LFS301-JP

Linux システム管理

Version 7.1

Version 7.1

発行元：

The Linux Foundation
https://www.linuxfoundation.org

目次

注目

** これらのセクションの一部または全体をオプション扱いとする場合があります。これらには補足資料、専門トピック、または高度な話題が含まれインストラクターが教室の状況や時間制約に応じてこれらの内容を紹介するかを判断します。

List of Figures

List of Tables

章 1

はじめに

1.1 Linux Foundation

Linux Foundation とは

- 次項の成長を推進することに専念した非営利のコンソーシアム:
  - **Linux**
  - その他多くの**オープンソースソフトウェア** (**OSS**) プロジェクトとコミュニティ
- 次項を提供することで持続可能な **OSS** エコシステムの創造をサポート:
  - 財政的・知的資源とサービス
  - トレーニング
  - イベント
- 元々は **Linux** 開発を保護・サポート・改善し **Linux** 創始者リーナス・トーバルズ氏のスポンサーとして設立
- 中立的な共同作業環境における大手テック企業や開発者が支援
- 詳細: https://www.linuxfoundation.org

Linux Foundation は **Linux** の成長を推進することに専念した非営利のコンソーシアムです。2000 年に設立され、**Linux** 創始者である**リーナス・トーバルズ氏**のスポンサーです。また、世界中の大手テクノロジー企業や開発者によって支えられています。

今日ではこの使命を越えて、他の **OSS** プロジェクトとコミュニティのホストをサポートするまでに成長しました。ホストしている財団や組織は日々増え続けています。**Linux Foundation** は会員やオープンソースコミュニティのリソースを集結し、**Linux** がオープンでありながらも高度な技術を保つことを保証します。そのような活動を通して **Linux** を推奨し保護し発展させているのです。

これらのプロジェクトのための中立的なスポークスパーソンという役割を担い、それらのプラットフォームの理解を深める独自のリサーチとコンテンツを作成しています。https://www.linux.comを含むウェブページには毎月数 100 万人がアクセスします。また多くのコラボレーションイベントを主催することによって技術革新を促進します。このようなイベントでは技術コミュニティやアプリケーション開発者、熟練者やエンドユーザーが集まりエコシステムが抱えている緊急な課題を共に解決します。こうしてエンドユーザーや開発者、企業が技術的や法的な、またはマーケティングに関する問題について協力し合うことができるのです。

本来の使命に忠実であるために **Linux** 開発者のリーナス・トーバルズ氏や他の中心的なカーネル開発者達がフルタイムで **Linux** の改善作業に取り組めるように **Linux Foundation** が直接スポンサーしています。彼らの独立性を維持することは非常に重要なことだからです。

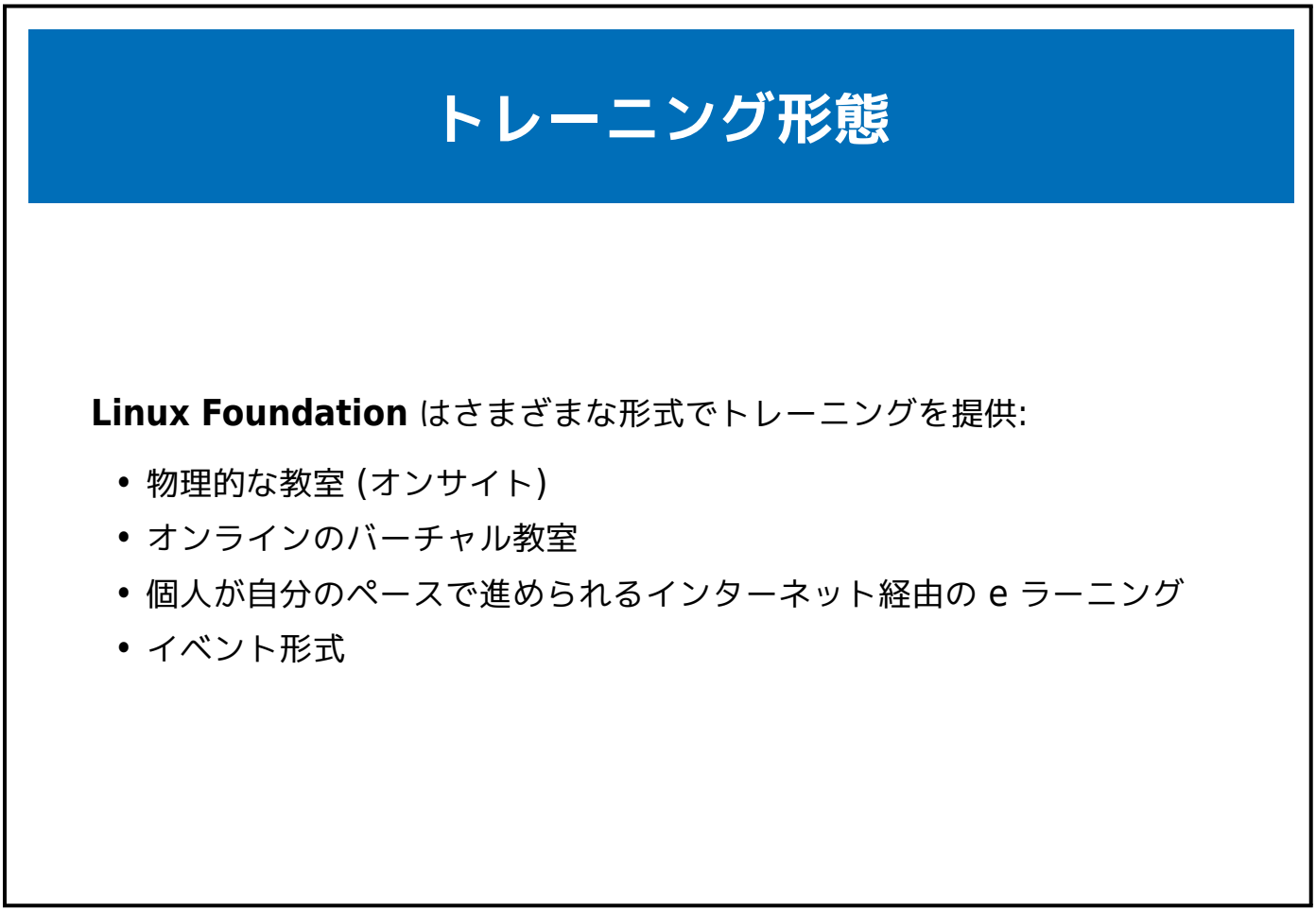
Linux Foundation 主催のイベント

Linux Foundation が主催するイベントのごく一部
（北米・ヨーロッパ・アジアなど各地で行われるイベントを含む）

- Open Source Summit (北米 / 欧州 / 日本)
- Linux Kernel Summit
- Embedded Linux Conference (北米 / 欧州)
- Automotive Linux Summit
- Linux Security Summit
- Hyperledger Global Forum
- KubeCon + CloudNativeCon (欧州 / 北米)
- Open Networking Summit
- LF Energy Summit
- OpenIot Summit
- OpenFinTech Summit
- Cloud Foundry Summit
- Vault
- KVM Forum
- Linux Storage Filesystem and MM Summit
- 2018 年 11 月現在

Linux Foundation は世界中でテクノロジー関係のイベントを開催しています。目的はさまざまです。例えば次のカーネルリリース開発のためのオープンフォーラムの提供や、リアルタイムな環境で問題解決できるように開発者を集めること、またはアクティブなディスカッションができるようなワークグループやコミュニティグループをホストすることだったりします。あるいは企業レベルで **Linux** 普及を促進するコミュニティ内のコラボレーションを奨励するためにカーネル開発者とエンドユーザーが交流できる場を提供することが目的だったりします。目的はさまざまでも私達が主催する技術者会議が **Linux** プラットフォームの前進を促す比類ない雰囲気を作り出していると自負しています。

1.2 Linux Foundation が提供するトレーニング



Linux Foundation のトレーニングはコミュニティによるコミュニティのためのものであり、開発コミュニティのリーダー達自身が提供するトレーニング内容を使用したり彼らを直々に講師として迎えます。

受講者は OS や **Linux** ディストリビューションに囚われない、高度な技術を取り入れた、そして開発コミュニティの実際のリーダー達自身によって作成されたトレーニングを受けることができます。**Linux Foundation** のさまざまな講座は受講者が今の仕事で活き活きできるために必要な幅広い、基盤となる知識やネットワーキングを提供します。オンラインのトレーニングでもそうでなくても **Linux Foundation** の講座はあなたに、そしてあなたの開発者達に、最新のオープンソースの管理と開発の知識を提供いたします。

Linux Foundation が提供するトレーニング

現在受講可能な講座の種類:

- Linux プログラミング & 開発トレーニング
- エンタープライズ IT & Linux システム管理講座
- オープンソースコンプライアンス講座

詳細: https://training.linuxfoundation.org

Linux Foundation は他にも幅広い分野の無料 **MOOC**（ムーク）を提供しています。**MOOC**(**M**assively **O**pen **O**nline **C**ourses) とはインターネット上で誰もが無料で受講可能な大規模公開講座です。**edX** を利用しており https://www.edx.org でアクセスできます。これらの講座にはオープンソースに関する基礎的な内容から上級者向けの内容まで含みます。

edX のウェブサイトから、"Linux Foundation"のキーワードで検索可能です。

著作権

- この講座のコンテンツと関連するすべての教材（ハンドアウトを含む）は Copyright 2020 The Linux Foundation によって保護されています。
- このトレーニングのいかなる資料も、The Linux Foundation の書面による事前の許可なく、いかなる形式または電子的、磁気的、機械的、写真複写的、記録的、その他のいかなる手段によっても、複製、配布、再出版、ダウンロード、表示、掲示、転送、または情報検索システムに格納することを禁じます。

The Linux Foundation は本トレーニングで使用する一切の資料についていかなる保証を行いません。トレーニングのコンテンツの正当なあるいは不正な利用によって生じた損害や法的行為につき **The Linux Foundation** は責任を一切負わないものとします。

The Linux Foundation の資料の不正な配布、複製、および利用を見かけましたら、training@linuxfoundation.org にメールで、もしくは (USA) +1-415-723-9709 に電話でご連絡ください。

1.3 Linux Foundation の資格

LFCS と LFCE

2 段階の**資格プログラム**を提供:

- **LFCS**: **L**inux **F**oundation **C**ertified **S**ysadmin（LF 認証システム管理者)
- **LFCE**: **L**inux **F**oundation **C**ertified **E**ngineer（LF 認証エンジニア）

資格プログラムの詳細:
https://training.linuxfoundation.org/certification （各試験の**出題範囲**と**必要なスキル**などを参照可能）

Linux Foundation は **LFCS** と **LFCE** 試験の他にも、**Linux Foundation** が参加している共同プロジェクトにてオープンソース認証プログラム用に試験を作成しました（あるいは今現在作成中です）：

- **Certified Kubernetes Administrator** (**CKA**)
- **Certified Kubernetes Application Developer** (**CKAD**)
- **Certified Hyperledger Fabric Administrator** (**CHFA**)
- **Cloud Foundry Certified Developer** (**CFCD**)
- **Certified Hyperledger Sawtooth Administrator** (**CHSA**)

トレーニングと試験の「ファイアウォール」

- **Linux Foundation** はトレーニングを 2 つに分割:
  - 試験
  - 講座
- これらは論理的な**ファイアウォール**によって隔離:
  - 第三者の企業や組織が **LF** 認証試験に向けた講座を作成し提供することができる
  - **LF** の講座でしか得られない秘訣（いわゆる秘密のレシピ）が生まれるのを防ぐ
  - **LF** の講座で **テスト対策だけを目標とした**教え方を防ぐ
- 講師は一般的な公開情報のみに基いた講義を行う

カリキュラムの作成と管理を担当する **Linux Foundation** トレーニング部署は認証試験の作成・管理・採点に直接関わることは一切ありません。

このように自ら**「ファイアウォール」**を立てることにより、**Linux Foundation** とは関係のない組織や企業でも受験者が認証試験に受かる手助けとなるトレーニング教材を第三者として作成することができるのです。

また試験に受かるために必要な秘策（秘密の勉強法など）がないことを保証できるのです。

さらに **Linux Foundation** が提供するトレーニングが単なるテスト対策に留まらず、受講者が **Linux** システム管理者として成功するために必要な幅広い知識を提供できるような堅牢で充実したものであることを保証できるのです。

試験準備用のリソース

- まずこちらをダウンロード:
  https://training.linuxfoundation.org/download-free-certification-prep-guide
- 次のセクションを含む:
  - ドメインとコンピテンシー（トレーニングを通して身につくスキル）
  - 無料のトレーニングリソース
  - 有料のトレーニングリソース
  - 試験を受ける
- **受験者ハンドブック**はこちら:

https://training.linuxfoundation.org/go/candidate_handbook

- 各試験の詳細はこちら:

https://training.linuxfoundation.org/certification/lfcs
https://training.linuxfoundation.org/certification/lfce

上記のドキュメントで、すべてではありませんが、多くの内容について触れています。そしてこれらのドキュメントはすべて **Linux Foundation** のウェブサイトでも確認できます: https://training.linuxfoundation.org/certification
次のような内容を含みます:

- 試験範囲
- 試験を受ける必要条件（身分証明・認証・適任性・アクセシビリティ・機密性に関する必要条件）
- 受験費や返金ポリシーなど
- 利用可能な **Linux** ディストリビューション
- ハードウェア環境とソフトウェア環境が試験に適応しているかの確認
- 試験の始め方・終わり方
- 試験のインターフェイスとフォーマット
- 試験結果・採点と再採点の依頼・再受験ポリシー
- 認定証の発行・検証・失効・更新など
- 技術的サポートへのアクセス

1.4 Linux Foundation のデジタルバッジ

Linux Foundation のデジタルバッジ

図 1.1: **Linux Foundation のデジタルバッジ**

- デジタルバッジで技能と証明を提示
- メールの署名、電子レジュメ、ソーシャルメディア (**LinkedIn**、**Facebook**、**Twitter** など) で利用可能
- 資格と獲得手順を記載した検証済みメタデータを含む
- **Linux Foundation** のトレーニングコースと認定試験に合格した受講者に提供
- 詳細: https://training.linuxfoundation.org/badges/

Linux Foundation は専門的な目標を達成するために必要な機能を提供することをお約束しています。私達は技能や証明の提示が難しいことを理解しています。このため、私達は **Credly** 社とパートナー契約を結び、**Acclaim** プラットフォームを通してあなたの証明書の電子版を提供することにしました。これらのバッジは、メールの署名や電子レジュメ、**LinkedIn**、**Facebook**、**Twitter** のようなソーシャルメディアでも利用可能です。このデジタルイメージは、資格と獲得手順を記載した検証済みメタデータを含みます。

- バッジはあらゆる電子プラットフォーム経由で共有可能です: ソーシャルメディア、レジュメに埋め込み、電子メールの署名、ウェブ上など。
- すべてのバッジはクリックするだけで誰にでも検証可能です。
- バッジは認定試験にパスした人なら誰にでも発行します。また
- **Linux Foundation** や認定トレーニングパートナーが提供するトレーニングを受講した人にも発行します。
- バッジは試験やトレーニングコース開発チームに貢献した方にも発行します。

動作方法

1. 私達のパートナーの **Credly** 社の **Acclaim** プラットフォームウェブサイトにバッジ要求したことを通知するメールを受信
2. メールに記載のリンクをクリック
3. **Acclaim** プラットフォームサイトにアカウントを作成しメールを確認
4. バッジを確認
5. 共有を開始

1.5 演習・解答・リソース

演習

- 各セッションの終わりにハンズオン演習を実施
- 仮想マシンでもベアメタルサーバでも実施可能
  - 一部、仮想マシンでは出来ないものもあり
- 課題の一部は任意
- 各課題の終わりに解答あり

講座の解答とその他リソースの入手方法

- **本講座用の演習の解答・参考資料などがある場合**
  https://training.linuxfoundation.org/cm/LFS301-JPからダウンロード可能
- 利用できるブラウザがない場合:

```
$ wget --user=LFtraining --password=<講師提供> \
  https://training.linuxfoundation.org/cm/LFS301-JP/LFS301-JP_V7.1_SOLUTIONS.tar.bz2
```

