactive (dead)
) = 193
[pid  9130] ioctl(3, SNDCTL_TMR_STOP or TCSETSW, {B38400 opost isig icanon echo ...}) = 0
[pid  9130] exit_group(0)               = ?
[pid  9130] +++ exited with 0 +++
<... waitid resumed>{si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=9130, \
 si_uid=0, si_status=0, si_utime=0, si_stime=0}, WEXITED, NULL) = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=9130, si_uid=0, \
  si_status=0, si_utime=0, si_stime=0} ---
exit_group(3)                           = ?
+++ exited with 3 +++
```

`strace` の出力を分析したいものの、コンソールウインドウ内に表示してしまうと長すぎるような場合は、`-o` オプションをお使いください。この場合、プロセスへの接続や接続終了などのメッセージが省略されます。また、通常は標準出力に書かれるようなメッセージについても、`-q` を指定することで省略することができます。このほか、それぞれのシステムコールの行に対して、行頭に時刻を表示させたい場合は、`-t` を指定します:

```
> strace -t -o strace_sleep.txt sleep 1; more strace_sleep.txt
08:44:06 execve("/bin/sleep", ["sleep", "1"], [/* 81 vars */]) = 0
08:44:06 brk(0)                         = 0x606000
08:44:06 mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, \
-1, 0) = 0x7f8e78cc5000
[...]
08:44:06 close(3)                       = 0
08:44:06 nanosleep({1, 0}, NULL)        = 0
08:44:07 close(1)                       = 0
08:44:07 close(2)                       = 0
08:44:07 exit_group(0)                  = ?
```

strace の動作や出力形式は変更することができます。詳しくは strace のマニュアルページ (man 1 strace) をお読みください。

17.2 ltrace によるライブラリコールの追跡

`ltrace` はプロセスの動的なライブラリコールを追跡することができます。`strace` と同様の用途で使用するツールであり、ほとんどのパラメータは似通っているか、同じ意味になっています。既定では `ltrace` は、`/etc/ltrace.conf` もしくは `~/.ltrace.conf` にある設定ファイルを使用します。それ以外の設定ファイルを使用したい場合は、`-F 設定ファイル` オプションを指定してください。

`ltrace` で `-S` オプションを指定すると、ライブラリコールに加えて システムコールについても追跡を行うことができます:

```
> ltrace -S -o ltrace_find.txt find /etc -name \
xorg.conf; more ltrace_find.txt
SYS_brk(NULL)                                              = 0x00628000
SYS_mmap(0, 4096, 3, 34, 0xffffffff)                       = 0x7f1327ea1000
SYS_mmap(0, 4096, 3, 34, 0xffffffff)                       = 0x7f1327ea0000
[...]
fnmatch("xorg.conf", "xorg.conf", 0)                       = 0
free(0x0062db80)                                           = <void>
__errno_location()                                         = 0x7f1327e5d698
__ctype_get_mb_cur_max(0x7fff25227af0, 8192, 0x62e020, -1, 0) = 6
__ctype_get_mb_cur_max(0x7fff25227af0, 18, 0x7f1327e5d6f0, 0x7fff25227af0,
0x62e031) = 6
__fprintf_chk(0x7f1327821780, 1, 0x420cf7, 0x7fff25227af0, 0x62e031
<unfinished ...>
SYS_fstat(1, 0x7fff25227230)                                = 0
SYS_mmap(0, 4096, 3, 34, 0xffffffff)                       = 0x7f1327e72000
SYS_write(1, "/etc/X11/xorg.conf
", 19)                    = 19
[...]
```

`-e` オプションを指定することで、追跡するイベントの種類を変更することができます。下記の例では、`fnmatch` と `strlen` に関連したライブラリコールを表示しています:

```
> ltrace -e fnmatch,strlen find /etc -name xorg.conf
[...]
fnmatch("xorg.conf", "xorg.conf", 0)             = 0
strlen("Xresources")                             = 10
strlen("Xresources")                             = 10
strlen("Xresources")                             = 10
fnmatch("xorg.conf", "Xresources", 0)            = 1
strlen("xorg.conf.install")                      = 17
[...]
```

特定のライブラリ内に含まれているシンボルのみを表示したい場合は、`-l ライブラリのパス` オプションを指定します:

```
> ltrace -l /lib64/librt.so.1 sleep 1
clock_gettime(1, 0x7fff4b5c34d0, 0, 0, 0)                    = 0
clock_gettime(1, 0x7fff4b5c34c0, 0xffffffffff600180, -1, 0) = 0
+++ exited (status 0) +++
```

入れ子になった呼び出し構造を読みやすくするため、`-n スペース数` というオプションを指定すると、インデントを設定することもできます。

17.3 Valgrind によるデバッグとプロファイリング

Valgrind はプログラムをデバッグしたりプロファイルしたりするためのツール集で、これによってより高速に、エラーの少ないプログラムを作ることができるようになります。Valgrind はメモリ管理やスレッドに関する問題点を検出することができるほか、新しいデバッグツールを作成するためのフレームワークとしても使用することができます。ただし、このツールはオーバーヘッドが大きく、複数の処理を同時に実行するような状況では、通常よりもずっと遅い速度になってしまうことに注意する必要があります。

17.3.1 一般的な情報

Valgrind の主な利点は、コンパイル済みの実行ファイルの問題を直接検出できる点にあります。プログラムをコンパイルし直したり、修正したりする必要はありません。Valgrind は下記のようにして使用します:

`valgrind` `Valgrind_のオプション` `プログラム名` `プログラムのオプション`

Valgrind には、様々な機能を持つツールが含まれています。本章で説明しているのは非常に一般的なもので、ツールとは無関係に使用できるものばかりです。Valgrind の設定オプションで最も重要なものは `--tool` で、ここではどのツールを使用するのかを指定することができます。このオプションを省略すると、既定では `memcheck` が選択されたものと見なされます。たとえば Valgrind の memcheck ツールで `find ~` `-name` .bashrc を実行したい場合は、下記のように入力して実行します:

`valgrind` `--tool` =memcheck find ~ -name .bashrc

Valgrind で提供されているツールの一覧と、その簡潔な説明は下記のとおりです:

`memcheck`

メモリエラーを検出します。お使いのプログラムの動作をチェックして、正しく動作するように支援するためのツールです。

`cachegrind`
: キャッシュ予測のプロファイルを行います。お使いのプログラムをより高速に動作させるためのヒントを提示する仕組みです。

`callgrind`
: `cachegrind` と同様の処理を行うツールですが、追加でキャッシュプロファイル情報も収集します。

`exp-drd`
: スレッドのエラーを検出します。マルチスレッド型のプログラムの動作をチェックして、正しく動作するように支援する仕組みです。

`helgrind`
: もう 1 つのスレッドエラー検出ツールです。`exp-drd` に似ていますが、問題の分析に際して異なる技術を使用します。

`massif`
: ヒーププロファイラです。ヒープは動的なメモリ管理に使用するメモリ領域で、このツールは少ないメモリ量で動作するように、プログラムをチューニングする支援を行います。

`lackey`
: 基本的な仕組みを説明するためのサンプルツールです。

17.3.2 既定のオプション

Valgrind はその起動時にオプションを読み込みます。Valgrind がオプションをチェックする箇所には、下記の 3 種類のものがあります:

1. Valgrind を起動するユーザのホームディレクトリ内にある、`.valgrindrc` ファイル。
2. 環境変数 `$VALGRIND_OPTS` 。
3. Valgrind を起動する時点でのカレントディレクトリ内にある `.valgrindrc` ファイル。

これらのリソースは上記の順序で処理が行われます。そのため、前のほうのファイルで設定を行っていても、後ろのほうのファイル内に同じ設定があれば、上書きされることになります。また、特定の Valgrind ツールに固有のオプションを指定する場合は、ツールの名前とコロンを付けなければなりません。たとえば `cachegrind` に対して、プロファイルデータを `/tmp/cachegrind_PID.log` に書き込むように指定するには、ホームディレクトリの `.valgrindrc` ファイル内に、下記の内容を記述します:

```
--cachegrind:cachegrind-out-file=/tmp/cachegrind_%p.log
```

17.3.3 Valgrind の動作原理

Valgrind は実行ファイルの起動が行われる前から、その制御を実施します。実行ファイルのデバッグ情報を読み込み、関連する共有ライブラリを読み込みます。実行ファイルのコードは、選択された Valgrind ツールに転送され、ツール側でデバッグ処理のためのコードを追加したあと、Valgrind のコアに戻され、実際の実行が行われます。

たとえば `memcheck` では、それぞれのメモリアクセスに対してチェックを行うコードを追加します。そのため、プログラムは通常の実行よりもずっと遅くなります。

Valgrind はプログラム内の各インストラクションを擬似します。そのため、プログラムのコードをチェックするだけでなく、関連するライブラリ (C ライブラリを含む) やグラフィカル環境で使用されるライブラリまでも、チェックすることができます。Valgrind でエラーを検出しようとすると、関連するライブラリ (C ライブラリ, X11, Gtk ライブラリなど) のエラーも検出することになります。これらのエラーについては通常検出する必要はありませんので、省略設定ファイルを作成して Valgrind 側で省略するように指定することができます。`--gen-suppressions=yes` を指定して実行すると、Valgrind に対して省略用の設定を作成させることができます。

Valgrind のパラメータには実際の実行ファイル (マシンコード) を指定してください。シェルや Perl のスクリプトなどからお使いのアプリケーションを起動してしまうと、`/bin/sh` (または `/usr/bin/perl`) を起動することによる無関係なエラーが表示されることがありますので、この場合は `--trace-children=yes` を指定して、エラーを回避してください。ただし、通常は実行ファイルを直接指定して実行してください。

17.3.4 メッセージ

Valgrind は、その実行中に詳細なエラーメッセージや重要なイベントを報告することがあります。下記にメッセージの出力例を示します:

```
> valgrind --tool=memcheck find ~ -name .bashrc
[...]
==6558== Conditional jump or move depends on uninitialised value(s)
==6558==    at 0x400AE79: _dl_relocate_object (in /lib64/ld-2.11.1.so)
==6558==    by 0x4003868: dl_main (in /lib64/ld-2.11.1.so)
[...]
==6558== Conditional jump or move depends on uninitialised value(s)
==6558==    at 0x400AE82: _dl_relocate_object (in /lib64/ld-2.11.1.so)
==6558==    by 0x4003868: dl_main (in /lib64/ld-2.11.1.so)
[...]
==6558== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
==6558== malloc/free: in use at exit: 2,228 bytes in 8 blocks.
==6558== malloc/free: 235 allocs, 227 frees, 489,675 bytes allocated.
==6558== For counts of detected errors, rerun with: -v
==6558== searching for pointers to 8 not-freed blocks.
```

```
==6558== checked 122,584 bytes.
==6558==
==6558== LEAK SUMMARY:
==6558==    definitely lost: 0 bytes in 0 blocks.
==6558==      possibly lost: 0 bytes in 0 blocks.
==6558==    still reachable: 2,228 bytes in 8 blocks.
==6558==         suppressed: 0 bytes in 0 blocks.
==6558== Rerun with --leak-check=full to see details of leaked memory.
```

上記の Valgrind の出力例には `==6558==` が含まれていますが、これはプロセス ID 番号 (PID) を表しています。これにより、Valgrind のメッセージとプログラム自身の出力を区別できるようになっているほか、Valgrind のメッセージがどのプロセスに対するものであるのかがわかるようにもなっています。

Valgrind のメッセージをより詳細に出力させたい場合は、`-v` もしくは `-v -v` のように指定してください。

Valgrind の出力するメッセージを制御するには、下記のようにして行います:

1. 既定では、Valgrind がメッセージを送信する際、出力先をファイルディスクリプタの 2 番 (つまり標準エラー出力) に設定します。それ以外のファイルディスクリプタ番号を使用したい場合は、`--log-fd=ファイルディスクリプタ番号` のように指定してください。

2. おそらくはこちらの方法のほうが期待されるとは思いますが、もう 1 つの方法として、`--log-file=ファイル名` のように指定することで、ログを指定したファイルに書き込むこともできます。このオプションを指定するにあたっては、いくつかのプレースホルダを使用することができます。たとえばファイル名の一部に `%p` を指定すると、その部分にはプロファイル対象のプログラムのプロセス ID (PID) が埋め込まれるようになります。また、`%q{env_var}` のように指定すると、`env_var` という名前の環境変数の値を展開して出力することができます。
   下記の例では、Apache Web サーバの再起動時にメモリエラーが発生していないかどうかを確認している例となります。この場合、子プロセスも追跡して、詳細な Valgrind メッセージをプロセス ID ごとに分けて別々のファイルに書き込むように指定しています:

   ```
   > valgrind -v --tool=memcheck --trace-children=yes \
   --log-file=valgrind_pid_%p.log systemctl restart apache2.service
   ```

   この処理を実行すると、テストで使用した環境の場合 52 個のログファイルを作成していたほか、通常であれば 7 秒程度で `sudo systemctl restart apache2.service` を処理できていたところ、Valgrind を入れることによって、おおよそ 10 倍の 75 秒程度かかるようになっていました。

   ```
   > ls -1 valgrind_pid_*log
   valgrind_pid_11780.log
   valgrind_pid_11782.log
   valgrind_pid_11783.log
   ```

```
[...]
valgrind_pid_11860.log
valgrind_pid_11862.log
valgrind_pid_11863.log
```

3. Valgrind のメッセージをネットワーク経由で送信したいと思うことがあるかもしれません。そのような場合は、`--log-socket=aa.bb.cc.dd:port_num` のような形式でオプションを指定してください。ここで、`aa.bb.cc.dd` には送信先の IP アドレスを、`port_num` には送信先のポート番号をそれぞれ指定します。ポート番号を省略した場合は、1500 を使用します。
Valgrind のメッセージをネットワークソケット経由で送信する場合、受信側のアプリケーションを用意しないと意味がありません。Valgrind では `valgrind-listener` と呼ばれるシンプルなリスナーが同梱されていて、指定したポートで接続を待ち受けて、受信した内容をそのまま標準出力に出力することができます。

17.3.5 エラーメッセージ

Valgrind では全てのエラーメッセージを記憶していて、新しいエラーを検出すると既存のエラーメッセージと比較を行います。この方法により、Valgrind はエラーを繰り返し表示するような事態を防いでいます。同じエラーが発生した場合、Valgrind は単純にそのメッセージを記録するだけで、メッセージを表示しなくなります。この仕組みにより、似たようなエラーを多数受け取って混乱を起こすことがないようになっています。

`-v` オプションを指定すると、Valgrind の実行出力の最後に全ての概要レポート (発生回数順に並べたもの) を表示することができます。これに加えて、Valgrind では 1000 種類以上のエラーが発生した場合や、合計で 10,000,000 回以上のエラーが発生した場合、エラーの収集処理を停止する機能があります。このような制限を無効化して、全てのエラーメッセージを表示するようにしたい場合は、`--error-limit=no` を指定してください。

エラーによっては他のエラーを引き起こす種類のものもあります。そのため、エラーを修正する場合は、その発生順に行うものとし、プログラムを繰り返しチェックし直してください。

17.4 さらなる情報

- 上記のトレーシングツールに関連する全オプションの一覧を確認するには、それぞれ対応するマニュアルページ (`man 1 strace` , `man 1 ltrace` , `man 1 valgrind`) をお読みください。
- Valgrind の高度な使用方法については、本書では説明していません。詳しい説明を読みたい場合は、Valgrind User Manual (https://valgrind.org/docs/manual/manual.html) (英語) をお読みください。これらのページは Valgrind をより高度に使用したり、標準ツールの使い方や目的を知ったりしたい場合には、必須の情報源です。

18 Kexec と Kdump

改訂履歴

2024-03-14

Kexec は現在動作中のカーネルから他のカーネルを高速に起動するための仕組みです。これにより、ハードウエアの初期化を行わず、高速にシステムを再起動することができます。必要であれば、システムがクラッシュした際、もう 1 つのカーネルを起動することもできます。

18.1 概要

Kexec を使用することで、ハードウエアによる再起動を行うことなく、別のカーネルを起動することができるようになります。このツールは、下記のような用途で使用されます:

- 高速な再起動
  システムを頻繁に再起動する必要があるような場合、Kexec は時間をかなり削減できるようになります。

- 不正なファームウエアやハードウエアの回避
  コンピュータのハードウエアは複雑な構造であり、システムの起動時に深刻な問題が発生する場合があります。このような場合、すぐにハードウエアを交換できればよいのですが、そうは行かない場合があります。Kexec では、ハードウエアの初期化が完了した状態から、特定の制御環境下に移行することができます。これにより、システムの起動が失敗するようなリスクを抑えることができます。

- カーネルがクラッシュした際のダンプの保存
  Kexec は物理メモリ内の情報を保持して動作します。そのため、本番用 のカーネルに何らかの障害が発生した場合、情報採取用 のカーネル (限られたメモリ範囲内だけで動作する追加のカーネル) を起動することで、障害情報を保存することができるようになります。保存されたイメージは、後から解析用に使用することができます。

- GRUB 2 の設定を使用しない起動
  Kexec 経由でカーネルを起動した場合、ブートローダの段階は省略されます。通常はブートローダの設定が誤っていると、起動の処理を行うことができなくなりますが、Kexec を使用することで、ブートローダの設定に依存せずに起動することができるようになります。

18.2 必要なパッケージ

openSUSE® Leap で Kexec を使用すると、再起動を高速化したり潜在的なハードウエア問題を回避したりすることができます。Kexec を使用するには、`kexec-tools` パッケージをインストールしてください。このパッケージには `kexec-bootloader` というスクリプトが含まれていますが、これはブートローダの設定からカーネルのコマンドラインオプションを読み出し、そのコマンドラインオプションを利用して新しいカーネルを起動することができます。

カーネルのクラッシュ時にデバッグ情報を取得する環境を設定したい場合は、`makedumpfile` パッケージをインストールしておいてください。

openSUSE Leap での Kdump の使用方法は、YaST の Kdump モジュールで設定することができます。YaST モジュールを使用する場合は、`yast2-kdump` パッケージをインストールしておいてください。

18.3 Kexec の内部構造

Kexec で最も重要なコンポーネントは `/sbin/kexec` です。Kexec でカーネルを読み込む際、2 種類の方法で行うことができます:

- 通常の再起動と同様に、本番用のカーネルのアドレス領域にカーネルを読み込む方法:

  ```
  # kexec -l カーネルイメージ
  ```

  読み込んだあと、そのカーネルを実行するには、`kexec -e` と入力して実行します。

- メモリ内の専用の領域にカーネルを読み込む方法:

  ```
  # kexec -p カーネルイメージ
  ```

  このカーネルは、システムがクラッシュした際に自動的に起動されます。

システムがクラッシュした際、本番用のカーネルのデータを保持したまま、もう 1 つのカーネルを起動したい場合は、システムメモリ内に専用の領域を用意する必要があります。その領域は常に空けておかなければならないものであることから、この専用領域には本番用のカーネルが読み込まれることはありません。この領域は情報採取用のカーネルが使用するもので、このような仕組みによって、本番用のカーネルのメモリページを保持させることができるようになっています。

このような領域を予約するには、本番用のカーネルの起動コマンドライン内に、`crashkernel` というオプションを追加します。`crashkernel` で必要な値を判断するには、18.4項「crashkernel 割り当てサイズの計算」に書かれた手順に従ってください。

なお、上記のパラメータは情報採取用のカーネルのパラメータではありません。情報採取用のカーネルでは Kexec を使用しません。

情報採取用のカーネルは専用の領域に読み込まれ、カーネルがクラッシュするのを待ちます。いったんクラッシュが発生すると、Kdump は情報採取用のカーネルに制御を移します。これは、クラッシュした時点で、元のカーネルは信頼できる状態ではなくなるためです。これはつまり、Kdump 自身も失敗する可能性があることを示しています。

情報採取用のカーネルを読み込むには、カーネルの起動パラメータを含める必要があります。ほとんどの場合、初期 RAM ファイルシステムを起動時に使用します。初期 RAM ファイルシステムを指定するには、`--initrd = ファイル名` のようなオプションを指定します。なお、`--append = コマンドライン` のように指定すると、起動するほうのカーネルのコマンドラインを設定することができます。

なお、本番用のカーネルで指定していたコマンドラインを含めておく必要があります。このような場合は、`--append = "$(cat /proc/cmdline)"` のように入力して実行することで、既存のコマンドラインをそのまま受け渡すことができます。また、オプションを追加したい場合は、`--append = "$(cat /proc/cmdline) オプション"` のようにして追加のオプションを指定してもかまいません。

たとえば、`/boot/vmlinuz-6.4.0-150600.9-default` というカーネルイメージファイルに対して、現在動作中のカーネルに指定しているものと同じ `/boot/initrd` ファイルを使用するように指定したい場合は、下記のようなコマンドになります:

```
#  kexec -l /boot/vmlinuz-6.4.0-150600.9-default \
 --append="$(cat /proc/cmdline)" --initrd=/boot/initrd
```

読み込んでおいたカーネルはいつでも解放することができます。`-l` オプションで読み込んでおいたカーネルを解放したい場合は、`kexec -u` コマンドを実行してください。また、`-p` オプションで読み込んでおいたカーネルの場合は、`kexec -p -u` コマンドを実行してください。

18.4 crashkernel 割り当てサイズの計算

Kexec で情報採取用のカーネルを起動する 場合、このカーネル用のメモリ領域を割り当てておく必要があります。割り当てるべきサイズはコンピュータのハードウエア設定によって異なりますので、あらかじめ指定しておく必要があります。

割り当てるべきサイズは、お使いのコンピュータのハードウエアアーキテクチャによっても異なります。それぞれ下記の手順をお読みのうえ、必要なサイズを判断してください。

手順 18.1: AMD64/INTEL 64 でのサイズの割り当て

1. コンピュータに対する基礎値を得るには、端末内で下記のコマンドを入力し実行します:

   ```
   # kdumptool calibrate
   Total: 49074
   ```

```
Low: 72
High: 180
MinLow: 72
MaxLow: 3085
MinHigh: 0
MaxHigh: 45824
```

それぞれの値はメガバイト単位です。

2. 上記の `Low` および `High` の値を記録しておきます。



注記: Low と High の値の意味について

AMD64/Intel 64 コンピュータの場合、`High` は利用可能な全てのメモリを予約する意味になり、`Low` は DMA32 ゾーン内のメモリ (つまり 4 GB 以下の領域) を予約する意味になります。

SIZE_LOW は 32 ビットのみに対応したデバイスで必要なメモリ量を表します。カーネルは DMA32 のバウンスバッファに対して 64M を割り当てますが、お使いのサーバ内に 32 ビットのみに対応したデバイスが存在しない場合は、`SIZE_LOW` の値は既定の 72M で問題なく動作するはずです。ただし、NUMA マシンの場合は例外で、より多くの `Low` が必要となる場合があります。そのため、通常のカーネルの割り当てで `Low` メモリを使用しないようにするため、Kdump カーネルのパラメータに `numa=off` を設定してもかまいません。

3. 上記の手順の `High` の値に、お使いのコンピュータに接続されている LUN カーネルパス (ストレージデバイスのパス数) を元にした値を足します。具体的には、メガバイト単位で下記のような計算を行います:

```
SIZE_HIGH = 推奨値 + (LUN_数 / 2)
```

上記の数式に当てはめるべき値は下記のとおりです:

- SIZE_HIGH: `High` の計算結果です。
- RECOMMENDATION: `kdumptool calibrate` で表示された `High` の値です。
- LUNs: お使いのコンピュータで作成する予定の LUN カーネルパス数の最大値を指定します。マルチパスデバイスについては無視されるため、数から除外してください。お使いのシステムで利用可能な 現在の LUN 数を取得するには、下記のようなコマンドを入力して実行します:

```
> cat /proc/scsi/scsi | grep Lun | wc -l
```

4. お使いのデバイスに対応するドライバが DMA32 ゾーン内に多数の予約領域を作成する場合は、`Low` の値も調整する必要があります。しかしながら、この値は簡単に計算できるものではありません。正しい値を得るには、試行錯誤を繰り返す必要があります。
   まずは `Low` の値を `kdumptool calibrate` で出力された値から試してみてください。

5. あとは値を正しい場所に設定します。

   カーネルのコマンドラインを直接変更する場合:

   お使いのブートローダの設定で、カーネルオプションに下記を追加してください:

   ```
   crashkernel=SIZE_HIGH,high crashkernel=SIZE_LOW,low
   ```

   `SIZE_HIGH` と `SIZE_LOW` の値は、それぞれ上記の手順で算出もしくは試行錯誤した結果を入力してください。また、末尾には `M` (メガバイトの意味です) を付与してください。
   たとえば下記のようになります:

   ```
   crashkernel=36M,high crashkernel=72M,low
   ```

   YaST GUI を使用している場合:

   [Kdump の低メモリ] に `Low` の値を入力してください。
   [Kdump の高メモリ] に `High` の値を入力してください。

   YaST のコマンドラインインターフェイスを使用している場合:

   下記のコマンドを入力して実行してください:

   ```
   # yast kdump startup enable alloc_mem=LOW,HIGH
   ```

   ここで、`LOW` には `Low` の値を、`HIGH` には `High` の値をそれぞれ入力してください。



ヒント: IBM Z における未使用および無効化された CCW デバイスの除外について

利用可能なデバイスの数にもよりますが、`crashkernel` カーネルパラメータで指定したメモリ量が不足する場合があります。不足した場合は、メモリを増やす代わりに、カーネル側で見えるデバイスの量を制限することができます。これにより、crashkernel の設定で必要なメモリ量を減らすことができます。

1. デバイスを無視したい場合は、`cio_ignore` ツールを利用することで、現時点で有効化されているデバイスや使用しているデバイスを除き、それ以外の全てのデバイスを無視する設定を生成することができます。

   ```
   > sudo cio_ignore -u -k
   ```