  RESOURCESファイルも次の手順で入手可能:

```
$ wget --user=LFtraining --password=<講師提供> \
  https://training.linuxfoundation.org/cm/LFS301-JP/LFS301-JP_V7.1_RESOURCES.tar
```

- 誤字などの正誤表や更新したソリューションもこのサイトで確認可能
- これらのファイルは次の手順で展開可能:

```
$ tar xvf LFS301-JP_*SOLUTIONS*.tar.bz2
$ tar xvf LFS301-JP_*RESOURCES*.tar
```

SOLUTIONSディレクトリとRESOURCESディレクトリに各セッション用に s_01, s_10, s_13 のようなサブディレクトリがあります。ここには必要なファイルや参考資料が入っています。これからのセクションでもこれらのファイルに触れることがあるでしょう。

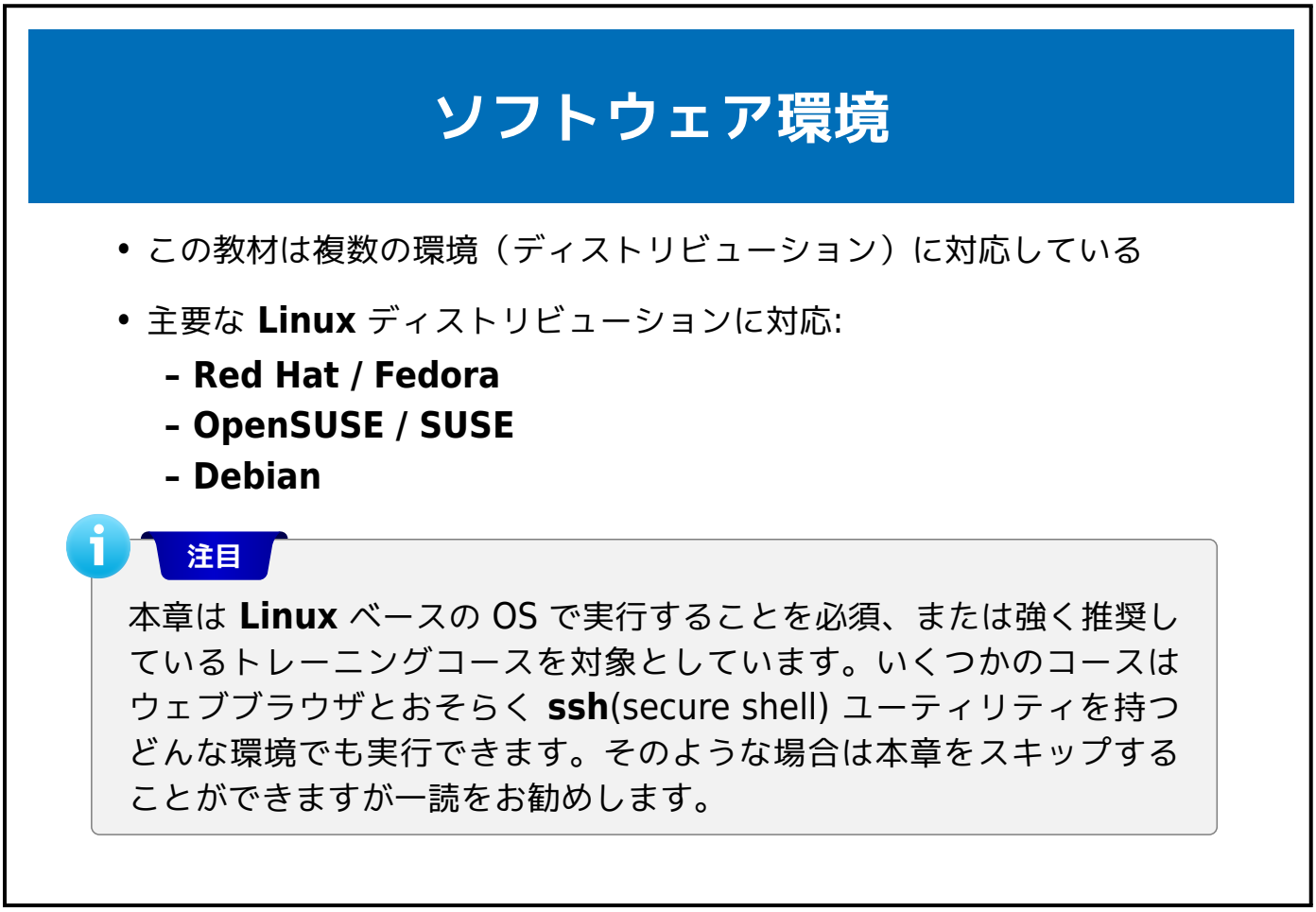
講座によっては、RESOURCESファイルがない可能性もあります。RESOURCESファイルはアーカイブや動画のようなバイナリファイルのためのものです。演習のためのソースの tar ファイルなどのバイナリファイルは、必要に応じて説明付きで提供します。デモ動画を含む講座もあります。その場合は実際の講座の時間以外で自習として観るものとして提供しています。しかしデモを見せるために、あるいは他の **Linux** ディストリビューションでの実行例を見せるために講師が見せる場合もあります。

1.6 e ラーニング講座: LFS201-JP

関連 e ラーニング講座: LFS201-JP

- **Linux Foundation** はこの講座と非常によく似た内容の自分のペースで進められる e ラーニング版講座も提供しています。
  LFS201-JP: Linux システム管理入門
- この e ラーニング講座の大部分は本講座と共通のコンテンツや演習を使用しています。
- １年間オンラインで教材にアクセスできるのですべてのコンテンツや演習を終えるために必要な時間をとることができます。
- 講師付きの場合はすべての題材で必要な深さと幅を追求するための充分な時間をとることができません。またいくつかの題材はオプショナル扱いとなっています。
- この講座への登録時に **LFS201-JP** へのサブスクリプションを受け取ったかもしれません。受けとっていない場合は https://training.linuxfoundation.org/ からアクセスすることができます。

1.7 ディストリビューションについての詳細



Linux Foundation が提供する教材は複数のディストリビューションに対応しているので１つのディストリビューションに縛られません。
つまり技術的な説明、演習、手順の説明などはほとんどの最近のディストリビューションでそのまま実行可能です。また特定のベンダの商品を推奨することはありません（ただし具体例で紹介することはあります）。

実践的にはこの教材の大部分は３つの主要な **Linux** ディストリビューションを対象として書いています: **Red Hat / Fedora**、**OpenSUSE / SUSE**、**Debian**。受講者は通常この３つの系列のディストリビューション、あるいはそれらを元にした商品を利用しています。

ディストリビューションの選択

- 考慮すべき点:
  - 会社で決められたディストリビューションがあるか
  - 新しいことを学びたいか
  - 保証が必要か
- ２つ以上のディストリビューションの試用を推奨

新しくディストリビューションを選ぶときはいくつかの質問を自分に投げかけるべきです。ある特定の **Linux** ディストリビューションに集中したくなる理由は多くあるでしょう。しかし私達はすべてのディストリビューションを体験することをおすすめします。技術的な違いは主にパッケージ管理システム・ソフトウェアのバージョン・ファイルの位置などに限るということがすぐにわかるはずです。その違いさえ理解できれば１つの **Linux** ディストリビューションから他のディストリビューションへの切り替えが非常に簡単になります。あるツールや機能には（特に特定の、あるいは複雑なレポートの場合）ベンダが提供するフロントエンドがあります。違うプラットフォームで実行するには教材に記載されている手順をところどころ修正する必要があるかもしれません。

Red Hat / Fedora 系

- 教材は出版時での最新の **Red Hat Enterprise Linux (RHEL)** 7.x のリリースに基づく
- **x86** · **x86-64** · **Itanium** · **PowerPC** · **IBM System Z** をサポート
- RPM ベースでインストールやアップデートには **yum**(あるいは **dnf**) を使用
- リリースサイクルが長く、エンタープライズ級のサーバ環境向け
- **CentOS** · **Scientific Linux** · **Oracle Linux** の元となったディストリビューション

注目

デモや演習では無料の **CentOS** を使用しています。

Fedora は、**Red Hat Enterprise Linux** · **CentOS** · **Scientific Linux** · **Oracle Linux** の基盤となるコミュニティ主導ディストリビューションです。**Fedora** は **Red Hat** の商用版のバージョンよりはるかに多くのソフトウェアを含んでいます。その理由の１つは **Fedora** が１つの会社ではなく多様なコミュニティによって開発されているからです。

Fedora コミュニティは半年に一度新しいバージョンをリリースしています。そのため教材の **Red Hat / Fedora** の部分ではリリースサイクルが長い **CentOS 7** の最新のバージョンを元に標準化することにしました。インストールしてしまえば **CentOS** はエンタープライズ環境で最もポピュラーな **Linux** ディストリビューションである **Red Hat Enterprise Linux** (**RHEL**) とほぼ同じなのです。

OpenSUSE 系

- 教材は、出版時での最新の **OpenSUSE** のリリースに基づく
- RPM ベースで、インストールやアップデートには **zypper** を使用
- 管理者用に **YaST** も存在
- **x86** と **x86-64**
- **SUSE Linux Enterprise Server (SLES)** の元となったディストリビューション

注目

デモや演習では無料の **OpenSUSE** を使用しています。

OpenSUSE と **SUSE Linux Enterprise Server** の関係は、前述の **Fedora** と **Red Hat Enterprise Linux** の関係と似ています。しかし **SUSE Linux Enterprise Server** の無料版を見つけることが困難なため **OpenSUSE** 系の部分では **OpenSUSE** を元にすることにしました。この 2 つの商品は非常によく似ていて **OpenSUSE** に関する教材はほとんどの場合、問題なく **SUSE Linux Enterprise Server** にも対応します。

Debian 系

- サーバやデスクトップパソコンでの利用が多い
- **DPKG** ベースで、インストールやアップデートには **apt** とフロントエンドアプリを使用
- **Ubuntu** · **Linux Mint** などの元となったディストリビューション
- 教材は、最新の **Ubuntu** のリリースに基づく
- **x86** と **x86-64**
  - 長期保守版（Long Term Release = LTS）

注目

デモや演習では **Ubuntu** を使用しています。**Debian** も無償で利用可能ですが **Ubuntu** の方が新規 **Linux** ユーザーに多く使われているためです。

Debian ディストリビューションは、**Ubuntu** や **Linux Mint** を含むいくつかのディストリビューションの元となっています。**Debian** は純粋なオープンソースプロジェクトであり、安定性を重視しています。最も大きく完全なソフトウェアリポジトリをユーザーに提供します。一方 **Ubuntu** は使いやすさと長期的な安定性をうまくバランスすることを目標としています。**Ubuntu** はほとんどのパッケージを **Debian** の不安定版ブランチから取っているので **Ubuntu** でも非常に大きなソフトウェアリポジトリへアクセスできます。このような理由を元に **Debian** 系ディストリビューションに関しての演習では **Ubuntu** を使用することにしました。

最近のディストリビューション間の類似点

- ディストリビューションのトレンドの変化により、ディストリビューション間の違いは減少している
  - **systemd**: システムの起動や、サービス管理
  - **journald**: システムログの管理
  - **firewalld**: ファイアウォール管理デーモン
  - **ip**: ネットワークディスプレイ、設定ツール

注目

これらの機能は多くのディストリビューションに共通しているため、講座や演習はこれらの機能に基づいています。利用中のディストリビューションあるいはリリースがこれらのコマンドをサポートしていない場合は適宜読み替えてください。

systemd は **SysVinit** や **Upstart** パッケージに代わり、最も一般的なディストリビューションで使用されています。**service** や **chkconfig** コマンドを置き換えています。**journald** はログのデータの収集と格納を行う **systemd** のサービスです。さまざまなプロセスやアプリケーションから受け取ったログ情報を元に構造化された索引付きのジャーナルを作成・管理します。ディストリビューションによりテキストベースのシステムログが置き換えられている可能性があります。

firewalld は、ネットワークあるいはファイアウォールゾーンがネットワーク接続やインターフェイスの信頼レベルを定義できるようにする、動的に管理できるファイアウォールを提供します。IPv4 や IPv6 のファイアウォール設定やイーサネットブリッジをサポートしています。これは **iptables** 設定を置き換えるものです

ip プログラムは、**net-tools** パッケージの一部であり、**ifconfig** コマンドを置き換えるものです。**ip** コマンドは、ルーティング・ネットワークデバイス・ルーティングの情報とトンネルの表示と操作をします。こちらの資料は古いコマンドを相当する **systemd** のコマンドに書き換える手助けとなるでしょう:
https://fedoraprojectorg/wiki/SysVinit_to_Systemd_Cheatsheet
https://wiki.debian.org/systemd/CheatSheet
https://en.opensuse.org/openSUSE:Cheat_sheet_13.1#Services

AWS 無料利用枠

- **Amazon Web Services** (**AWS**) はリモートユーザーがクラウド上でアクセスできる幅広い仮想マシン商品 (**インスタンス**) を提供
- **無料枠**のアカウントレベルが 1 年間利用可能
  Linux に慣れ **Linux Foundation** の講座・演習を行う上で必要な仮想ハードウェアとソフトウェアを利用可能

  ２つ以上の **Linux** ディストリビューションを学習する機会を得られる **AWS** の無料利用枠を試用する上で役に立つ説明:
  https://training.linuxfoundation.org/cm/prep/aws.pdf
- **AWS** はコンソールへアクセスできず GUI を提供していないので、これらの機能を必要とするタスクが実行できないかもしれないことに注意
  特にカーネルレベルのトレーニングは困難である

1.8 演習

課題 1.1: sudo 用にシステムを設定

root シェルの実行は非常に危険です: 1 つのミスタイプや間違いで重大な（致命的な）損害を与えてしまう可能性があります。絶対に必要なとき以外は控えるべきです。

そこで、**sudo** メカニズムを用いて 1 つずつのコマンドをスーパーユーザーの権限で実行できるように設定するのが賢明です。**sudo** を使えば、ユーザーは自分のパスワードのみを知っていればよいので、root パスワードを知る必要はありません。

この演習は、本講座用に **sudo** を設定するものです。そのため、**Ubuntu** のようなディストリビューションを利用しているなら、この演習は必要ないかもしれません。しかし、手順は知っておくとよいでしょう。

利用しているシステムが既にユーザーアカウントから **sudo** が実行できるように設定されているかどうかを確認するには、次の単純なコマンドを実行します:

```
$ sudo ls
```

ユーザーパスワードを求められ、コマンドが実行されるはずです。もし、代わりにエラーメッセージが表示されれば、次の手順を行ってください。

su と入力し、自分のユーザーパスワードではなく **root** パスワードを与えることにより、root シェルを起動します。

最近のすべての **Linux** ディストリビューションでは、`/etc/sudoers.d`サブディレクトリへ移動し、**sudo** アクセスを与えようとしているユーザーの名前でファイルを作成するべきです。しかし、この手順は実際は必要のない手順です。なぜなら、**sudo** がこのディレクトリ内のすべてのファイルを必要に応じてスキャンするからです。ファイルには次の一文が含まれていれば充分です:

```
student ALL=(ALL)  ALL
```

前述の例は、ユーザーが`student`の場合です。

（今でも利用できる）古いやり方は`/etc/sudoers`ファイルにそのような一行を追加する方法です。これには、文法エラーを防いでくれる **visudo** プログラムを利用することを推奨します。

また、次の文を入力することによってファイルに適切なパーミッションを設定する必要もあるでしょう:

```
$ chmod 440 /etc/sudoers.d/student
```

（ディストリビューションによっては、パーミッションを`440`の代わりに`400`と設定する必要があるかもしれません。）

これらの手順が終わったら、`exit`と入力し、root シェルからログアウトしてから、もう一度`sudo ls`を試しましょう。

特定のユーザーに必要なパーミッションを付与することや、検索パスの限定など、管理者が **sudo** を設定する方法は他にもあります。`/etc/sudoers`ファイルは初めて触る人でもファイル自身が良く書かれているので読むだけで内容がわかるものです。

しかし、システムに **sudo** が既に設定されている場合にも設定して頂きたい設定が 1 つあります。多くのディストリビューションでは、一般ユーザーと root ユーザーとでは実行ファイルを見つけるためのパスが違います。特に、**sudo** は root ユーザーではなく普通のユーザーのPATHを受け継ぐので、`/sbin`ディレクトリと`/usr/sbin`ディレクトリは検索対象に含まれません。

この場合だと、多くのツール・コマンドへのフルパスを常にこの講座でお伝えしなければいけないことになります。これほどの入力量の増加とディレクトリの位置を探し出す手間は、セキュリティの強化のためとはいえ、あまりに大きな手間です。従って、ホームディレクトリにある`.bashrc`ファイルに次の一行を追加することをおすすめします:

```
PATH=$PATH:/usr/sbin:/sbin
```

ログアウトしてもう一度ログインすれば、設定が有効になります（リブートの必要はありません）。

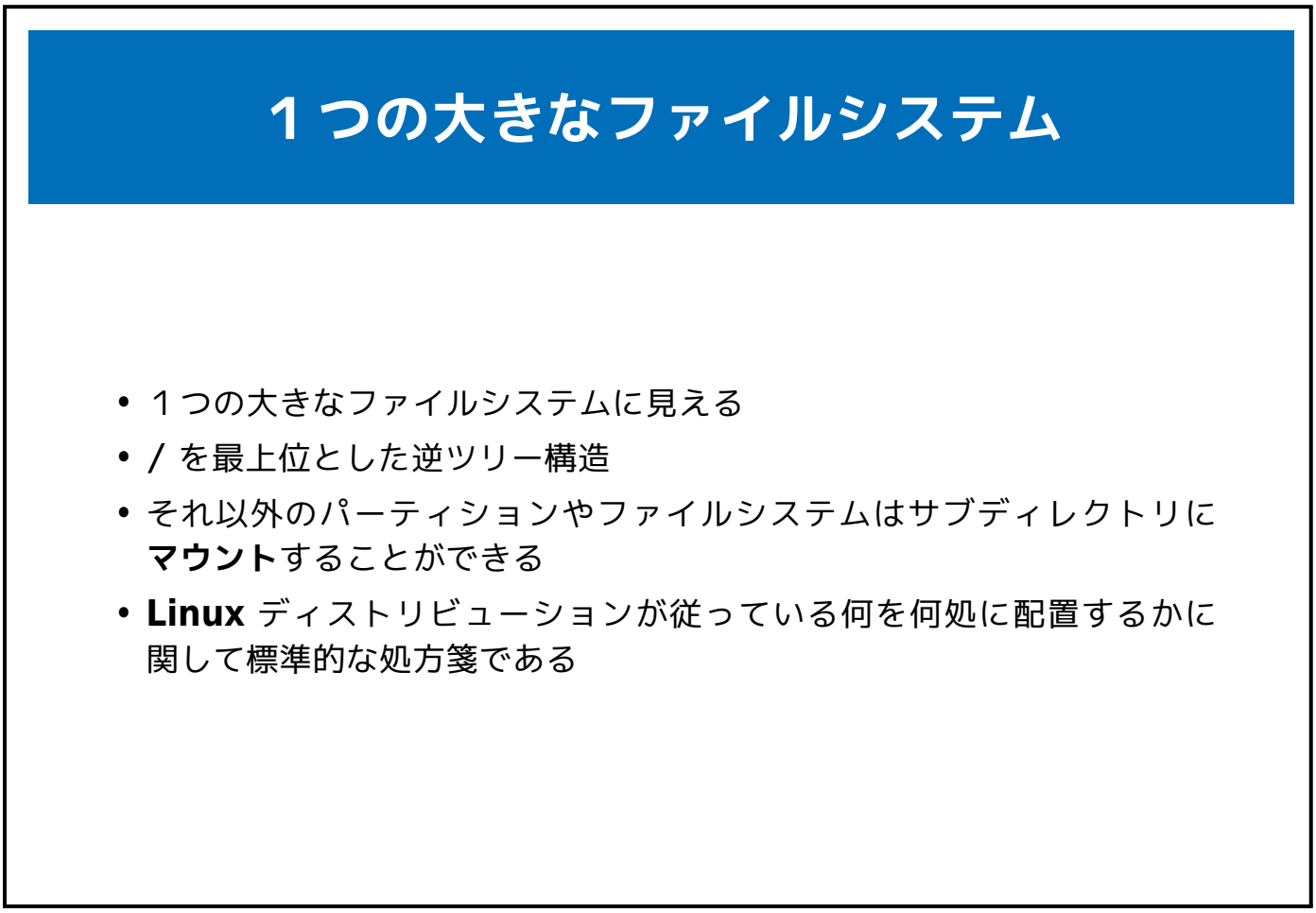
課題 1.2: トレーニングへの登録

- こちらで登録: https://training.linuxfoundation.org/surveys/registration
- ウェブフォームに入力するキーは講師が提供します。
- このキーは、講座の一番最後にするアンケートでも必要になるので、教材の一番最後の演習のページにメモしてください。
- トレーニングを完了したら、修了証書が送られます。

章 2

Linux ファイルシステム階層構造

2.1 １つの大きなファイルシステム



Linux はすべての UNIX ベースのオペレーティングシステムと同様に、１つの大きなファイルシステムツリー（階層）で構成されています。実際にはルートディレクトリ / がツリー（階層）の最上位にある逆ツリーの形で図示されることが一般的です。

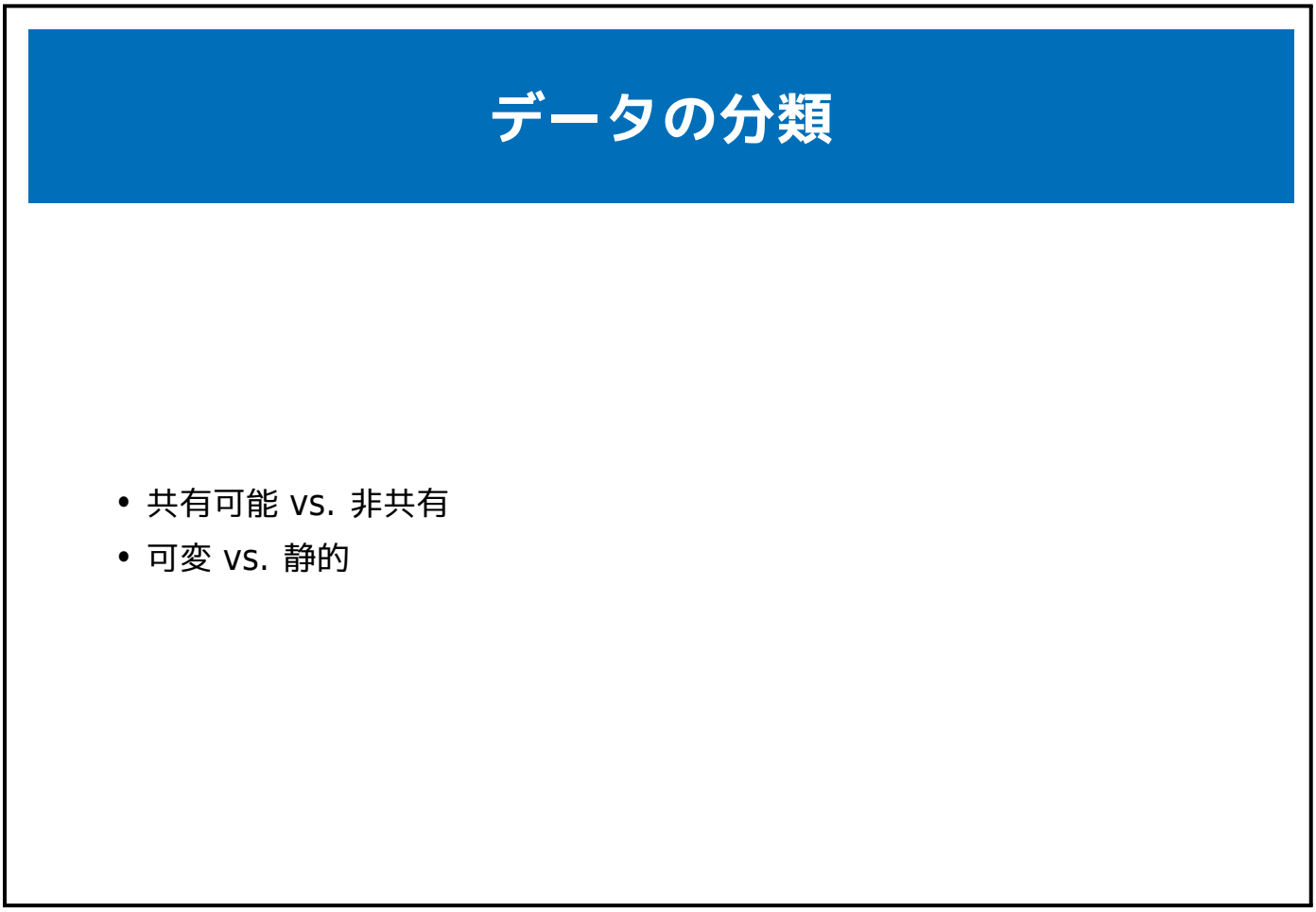
この１つの大きな論理ファイルシステム内には複数の異なるファイルシステムが複数のポイントで**マウント**され、サブディレクトリとして表示される場合があります。これらの個別のファイルシステムは通常は異なるパーティションにあり、ネットワーク上のデバイスを含む任意の数のデバイスに存在します。

どのように結合されているかに関わらず、すべてが１つの大きなファイルシステムのように見えます。アプリケーションは実際にどの物理デバイスファイルが存在するかについてまったく気にしません。

昔は、さまざまな **UNIX** 系のオペレーティングシステムがこの１つの大きなツリー（階層）をさまざまな方法で編成しました。**Linux** ディストリビューションの間でも多くの違いがありました。このためアプリケーションの作成と複数の種類のシステムでのシステム管理タスクの実行の両方が困難になり、イライラすることが多くありました。

結果として **Linux** エコシステムはそのような苦労を最小限に抑えるための標準化手順の確立に努めてきました。

2.2 データの分類



1 つの大きなディレクトリツリーでファイルとデータをどのように整理するかにおいては、どの種類の情報を読み書きする必要があるかという分類が重要になります。大きくは 2 種類あります。

1. **共有可能 vs. 非共有**
   共有可能なデータとは異なるホスト間で共有できるデータです。共有できないデータとは特定のホストに固有なデータです。たとえばユーザーのホームディレクトリは共有可能ですが、デバイスロックファイルは共有できません。
2. **可変 vs. 静的**
   静的データにはバイナリ、ライブラリ、ドキュメント、そしてシステム管理者のみが変更できるデータが含まれます。
   可変データはシステム管理者以外でも変更できるデータです。

これらの論理的な違いは、いろいろなディレクトリ、パーティション、ファイルシステムに存在するさまざまな種類の情報で表わされます。

2.3 FHS Linux 標準ディレクトリツリー

標準ファイルシステム階層構成

- ファイルタイプや場所の標準レイアウト
  - ファイルの場所を予測しやすくする
- BSD、その他の歴史的な標準配置がベース
- **Linux Foundation**: によって管理されている https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf.

Filesystem Hierarchy Standard（FHS）は当初 **Free Standards Group** が管理していましたが、現在は The **Linux Foundation** が管理しています。FHS は主なディレクトリとその内容を定義しています。Filesystem Hierarchy Standard のドキュメントは以下の URL から取得できます。https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf

FHS は標準レイアウトを定義することによりファイルの場所を予測し易くしています。ほとんどの **Linux** ディストリビューションは **FHS** を尊重しますが正確には従っていません。最後の公式バージョンはとても古くいくつかの新しい開発を考慮していないためです。

ディストリビューションは実験を好み、最終的には実験の一部を受け入れています。

FHS Linux ディレクトリツリー

Table 2.1: **主なディレクトリ　パート 1**

ディレクトリ	目　的
/	ファイルシステムの階層全体の最上位ディレクトリ
/bin	**シングルユーザーモード**で使用できる必要がある必須の実行可能プログラム
/boot	カーネル、**initrd**、**initramfs** イメージなどのシステムのブートに必要なファイル、ブート構成ファイルおよびブートローダープログラム
/dev	ハードウェアおよびソフトウェアデバイスと対話するために使用される**デバイスノード**
/etc	システム全体の構成ファイル
/home	個人の設定、ファイルなどを含むユーザーの**ホームディレクトリ**
/lib	/bin と /sbin の実行可能バイナリに必要なライブラリ
/lib64	32 ビットプログラムと 64 ビットプログラムの両方を実行できるシステムの場合の /binと /sbin の実行可能バイナリに必要な 64 ビットライブラリ。
/media	**CD**、**DVD**、**USB** スティックなどのリムーバブルメディアのマウントポイント
/mnt	一時的にマウントされたファイルシステム

ルートディレクトリの下に、ディストリビューション固有のディレクトリが追加されている場合があります。これらにはその他のデータに使用できる /misc や、**tftp** を使用した起動に使用される /tftpboot が含まれる場合があります。ディレクトリにファイルがある場合、それらはディスクレスワークステーションの起動に関連しています。他のディレクトリを持つことは **FHS** には違反しません。ただし標準で規定されているディレクトリ以外にコンポーネントを配置することには違反します。

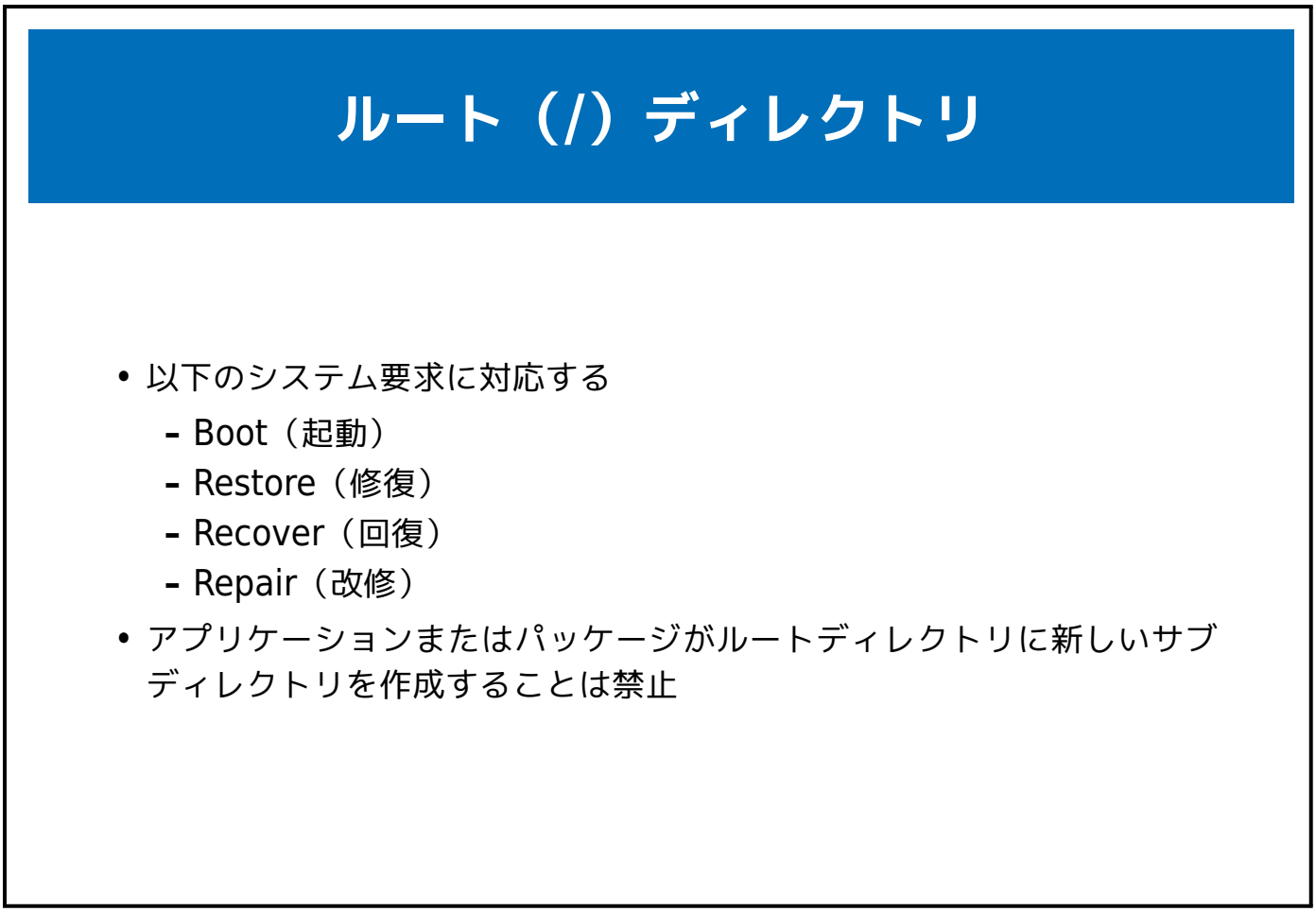
FHS Linux ディレクトリツリー（つづき）

Table 2.2: **主なディレクトリ　パート 2**

ディレクトリ	目　的
/opt	オプションのアプリケーションソフトウェアパッケージ
/proc	システムおよびその上で実行されているプロセスに関する情報を提供する 仮想擬似ファイルシステム、システムパラメータの変更に使用可能
/run	ランタイム変数データ、システム起動以降の記録が保存される 以前の /var/run の置き換え
/sys	システムおよびそこで実行されているプロセスに関する情報を提供する仮想擬似 ファイルシステム、システムパラメータの変更に使用可能**デバイスツリー**に似て おり**統合デバイスモデル**の一部 **統合デバイスモデル**の一部
/root	root ユーザーのホームディレクトリ
/sbin	必須のシステムバイナリ
/srv	システムが提供するサイト固有のデータ、めったに使用されない
/tmp	一時ファイル、多くのディストリビューションでは再起動すると失われる メモリー内の RAM ディスクである可能性もある
/usr	マルチユーザーアプリケーション、ユーティリティおよびデータ、読み取り専用
/var	システム操作中に変化する可変データ