```
cio_ignore=all,!da5d,!f500-f502
```

`cio_ignore -u -k` と入力して実行すると、ブラックリスト設定が即時に有効化され、既存のブラックリストを置き換えるようになります。未使用のデバイスが消えることはなく、チャネルサブシステム内で変わらず現れるようになります。ただし、新しいチャネルデバイスを追加 (z/VM 下での CP ATTACH か、LPAR での動的な I/O 設定変更) すると、それらはブラックリストとして扱われます。この動作をしないようにするには、まず既存の設定を `sudo cio_ignore -l` を実行して保存し、`cio_ignore -u -k` コマンドの実行後にその状態に戻すようにしてください。それ以外の方法としては、生成された設定を通常のカーネルパラメータに追加する方法もあります。

2. この方法の場合は、`/etc/sysconfig/kdump` 内にある `KDUMP_CMDLINE_APPEND` の値の中に、`cio_ignore` パラメータを追加してください。たとえば下記のようになります:

```
KDUMP_COMMANDLINE_APPEND="cio_ignore=all,!da5d,!f500-f502"
```

3. 設定を反映させるには、`kdump` を再起動します:

```
systemctl restart kdump.service
```

18.5 基本的な Kexec の使用方法

Kexec を使用するには、まず対応するサービスを有効化し、動作させる必要があります:

- Kexec のサービスをシステムの起動時に開始するようにするには、下記のように入力して実行します:

```
> sudo systemctl enable kexec-load.service
```

- Kexec サービスを開始するには、下記のように入力して実行します:

```
> sudo systemctl start kexec-load.service
```

お使いの Kexec 環境が正しく動作していることを確認するには、Kexec を利用して新しいカーネルを起動してください。ただし、お使いのシステム内には誰もログインしていない状態で、重要なサービスを動作させていない状態であることを確認しておいてください。あとは下記のコマンドを入力して実行します:

```
systemctl kexec
```

これにより、あらかじめ読み込んでおいた新しいカーネルが古いカーネルを置き換えて、制御を行うようになります。制御が始まると、通常の起動メッセージが表示されます。新しいカーネルが起動しても、ハードウエアやファームウエアのチェックは省略されます。このとき、何も警告メッセージが表示されないことを確認してください。



ヒント: reboot コマンドでの Kexec の使用について

`reboot` コマンドで通常の再起動を行わず、Kexec を使用するように設定したい場合は、下記のように入力して実行します:

```
ln -s /usr/lib/systemd/system/kexec.target /etc/systemd/system/reboot.target
```

なお、`etc/systemd/system/reboot.target` ファイルを削除することで、元の再起動に戻すことができます。

18.6 日々の再起動に対する Kexec の設定方法

Kexec は定期的な再起動処理でも使用することができます。たとえばハードウエア検出ルーチンに長い時間がかかるような環境や、起動時の信頼性があまり高くないシステムなどで有用です。

ただし、Kexec でシステムを再起動する際は、ファームウエアやブートローダを使用しないことに注意してください。ブートローダ側で設定を変更していても、物理的な (Kexec を使用しない) 再起動を行うまで、それらは反映されません。

18.7 基本的な Kexec の設定

Kdump を使用することで、カーネルのダンプ情報を保存することができます。これは、カーネルがクラッシュしてしまった場合、その時点のメモリ内容をファイルシステムに保存する機能です。この仕組みにより、カーネルがクラッシュした時点で何が起こっていたのかを調べることができるようになります。これを「コアダンプ」と呼びます。

Kdump は Kexec と同じ仕組みで動作するものです (詳しくは 第18章「Kexec と Kdump」をお読みください) 。この場合、本番用のカーネルがクラッシュした際に、情報採取用のカーネルを起動します。ただし、Kexec では本番用のカーネルをそのまま置き換えて動作するのに対して、Kdump ではクラッシュした本番用のカーネルのメモリを保持し続ける点が異なります。これにより、Kdump カーネルの環境内から、クラッシュしたカーネルのメモリ領域を保存できることになります。

ヒント: ネットワーク経由でのダンプについて

ローカルストレージのサイズが少ないシステムでは、カーネルダンプをネットワーク経由で採取したいこともあります。Kdump では、`initrd` を介してネットワークインターフェイスの設定を行い、それを動作状態に移行させることができます。また、通常の LAN だけでなく VLAN にも対応しています。設定作業は YaST を使用して行うか、`/etc/sysconfig/kdump` ファイル内にある `KDUMP_NETCONFIG` 変数を設定してください。

重要: Kdump での保存先ファイルシステムは設定の時点でマウントしておかなければならない問題について

Kdump を設定する際、採取したダンプイメージの保存先の場所を指定することができます (既定値: `/var/crash`)。この保存先は Kdump の設定時点でマウントしておかなければなりません。設定時点でマウントを行っていないと、設定が失敗します。

18.7.1 手作業での Kdump の設定

Kdump では設定を `/etc/sysconfig/kdump` ファイルから読み込みます。お使いのシステムで Kdump を動作させるにあたって、事前の設定は特に必要ありません。Kdump を既定の設定値のままで動作させたい場合は、下記の手順を実施してください:

1. 18.4項「crashkernel 割り当てサイズの計算」の手順に従って、Kdump で必要なメモリ量を決定します。決定後、`crashkernel` のカーネルパラメータを設定します。
2. システムを再起動します。
3. Kdump サービスを有効化します:

   ```
   > sudo systemctl enable kdump
   ```

4. 必要であれば `/etc/sysconfig/kdump` ファイルを編集します。それぞれのオプションの意味について、詳しくはファイル内のコメント (英語) をお読みください。
5. - 初期化スクリプトを一度だけ動作させます。

     ```
     > sudo systemctl start kdump
     ```

   - システムを再起動します。

既定値で Kdump の設定を行ったら、あとは期待通りに動作するかどうかを確認します。まずはお使いのシステム内に誰もログインしていないことと、システム内で重要なサービスを動作させていないことを確認して、下記の手順を実施します:

1. rescue ターゲットに切り替えます。

```
> sudo systemctl isolate rescue.target
```

2. Kdump サービスを再起動します:

```
> sudo systemctl start kdump
```

3. 下記のコマンドを入力して実行し、ルートファイルシステム以外の全てのマウントを解除します:

```
> sudo umount -a
```

4. ルートファイルシステムを読み込み専用モードで再マウントします:

```
> sudo mount -o remount,ro /
```

5. `procfs` インターフェイス内の Magic SysRq キー機能を利用して、「カーネルパニック」を動作させます:

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

重要: カーネルダンプのサイズについて

`KDUMP_KEEP_OLD_DUMPS` オプションは、カーネルダンプの保存数 (既定値: 5) を指定するためのものです。圧縮を行わない場合、ダンプのサイズは最大で物理メモリ (RAM) のサイズそのものになります。そのため、`/var` のパーティションに対しては、十分な空き領域を用意しておいてください。

情報採取用のカーネルが起動して、クラッシュしたほうのカーネルのメモリ状態をファイルシステムに保存します。保存先は `KDUMP_SAVEDIR` オプションで設定します (既定値: `/var/crash`)。`KDUMP_IMMEDIATE_REBOOT` を `yes` に設定していると、本番用のカーネルを利用してシステムを自動的に再起動します。再起動が終わったらログインを行い、`/var/crash` 内にダンプファイルが作成されていることを確認してください。

18.7.2 YaST での設定

YaST で Kdump を設定するには、まず `yast2-kdump` パッケージをインストールする必要があります。インストール後は `root` で［YaST コントロールセンター］内の［システム］カテゴリにある［カーネル Kdump］を起動するか、もしくはコマンドラインで `yast2 kdump` と入力して実行します。

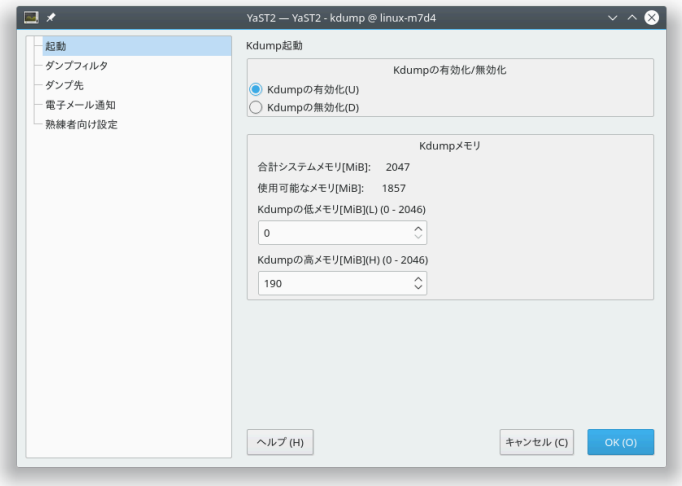

図 18.1: YAST KDUMP モジュール: 最初のページ

まずは［起動］ウインドウ内で［Kdump の有効化］を選択します。

ウインドウを始めた開いたタイミングで、［Kdump メモリ］の値が自動的に設定されます。しかしながら、環境によっては自動生成の値では不十分な場合があります。正しい値を設定するには、18.4項「crashkernel 割り当てサイズの計算」の手順に従って判断を行い、値を入力してください。

重要: ハードウエア変更後の［Kdump メモリ］の再設定について

コンピュータ内で Kdump を設定し、後からメモリ量や利用可能なハードディスクを変更した場合、YaST では古い環境での値を表示し続けてしまいます。

この問題を回避するには、18.4項「crashkernel 割り当てサイズの計算」で書かれている手順に従って必要なメモリ量を判断して、YaST で設定をやり直してください。

左のペイン内で［ダンプフィルタリング］を選択して、ダンプ内に含めるべきページを選択します。カーネルの問題をデバッグするだけであれば、これらのページは必要ではありません:

- ゼロ充填のページ
- キャッシュページ

- ユーザデータページ
- フリーページ

［ダンプ先］ウインドウでは、ダンプの保存先の種類を選択したあと、保存先の場所を指定します。FTP や SSH などのネットワークプロトコルを指定した場合は、必要なアクセス情報についても指定する必要があります。



ヒント: 他のアプリケーションとのダンプディレクトリ共有について

Kdump のダンプの保存先と他のアプリケーションのダンプ保存先を同じパスに設定することもできます。この場合、古いダンプファイルの削除処理では、他のアプリケーションのダンプが除外され、削除は行われません。

Kdump に対して、そのイベントが発生した際に電子メールで通知を行いたい場合は、［電子メール通知］のウインドウ内で設定を行います。また、必要であれば［熟練者向け設定］で必要な設定を行います。設定が完了したら［OK］を押すことで、設定が完了します。

18.7.3 SSH 経由での Kdump

ダンプファイルには不用意に開示してはならない様々な機密情報が含まれています。このような機密情報を含むデータを信頼できないネットワーク経由で送信できるようにするため、Kdump には、SSH プロトコルを介してリモートのマシンにダンプファイルを送信できる機能が用意されています。

1. 送信先のホストの識別情報は、あらかじめ Kdump 側で既知のものでなければなりません。これは、機密情報を含むデータを誤った (場合によっては悪意を持って偽装された) ホストに送信しないための仕組みです。Kdump が新しい `initrd` を生成する際、送信先のホストの識別情報を問い合わせるために `ssh-keygen -F 宛先ホスト` を実行しますが、それは `宛先ホスト` の公開鍵が既知である必要があるためです。公開鍵を既知のものにしておくための最も簡単な方法は、`root` で `宛先ホスト` に接続してみることです。
2. また、Kdump は送信先のホストとの間で認証できなければなりません。現時点では公開鍵認証にのみ対応しています。既定では Kdump は `root` の機密鍵を使用して接続しようとしますが、通常は Kdump 用に個別の鍵を用意しておくことが望ましい設定です。個別の鍵は `ssh-keygen` を利用して、下記のように実行することで作成することができます:
    a. `# ssh-keygen -f ~/.ssh/kdump_key`
    b. なお、パスフレーズの入力を求められた際には、何も入力せずに `Enter` を押してください (これでパスフレーズを使用せずに鍵を作成する意味になります) 。

c. あとは `/etc/sysconfig/kdump` ファイルを開いて、`KDUMP_SSH_IDENTITY` の値を `kdump_key` に設定します。なお、`~/.ssh` 以外のディレクトリに配置している場合は、フルパスで指定してください。

3. 次に、生成した Kdump の SSH 鍵を送信先のホストに送信して認証できるようにします。

```
# ssh-copy-id -i ~/.ssh/kdump_key TARGET_HOST
```

4. `KDUMP_SAVEDIR` を設定します。これには 2 つの方法があります:

Secure File Transfer Protocol (SFTP)

SFTP は SSH 経由でファイルを送信するための推奨される方法です。ただし、接続先のホストで SFTP サブシステムが有効化されていなければなりません (openSUSE Leap では既定で有効化されています) 。具体的には、下記のように指定します:

```
KDUMP_SAVEDIR=sftp://宛先ホスト/ダンプファイルの保存先パス
```

Secure Shell Protocol (SSH)

ディストリビューションによっては、接続先のホストで SSH 経由でコマンドを実行しなければならないものもあります。openSUSE Leap では、この方法にも対応しています。この場合、接続先のホストでの Kdump のユーザには、ログインシェルが設定されていなければならず、`mkdir` , `dd` , `mv` の各コマンドを実行できなければなりません。具体的には下記のように指定します:

```
KDUMP_SAVEDIR=ssh://宛先ホストダンプファイルの保存先パス
```

5. 新しい設定を適用するには、Kdump のサービスを再起動してください。

18.8 クラッシュダンプの解析

ダンプファイルを保存することができたら、あとは解析するだけです。解析方法にはいくつかあります。

ダンプを解析する際に使用する最も基本的なツールは GDB です。このツールは最新の環境でも使用することができますが、いくつかの欠点や制限が存在しています:

- GDB はカーネルのダンプを解析するための専用ツールではないこと。
- GDB では 32 ビットプラットフォームで ELF64 バイナリ形式に対応していないこと。
- GDB では ELF ダンプ以外の形式 (圧縮ダンプを含む) に対応していないこと。

上記のような理由から、`crash` ユーティリティが作成されました。このユーティリティはクラッシュダンプを解析するだけでなく、動作中のシステムもデバッグすることができます。また、Linux カーネルのデバッグ専用の機能が用意され、高度なデバッグを行いたい場合に最適な仕組みです。

Linux カーネルをデバッグするには、まずデバッグ情報パッケージをもインストールしておく必要があります。お使いのシステムにインストールされているかどうかを確認するには、下記のコマンドを入力して実行します:

```
> zypper se kernel | grep debug
```



重要: デバッグ情報パッケージのリポジトリについて

openSUSE Leap 15.7 のシステムでオンライン更新を受け取るように設定している場合、`*-Debuginfo-Updates` という名前のオンラインリポジトリ内に「debuginfo」パッケージが存在しています。YaST を利用するなどしてリポジトリを有効化してください。

ダンプを生成したマシンと同じマシンで `crash` コマンドを実行し、採取したダンプを開きたい場合は、下記のようなコマンドを入力して実行します:

```
crash /boot/vmlinux-6.4.0-150600.9-default.gz \
/var/crash/2024-04-23-11\:17/vmcore
```

ここで、最初のパラメータはカーネルイメージのパスを表しています。2 つめのパラメータには、Kdump で採取したダンプファイルを指定します。既定では、`/var/crash` ディレクトリ内に存在しています。



ヒント: カーネルクラッシュダンプからの基本情報の取得について

openSUSE Leap には `kdumpid` と呼ばれるユーティリティ (同名のパッケージに含まれています) が存在し、これによって未知のカーネルダンプの情報を表示することができます。このユーティリティは、アーキテクチャやカーネルリリースなど、基本的な情報を抽出することができます。また、このユーティリティは lkcd, diskdump, Kdump, ELF ダンプの各形式に対応しているほか、`-v` オプションを付けて実行すると、マシンの種類やカーネルのバナー文字列、カーネルの設定フレーバーなども表示することができます。

18.8.1 カーネルのバイナリ形式

Linux カーネルは Executable and Linkable Format (ELF) 形式で提供されます。このファイルは通常 `vmlinux` と呼ばれ、コンパイル処理時に直接生成されます。ただし、特に AMD64/Intel 64 アーキテクチャで顕著ではありますが、全てのブートローダが ELF 形式に対応しているというわけではありません。openSUSE® Leap で対応しているアーキテクチャごとに、下記のような仕組みになっています。

18.8.1.1 AMD64/Intel 64

SUSE が提供する AMD64/Intel 64 向けのカーネルパッケージには、`vmlinuz` と `vmlinux.gz` という 2 つのファイルが含まれています。

- `vmlinuz`: ブートローダから直接実行するほうのファイルです。
  Linux カーネルは 2 種類のパーツから構成されています。1 つはカーネルそれ自身 (`vmlinux`) 、もう 1 つはブートローダが実行するセットアップコードです。これら 2 つのパーツを 1 つにまとめて `vmlinuz` というファイルに仕立て上げています (末尾が `x` ではなく `z` であることに注意してください) 。
  カーネルのソースツリー内では、このファイルは `bzImage` と呼ぶこともあります。

- `vmlinux.gz`: こちらは `crash` や GDB が使用するファイルで、圧縮された ELF イメージの形態になっています。ELF イメージは AMD64/Intel 64 のブートローダから使用されることがありませんので、圧縮されたファイルのみを提供しています。

18.8.1.2 POWER

POWER 上で動作する `yaboot` ブートローダは ELF イメージの読み込みに対応していますが、圧縮されたファイルには対応していません。そのため、POWER アーキテクチャ向けのカーネルパッケージには、`vmlinux` という ELF 形式の Linux カーネルファイルが含まれています。`crash` を動作させる前提で考えると、最も単純なアーキテクチャであると言えます。

なお、他のマシンでダンプを解析する場合は、コンピュータのアーキテクチャとデバッグに必要なファイルの両方を確認しなければならないことに注意してください。

他のコンピュータでダンプを解析する場合は、同じアーキテクチャの Linux マシンのみを使用することができます。アーキテクチャを確認したい場合は、`uname -i` と入力して実行し、出力された内容を比較してください。

このほか、他のコンピュータでダンプを解析しようとしている場合は、`kernel` パッケージと `kernel debug` パッケージが提供するファイルが必要となります。

1. まずはカーネルダンプと `/boot` にあるカーネルイメージ、そしてそれに対応し、`/usr/lib/debug/boot` 内にあるデバッグ情報ファイルを、1 つのディレクトリ内にコピーしてまとめます。
2. また、`/lib/modules/$(uname -r)/kernel/` にあるカーネルモジュールと `/usr/lib/debug/lib/modules/$(uname -r)/kernel/` にあるカーネルモジュール用のデバッグ情報ファイルを、`modules` サブディレクトリ内にコピーします。
3. ダンプファイルとカーネルイメージ、デバッグ情報ファイルを 1 つのディレクトリにまとめ、`modules` サブディレクトリ以下にカーネルモジュールとデバッグ情報ファイルを配置したら、あとは `crash` ユーティリティを実行します:

```
> crash VMLINUX-バージョン vmcore
```

ダンプを解析しているコンピュータに関わらず、crash ユーティリティは下記のような出力を生成するはずです:

```
> crash /boot/vmlinux-6.4.0-150600.9-default.gz \
/var/crash/2024-04-23-11\:17/vmcore
crash 7.2.1
Copyright (C) 2002-2017  Red Hat, Inc.
Copyright (C) 2004, 2005, 2006, 2010  IBM Corporation
Copyright (C) 1999-2006  Hewlett-Packard Co
Copyright (C) 2005, 2006, 2011, 2012  Fujitsu Limited
Copyright (C) 2006, 2007  VA Linux Systems Japan K.K.
Copyright (C) 2005, 2011  NEC Corporation
Copyright (C) 1999, 2002, 2007  Silicon Graphics, Inc.
Copyright (C) 1999, 2000, 2001, 2002  Mission Critical Linux, Inc.
This program is free software, covered by the GNU General Public License,
and you are welcome to change it and/or distribute copies of it under
certain conditions.  Enter "help copying" to see the conditions.
This program has absolutely no warranty.  Enter "help warranty" for details.

GNU gdb (GDB) 7.6
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu".

      KERNEL: /boot/vmlinux-6.4.0-150600.9-default.gz
   DEBUGINFO: /usr/lib/debug/boot/vmlinux-6.4.0-150600.9-default.debug
    DUMPFILE: /var/crash/2024-04-23-11:17/vmcore
        CPUS: 2
        DATE: Thu Apr 23 13:17:01 2024
      UPTIME: 00:10:41
LOAD AVERAGE: 0.01, 0.09, 0.09
       TASKS: 42
```

```
    NODENAME: eros
     RELEASE: 6.4.0-150600.9-default
     VERSION: #1 SMP 2024-03-31 14:50:44 +0200
     MACHINE: x86_64  (2999 Mhz)
      MEMORY: 16 GB
       PANIC: "SysRq : Trigger a crashdump"
         PID: 9446
     COMMAND: "bash"
        TASK: ffff88003a57c3c0  [THREAD_INFO: ffff880037168000]
         CPU: 1
       STATE: TASK_RUNNING (SYSRQ)
crash>
```

このコマンドは最初に、有益なデータを出力します。上記の例では、カーネルがクラッシュした時点で 42 個のタスクが動作していて、クラッシュの原因は PID 9446 のタスクから送信された SysRq トリガー、そしてそのプロセスは bash である (bash 内蔵コマンドである `echo` コマンドでクラッシュを発生させたため) ことを示しています。

`crash` ユーティリティは GDB を利用して動作する仕組みであるため、様々な追加コマンドも用意されています。たとえば `bt` コマンドを何もパラメータを指定せずに実行すると、クラッシュが発生した時点での、そのタスクのバックトレースを表示することができます:

```
crash> bt
PID: 9446   TASK: ffff88003a57c3c0  CPU: 1   COMMAND: "bash"
 #0 [ffff880037169db0] crash_kexec at ffffffff80268fd6
 #1 [ffff880037169e80] __handle_sysrq at ffffffff803d50ed
 #2 [ffff880037169ec0] write_sysrq_trigger at ffffffff802f6fc5
 #3 [ffff880037169ed0] proc_reg_write at ffffffff802f068b
 #4 [ffff880037169f10] vfs_write at ffffffff802b1aba
 #5 [ffff880037169f40] sys_write at ffffffff802b1c1f
 #6 [ffff880037169f80] system_call_fastpath at ffffffff8020bfbb
    RIP: 00007fa958991f60  RSP: 00007fff61330390  RFLAGS: 00010246
    RAX: 0000000000000001  RBX: ffffffff8020bfbb  RCX: 0000000000000001
    RDX: 0000000000000002  RSI: 00007fa959284000  RDI: 0000000000000001
    RBP: 0000000000000002   R8: 00007fa9592516f0   R9: 00007fa958c209c0
    R10: 00007fa958c209c0  R11: 0000000000000246  R12: 00007fa958c1f780
    R13: 00007fa959284000  R14: 0000000000000002  R15: 00000000595569d0
    ORIG_RAX: 0000000000000001  CS: 0033  SS: 002b
crash>
```

これで何が起こったのかを知ることができます。bash シェルの内蔵コマンドである `echo` が、`/proc/sysrq-trigger` に文字を送信したために、クラッシュが発生していることになります。その文字に対応するハンドラが検出されると、`crash_kexec()` を実行しています。この関数は `panic()` と呼ばれ、ここで Kdump がダンプを保存しています。

GDB の基本的なコマンドと `bt` の拡張版に加えて、crash ユーティリティには Linux カーネルの構造に関連したその他のコマンドも用意されています。これらのコマンドは Linux カーネルの内部データ構造を理解し、それを人間にとって読みやすい形式で表示します。たとえばクラッシュが発生した時

点でのタスクの一覧は、`ps` で表示することができます。また `sym` コマンドは、全てのカーネルシンボルの一覧とアドレスを表示したり、指定したシンボルに対して値を取得したりすることもできます。さらに `files` コマンドでは、プロセスが開いている全てのファイルディスクリプタを表示することができます。`kmem` では、カーネルのメモリ使用率に関する詳細を表示することができます。`vm` では、個別のページマッピングのレベルに至るまで、プロセスの仮想メモリを調査することができます。分かりやすいコマンドの一覧は多く用意されているほか、様々なオプションを受け付けるようになっています。

上述のコマンドは、一般的な Linux コマンドである `ps` や `lsof` の機能に対応しているものです。なお、デバッガでイベントの正確な順序を確認したい場合は、GDB の使用方法について知る必要があるほか、強固なデバッグスキルを必要とします。この種類の話題は本文書の範疇外にあたるものであるため、本書では説明していません。これに加えて、Linux カーネルそのものの知識も必要となります。こちらについては、本文書の末尾にある参照情報をご確認ください。

18.9 高度な Kdump の設定

Kdump の設定は `/etc/sysconfig/kdump` 内に保存されています。設定は YaST を利用しても行うことができます。Kdump の設定オプションは、[YaST コントロールセンター] 内の [システム] › [カーネル Kdump] 内に存在しています。下記の Kdump オプションが用意されています。

カーネルダンプの保存先は、`KDUMP_SAVEDIR` オプションで指定します。ただし、カーネルダンプのサイズは大きくなることに注意してください。ディスクの空き容量から予想されるダンプファイルのサイズを引いた値が、`KDUMP_FREE_DISK_SIZE` よりも少ない場合、Kdump はダンプの保存を行わなくなります。また、`KDUMP_SAVEDIR` では、URL 形式にも対応しています。URL 形式は `プロトコル://仕様` の形式で指定し、`プロトコル` は `file`, `ftp`, `sftp`, `nfs`, `cifs` のいずれかを指定します。`仕様` はそれぞれのプロトコルに従った値を指定します。たとえばカーネルダンプを FTP サーバ内に保存したい場合は、`ftp://ユーザ名:パスワード@ftp.example.com:123/var/crash` のように指定します。

カーネルダンプは巨大なものであり、解析には必要のない多数のページをも含んでしまっています。`KDUMP_DUMPLEVEL` オプションを指定することで、これらの不要なページを省略することができます。設定可能な値は 0 から 31 までで、`0` を指定すると最も巨大なダンプファイルに、`31` を指定すると最も小さなダンプファイルになります。設定可能な値について、詳しくは `kdump` のマニュアルページ (`man 7 kdump`) をお読みください。

場合によっては、カーネルダンプのサイズを小さくしておいたほうが都合のよい場合があります。たとえばネットワーク経由でダンプを転送するような場合や、ダンプディレクトリのディスク領域を節約したい場合などがそれにあたります。この場合は、`KDUMP_DUMPFORMAT` の値を `compressed` に設定してください。なお、`crash` ユーティリティでは、圧縮されたダンプを動的に展開して使用することができます。

重要: Kdump の設定ファイルの変更について

`/etc/sysconfig/kdump` ファイルを変更した場合は、その変更点を反映させるため、必ず `systemctl restart kdump.service` を実行する必要があります。実行しておかないと、次にシステムを再起動するまで、変更が反映されなくなってしまいます。

18.10 さらなる情報

Kexec や Kdump の使用方法については、1 箇所にまとまった情報源が存在していないのが現状です。下記にさまざまな側面で使用できる情報源 (いずれも英語のみ) を示します:

- Kexec ユーティリティの使用方法については、`kexec` のマニュアルページ (`man 8 kexec`) をお読みください。
- Kexec に関する一般的な情報については、https://developer.ibm.com/technologies/linux/ をお読みください。
- openSUSE Leap における Kdump の詳細については、https://ftp.suse.com/pub/people/tiwai/kdump-training/kdump-training.pdf をお読みください。
- Kdump の内部仕様に関する詳細については、https://lse.sourceforge.net/kdump/documentation/ols2oo5-kdump-paper.pdf をお読みください。

`crash` ダンプ解析ユーティリティやデバッグツールについての詳細は、それぞれ下記 (いずれも英語) をお読みください:

- GDB の info ページ (`info gdb`) に加えて、印刷可能なガイド https://sourceware.org/gdb/documentation/ も提供されています。
- crash ユーティリティには幅広い分野に対応したオンラインヘルプが用意されています。`help コマンド` のように入力すると、`コマンド` に対するオンラインヘルプを表示することができます。
- Perl の知識をお持ちであれば、Alicia を利用してより簡単にデバッグを行うことができます。これは Perl ベースの crash ユーティリティ向けフロントエンドです。詳しくは https://alicia.sourceforge.net/ をお読みください。
- Python をご利用になりたい場合は、Pykdump をインストールして使用することをお勧めします。このパッケージは Python スクリプト経由で GDB を制御することができるものです。
- Linux カーネルの内部情報に関する幅広い情報については、Daniel P. Bovet 氏および Marco Cesati 氏による Understanding the Linux Kernel (ISBN 978-0-596-00565-8) をお読みください。

19 アプリケーションクラッシュ時の systemd-coredump の使用

改訂履歴

2023-08-08

`systemd-coredump` はアプリケーションのクラッシュを解析するため、コアダンプ情報を採取して表示できる仕組みです。コアダンプ内にはアプリケーションがクラッシュした時点でのメモリイメージが含まれています。既定では、プロセス (もしくはアプリケーションに属する複数のプロセス) がクラッシュした場合、可能であればバックトレースを含めて `/var/lib/systemd/coredump` にコアダンプを保存して、`systemd` のジャーナル内にその旨を記録します。この出力されたコアダンプをもとに、`gdb` や `crash` (詳しくは 18.8項「クラッシュダンプの解析」をお読みください) を利用して解析することができます。

なお、`/var/lib/systemd/coredump` 内に保存されたコアダンプは、3 日間が経過すると自動的に削除されます (詳しくは `/usr/lib/tmpfiles.d/systemd.conf` ファイル内の `d /var/lib/systemd/coredump` の行をお読みください) 。

また、コアダンプを全く保存せず、ジャーナルにのみ記録を行うように設定することもできます。これは、収集によるディスク容量の圧迫と機密情報の保存を防ぐ目的で有用です。

19.1 使用と設定

`systemd-coredump` は既定で有効化され、すぐに使用できるようになっています。既定の設定ファイルは `/etc/systemd/coredump.conf` 内に存在しています:

```
[Coredump]
#Storage=external
#Compress=yes
#ProcessSizeMax=2G
#ExternalSizeMax=2G
#JournalSizeMax=767M
#MaxUse=
#KeepFree=
```

サイズを指定する箇所では、それぞれ B (バイト), K (キロバイト), M (メガバイト), G (ギガバイト), T (テラバイト), P (ペタバイト), E (エクサバイト) の接尾辞を設定することができるほか、`infinity` (無制限) を指定することもできます。

下記の例では、vim を利用して SEGFAULT を発生させ、ジャーナルとコアダンプを出力させるテスト方法を説明しています。

手順 19.1: VIM を利用したコアダンプの作成

1. まずは `debuginfo-pool` および `debuginfo-update` の各リポジトリを有効化します
2. `vim-debuginfo` をインストールします
3. `vim testfile` と入力して実行し、文字を入力します
4. PID を取得して SEGFAULT を発生させます:

```
> ps ax | grep vim
2345 pts/3    S+     0:00 vim testfile

# kill -s SIGSEGV 2345
```

vim は下記のようなエラーメッセージを出力するはずです:

```
Vim: Caught deadly signal SEGV
Vim: Finished.
Segmentation fault (コアダンプ)
```

5. 生成されたコアダンプファイルを調査します:

```
# coredumpctl
TIME                         PID  UID  GID SIG PRESENT EXE
Wed 2019-11-12 11:56:47 PST 2345 1000 100 11  *       /bin/vim

# coredumpctl info
PID: 2345 (vim)
UID: 0 (root)
GID: 0 (root)
Signal: 11 (SEGV)
Timestamp: Wed 2019-11-12 11:58:05 PST
Command Line: vim testfile
Executable: /bin/vim
Control Group: /user.slice/user-1000.slice/session-1.scope
    Unit: session-1.scope
    Slice: user-1000.slice
    Session: 1
    Owner UID: 1000 (tux)
    Boot ID: b5c251b86ab34674a2222cef102c0c88
    Machine ID: b43c44a64696799b985cafd95dc1b698
    Hostname: linux-uoch
    Coredump: /var/lib/systemd/coredump/core.vim.0.b5c251b86ab34674a2222cef102
    Message: Process 2345 (vim) of user 0 dumped core.

         Stack trace of thread 2345:
         #0  0x00007f21dd87e2a7 kill (libc.so.6)
         #1  0x000000000050cb35 may_core_dump (vim)
         #2  0x00007f21ddbfec70 __restore_rt (libpthread.so.0)
```

```
          #3  0x00007f21dd92ea33 __select (libc.so.6)
          #4  0x000000000050b4e3 RealWaitForChar (vim)
          #5  0x000000000050b86b mch_inchar (vim)
[...]
```

複数のコアダンプが生成されている場合は、`coredumpctl info` と入力することで、全てのものを表示することができます。このほか、`PID`，`COMM` (コマンド)，`EXE` (実行ファイルのフルパス) などで絞り込むこともできます。たとえば vim に対する全てのコアダンプを表示したい場合は、下記のように入力して実行します:

```
# coredumpctl info /bin/vim
```

特定の `PID` が生成したコアダンプを表示したい場合は、下記のように入力して実行します:

```
# coredumpctl info 2345
```

選択したコアダンプを `gdb` に出力します:

```
# coredumpctl gdb 2345
```

`PRESENT` 列にアスタリスク記号が表示されている場合、これは選択したコアダンプが存在していることを表します。この列に何も書かれていない場合はコアダンプが存在していないため、`coredumpctl` はジャーナルから情報を採取するようになります。この動作は、`/etc/systemd/coredump.conf` ファイル内の `Storage` セクションの項目を編集することで、変更することができます:

- `Storage=none`: コアダンプをジャーナル内に記録しますが、保存を行わないようにします。これは General Data Protection Regulation (GDPR) の規則に準拠するなどの目的で収集される情報を最小限に絞ったり、機密情報を記録したりしないようにするための選択肢です。
- `Storage=external`: コアダンプを `/var/lib/systemd/coredump` 内に記録するようにします。
- `Storage=journal`: コアダンプを `systemd` のジャーナル内に記録するようにします。

それぞれコアダンプが出力されるたびに `systemd-coredump` の新しいインスタンスが起動するようになっていますので、設定を変更したあとにサービスの再起動を行う必要はありません。次回のコアダンプから新しい設定が適用されます。

なお、システムを再起動してしまうと、コアダンプは消去されてしまいます。恒久的にコアダンプを保存したい場合でも、`coredumpctl` 側で対応することができます。下記の例では、指定した `PID` のコアダンプを `vim.dump` ファイル内に保存しています:

```
# coredumpctl -o vim.dump dump 2345
```

コマンドの詳しい説明やオプションの一覧について、詳しくは `man systemd-coredump` , `man coredumpctl` , `man core` , `man coredump.conf` をそれぞれお読みください。

VII Precision Time Protocol による時刻同期

20 Precision Time Protocol 205

20 Precision Time Protocol

改訂履歴

2024-06-25

ネットワーク環境ではコンピュータやデバイス類の時刻を同期して、正確性を保つことが必要不可欠です。時刻の同期にあたってはさまざまな手法が存在しますが、その中で最もよく使用されているのが Network Time Protocol (NTP) です。NTP については『リファレンス』、第18章「NTP を利用した時刻同期」で説明しています。

Precision Time Protocol (PTP) はマイクロ秒未満の精度に対応するプロトコルで、NTP よりもさらに正確な時刻同期を実現することができます。PTP はカーネルとユーザスペースのパーツから構成され、openSUSE Leap にも PTP クロック向けの対応 (ネットワークドライバでの提供) が含まれています。

20.1 PTP の紹介

PTP での時刻同期は、マスター／スレーブ型の階層構造を取ります。スレーブは対応するマスターに問い合わせを行い、同期を実施します。階層構造は、それぞれのスレーブが best master clock (BMC) (「最適なマスタークロック」の意味) アルゴリズムを利用して更新されます。また、1 つのポートしか持たないクロック (時刻同期を行うマシン) は、マスターもしくはスレーブのいずれかにしか、なることができません。このようなクロックを ordinary clock (OC) (通常クロック) と呼びます。逆に、複数のポートを持つクロックは、一方をマスターに、他方をスレーブに設定することができます。このようなクロックを boundary clock (BC) (境界クロック) と呼びます。また、最上位のマスタークロックのことを、grandmaster clock (最上位クロック) と呼びます。Global Positioning System (GPS) を利用したクロックなどがそれにあたります。このような仕組みにより、異種混合型ネットワークでも高い精度の時刻同期を行うことができます。

なお、PTP ではハードウエア側の対応が含まれていることが大きな特長です。いくつかのネットワークスイッチやネットワークインターフェイスコントローラ (NIC) に、そのような仕組みが用意されています。ネットワーク内に PTP 非対応のハードウエアが存在していてもかまいませんが、全ての PTP クロックでハードウエア対応が行われていると、最大限の正確性を実現することができます。

20.1.1 PTP の Linux 実装

openSUSE Leap では、PTP の実装は `linuxptp` パッケージに含まれています。`zypper install linuxptp` と入力して実行し、インストールを行ってください。このパッケージには、時刻同期用の `ptp4l` と `phc2sys` というプログラムが用意されています。`ptp4l` は boundary clock の機能と ordinary clock の両方の機能を提供しています。ハードウエア側にタイムスタンプ機能が用

意されていれば、`ptp4l` は PTP のハードウエアクロックをマスタークロックに同期します。ソフトウエア側でのタイムスタンプ機能を利用する場合は、システムクロックをマスタークロックに同期します。`phc2sys` はハードウエアタイムスタンプの際にのみ使用するもので、システムクロックをネットワークインターフェイス (NIC) に搭載されたハードウエアクロックに同期させるために使用します。

20.2 PTP の使用

20.2.1 ネットワークドライバとハードウエアのサポート

PTP ではカーネルのネットワークドライバに対して、ソフトウエアもしくはハードウエアのタイムスタンプ機能を必要とします。これに加えて、NIC の物理ハードウエア側では、タイムスタンプ機能に対応していなければなりません。ドライバと NIC にタイムスタンプ機能があるかどうかを確認するには、`ethtool` コマンドを使用します:

```
> sudo ethtool -T eth0
Time stamping parameters for eth0:
Capabilities:
hardware-transmit     (SOF_TIMESTAMPING_TX_HARDWARE)
        software-transmit     (SOF_TIMESTAMPING_TX_SOFTWARE)
        hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
        software-receive      (SOF_TIMESTAMPING_RX_SOFTWARE)
        software-system-clock (SOF_TIMESTAMPING_SOFTWARE)
        hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
        off                   (HWTSTAMP_TX_OFF)
        on                    (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
        none                  (HWTSTAMP_FILTER_NONE)
        all                   (HWTSTAMP_FILTER_ALL)
```

ソフトウエアタイムスタンプ機能を使用するには、下記のパラメータが表示されていなければなりません:

```
SOF_TIMESTAMPING_SOFTWARE
SOF_TIMESTAMPING_TX_SOFTWARE
SOF_TIMESTAMPING_RX_SOFTWARE
```

ハードウエアタイムスタンプ機能を使用するには、下記のパラメータが表示されていなければなりません:

```
SOF_TIMESTAMPING_RAW_HARDWARE
SOF_TIMESTAMPING_TX_HARDWARE
```

```
SOF_TIMESTAMPING_RX_HARDWARE
```

20.2.2 ptp4l の使用

`ptp4l` は既定ではハードウエアタイムスタンプを使用します。`root` で `-i` オプションを指定して、ハードウエアタイムスタンプ機能に対応したネットワークインターフェイスを指定してください。なお、`-m` オプションを指定すると、`ptp4l` からの出力が、システムのログ機能経由ではなく標準出力に書き込まれるようになります:

```
> sudo ptp4l -m -i eth0
selected eth0 as PTP clock
port 1: INITIALIZING to LISTENING on INITIALIZE
port 0: INITIALIZING to LISTENING on INITIALIZE
port 1: new foreign master 00a152.fffe.0b334d-1
selected best master clock 00a152.fffe.0b334d
port 1: LISTENING to UNCALIBRATED on RS_SLAVE
master offset -25937 s0 freq +0 path delay       12340
master offset -27887 s0 freq +0 path delay       14232
master offset -38802 s0 freq +0 path delay       13847
master offset -36205 s1 freq +0 path delay       10623
master offset  -6975 s2 freq -30575 path delay   10286
port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
master offset  -4284 s2 freq -30135 path delay    9892
```

`master offset` 以下には、マスターとの時刻差 (ナノ秒単位) が表示されます。

`s0` , `s1` , `s2` の各表示は、クロックサーボの状態を表しています。それぞれ `s0` はロック解除状態を、`s1` はクロックステップを、`s2` はロック済み状態を表しています。サーボがロック済みの状態 (`s2`) にある場合、`pi_offset_const` オプションが負の値に設定されていると、クロックをステップさせる (大きく変更する) ことは行わず、徐々に調整するようになります (詳しくは `man 8 ptp4l` をお読みください) 。

`freq` で示されている値は、クロックの周波数調整値 (単位は ppb で、10 億あたりの値) を表しています。

`path delay` には、マスターから送信された同期メッセージの見積もり遅延時間 (単位はナノ秒) を表しています。

Port 0 はローカルの PTP 管理向けに Unix ドメインソケットを使用する意味です。Port 1 は `eth0` インターフェイスの意味です。

`INITIALIZING` , `LISTENING` , `UNCALIBRATED` , `SLAVE` の表示は、それぞれ `INITIALIZE` , `RS_SLAVE` , `MASTER_CLOCK_SELECTED` のイベントに対するポート状態の変更例です。ポートの状態が `UNCALIBRATED` から `SLAVE` に変化すると、コンピュータは PTP のマスタークロックと同期が成功していることを表します。

`-S` オプションを指定することで、ソフトウエアのタイムスタンプ機能を使用することができるようになります。

```
> sudo ptp4l -m -S -i eth3
```

`ptp4l` をサービスとして動作させる場合は、下記のように入力して実行します:

```
> sudo systemctl start ptp4l
```

この場合、`ptp4l` のオプションは `/etc/sysconfig/ptp4l` ファイルから読み込みが行われます。既定では、このファイルには `ptp4l` が `/etc/ptp4l.conf` から設定を読み込むように指定しています。`ptp4l` のオプションや設定ファイルの構造について、詳しくは `man 8 ptp4l` をお読みください。

`ptp4l` サービスを恒久的に有効化したい場合は、下記のとおり入力して実行します:

```
> sudo systemctl enable ptp4l
```

無効化したい場合は、下記のとおり入力して実行します:

```
> sudo systemctl disable ptp4l
```

20.2.3 `ptp4l` 設定ファイル

`ptp4l` は設定を設定ファイルから読み込むことができます。既定では設定ファイルを使用していませんので、設定ファイルを使用する場合は `-f` で指定する必要があります。

```
> sudo ptp4l -f /etc/ptp4l.conf
```

設定ファイルは 2 つのセクションに分かれています。グローバルセクション（`[global]`）内には、プログラムのオプションとクロックのオプション、そして既定のポートオプションが含まれます。その他のセクションはポートごとに固有のもので、既定のポートオプションを上書きするためのものです。セクション名は設定されているポートの名前そのもので、たとえば `[eth0]` 等のようになります。ポートのセクション内に何も設定していないと、コマンドラインオプションを置き換えることができます。

```
[global]
verbose               1
time_stamping         software
[eth0]
```

上記のように設定を行うと、コマンドラインでは下記のような意味になります:

```
> sudo ptp4l -i eth0 -m -S
```

`ptp4l` で利用可能な設定オプションの一覧については、`man 8 ptp4l` をお読みください。

20.2.4 遅延の測定

`ptp4l` では遅延の測定にあたって、2 種類の方式のうちのいずれかを使用することができます。1 つは peer-to-peer (P2P) 、もう 1 つは end-to-end (E2E) です。

P2P
: この方式を使用するには、`-P` オプションを指定します。
この方法は、ネットワーク環境の変更に素早く反応できる方式であるほか、遅延の測定をより正確に行うことができます。この方法は、それぞれのポートが他のポートと PTP メッセージを交換する場合にのみ使用します。P2P は通信パス内の全てのハードウエア側で対応している必要があります。

E2E
: この方式を使用するには、`-E` オプションを指定します。こちらが既定値です。

自動方式選択
: この方式を使用するには、`-A` オプションを指定します。自動方式選択では、まず `ptp4l` が E2E モードで動作し、その後 peer delay (対向遅延) 要求が届くと、P2P モードに切り替える動作を行います。

重要: 一般的な測定方法について

単一の PTP 通信経路内にある全てのクロックは、遅延の測定方式を全て同じに設定しなければなりません。E2E 方式で動作しているポートに peer delay 要求が届いた場合や、P2P 方式で動作しているポートに E2E delay 要求が届いた場合は、警告メッセージを表示します。

20.2.5 PTP 管理クライアント: `pmc`

`ptp4l` に関するより詳しい情報を取得したい場合は、`pmc` クライアントをお使いください。このコマンドは標準入力やコマンドラインからの入力を受け付け、名前で指定したアクションや管理 ID を取得します。あとはアクションを選択したトランスポート経由で送信し、受信した応答を表示します。アクションには 3 種類のものがあります。`GET` は指定した情報の取得、`SET` は指定した情報の更新、そして `CMD` (もしくは `COMMAND`) は指定したイベントの開始に使用します。

既定では、管理コマンドは全てのポートを対象として実行されます。`TARGET` コマンドは特定のクロックとポートを選択して、後続のコマンドを実行させるために使用します。管理 ID の完全な一覧を表示するには、`pmc help` と入力して実行してください。

```
> sudo pmc -u -b 0 'GET TIME_STATUS_NP'
sending: GET TIME_STATUS_NP
```

```
        90f2ca.fffe.20d7e9-0 seq 0 RESPONSE MANAGEMENT TIME_STATUS_NP
                master_offset              283
                ingress_time               1361569379345936841
                cumulativeScaledRateOffset   +1.000000000
                scaledLastGmPhaseChange    0
                gmTimeBaseIndicator        0
                lastGmPhaseChange          0x0000'0000000000000000.0000
                gmPresent                  true
                gmIdentity                 00b058.feef.0b448a
```

`-b` オプションを指定すると、送信されるメッセージに設定する境界ホップ数を指定することができます。境界ホップ数に 0 を指定すると、ローカルの `ptp4l` インスタンスに制限する意味になります。この値を増やして行くことで、ローカルから離れた遠くの PTP ノードからのメッセージを受け取れるようになります。なお、返却された情報には、下記の項目が含まれることがあります:

stepsRemoved
: 最上位クロックに至るまでの通信ノード数を表しています。

offsetFromMaster, master_offset
: マスタークロックとのクロック差 (ナノ秒単位) 。

meanPathDelay
: マスタークロックから送信された同期メッセージの遅延見積値 (ナノ秒単位) 。

gmPresent
: `true` が書かれていると、PTP のクロックはマスタークロックとの間で同期ができていて、最上位のクロックではないことを表しています。

gmIdentity
: 最上位クロックの識別情報が表示されます。

`pmc` のコマンドラインオプションの完全な一覧について、詳しくは `man 8 pmc` と入力して実行してください。

20.3 `phc2sys` による時刻同期

`phc2sys` を使用することで、システムクロックをネットワークカードに搭載された PTP ハードウエアクロック (PHC) に同期させることができます。この場合、システムクロックは スレーブ 、ネットワークカード側は マスター として動作することになります。PHC それ自身は `ptp4l` (詳しくは 20.2項「PTP の使用」をお読みください) で同期を行います。なお、`-s` を指定することで、デバイスやネットワークインターフェイスでマスタークロックを指定することができます。また `-w` を指定すると、`ptp4l` が同期状態になるまで待機させることができます。

```
> sudo phc2sys -s eth0 -w
```

PTP は International Atomic Time (TAI) を基準に動作するのに対して、システムクロックは Coordinated Universal Time (UTC) を基準に動作します。`ptp4l` に対して `-w` オプションを指定して待機させるように指定しない場合、`-O` オプションで TAI と UTC の時刻差を秒単位で指定することができます:

```
> sudo phc2sys -s eth0 -O -35
```

`phc2sys` についても、同様にサービスとして動作させることができます:

```
> sudo systemctl start phc2sys
```

この場合、`phc2sys` のオプションは `/etc/sysconfig/phc2sys` ファイルから読み込みが行われます。`phc2sys` のオプションや設定ファイルの構造について、詳しくは `man 8 phc2sys` をお読みください。

`phc2sys` サービスを恒久的に有効化したい場合は、下記のとおり入力して実行します:

```
> sudo systemctl enable phc2sys
```

無効化したい場合は、下記のとおり入力して実行します:

```
> sudo systemctl disable phc2sys
```

20.3.1 時刻同期の検証

PTP による時刻同期が正しく動作し、ハードウエアのタイムスタンプ機能を使用するように設定すると、`ptp4l` と `phc2sys` は、時刻のズレに関する情報と周期的な調整に関するメッセージを、システムログ内に出力するようになります。

`ptp4l` の出力例:

```
ptp4l[351.358]: selected /dev/ptp0 as PTP clock
ptp4l[352.361]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[352.361]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[353.210]: port 1: new foreign master 00a069.eefe.0b442d-1
ptp4l[357.214]: selected best master clock 00a069.eefe.0b662d
ptp4l[357.214]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[359.224]: master offset       3304 s0 freq      +0 path delay      9202
ptp4l[360.224]: master offset       3708 s1 freq  -28492 path delay      9202
ptp4l[361.224]: master offset      -3145 s2 freq  -32637 path delay      9202
ptp4l[361.224]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[362.223]: master offset       -145 s2 freq  -30580 path delay      9202
ptp4l[363.223]: master offset       1043 s2 freq  -28436 path delay      8972
```

```
[...]
ptp4l[371.235]: master offset        285 s2 freq  -28511 path delay      9199
ptp4l[372.235]: master offset        -78 s2 freq  -28788 path delay      9204
```

`phc2sys` の出力例:

```
phc2sys[616.617]: Waiting for ptp4l...
phc2sys[628.628]: phc offset     66341 s0 freq      +0 delay   2729
phc2sys[629.628]: phc offset     64668 s1 freq  -37690 delay   2726
[...]
phc2sys[646.630]: phc offset      -333 s2 freq  -37426 delay   2747
phc2sys[646.630]: phc offset       194 s2 freq  -36999 delay   2749
```

`ptp4l` は通常、メッセージを頻繁に出力します。メッセージの頻度を減らしたい場合は、`summary_interval` ディレクティブを使用して減らしてください。なお、設定する値には 2 の乗数を指定します。たとえば 1024 (= 2^10) 秒ごとにメッセージを出力するようにしたい場合、下記の行を `/etc/ptp4l.conf` ファイルに追加します:

```
summary_interval 10
```

`phc2sys` についても、出力を減らすことができます。こちらの出力を減らしたい場合は、`-u 更新間隔` オプションを指定してください。

20.4 設定例

本章では、`ptp4l` の設定例をいくつか示します。設定例は設定すべき内容を網羅したものではなく、必要な箇所のみに絞った最小限のものです。なお、`ethX` の箇所はお使いのネットワークインターフェイスに置き換えてご使用ください。

例 20.1: ソフトウエアタイムスタンプ機能を利用した従属クロック

> `/etc/sysconfig/ptp4l` :
>
> ```
> OPTIONS="-f /etc/ptp4l.conf -i ethX"
> ```
>
> `/etc/ptp4l.conf` ファイルを変更する必要はありません。

例 20.2: ハードウエアタイムスタンプ機能を利用した従属クロック

> `/etc/sysconfig/ptp4l` :
>
> ```
> OPTIONS="-f /etc/ptp4l.conf -i ethX"
> ```
>
> `/etc/sysconfig/phc2sys` :
>
> ```
> OPTIONS="-s ethX -w"
> ```

> `/etc/ptp4l.conf` ファイルを変更する必要はありません。

例 20.3: ハードウエアタイムスタンプ機能を利用した自律クロック

> `/etc/sysconfig/ptp4l` :
>
> ```
> OPTIONS="-f /etc/ptp4l.conf -i ethX"
> ```
>
> `/etc/sysconfig/phc2sys` :
>
> ```
> OPTIONS="-s CLOCK_REALTIME -c ethX -w"
> ```
>
> `/etc/ptp4l.conf` :
>
> ```
> priority1 127
> ```

例 20.4: ソフトウエアタイムスタンプ機能を利用した自律クロック (一般的には非推奨)

> `/etc/sysconfig/ptp4l` :
>
> ```
> OPTIONS="-f /etc/ptp4l.conf -i ethX"
> ```
>
> `/etc/ptp4l.conf` :
>
> ```
> priority1 127
> ```

20.5 PTP と NTP

NTP と PTP の時刻同期ツールは同居させることができます。また、双方向の時刻同期を行うこともできます。

20.5.1 NTP から PTP への同期

`chronyd` を利用してローカルのシステム時計を同期させている場合、`ptp4l` を設定して最上位クロックとして動作させ、PTP 経由でローカルのシステム時計を配信することができます。この場合、`/etc/ptp4l.conf` の `priority1` オプションに下記を設定してください:

```
[global]
priority1 127
[eth0]
```

あとは `ptp4l` を起動します:

```
> sudo ptp4l -f /etc/ptp4l.conf
```

ハードウエアタイムスタンプ機能を使用している場合、`phc2sys` を利用して、PTP ハードウエアクロックをシステムクロックと同期させる必要があります:

```
> sudo phc2sys -c eth0 -s CLOCK_REALTIME -w
```

20.5.2 PTP-NTP ブリッジの設定

非常に精度の高い PTP 最上位クロックがネットワーク内に存在しているものの、PTP に対応するスイッチやルータが存在していない場合、コンピュータを PTP スレーブに設定するとともに、stratum-1 の NTP サーバとして設定することができます。このようなコンピュータには複数のネットワークインターフェイスが必要となるほか、最上位のクロックとネットワーク的に非常に近い場所に存在しているか、直結している必要があります。これにより、ネットワーク内でも非常に精度の高い同期を実現することができます。

`ptp4l` と `phc2sys` のプログラムを設定して、1 つのネットワークインターフェイスで PTP によるシステムクロックの同期を行うようにします。あとは `chronyd` を設定して、その他のインターフェイス経由でシステム時刻を提供するようにします:

```
bindaddress 192.0.131.47
hwtimestamp eth1
local stratum 1
```



注記: NTP と DHCP について

DHCP クライアントである `dhclient` が NTP サーバの一覧を取得すると、既定では NTP の設定ファイル内にそれを追加します。このような動作を防ぐには:

```
NETCONFIG_NTP_POLICY=""
```

上記の内容を、`/etc/sysconfig/network/config` ファイルに設定してください。

A GNU ライセンス

本付録には、GNU Free Documentation License バージョン 1.2 とその日本語訳 (八田真行氏 (mhatta@gnu.org) による翻訳) を収録しています。

GNU Free Documentation License

Copyright (C) 2000, 2001, 2002 Free Software Foundation, Inc. 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA. Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See https://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

```
Copyright (c) YEAR YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled “GNU
Free Documentation License”.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

GNU フリー文書利用許諾契約書

Copyright (C) 2000, 2001, 2002 Free Software Foundation, Inc. 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA. この利用許諾契約書を、一字一句そのままに複製し頒布することは許可する。しかし変更は認めない。

0. はじめに

この利用許諾契約書の目的は、この契約書が適用されるマニュアルや教科書、その他機能本位で実用的な文書を(無料ではなく)自由という意味で「フリー」とすること、すなわち、改変の有無あるいは目的の営利非営利を問わず、文書を複製し再頒布する自由をすべての人々に効果的に保証することです。加えてこの契約書により、著者や出版者が自分たちの著作物に対して相応の敬意と賞賛を得る手段も保護されます。また、他人が行った改変に対して責任を負わずに済むようになります。

この利用許諾契約書は「コピーレフト」的なライセンスの一つであり、この契約書が適用された文書から派生した著作物は、それ自身もまた原本と同じ意味でフリーでなければなりません。この契約書は、フリーソフトウェアのために設計されたコピーレフトなライセンスであるGNU一般公衆使用許諾契約書を補足するものです。

この利用許諾契約書は、フリーソフトウェア用のマニュアルに適用することを目的として書かれました。フリーソフトウェアはフリーな文書を必要としており、フリーなプログラムはそのソフトウェアが保証するのと同じ自由を提供するマニュアルと共に頒布されるべきだからです。しかし、この契約書の適用範囲はソフトウェアのマニュアルに留まりません。対象となる著作物において扱われる主題が何であれ、あるいはそれが印刷された書籍として出版されるか否かに関わらず、この契約書は文字で書かれたいかなる著作物にも適用することが可能です。私たちとしては、主にこの契約書を解説や参照を目的とする著作物に適用することをお勧めします。

1. この利用許諾契約書の適用範囲と用語の定義

著作物がこの利用許諾契約書の定める条件の下で頒布される旨の告知を、著作権者がその中に書いたすべてのマニュアルあるいはその他の著作物は、いかなる媒体上にあってもこの契約書の適用対象となる。そのような告知を置くことで、全世界において、著作権使用料を必要とせず、許可の存続期間を限定されること無く、この契約書の中で述べられている条件の下で当該著作物を利用できるという許可を与えることとする。以下において、「『文書』(Document)」とはそのような告知が記載されたマニュアルないし著作物すべてを指す。公衆の一員ならば誰でも契約の当事者となることができ、この契約書中では「あなた」と表現される。あなたは、著作権法の下で許可を必要とするような方法で著作物を複製や改変、あるいは頒布することにより、この契約書を受諾することになる。

『文書』の「改変版 (Modified Version)」とは、一字一句忠実に複製したか、あるいは改変や他言語への翻訳を行ったかどうかに関わらず、その『文書』の全体あるいは一部分を含む著作物すべてを意味する。

「補遺部分 (Secondary Section)」とは、『文書』中でその旨指定された補遺ないし本文に先だって前付けとして置かれる一部分であり、『文書』の出版者あるいは著者と、『文書』全体の主題(あるいはそれに関連する事柄)との関係のみを論じ、全体としての主題の範疇に直接属する内容を全く含まないものである(たとえば、『文書』の一部が数学の教科書だった場合、補遺部分では数学について何も解説してはならない)。補遺部分で扱われる関係は、その主題あるいは関連する事柄との歴史的なつながりのことかも知れないし、それらに関する法的、商業的、哲学的、倫理的、あるいは政治的立場についてかも知れない。

「変更不可部分 (Invariant Sections)」とは補遺部分の一種で、それらが変更不可部分であることが、『文書』をこの利用許諾契約書の下で発表する旨述べた告知中においてその部分の題名と共に明示されているものである。ある部分が上記のような「補遺」性の定義にそぐわない場合は、その部分を「変更不可」として指定することは認められない。『文書』は、変更不可部分を全く含まなくても良い。『文書』において変更不可部分が全く指定されていなければ、その『文書』に変更不可部分は存在しないということである。

「カバーテキスト(Cover Texts)」とは、『文書』がこの利用許諾契約書の指定する条件の下で発表される旨述べた告知において、「表カバーテキスト」あるいは「裏カバーテキスト」として列挙された短い文章のことを指す。表カバーテキストは最大で5語、裏カバーテキストは最大で25語までとする。

『文書』の「透過的」複製物とは、機械による読み取りが可能な『文書』の複製物のことを指す。透過的な複製物の文書形式は、その仕様が一般の人々に入手可能で、『文書』の内容を一般的なテキストエディタ、または(画素で構成される画像ならば)一般的なペイントプログラム、あるいは(図面ならば)いくつかの広く入手可能な製図エディタで簡単に改訂するのに適しており、なおかつテキストフォーマッタへの入力に適する(あるいはテキストフォーマッタへの入力に適する諸形式への自動的な変換に適する)ものでなければならない。透過的なファイル形式への複製であっても、マークアップ、あるいはマークアップの不在が読者によるそれ以降の改変をわざと邪魔し阻害するように仕組まれたものは透過的であるとは見做されない。ある画像形式が、相当量のテキスト文章を表現するために使われた場合、それは透過的ではない。透過的ではない複製は「非透過的」複製と呼ばれる。

透過的複製に適した形式の例としては、マークアップを含まないプレーンなASCII形式、Texinfo入力形式、LaTeX入力形式、一般に入手可能なDTDを用いたSGMLあるいはXML、または人間による改変を想定して設計された、標準に準拠したシンプルなHTMLやPostScript、PDFなどが挙げられる。透過的な画像形式の例には、PNGやXCF、JPGが含まれる。非透過な形式としては、独占的なワードプロセッサでのみ閲覧編集できる独占的なファイル形式、普通には入手できないDTDまたは処理系を使ったSGMLやXML、ある種のワードプロセッサが生成する、出力のみを目的とした機械生成のHTMLやPostScript、PDFなどが含まれる。

「題扉 (Title Page)」とは、印刷された書籍に於いては、実際の表紙自身のみならず、この利用許諾契約書が表紙に掲載することを義務づける文章や図などを、読みやすい形で載せるのに必要なだけの、表紙に引き続く数ページをも意味する。表紙に類するものが無い形式で発表される著作物においては、「題扉」とは本文の始まりに先だって、その著作物の題名が最も目立つ形で現れる場所の近くに置かれる文章のことを指す。

「XYZと題された (Entitled XYZ)」部分とは、『文書』において「XYZ」と名付けられた一部分であり、その題名は正確に「XYZ」であるか、「XYZ」を他の言語に翻訳した上でその後ろに「XYZ」をそのまま括弧で括ったものを含む記述のどちらかである(ここでの「XYZ」とは、この利用許諾契約書において以下で言 及される特定の部分名を意味している。例えば「謝辞 (Acknowledgements)」、「献辞 (Dedications)」、「推薦の辞 (Endorsements)」、「履歴 (History)」)。あなたが『文書』を改変する場合、そのような部分の「題名を保存する (Preserve the Title)」とは、「XYZと題された」部分として、ここでの定義に従い題名を残すということである。

『文書』は、「保証否認警告 (Warranty Disclaimers)」を、この利用許諾契約書が『文書』に適用されると述べた告知の次に含んでも良い。この種の保証否認警告は、この契約書からの言及という形で利用条件に含まれるものと解されるが、保証の否認に関することについてのみ有効とする。こういった保証否認警告で示しうるその他のいかなる含意も無効であり、この契約書の効能には何ら影響を持たない。

2. 逐語的に忠実な複製

この利用許諾契約書、著作権表示、この契約書が『文書』に適用される旨述べた告知の三つがすべての複製物に複製され、かつあなたがこの契約書で指定されている以外のいかなる条件も追加しない限り、あなたはこの『文書』を、商用であるか否かを問わずいかなる形でも複製頒布することができる。あなたは、あなたが作成あるいは頒布する複製物に対して、閲覧や再複製を技術的な手法によって妨害、規制してはならない。しかしながら、複製と引き換えに代価を得てもかまわない。あなたが相当量の複製物を頒布する際には、本契約書第3項で指定される条件にも従わなければならない。

またあなたは、上記と同じ条件の下で、複製物を貸与したり複製物を公に開示することができる。

3. 大量の複製

もしあなたが、『文書』の印刷された(あるいは通常は印刷された表紙を持つ媒体における)複製物を100部を超えて出版し、また『文書』の利用許諾告知がカバーテキストの掲載を要求している場合には、指定されたすべてのカバーテキストを、表カバーテキストは表表紙に、裏カバーテキストは裏表紙に、はっきりと読みやすい形で載せた表紙の中に複製物本体を綴じ込まなければならない。また、両方の表紙において、それらの複製物の出版者としてのあなたをはっきりとかつ読みやすい形で確認できなければならない。表表紙では『文書』の完全な題名を、題名を構成するすべての語が等しく目立つようにして、視認可能な形で示さなければならない。それらの情報に加えて、表紙に他の文章や図などを加えることは許可される。表紙のみを変更した複製物は、それが『文書』の題名を保存し上記の条件を満たす限り、ほかの点では逐語的に忠実な複製物 として扱われる。

もしどちらかの表紙に要求されるカバーテキストの量が多すぎて読みやすく収めることが不可能ならば、あなたはテキスト先頭の一文(あるいは適切に収まるだけ)を実際の表紙に載せ、続きは隣接したページに載せるべきである。