これらは、メインの **root** (/) ディレクトリとは別の追加ディレクトリである

2.4 ルート（/）ディレクトリ



先ほど述べたように、ファイルシステム全体を 1 つの大きなツリー（階層）と見なすことができますが、複数のパーティションとファイルシステムが結合されている場合があります。

ルートディレクトリ自体が含まれるパーティションとファイルシステムはかなり特別です。多くの場合、後でマウントされる `/home`、`/var`、`/opt` などのディレクトリを持つ他のコンポーネントとともに特別な専用パーティションにあります。

ルートパーティションには、システムの起動後に他のすべてのファイルシステムをマウントするために必要な、すべての重要なファイルが含まれている必要があります。したがって、ユーティリティ、構成ファイル、ブートローダー情報、およびその他の重要なスタートアップデータが含まれている必要があります。それは次のことを適切に行うためです。

- システムを起動します。
- テープやその他のリムーバブルメディア、**NAS** などの外部メディアのシステムバックアップからシステムを復元します。
- システムの回復か修復、またはその両方を行います。経験豊富なメンテナは損傷したシステムを診断および再構築するツールを持っている必要があります。

FHS はアプリケーションまたはパッケージがルートディレクトリに新しいサブディレクトリを作成することを禁止しています。

/ は /root とは違います

`/root` はスーパーユーザーのホームディレクトリです。名前が紛らわしいので注意してください！

2.5 /bin

/bin

- システム管理者と非特権ユーザーの両方が必要とする実行可能プログラムとスクリプトが含まれる。これらは他のファイルシステムがまだマウントされていない場合、たとえば**シングルユーザーモード**またはリカバリモードで起動する場合に必要
- スクリプトによって間接的に使用される、実行可能ファイルが含まれる場合もある
- サブディレクトリを含めることはできない

/bin と /usr/bin

最新の一部のディストリビューションでは、/binと /usr/bin（および/sbinと/usr/sbin）を区別せず、シンボリックリンクを使って 1 つのディレクトリで 2 つのディレクトリビューを持ちます。そのディストリビューションでは、ブート後にマウントされる別のパーティションに/usrを配置するのは、時代遅れとみなしています。

/bin/に存在する必要があるプログラムには、次のものがあります。

cat, chgrp, chmod, chown, cp, date, dd, df, dmesg, echo, false, hostname, kill, ln, login, ls, mkdir, mknod, more, mount, mv, ps, pwd, rm, rmdir, sed, sh, stty, su, sync, true, umount and **uname**.

[と **test** もここに置かれることがあります。オプションで以下のものも含まれることがあります

csh, ed, tar, cpio, gunzip, zcat, netstat と **ping**.

/binの場所に値するほど重要ではないとみなされるコマンドバイナリが /usr/bin に置かれます。
非 root ユーザーだけが必要とするプログラムがここに置かれます。

/bin の例

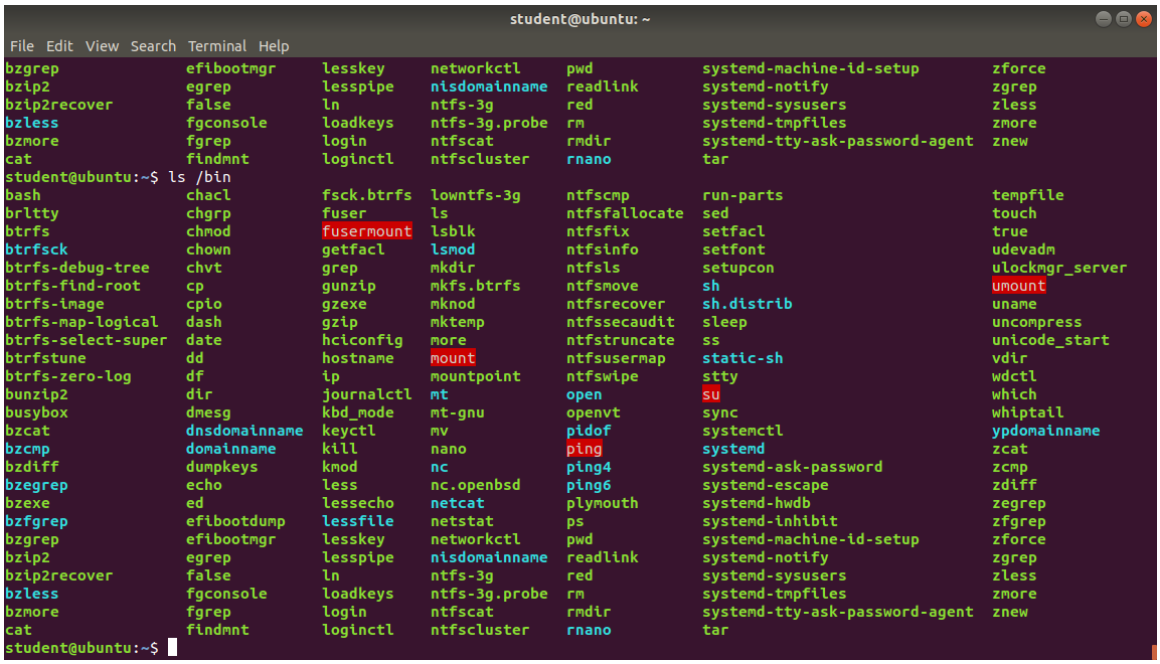


図 2.1: **/bin ディレクトリ: Ubuntu 18.04**

RHEL、**CentOS**、**Fedora** や **Ubuntu** などの最近のディストリビューションでは、/bin と/usr/bin は実際には同じディレクトリです。

このスクリーンショットは **Ubuntu 18.04** のものです。最新バージョンではリンクが切れた独立ディレクトリーは無くなりました。

2.6 /boot

/boot

- システムの起動に必要な全てのファイルが含まれる
  以下の 2 つのファイルは特に重要
  - `vmlinuz`: 圧縮された **Linux** カーネル
  - `initramfs`: 実際のルートファイルシステムが使用可能になる前にマウントされる**初期 RAM ファイルシステム**
- カーネルがユーザーモードプログラムの実行開始前に使用するデータを格納
- また情報とデバッグに使用される 2 つのファイルも含まれる
  - `config` カーネルのコンパイルを構成するために使用
  - `System.map`: デバッグ用のカーネル **シンボルテーブル**
- ディストリビューションによって他のファイルが置かれることもある

`/boot`の正確な内容はディストリビューションと時代によって異なります。以下に **Ubuntu** システムの例を示します。

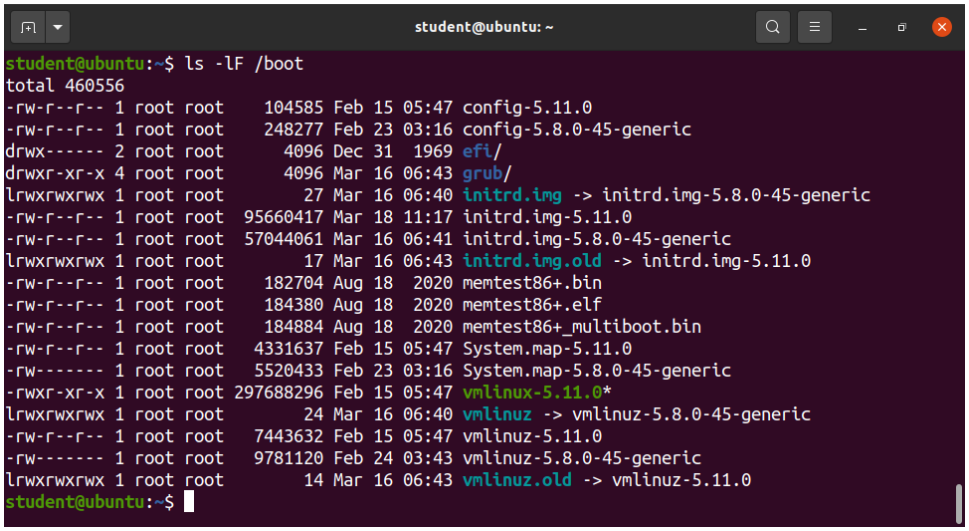


図 2.2: **/boot Directory on Ubuntu 20.04**

これらのファイルにはカーネルのバージョンに依存する長い名前があり、正確な形式は **Linux** ディストリビューションによって異なります。さらに `initramfs` の代わりに `initrd` (**init**ial **r**am **d**isk) と呼ばれる場合があります。

2.7 /dev

/dev

- キャラクター型／ブロック型の**デバイスノード**又はファイル
- 適切なシステム運用の要
- 現代のシステムは **udev** でデバイスファイルの管理を自動化
- 古いシステム（または組込み機器）では必要に応じインストール時又は別の時間に **MAKEDEV** や **mknod** を使ってデバイスを生成する

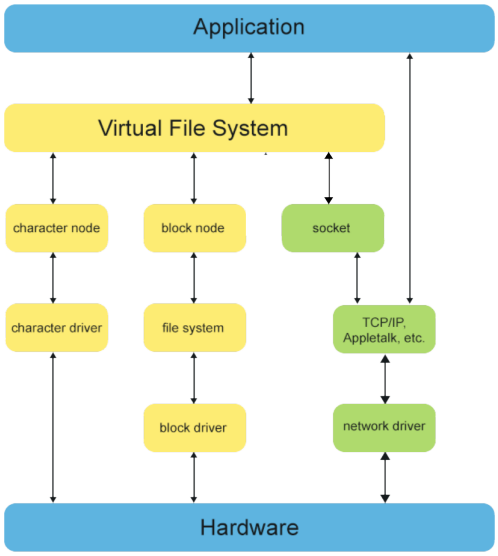


図 2.3: **デバイスノード**

このディレクトリにはシステムに組み込まれた、またはシステムに接続されたデバイスを表す**スペシャルデバイスファイル**（**デバイスノード**とも呼ばれます）が含まれています。これらの特殊ファイルはシステムが正しく機能するために不可欠です。このようなデバイスファイルには **キャラクタデバイス**（バイトストリーム）と**ブロック I/O デバイス**があります。**Linux** の**ネットワーク**デバイスにはデバイスノードがなく、代わりに `eth1` や `wlan0` などの名前で参照されます。

```
File  Edit  View  Search  Terminal  Help
c7:/tmp>ls -l /dev
total 0
....
crw-------   1 root root        5,   1 May 26 07:05 console
....
lrwxrwxrwx   1 root root             13 May 26 07:04 fd -> /proc/self/fd
....
brw-rw----   1 root disk        7,   0 May 26 07:05 loop0
crw-rw----   1 root disk       10, 237 May 26 07:05 loop-control
.....
crw-rw----   1 root lp          6,   0 May 26 07:05 lp0
crw-rw----   1 root lp          6,   1 May 26 07:05 lp1
....
brw-rw----   1 root disk        8,   0 May 26 07:05 sda
brw-rw----   1 root disk        8,   1 May 26 07:05 sda1
brw-rw----   1 root disk        8,   2 May 26 07:05 sda2
brw-rw----   1 root disk        8,   3 May 26 07:05 sda3
....
lrwxrwxrwx   1 root root             15 May 26 07:04 stderr -> /proc/self/fd/2
lrwxrwxrwx   1 root root             15 May 26 07:04 stdin -> /proc/self/fd/0
lrwxrwxrwx   1 root root             15 May 26 07:04 stdout -> /proc/self/fd/1
crw-rw-rw-   1 root tty         5,   0 May 26 07:05 tty
crw--w----   1 root tty         4,   0 May 26 07:05 tty0
....
c7:/tmp>
c7:/tmp>
```

図 2.4: **/dev ディレクトリ**

最新の **Linux** ディストリビューションはすべて **udev** システムを使用し、デバイスが見つかったときのみ必要に応じて `/dev` にノードを作成します。マウントされていないファイルシステムの `/dev` ディレクトリを見ると空であることがわかります。

2.8 /etc

このディレクトリには、マシンローカルの構成ファイルといくつかの起動スクリプトが含まれています。実行可能なバイナリプログラムはありません。**FHS** がこのディレクトリに配置することを求めているファイルとディレクトリは次のとおりです。

```
csh.login, exports, fstab, ftpusers, gateways, gettydefs, group, host.conf,
hosts.allow, hosts.deny, hosts,equiv, hosts.lpd, inetd.conf, inittab, issue,
ld.so.conf, motd, mtab, mtools.conf, networks, passwd, printcap, profile,
protocols, resolv.conf, rpc, securetty, services, shells, syslog.conf.
```

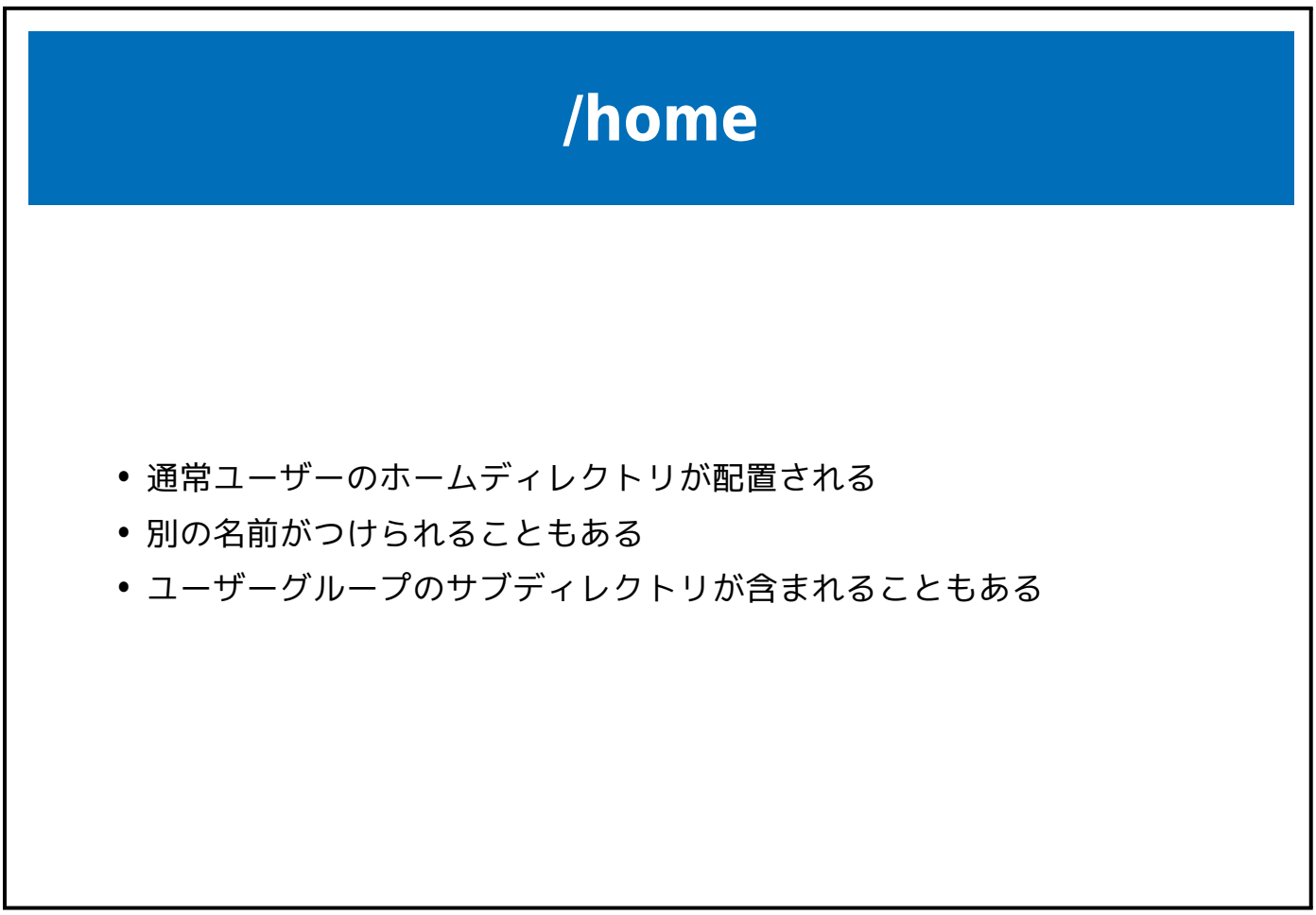
フロッピーディスクで使用される `mtools.conf` など、この中の一部のファイルは現在重要ではなくなっています。**FHS** が定義していてもソフトウェアが陳腐化したためもう見つからないものもあります。

ディストリビューションは、`/etc` に構成ファイルとディレクトリを追加することがよくあります。たとえば **Red Hat** は `/etc/sysconfig` を含むいくつかのディレクトリを追加しています。そのディレクトリにはいくつかのシステム構成ファイルとディレクトリが存在します。

他の重要なサブディレクトリにとしては

- `/etc/skel`: **スケルトン**ファイルが置かれています。このファイルは新しく作成したホームディレクトリの追加に使用されます。
- `/etc/systemd`: **systemd** を使用する時にシステムサービスの開始と停止をするための、構成スクリプトが置かれているかそれをポイントしています。
- `/etc/init.d`: **System V** の初期化を行う場合の起動スクリプトとシャットダウンスクリプトが置かれています。

2.9 /home



Linux システムでは、通常、ユーザーディレクトリは /home/coop、/home/student などのように /home の下に置かれます。すべての個人設定、データ、および実行可能プログラムはこのディレクトリの階層に置かれます。/home には /home/students、/home/staff、/home/aliens など、ユーザーのさまざまなグループや関連付けされたサブディレクトリが置かれる場合もあります。

他の **UNIX** 系のオペレーティングシステムでも /home ディレクトリツリーの概念は存在しますが微妙に異なる場合があります。たとえば、**Solaris** ではユーザーディレクトリは/export/home に作成され、最終的には**自動マウント**機能によって /home にマウントされます。これは一般的にはホームディレクトリが企業ネットワーク上のどこか、おそらく **NFS** サーバー上にあり、使用時にホームディレクトリが自動的にマウントされるためです。

Linux にも同じ自動マウント機能がありますが、多くのユーザーはそれらを意識していません。また自己完結型システムではおそらく **NFS** マウントは使われないでしょう。

特定のユーザーは環境変数 $HOME をルートディレクトリの代わりに使用することも、略記〜; を使用することもできます。
従って以下は同じ意味です。

```
$ ls -l $HOME/public_html
$ ls -l ~/public_html
```

例外が１つあります。**Linux** システムの **root** ユーザーのホームディレクトリは、常に/root の下にあります。
古い **UNIX** システムの中には / を代わりに使用するものがあり混乱するかもしれません。

/home の例

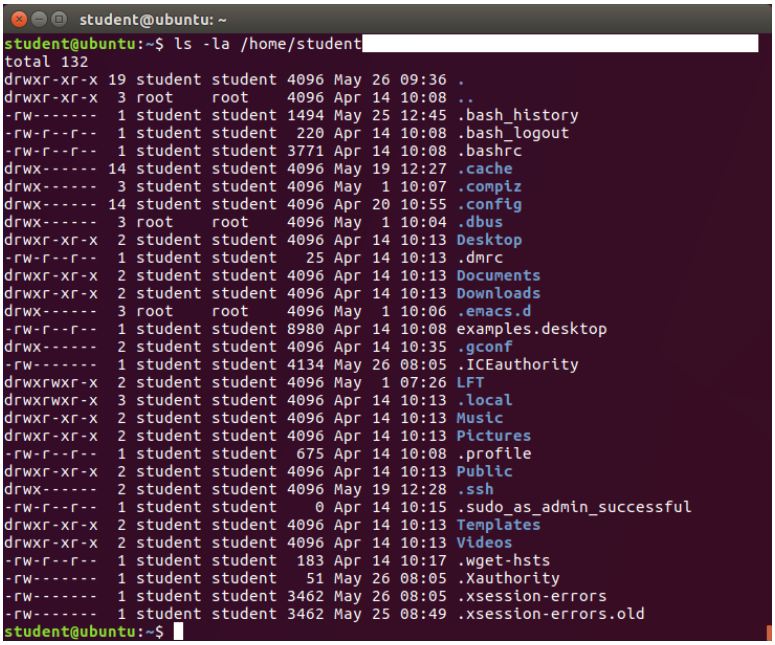


図 2.5: **/home Directory**

2.10 /lib と /lib64

/lib と /lib64

- /bin and /sbin にあるバイナリーを実行するのに必要がライブラリーだけが配置される
- カーネルモジュールは /lib/modules に置かれる
- PAM モジュールは /lib/security に置かれる
- いくつかのディストリビューションは 64bit ライブラリーを /lib64 に置く

/lib と /usr/lib

最近のディストリビューションの中には、/libと /usr/lib（および/lib64と/usr/lib64）を区別せずにディレクトリを１つだけ用意してシンボリックリンクを使うことで、あたかも２つのディレクトリが存在するかのように見せるものがあります。そのディストリビューションではブート後にマウントされる別のパーティションに/usr を置くのは時代遅れと考えています。

これらのディレクトリには、/bin と /sbin 内のバイナリを実行するために必要なライブラリのみが置かれています。これらのライブラリはシステムを起動しルートファイルシステム内でコマンドを実行する上で特に重要なものです。

カーネルモジュール（多くの場合デバイスまたはファイルシステムのドライバ）は /lib/modules/<kernel-version-number> にあります。

PAM (**P**luggable **A**uthentication **M**odules) ファイルは /lib/security にあります。

32 ビットと 64 ビット両方のバイナリをサポートするシステムではシステムに両方のライブラリを保持する必要があります。Red Hat ベースのシステムでは 32 ビットライブラリ (/lib) と 64 ビットライブラリ (/lib64) のディレクトリがあります。

```
$ ls -l /lib*
```

```
rwxrwxrwx 1 root root 8 Apr 23  2020 /sbin -> usr/sbin
lrwxrwxrwx 1 root root 7 Apr 23  2020 /lib -> usr/lib
lrwxrwxrwx 1 root root 9 Apr 23  2020 /lib64 -> usr/lib64
```

2.11 /media

/media

- リムーバブルメディアのマウント用
  - CD ROMS
  - DVD
  - USB drives
- 動的にマウントされることもある

/media と /run

最近のシステムでは /media ではなく /run/media/[username]/... を使う

このディレクトリはファイルシステムをリムーバブルメディアにマウントするために使用されていました。これらには **CD**s、**DVD**s、**USB**、さらには旧石器時代のフロッピーも含まれます。

Linux システムはそのようなメディアを挿入時に動的にマウントし、構成ファイルにある **udev** ルールで設定した名前で **udev** がディレクトリを作成して、そこにリムーバブルファイルシステムをマウントします。アンマウントもしくは削除をするとマウントポイントとして使用されていたディレクトリが消えます。

メディアに複数のパーティションとファイルシステムがある場合、複数のエントリが表示されます。多くの **Linux** ディストリビューションではメディアがマウントされるとファイルマネージャ (**Nautilus** など) が表示されます。

/run は /media に置き換わっています

Note: 現在の **Linux** ディストリビューションでは、リムーバブルメディアは /media ではなく /run/media/[username]/... の下に表示されます。/run については後で説明します。

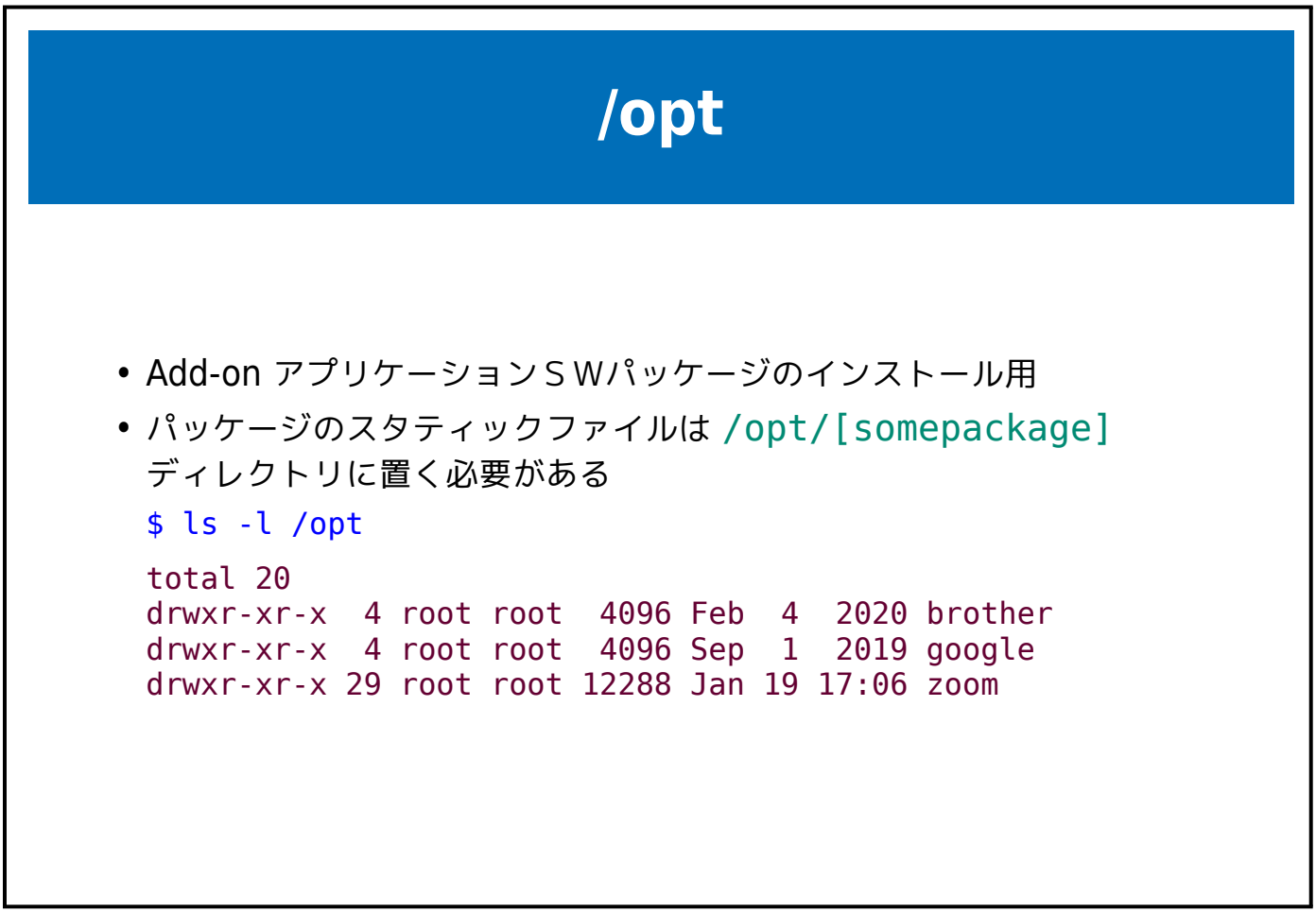
2.12 /mnt

このディレクトリはシステム管理者が必要に応じて一時的にファイルシステムをマウントできるようにするために提供されています。一般的な用途は次のような **ネットワークファイルシステム** です。

- **NFS**
- **Samba**
- **CIFS**
- **AFS**

歴史的に `/mnt` は現在のシステムの `media`（または `/run/media`）にマウントされるファイルにも使用されていました。
ここにはプログラムをインストールしないでください。現在使用していない別の一時ディレクトリが適切です。

2.13 /opt



このディレクトリは他のソフトウェアと共有するディレクトリを使ってシステム全体に分散させるのではなく、ファイルのすべてまたはほとんどを 1 つの専有場所に保持したいソフトウェアパッケージ用に設計されています。

たとえば **dolphy_app** が /opt に存在するパッケージの名前である場合、そのすべてのファイルは /opt/dolphy_app の下のディレクトリに存在する必要があります。バイナリの場合は /opt/dolphy_app/bin、**man** ページの場合は /opt/dolphy_app/man などです。

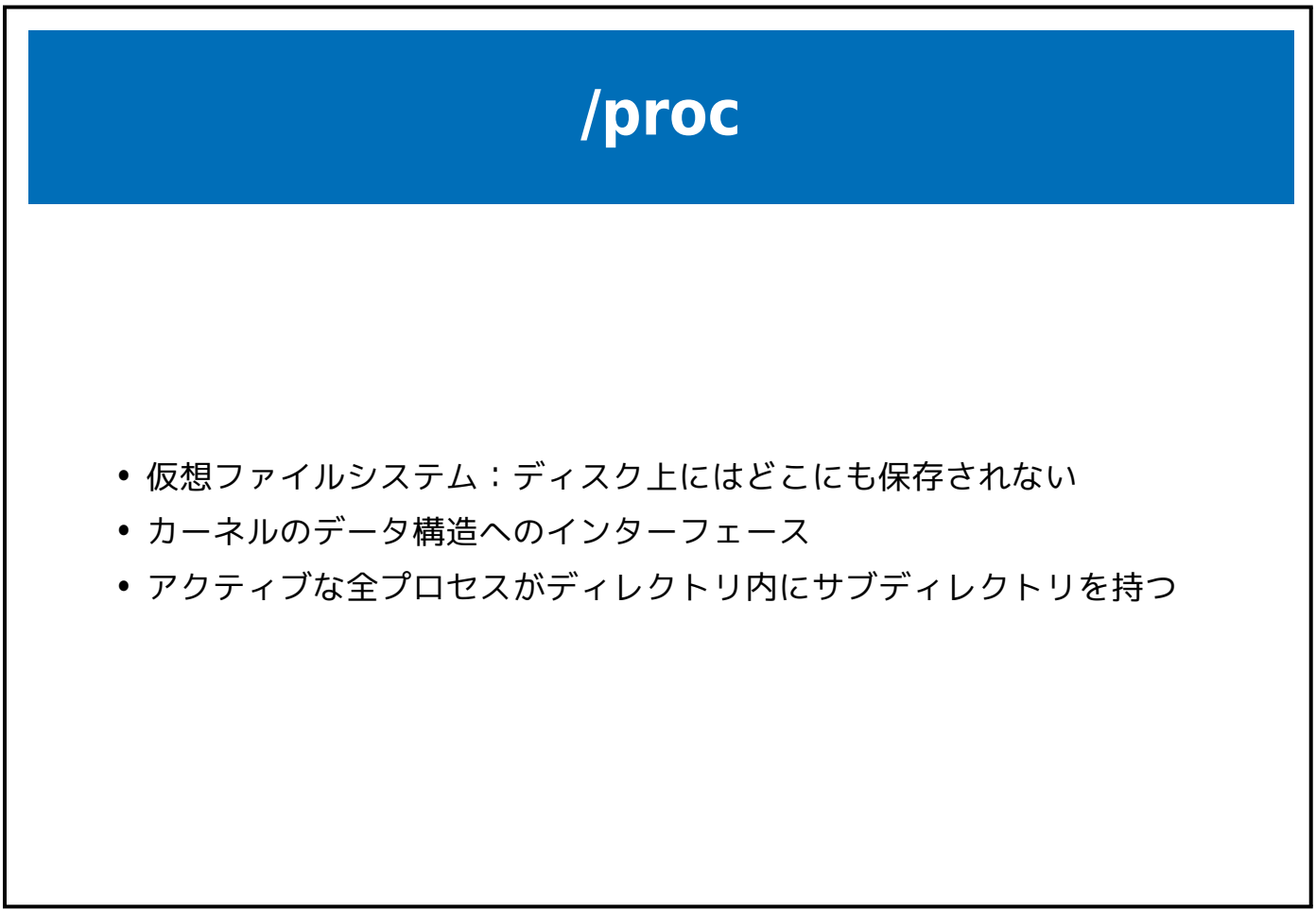
このようにファイルをまとめることで、ソフトウェアのインストールとアンインストールの両方を比較的簡単に行うことができます。すべてのファイルが、予測可能かつ構造化された１つの分離された場所にあるからです。またシステム管理者がパッケージ内の各ファイルの性質を簡単に判断できるようになります。