あなたが『文書』の「非透過的」複製物を100部を超えて出版あるいは頒布する場合、それぞれの非透過な複製物と一緒に機械で読み取り可能な透過的複製物を添付するか、それぞれの非透過な複製物(あるいはそれに付属する文書)中で、公にアクセス可能なコンピュータネットワーク上の所在地を記述しなければならない。その場所には、非透過な複製物と内容的に寸分違わず、余計なものが追加されていない完全な『文書』の透過的複製物が置かれ、またそこから、ネットワークを利用する一般公衆が、一般に標準的と考えられるネットワークプロトコルを使ってダウンロードすることができなければならない。もしあなたが後者の選択肢を選ぶならば、その版の非透過な複製物を公衆に(直接、あるいはあなたの代理人ないし小売業者が)最後に頒布してから最低1年間は、その透過的複製物が指定の場所でアクセス可能であり続けることを保証するよう、非透過な複製物の大量頒布を始める際に十分に慎重な手順を踏まなければならない。

これは要望であり必要条件ではないが、『文書』の著者に、『文書』の更新された版をあなたに提供する機会を与えるため、透過非透過を問わず大量の複製物を再頒布し始める前には彼らにきちんと連絡しておいてほしい。

4. 改変

『文書』の改変版を、この利用許諾契約書と細部まで同一の契約の下で発表する限り、すなわち原本の役割を改変版で置き換えた形での頒布と改変を、その複製物を所有するすべての人々に許可する限り、あなたは改変版を上記第2項および第3項が指定する条件の下で複製および頒布することができる。さらに、あなたは改変版において以下のことを行わなければならない。

A. 題扉に(もしあればその他の表紙にも)、『文書』お よび『文書』のそれ以前の版と見分けがつく題名を載せること(もし以前の 版があれば、『文書』の「履歴 (History)」の部分に列記されているはずである)。もし元の版の出版者から許可を得たならば、以前の版と同じ題名を 使っても良い。

B. 題扉に、改変版における改変を行った1人以上の人物か団体名を列記すること。あわせて元の『文書』の著者として、最低5人(もし5人以下ならばすべて)の主要著者を列記すること。ただし元の著者たちがこの条件を免除した場合は除く。

C. 題扉に、改変版の出版者名を出版者として記載すること。

D. 『文書』にあるすべての著作権表示を残すこと。

E. 他の著作権表示の近くに、あなたの改変に対する適当な著作権表示を追加すること。

F. 著作権表示のすぐ後に、改変版をこの契約書の条件の下で利用することを公衆に対して許可する告知を含めること。その形式はこの契約書の末尾にある付記で示されている。

G. 元の『文書』の利用許諾告知に書かれた、変更不可部分の完全な一覧と、要求されるカバーテキストとを、改変版の利用許諾告 知でもそのまま残すこと。

H. この契約書の、変更されていない複製物を含めること。

I. 「履歴 (History)」と題された部分とその題名を保存し、そこに改変版の、少なくとも題名、出版年、新しく変更した部分の著 者名、出版者名を、題扉に掲載するのと同じように記載した一項を加えること。もし『文書』中に「履歴」と題された部分が存在しない場合には、『文書』の題名、出版年、著者、出版者を題扉に掲載するのと同じように記載した部分を用意し、上記で述べたような、改変版を説明する一項を加えること。

J. 『文書』中に、『文書』の透過的複製物への公共的アクセスのために指定されたネットワーク的所在地が記載されていたならば、それを保存すること。同様に、その『文書』の元になった以前の版で指定されていたネットワーク的所在地も載っていたならば、それも保存すること。これらの情報は「履歴(History)」の部分に置いても良い。ただし、それが『文書』自身より少なくとも4年前に出版された著作物の情報であったり、あるいは改変版が参考にしている版の元々の出版者から許可を得たならば、その情報を削除してもかまわない。

K. 「謝辞 (Acknowledgement)」あるいは「献辞 (Dedication)」等と題されたいかなる部分も、その部分の題名を保存し、その部分の内容(各貢献者への謝意あるいは献呈の意)と語調を保存すること。

L. 『文書』の変更不可部分を、その本文および題名を変更せずに保存すること。章番号やそれに相当するものは部分の題名の一部とは見做さない。

M. 「推薦の辞 (Endorsement)」というような章名が題された部分はすべて削除すること。そのような部分を改変版に含めてはならない。

N. すでに存在する部分を「推薦の辞 (Endorsement)」と題されるように改名したり、題名の点で変更不可部分のどれかと衝突するように改名してはならない。

O. 保証否認警告を保存すること。

もし改変版に、補遺部分としての条件を満たし、かつ『文書』から複製物された文章や図などをいっさい含んでいない、前書き的な章あるいは付録が新しく含まれるならば、あなたは希望によりそれらの部分の一部あるいはすべてを変更不可と宣言することができる。変更不可を宣言するためには、それらの部分の題名を改変版の利用許諾告知中の変更不可部分一覧に追加すれば良い。これらの題名は他の章名とは全く別のものでなければならない。

含まれる内容が、さまざまな集団によるあなたの改変版に対する推薦の辞のみである限り、あなたは、「推薦の辞 (Endorsement)」と題された章を追加することができる。推薦の辞の例としては、ピアレビューの陳述、あるいは文書がある標準の権威ある定義としてその団体に承認されたという声明などがある。

あなたは、5語までの一文を表カバーテキストとして、25語までの文を裏表紙テキストとして、改変版のカバーテキスト一覧の末尾に加えることができる。一個人ないし一団体が直接(あるいは団体内で結ばれた協定によって)加えることができるのは、表カバーテキストおよび裏カバーテキストとしてそれぞれ一文ずつのみである。もし以前すでにその文書において、表裏いずれかの表紙にあなたの(またはあなたが代表する同じ団体内で為された協定に基づく)カバーテキストが含まれていたならば、あなたが新たに追加することはできない。し かしあなたは、その古い文を加えた以前の出版者から明示的な許可を得たならば、古い文を置き換えることができる。

『文書』の著者あるいは出版者は、この利用許諾契約書によって、彼らの名前を利用することを許可しているわけではない。彼らの名前を改変版の宣伝に使ったり、改変版への明示的あるいは黙示的な保証のために使うことを許可するものではない。

5. 文書の結合

あなたは、上記第4項において改変版に関して定義された条件の下で、この利用許諾契約書の下で発表された複数の文書を一つにまとめることができる。その際、原本となる文書にある変更不可部分を全て、改変せずに結合後の著作物中に含め、それらをあなたが統合した著作物の変更不可部分としてその利用許諾告知において列記し、かつ原本にある全ての保証否認警告を保存しなければならない。

結合後の著作物についてはこの契約書の複製物を一つ含んでいればよく、同一内容の変更不可部分が複数ある場合には一つで代用してよい。もし同じ題名だが内容の異なる変更不可部分が複数あるならば、そのような部分のそれぞれの題名の最後に、(もし分かっているならば)その部分の原著者あるいは出版者の名前で、あるいは他と重ならないような番号を括弧で括って記載することで、それぞれ見分けが付くようにしなければならない。結合後の著作物の利用許諾告知における変更不可部分の一覧においても、章の題名に同様の調整をすること。

結合後の著作物においては、あなたはそれぞれの原本の「履歴 (History)」と 題されたあらゆる部分をまとめて、「履歴 (History)」と題された一章にしなければならない。同様に、「謝辞 (Acknowledgements)」あるいは「献辞 (Dedications)」と題されたあらゆる部分もまとめなければならない。あなたは「推薦の辞 (Endowsements)」と題されたあらゆる部分も削除しなければならない。

6. 文書の収集

あなたは、この利用許諾契約書の下で発表された複数の文書で構成される収集著作物を作ることができる。その場合、それぞれの文書が逐語的に忠実に複製されることを保障するために他のすべての点でこの契約書の定める条件に従う限り、さまざまな文書中のこの契約書の個々の複製物を、収集著作物中に複製物を一つ含めることで代用することができる。

あなたは、このような収集著作物から文書を一つ取り出し、それをこの契約書の下で頒布することができる。ただしその際には、この契約書の複製物を抽出された文書に挿入し、またその他すべての点でこの文書の逐語的に忠実な複製に関してこの契約書が定める条件に従わなければならない。

7. 独立した著作物の集積

『文書』あるいはその派生物を、他の別の独立した文書あるいは著作物と一緒にし、一巻の記憶装置あるいは頒布媒体に収めた編集著作物は、編集に起因する著作権が編集著作物に含まれる個々の著作物がその利用者に許可した法的権利を制限するよう行使されない限り、「集積」著作物と呼ばれる。『文書』が集積著作物に含まれる場合、この契約書は、『文書』と共にまとめられた他の独立した著作物には、それら自身が『文書』の派生物で無い限り適用されることにはならない。

このような『文書』の複製物において、この利用許諾契約書の第3項によりカバーテキストの掲載が要求されている場合、『文書』の量が集積著作物全体の2分の1以下であれば、『文書』のカバーテキストは集積著作物中で『文書』そのものの周りを囲む中表紙、あるいは『文書』が電子的形式である場合には表紙の電子的等価物にのみ配置するだけでよい。その場合以外は、カバーテキストは集積著作物全体を取り巻く印刷された表紙に掲載されなければならない。

8. 翻訳

翻訳は改変の一種と見做すので、あなたは『文書』の翻訳をこの利用許諾契約書の第4項の定める条件の下で頒布することができる。変更不可部分を翻訳によって置き換えるには著作権者の特別許可を必要とするが、元の変更不可部分に追加する形で変更不可部分の全てないし一部の翻訳を含めることはかまわない。この契約書や『文書』中の利用許諾告知、保証否認警告すべての英語原本も含める限り、あなたはこの契約書、告知、警告の翻訳を含めることができる。契約書や告知、警告に関して翻訳と英語原本との間に食い違いが生じた場合、英語原本が優先される。

典型的な例として、『文書』のある部分が原文で「Acknowledgements」、「Dedications」、あるいは「History」と題されていた場合、実際の題名を変更するには、題名を保存する(この契約書の第1項)ための条件(同第4項)を満たすことが必要となる。

9. 契約の終了

この利用許諾契約書の下で明確に提示されている場合を除き、あなたは『文書』を複製、改変、サブライセンス、あるいは頒布してはならない。このライセンスで指定されている以外の、『文書』の複製、改変、サブライセンス、頒布に関するすべての企ては無効であり、この契約書によって保証されるあなたの権利を自動的に終結させることとなる。しかし、この契約書の下であなたから複製物ないし諸権利を得た個人や団体に関しては、そういった人々がこの契約書に完全に従ったままである限り、彼らに与えられた許諾は終結しない。

10. 将来における本利用許諾契約書の改訂

フリーソフトウェア財団は、時によってGNUフリー文書利用許諾契約書の新しい改訂版を出版することができる。そのような新版は現在の版と理念においては似たものになるであろうが、新たに生じた問題や懸念を解決するため細部においては違ったものになるだろう。詳しくは https://www.gnu.org/copyleft/ を参照せよ。

GNUフリー文書利用許諾契約書のそれぞれの版には、新旧の区別が付くようなバージョン番号が振られている。もし『文書』において、この契約書のある特定の版か「それ以降のどの版でも」適用して良いと指定されている場合、あなたはフリーソフトウェア財団から発行された(草稿として発表されたものを除く)指定の版かそれ以降の版のうちどれか一つを選び、その条項や条件に従うことができる。もし『文書』がこの契約書のバージョン番号を指定していない場合には、あなたはフリーソフトウェア財団から今までに出版された(草稿として発表されたものを除く)版のうちからどれか一つを選ぶことができる。

ADDENDUM: この利用許諾契約書をあなたの文書に適用するには

この利用許諾契約書をあなたが書いた文書に適用するには、この契約書の複製物一つを文書中に含め、以下に示す著作権表示と利用許諾告知を題扉のすぐ後に置いて下さい:

```
Copyright (c) YEAR YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled “GNU
Free Documentation License“.
```

(訳: Copyright (C) 西暦年 あなたの名前. この文書を、フリーソフトウェア財団発行のGNU フリー文書利用許諾契約書(バージョン1.2かそれ以降から一つを選択)が定める条件の下で複製、頒布、あるいは改変することを許可する。変更不可部分、表カバーテキスト、裏カバーテキストは存在しない。この利用許諾契約書の複製物は「GNU フリー文書利用許諾契約書」という章に含まれている。)

もし変更不可部分や表カバーテキスト、裏カバーテキストがあれば、「変更不可部分…は存在しない。」というところを以下で置き換えてください:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

(訳: (章の題名を列記)は変更不可部分であり、(表カバーテキストを列記)は表カバーテキスト、(裏カバーテキストを列記)は裏カバーテキストである。)

変更不可部分はあるがカバーテキストは存在しないなど、その他の三者の組み合せに関しては、状況に合わせて上記二つの選択肢を混ぜてください。

あなたの文書に、他に類を見ない独自のプログラムコードのサンプルが含まれる場合、フリーソフトウェアにおいてそのコードを利用することを許可するために、そういったサンプルに関してはこの利用許諾契約書と同時にGNU一般公衆許諾契約書のようなフリーソフトウェア向けライセンスのうちどれか一つを選択して適用してもよい、というような条件の下で発表することを推奨します。