ただし **RPM** や **APT** などのパッケージシステムを使用している場合は、このようなことを意識しなくてもファイルの管理情報と配置が明確なので簡単にインストールおよびアンインストールすることができます。

Linux では多くの場合、/opt ディレクトリはプロプライエタリなソフトウェアを使用するアプリケーションプロバイダ、またはディストリビューションの違いによる複雑さを回避したいアプリケーションプロバイダによって使用されます。たとえば、あるシステムではパッケージは /opt/brother、/opt/zoom や /opt/google に置かれ、後者は chrome と earth のためのサブディレクトリを持っている

またローカルシステム管理者が使用するためのディレクトリとして /opt/bin、/opt/doc、/opt/include、/opt/info、/opt/lib、/opt/man が用意されています。パッケージは、これら予約されたディレクトリにリンクまたはコピーされたファイルを提供する場合がありますが、一方で、これらの特別なディレクトリにプログラムが存在しなくても機能する必要があります。

2.14 /proc



このディレクトリは、すべての情報がディスクではなくメモリーにのみ存在する**疑似ファイルシステム**のマウントポイントです。/dev と同様に、実行されていないシステムでは /proc ディレクトリは空です。

カーネルは /proc エントリを介していくつかの重要なデータ構造を提示します。さらに、システム上のアクティブな各プロセスには、プロセスの状態、使用しているリソース、およびその履歴に関する詳細情報を提供する独自のサブディレクトリがあります。

/proc 下に存在するエントリは通常**仮想ファイル**と呼ばれ興味深い性質を持っています。ほとんどがサイズが 0 バイトとして表示されますが、表示すると大量の情報を含んでいることがわかります。

さらに仮想ファイルのほとんどの日時設定は現在の日時を反映しており常に変化していることを示しています。実際これらのファイルの情報は表示するときにのみ取得され、常時もしくは定期的に更新されるわけではありません。

/proc/interrupts、/proc/meminfo、/proc/mounts、/proc/partitions を含む重要な疑似ファイルはシステムのハードウェアの最新情報を提供します。

/proc/filesystems や /proc/sys/などはシステム構成情報とインターフェイスを提供します。

組織化する目的で類似のトピックに関する情報を含むファイルは、仮想ディレクトリとサブディレクトリにグループ化されています。たとえば /proc/scsi/ にはすべての物理 **SCSI** デバイスの情報が含まれます。同様に **プロセス**ディレクトリにはシステムで実行中の各プロセスに関する情報が含まれています。

このコースでは /proc のエントリを広範囲に見て行きます。カーネルの構成とシステムの監視に関する内容は、今後の章でさらに詳しく見ていきます。

/proc の実例

```
student@ubuntu:~$ ls -F /proc
1/     13/    200/   2288/  250/   33/    40/    5383/  99/          misc
10/    137/   201/   229/   251/   3315/  4047/  54/    990/         modules
100/   14/    202/   23/    2529/  3336/  41/    5410/  acpi/        mounts@
1005/  1457/  203/   230/   2572/  3353/  4156/  5436/  buddyinfo    mpt/
1007/  1465/  204/   231/   26/    3366/  4169/  5438/  bus/         mtrr
1008/  1478/  205/   232/   27/    3394/  418/   581/   cgroups      net@
1009/  15/    206/   233/   274/   34/    419/   583/   cmdline      pagetypeinfo
101/   16/    207/   234/   2756/  3419/  42/    6/     consoles     partitions
1010/  1649/  208/   235/   276/   3430/  420/   601/   cpuinfo      sched_debug
1011/  1780/  209/   236/   28/    3439/  422/   7/     crypto       schedstat
1012/  1782/  21/    2368/  280/   3441/  43/    715/   devices      scsi/
1013/  1798/  210/   237/   281/   35/    44/    717/   diskstats    self@
1014/  18/    211/   2373/  2979/  3512/  45/    718/   dma          slabinfo
102/   1822/  212/   238/   298/   3513/  4595/  721/   driver/      softirqs
1028/  188/   213/   2385/  2989/  3514/  4596/  723/   execdomains  stat
103/   189/   214/   239/   30/    3515/  4599/  728/   fb           swaps
104/   19/    2142/  24/    300/   3532/  46/    731/   filesystems  sys/
105/   190/   215/   240/   3081/  3534/  4601/  733/   fs/          sysrq-trigger
106/   191/   216/   241/   31/    3569/  47/    735/   interrupts   sysvipc/
107/   1916/  217/   242/   32/    3581/  473/   743/   iomem        thread-self@
108/   192/   218/   243/   3228/  3589/  478/   746/   ioports      timer_list
1083/  1922/  219/   2436/  3243/  36/    4792/  750/   irq/         timer_stats
1085/  193/   2196/  244/   3244/  3600/  483/   753/   kallsyms     tty/
109/   1930/  2197/  2443/  3245/  3613/  485/   755/   kcore        uptime
11/    194/   22/    2449/  3246/  3621/  486/   781/   keys         version
110/   1940/  220/   245/   3247/  3655/  492/   798/   key-users    version_signature
1121/  195/   221/   2456/  3251/  37/    50/    8/     kmsg         vmallocinfo
1134/  196/   222/   246/   3255/  370/   502/   828/   kpagecgroup  vmstat
1140/  197/   223/   2461/  3257/  38/    51/    9/     kpagecount   zoneinfo
1141/  198/   224/   247/   3264/  39/    52/    96/    kpageflags
1147/  1986/  225/   248/   3268/  3910/  53/    98/    loadavg
116/   199/   226/   2488/  3272/  393/   5310/  981/   locks
1180/  2/     227/   249/   3274/  399/   5332/  983/   mdstat
12/    20/    228/   25/    3294/  4/     5358/  988/   meminfo
student@ubuntu:~$
```

図 2.6: **/proc ディレクトリ**

以下のスクリーンショットでは、特定のプロセスに属する/proc ディレクトリの内容を確認できます。

```
student@ubuntu:~$ ls -F /proc/3589
attr/       coredump_filter  gid_map     mountinfo   oom_score      schedstat  status
autogroup   cpuset           io          mounts      oom_score_adj  sessionid  syscall
auxv        cwd@             limits      mountstats  pagemap        setgroups  task/
cgroup      environ          loginuid    net/        personality    smaps      timers
clear_refs  exe@             map_files/  ns/         projid_map     stack      timerslack_ns
cmdline     fd/              maps        numa_maps   root@          stat       uid_map
comm        fdinfo/          mem         oom_adj     sched          statm      wchan
student@ubuntu:~$
```

図 2.7: **/proc/[pid] ディレクトリ**

以下のスクリーンショットでは、重要なシステム情報を示す多くのファイルの 1 つである/proc/interrupts を確認できます。

```
File  Edit  View  Search  Terminal  Help
x7:/home/coop>cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  0:         88          0          0          0  IR-IO-APIC    2-edge      timer
  1:        566          1       2153          0  IR-IO-APIC    1-edge      i8042
  8:          1          0          0          0  IR-IO-APIC    8-edge      rtc0
  9:      17990         11        552         27  IR-IO-APIC    9-fasteoi   acpi
 12:      64336         21     186157         20  IR-IO-APIC   12-edge      i8042
 16:          0          0          0          0  IR-IO-APIC   16-fasteoi   i801_smbus
120:          0          0          0          0  DMAR-MSI    0-edge      dmar0
121:          0          0          0          0  DMAR-MSI    1-edge      dmar1
122:     217295       1607       4039      59571  IR-PCI-MSI 376832-edge      ahci[0000:00:17.0]
123:          3          0         46          0  IR-PCI-MSI 514048-edge      snd_hda_intel:card0
124:      95360         74      49535        192  IR-PCI-MSI 327680-edge      xhci_hcd
125:     591286          3     272983          2  IR-PCI-MSI 32768-edge      i915
126:         84         16        149         20  IR-PCI-MSI 520192-edge      enp0s31f6
127:     225390         29        383         46  IR-PCI-MSI 2097152-edge      iwlwifi
NMI:         24        120        130        119   Non-maskable interrupts
LOC:    2255506    2201465    2360863    2239138   Local timer interrupts
SPU:          0          0          0          0   Spurious interrupts
PMI:         24        120        130        119   Performance monitoring interrupts
IWI:          0          0          3          0   IRQ work interrupts
RTR:         24          3          0          0   APIC ICR read retries
RES:     185530     146421      95420      45924   Rescheduling interrupts
CAL:      76456      74989      78143      76063   Function call interrupts
TLB:      75401      73603      76833      75025   TLB shootdowns
ERR:          0
MIS:          0
PIN:          0          0          0          0   Posted-interrupt notification event
PIW:          0          0          0          0   Posted-interrupt wakeup event
x7:/home/coop>
```

図 2.8: **/proc/interrupts コマンドと出力のスクリーンショット**

2.15 /sys

/sys

- 別の仮想ファイルシステム：ディスク上にはどこにも保存されない
- デバイスとドライバー、カーネルモジュール、システム構成情報に関する情報が含まれる
- システムの挙動変更やパラメータ変更に利用される
- /proc よりうまくコントロールされている

このディレクトリは、すべての情報がディスクではなくメモリーにのみ存在する **sysfs 疑似ファイルシステム**のマウントポイントです。/dev や /proc と同様に、/sys ディレクトリは実行されていないシステムでは空です。デバイス、ドライバ、カーネルモジュール、システム構成構造などに関する情報が含まれています。

sysfs はシステムに関する情報を収集し、実行中にその動作を変更するために使用されます。その意味では /proc に似ていますが、それよりも歴史は新しくどのような種類のエントリを含めることができるかの厳しい基準に準拠しています。たとえば /sys 内のほとんどすべての疑似ファイルには、1 行または値しか含まれていません。/proc にあるような長いエントリはありません。

/proc の場合と同様に、このコースでは /sys のエントリを見て行きます。カーネルの構成とシステム監視に関してはこの後の章で見て行きます。

```
File  Edit  View  Search  Terminal  Help
c7:/tmp>ls -lF /sys
total 0
drwxr-xr-x  2 root root 0 May 26 12:08 block/
drwxr-xr-x 23 root root 0 May 26 12:08 bus/
drwxr-xr-x 41 root root 0 May 26 12:08 class/
drwxr-xr-x  4 root root 0 May 26 12:08 dev/
drwxr-xr-x 23 root root 0 May 26 12:08 devices/
drwxr-xr-x  5 root root 0 May 26 12:08 firmware/
drwxr-xr-x  5 root root 0 May 26 12:08 fs/
drwxr-xr-x 11 root root 0 May 26 12:08 kernel/
drwxr-xr-x 88 root root 0 May 26 12:08 module/
drwxr-xr-x  2 root root 0 May 26 12:08 power/
c7:/tmp>
```

図 2.9: **/sys ディレクトリ**

2.16 /root

/root

- ルートユーザーのホームディレクトリ user

```
File  Edit  View  Search  Terminal  Help
c7:/tmp>su
Password:
c7:/tmp>ls -saF /root
total 220
 4 ./                4 .dbus/              4 .lesshst        4 shit
 4 ../               4 Desktop/            4 .local/         4 .ssh/
 4 anaconda-ks.cfg   4 Documents/          4 lttng-traces/   4 .tcshrc
24 .bash_history     4 Downloads/          4 .macromedia/    4 Templates/
 4 .bash_logout      8 .emacs              4 Music/          4 Videos/
 4 .bash_profile     4 .emacs.d/           4 perl5/          4 .vmware/
 8 .bashrc           4 .esd_auth           4 Pictures/       4 .xauth2xyWfQ
 8 bashrc_ORIG       4 .galileo/           8 .pki/           4 .xautha52ibW
 4 .basrc_ORIG       4 .gitk               4 Public/         4 .Xauthority
 4 .cache/           8 .gnome2/            4 .rnd
 4 .config/          4 .ICEauthority       4 rpmbuild/
 4 .cshrc            4 initial-setup-ks.cfg 4 .sane/
c7:/tmp>
```

図 2.10: **/root ディレクトリ**

このディレクトリ（「スラッシュルート」と発音）は root ユーザーのホームディレクトリです。

このディレクトリを所有する **root** アカウントはスーパーユーザー権限が必要な作業の時にのみ使用してください。一般ユーザーが実行する作業には別のアカウントを使用します。

2.17 /sbin

/sbin

- システムの中核部分と保守ユーティリティ用
- このディレクトリー内のプログラムは特権ユーザーだけが利用できる
- このディレクトリ内の実行ファイルは (/bin にあるものに加え) 起動と /usr のマウントに使われる
- このディレクトリ内の実行ファイルは (/bin にあるものに加え) システムリカバリーに使われる

/sbin と /usr/sbin

前述したように、最近の一部のディストリビューションでは、(/bin と /usr/bin と同じように) /sbin と /usr/sbin を分離するという戦略を放棄し、１つのディレクトリにシンボリックリンクを配置し、２つのディレクトリビューを持つものがあります。

このディレクトリには /bin ディレクトリ内のバイナリに加えて起動、復元、回復、または修復、もしくはそのすべてに必須のバイナリが含まれています。ブート中にルートファイルシステムが正常であると確認したら、必要に応じて /usr, /home、およびその他の場所に他のファイルシステムをマウントします。

次のプログラムはこのディレクトリにあります（これらのサブシステムがインストールされている場合)。

fdisk, fsck, getty, halt, ifconfig, init, mkfs, mkswap, reboot, route, swapon, swapoff, update.

```
$ ls -l /sbin
rwxrwxrwx 1 root root 8 Apr 23  2020 /sbin -> usr/sbin
```

/sbin Example

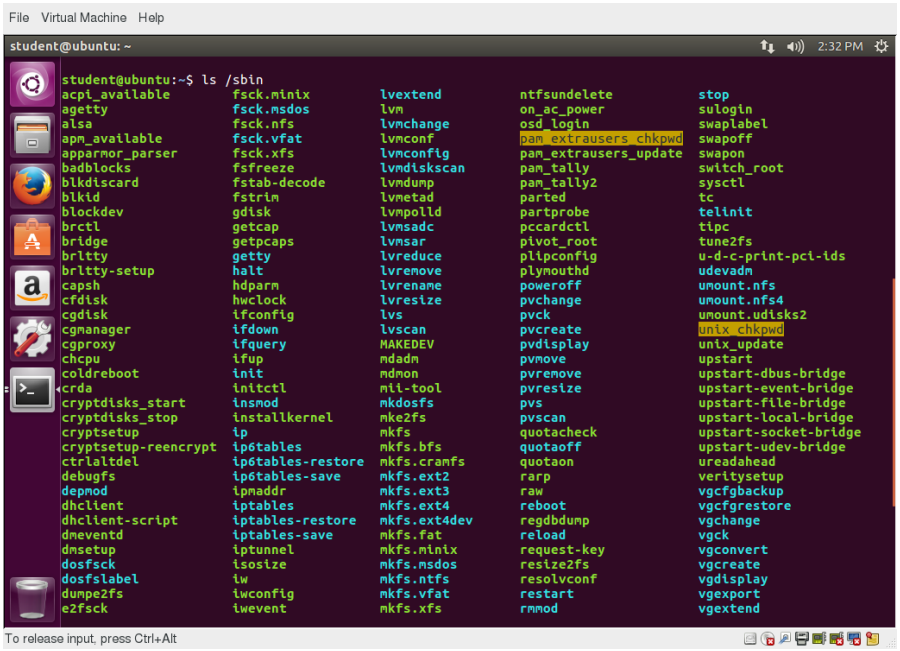
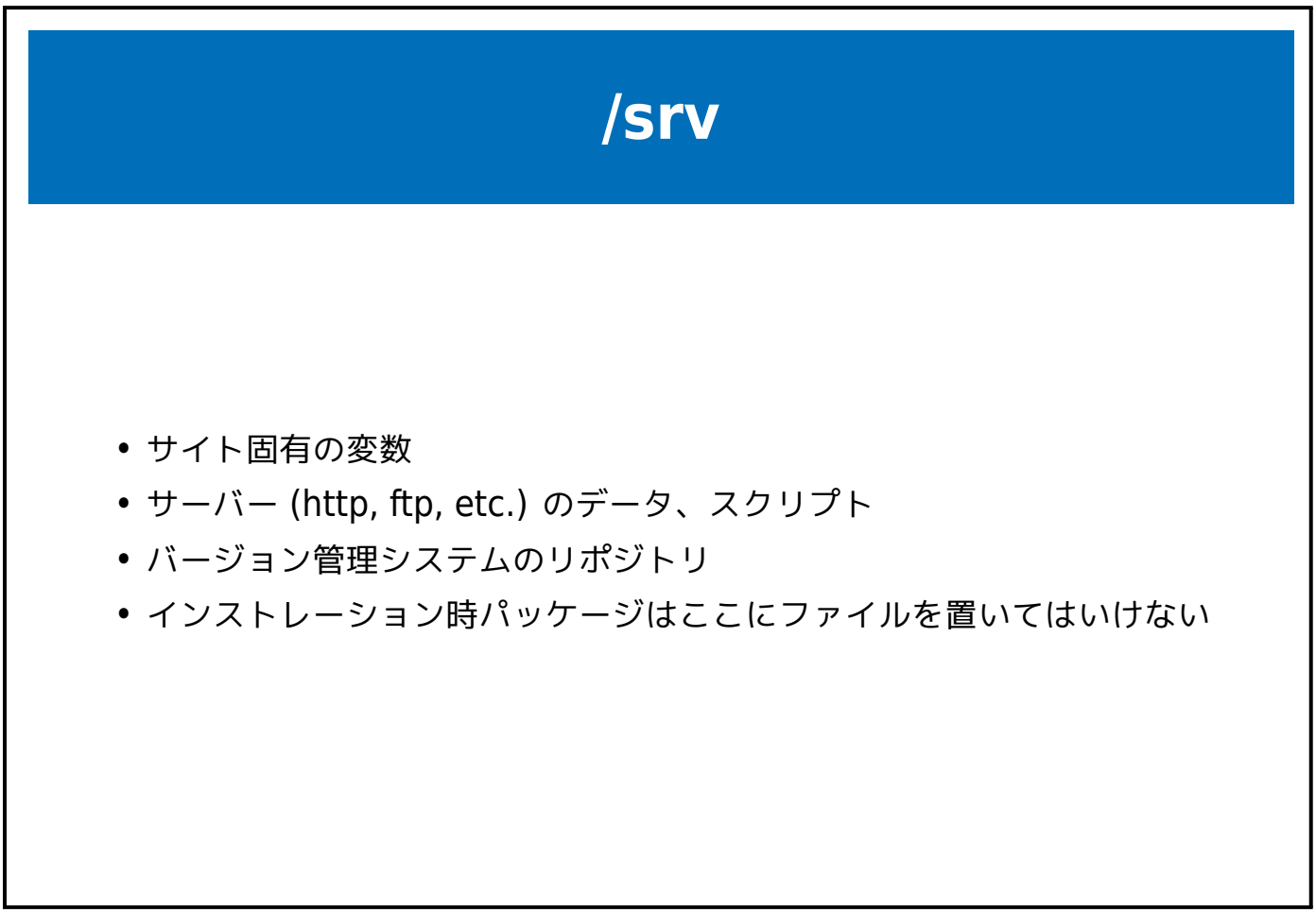


図 2.11: **/sbin Directory: Ubuntu 18.04**

RHEL、**CentOS**、**Fedora**、**Ubuntu** の最近のバージョンでは `/sbin` と `/usr/sbin` はシンボリックリンクになっているので実際には同じものです。

上のスクリーンショットは **Ubuntu 18.04** ですが最近のバージョンではリンクされていないディレクトリはありません。

2.18 /srv

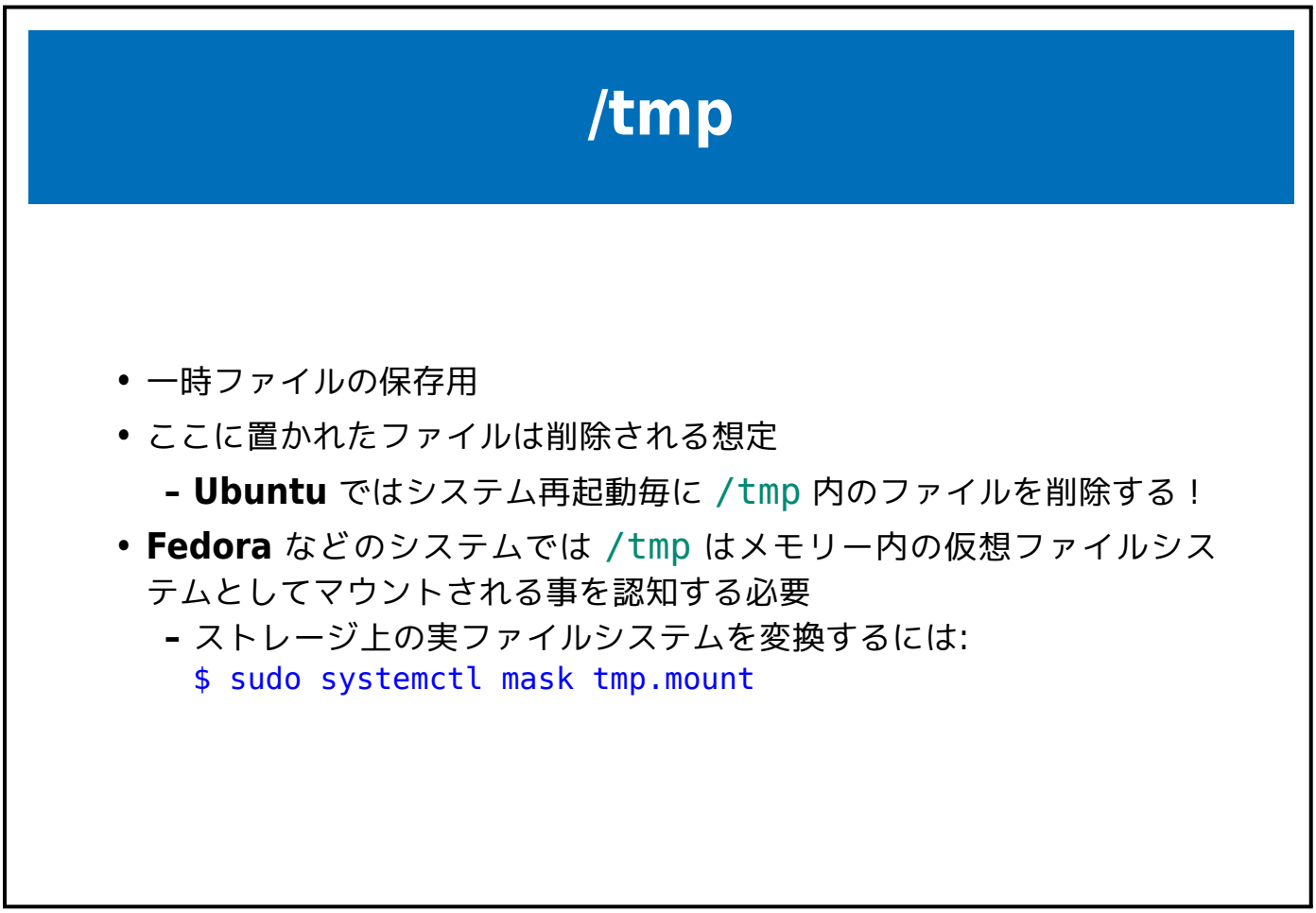


FHS は以下のように定義されています。

> /srv にはこのシステムによって提供されるサイト固有のデータが含まれています。
>
> これを指定する主な目的は、ユーザーが特定のサービスのデータファイルの場所を見つけることができるようにすることです。そして、読み取り専用データ、書き込み可能なデータ、およびスクリプト（cgi スクリプトなど）に対して単一のツリーを必要とするサービスを合理的に配置できるようにすることです。
>
> /srv のサブディレクトリに名前を付ける方法は、現時点ではこれがどのように行われるべきかのコンセンサスが得られていないため特定されていません。/srv の下でデータを構造化する 1 つの方法としてプロトコル（ftp、rsync、www、cvs など）があります。

システム管理者（およびディストリビューション）は `/srv` を使用するものと無視するものとに分かれます。
`/srv` とは対照的に `/var` に何を置くのが最善かについては、しばしば混乱が生じます。

2.19 /tmp



このディレクトリは一時ファイルを保存するために使用され、任意のユーザーまたはアプリケーションがアクセスできます。ただし /tmpにあるファイルを長期間使用することはできません。

- 一部のディストリビューションでは、パージスクリプトが変更されない限り 10 日以前のファイルを削除する自動 **cron** ジョブが実行されます。
- 一部のディストリビューションでは、リブートするたびに/tmp 内容が削除されます。これは **Ubuntu** のポリシーです。
- 最近の一部のディストリビューションは仮想ファイルシステムを利用しており /tmp ディレクトリを **tmpfs** ファイルシステムを使用する **RAM ディスク**のマウントポイントとしてのみ使用します。これは **Fedora** システムのデフォルトポリシーです。システムが再起動するとすべての情報が失われます。/tmp はまさに一時的なものです！

最後に注意ですが /tmp には大きなファイルを作成しないようにする必要があります。これらのファイルは実際にはディスクではなくメモリー内の領域を占有するので、メモリーの枯渇によってシステムに損害を加えたりクラッシュしたりしやすくなります。ガイドラインではアプリケーションが /tmp に大きなファイルを置かないようにしていますが、このポリシーに違反し /tmp に大きな一時ファイルを作成するアプリケーションはたくさんあります。（おそらく環境変数を指定することによって）それらを別の場所に配置することが可能であっても、多くのユーザーはその方法を認識しておらずすべてのユーザーが /tmp にアクセスしています。

Feodra の **systemd** を使用するシステムでは、次のコマンドを発行することによりこのポリシーをキャンセルできます。

```
$ sudo systemctl mask tmp.mount
```

その後、システムを再起動します。

2.20 /usr

Table 2.3: **/usr の下のディレクトリ**

/usr/bin	必須でなはいコマンドバイナリ
/usr/etc	必須ではない構成ファイル（通常は空）
/usr/games	ゲームデータ
/usr/include	アプリケーションのコンパイルに使用されるヘッダーファイル
/usr/lib	ライブラリファイル
/usr/lib64	64 ビット用のライブラリファイル
/usr/local	第三レベルの階層（マシンローカルファイル用）
/usr/sbin	必須ではないシステムバイナリ
/usr/share	読み取り専用のアーキテクチャ非依存のファイル
/usr/src	**Linux** カーネルのソースコードとヘッダー
/usr/tmp	第二の一時的なディレクトリ

/usr ディレクトリは **二次的な階層** と考えることができます。システムの起動に必要ではないファイルのために使用します。実際 /usr はルートディレクトリと同じパーティションに存在する必要はありません。ネットワーク越しに同じシステムアーキテクチャを使用するホスト間で共有できます。

ソフトウェアパッケージは /usr の直下にサブディレクトリを作成しないようにしてください。互換性のために他の場所へのシンボリックリンクが存在する場合があります。

通常このディレクトリは読み取り専用データです。シングルユーザーモードには必要のないバイナリが含まれています。/usr/local ディレクトリが含まれローカルバイナリなどが保存されます。**man** ページは /usr/share/man に保存されます。

非常に重要

最近の一部のディストリビューションは、/bin、/sbin、/lib、/lib64 を分離する戦略を放棄しており、/usr 以下にシンボリックリンクを使って１つのディレクトリで２つのディレクトリビューを持つものがあります。そのディストリビューションではブート後にマウントされるパーティションに /usr を配置するのは時代遅れとみています。

2.21 /var

/var

- 可変データファイルを配置、読み込み専用ではマウントできない
  - ログファイル
  - スプールディレクトリとスプールファイル
  - 管理用データファイル
  - キャッシュコンテンツなど一時ファイル

Table 2.4: **/var 以下のディレクトリ**

ディレクトリ	目　的
/var/ftp	ftp サーバーベースに使用
/var/lib	実行時にプログラムによって変更される恒久的なデータ
/var/lock	リソースへの同時アクセスを制御するために使用されるロックファイル
/var/log	ログファイル
/var/mail	ユーザーメールボックス
/var/run	ブート以降の実行中のシステムに関する情報
/var/spool	印刷キューなど **スプール** されたまたは処理を待機しているタスク
/var/tmp	システム再起動でも保持される一時ファイル。/tmp にリンクされることもある
/var/www	ウェブサイトの階層のルート

このディレクトリには、システム操作中に頻繁に変更 (=**var**iable) される変数（または **揮発性**）データファイルが含まれます。言うまでもなく **var** を読み取り専用ファイルシステムとしてマウントすることはできません。セキュリティ上の理由から、/var を別のファイルシステムとしてマウントすることは良い考えだと思います。さらにディレクトリがいっぱいになってもシステムはロックされるべきではありません。

/var/log はほとんどのログファイルが置かれている場所であり、/var/spool はメール、印刷、cron ジョブなどのプロセス用のローカルファイルがアクションを待っている間保存される場所です。

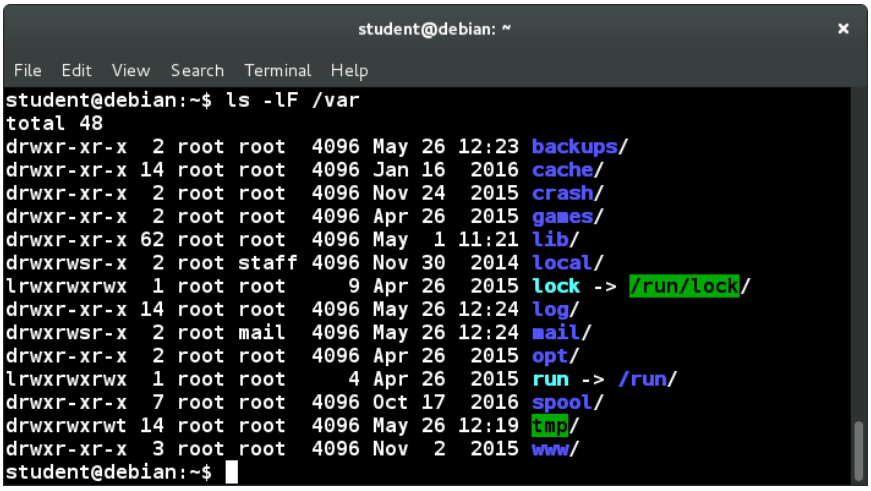


図 2.12: **/var ディレクトリ**

2.22 /run

/run

- 仮想ファイルシステム：実体はメモリー内
- 一時的なランタイムファイル用
- しばしばリムーバブルメディアのマウントに使われる

/run の目的は **一時的** なファイル保存です。これらはシステムの起動の早い段階で書き込まれ、再起動時に保持する必要がないランタイム情報が含まれます。

通常 /run は空のマウントポイントとして実装され、実行時に **tmpfs** RAM ディスク（/dev/shm など）がマウントされます。これはメモリーにのみ存在する疑似ファイルシステムです。

/var/run や /var/lock などは、/run の下のディレクトリへの単なるシンボリックリンクになります。ディストリビューションによっては他のディレクトリも /run の下の場所を指しているだけの場合があります。

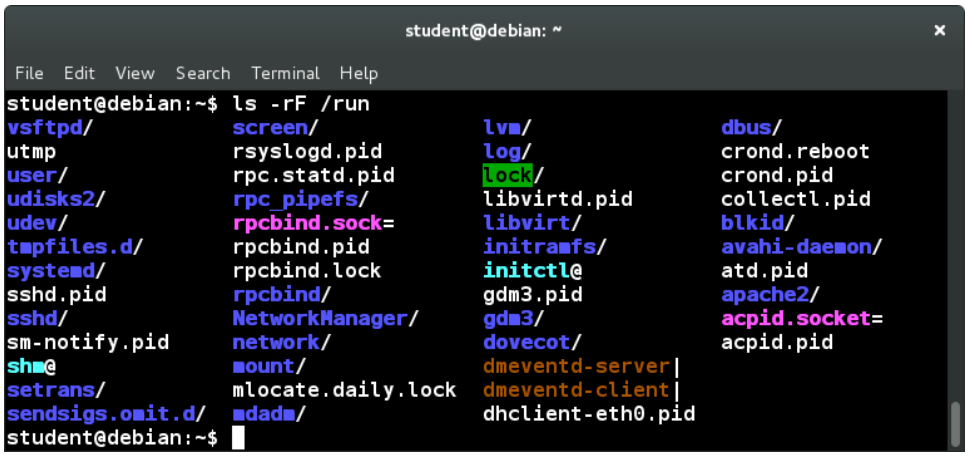


図 2.13: **/run ディレクトリ**

2.23 演習

デモ教材ビデオ

using_linux_distros_demo.mp4
using_disk_usage_demo.mp4

課題 2.1: Linux の標準的なディレクトリのサイズ

du ユーティリティを利用して、システムの各最上位ディレクトリのサイズを計算します。

以下のコマンドを入力:

```
$ du --help
```

結果を得るための方法のヒント。

解 2.1

/ 以下のディレクトリの一覧とサイズを表示:

```
$ sudo du --max-depth=1 -hx /
```

```
4.3M    /home
16K     /lost+found
39M     /etc
4.0K    /srv
3.6M    /root
178M    /opt
138M    /boot
6.1G    /usr
1.1G    /var
16K     /mnt
4.0K    /media
869M    /tmp
8.4G    /
```

以下のオプションを使用:

- `--maxdepth=1`: `/` の一段下の階層以下を全て合計します。
- `-h`: 人間に分かりやすい表示（KB、MB、GB）。
- `-x` 特定のパーティションに限定します。`/` パーティションに無いディレクトリは無視します。具体的には以下を無視します:

  `/dev /proc /run /sys`

  これらは疑似ファイルシステムであり、メモリーのみに存在します: システムが実行されていないときは、マウントポイントは空になります。上の例は **RHEL** システムで実行されたものなので、以下のマウントポイントも含まれていません:

  `/bin /sbin /lib /lib64`

  `/usr` 下の対応するものに、シンボリックリンクされています。

課題 2.2: /proc ファイルシステムを探索する

注目

本演習の内容は、カーネルのバージョンに依存します。したがって、必ずしも出力は一致しません。

1. root ユーザーで /proc に **cd** し、ディレクトリを表示します。以下のように表示されます:

```
$ cd /proc
$ ls -F
```

```
1/      128/    1510/   20/     2411/   30895/  53/    6925/  802/  951/         kmsg
10/     129/    1511/   2015/   2425/   31/     54/    7/     81/   952/         kpagecgroup
1002/   13/     1512/   2022/   2436/   31449/  55/    70/    813/  957/         kpagecount
1007/   130/    1513/   2023/   2444/   32/     56/    702/   814/  97/          kpageflags
10540/  131/    1514/   20300/  2451/   33/     58/    709/   816/  9742/        loadavg
10590/  13172/  152/    20354/  2457/   34/     585/   71/    817/  98/          locks
10798/  132/    15552/  20380/  2489/   35/     59/    718/   82/   99/          meminfo
10805/  133/    15663/  20388/  25/     36/     60/    719/   83/   9923/        misc
10806/  134/    15737/  20392/  2503/   37/     61/    72/    834/  acpi/        modules
10809/  135/    159/    20396/  2504/   374/    6193/  721/   835/  asound/      mounts@
10810/  136/    15981/  2086/   2531/   379/    62/    723/   84/   buddyinfo    mtrr
10813/  137/    16/     2090/   2546/   38/     63/    725/   841/  bus/         net@
10894/  138/    162/    211/    2549/   380/    634/   727/   842/  cgroups      pagetypeinfo
10925/  1384/   1632/   22/     2562/   40/     64/    73/    85/   cmdline      partitions
10932/  1385/   1636/   2205/   25794/  41/     65/    7300/  857/  config.gz    sched_debug
10934/  1387/   166/    2209/   26/     42/     662/   74/    86/   consoles     scsi/
10935/  139/    1670/   2212/   2610/   43/     663/   757/   864/  cpuinfo      self@
10941/  1390/   17/     2232/   26108/  44/     665/   758/   867/  crypto       slabinfo
10983/  1393/   17271/  2238/   2619/   4435/   666/   76/    87/   devices      softirqs
10998/  14/     17361/  2296/   2624/   45/     67/    761/   88/   diskstats    stat
11/     140/    1793/   2298/   2627/   46/     670/   762/   881/  dma          swaps
11047/  1410/   18/     23/     2644/   468/    671/   765/   886/  driver/      sys/
1105/   1415/   1831/   23042/  2645/   47/     673/   766/   887/  execdomains  sysrq-trigger
1121/   1429/   18880/  2344/   2679/   470/    674/   768/   888/  fb           sysvipc/
1123/   1437/   18903/  2348/   27/     484/    678/   769/   889/  filesystems  thread-self@
1135/   1445/   19/     2353/   2706/   49/     679/   77/    89/   fs/          timer_list
11420/  146/    19392/  2354/   2762/   492/    68/    771/   9/    interrupts   timer_stats
11499/  1463/   19488/  2365/   28/     493/    682/   78/    90/   iomem        tty/
11515/  147/    1954/   23683/  2858/   5/      683/   79/    92/   ioports      uptime
11530/  1476/   1963/   2370/   28730/  50/     686/   793/   921/  irq/         version
1163/   148/    19727/  2372/   28734/  51/     687/   794/   928/  kallsyms     vmallocinfo
1164/   1485/   19734/  2374/   29/     510/    69/    8/     930/  kcore        vmstat
12/     149/    19984/  24/     2973/   514/    690/   80/    931/  keys         zoneinfo
127/    15/     2/      2406/   3/      52/     691/   801/   944/  key-users
```

たくさんの数字のディレクトリが存在しています: これらはそれぞれプロセスに対応し、名称は **プロセス ID** です。重要なサブディレクトリとして、/proc/sys があります。その下に多数のシステムパラメータがあり、それらを参照したり、更新したりできます。このサブディレクトリについては、あとで議論します:

2. 以下のファイルを参照:

   - /proc/cpuinfo:
   - /proc/meminfo:
   - /proc/mounts:
   - /proc/swaps:
   - /proc/version:
   - /proc/partitions:
   - /proc/interrupts:

   名称から内容の想定がつきます。

   これらの値は、常時更新されているわけではなく、参照したときに更新されることに注意してください。

3. プロセスのディレクトリのどれかを参照してください（それ以外のディレクトリは **sudo** しないと見えない情報があります）:

```
$ ls -F 4435
```

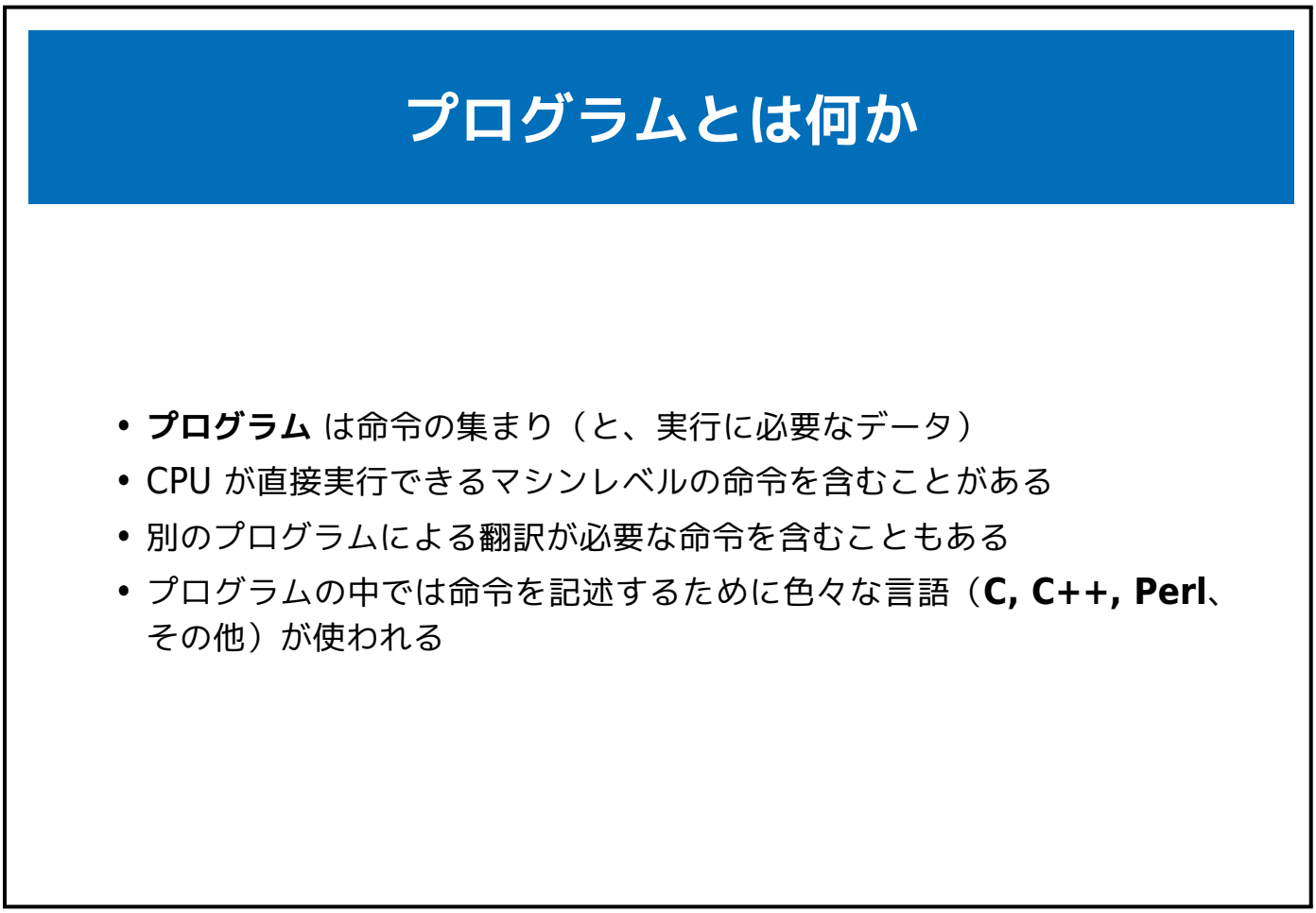
```
attr/        coredump_filter  gid_map     mountinfo   oom_score_adj  sessionid  syscall
autogroup    cpuset           io          mounts      pagemap        setgroups  task/
auxv         cwd@             limits      mountstats  personality    smaps      timerslack_ns
cgroup       environ          loginuid    net/        projid_map     stack      uid_map
clear_refs   exe@             map_files/  ns/         root@          stat       wchan
cmdline      fd/              maps        oom_adj     sched          statm
comm         fdinfo/          mem         oom_score   schedstat      status
```

以下の項目を見てください: `cmdline, cwd, environ, mem, status`

章 3

プロセス

3.1 プログラムとプロセス



プログラムとは、一連の命令と命令を実行する時に使用する内部データのことを言います。またプログラムが外部データを使用する場合もあります。内部データには、ユーザープロンプトの表示に使用される、プログラム内のテキスト文字列が含まれる場合があります。外部データにはデータベースからのデータを含む場合があります。

ls、**cat**、**rm** などの多くのユーザコマンドは、オペレーティングシステムのカーネルやシェルとは別のものです。（つまり、ディスク上にある独立した実行可能プログラムです）。

プロセスとは？

- **プロセス** とは実行中のプログラムのインスタンスである
- **Linux** は全ての実行済み、ないし実行中のプログラムに対して新しいプロセスを作る
- ひとつのプログラムが、複数のプロセスを同時に実行することがある
- オペレーティングシステムの重要な役割は、ユーザーに代わってプロセスの実行を管理することである

プロセスとは実行中プログラムのインスタンスです。実行中 や スリープ中 などさまざまな状態（ステート）を持ちます。

すべてのプロセスには **pid**（プロセス ID）、**ppid**（親プロセス ID）、**pgid**（プロセスグループ ID）があります。さらにすべてのプロセスはプログラムコード、データ、変数、ファイル記述子、および環境を持っています。

プロセスは完全にプリエンプティブに **スケジューリング** されます。プロセスをプリエンプションする権限を持つのはカーネルのみです。プロセス同士ではできません。

過去のいろいろな経緯から、**pid** の最大値は 16 ビット数、つまり 32768 に制限されています。`/proc/sys/kernel/pid_max` を変更することでこの値を変更することができます。大規模なサーバーでは、デフォルトの最大値では足りない場合があるためです。プロセスが生成され続けて行くと、最終的にその数は `pid_max` になります。そうすると再び `PID = 300` から割り当てられます。

プロセス属性

- プロセスは、全ていくつかの属性を持つ
  - 実行されたプログラム
  - コンテキスト（ステート）
  - 権限
  - 関連づけられたリソース

すべてのプロセスが何らかのプログラムを実行しています。プロセスは CPU レジスタの状態、プログラム内で実行されている箇所、プロセスのメモリーの内容、およびその他の情報を補足することにより、いつでも自身のスナップショットを取ることができます。このスナップショットがプロセスの **コンテキスト** です。

プロセスは CPU 時間を他のユーザーと共有する（または、ユーザー要求やデータ到着を待つなどの条件が満たされるまでスリープしなければならない）ときにプロセスのスケジュール切り替えができるので、プロセスをスワップアウトする時に全てのコンテキストを保存し実行時に復帰させる **コンテキストスイッチング** の機能は kernel の重要な役割になります。

すべてのプロセスには、実行のために呼び出したユーザーに基づいたパーミッションがあります。また、プログラムファイルの所有者に基づいたパーミッションもあります。「s」実行ビットでマークされたプログラムには、「実効」ユーザー ID とは異なる「実」ユーザー ID があります。これらのプログラムは setuid プログラムと呼ばれます。これらはプログラムを所有するユーザーのユーザー ID で実行されます。これに対して非 setuid プログラムは、プログラムを開始したユーザーのパーミッションの範囲内で実行されます。ただし root が所有する setuid プログラムの実行はセキュリティ上の問題になる可能性があります。

setuid プログラムの例として **passwd** コマンドがあります。これはすべてのユーザーが実行できます。ユーザーがこのプログラムを実行するとプロセスは root 権限で実行され、ユーザーのパスワードが保持されている書き込み制限があるファイル `/etc/passwd` と `/etc/shadow` を更新できるようになります。

すべてのプロセスは、割り当てられたメモリー、ファイルハンドラなどのリソースを持っています。

プロセスリソースの分離

- カーネルは（デフォルトでは）個々のプロセス同士を隔離する
  - セキュリティ確保
  - 安定性の向上
- カーネルはプロセスがハードウエア資源にアクセスする事を認めない
  - ハードウエアへのアクセスはカーネルが管理
  - **システムコール** を利用してアクセスさせる

プロセスが開始されると、そのプロセスは他のプロセスから保護するために独自のユーザー空間を持ちます。これによりセキュリティが向上し、安定性が向上します。

プロセスはハードウェアに直接アクセスできません。ハードウェアはカーネルにより管理されるため、プロセスは **システムコール** を使用して間接的にハードウェアにアクセスします。システムコールはアプリケーションとカーネルの間の基本的なインターフェイスです。

3.2 プロセスに対するリミット設定

ulimit でプロセスを制御する

- ulimit は **bash** に組み込まれたコマンド
- プロセスに関連するリソース制限の表示または再設定
- 個々のユーザーやプロセスが以下のようなシステムリソースを使い果たすことがないように機能を **制限** するために利用する
  - メモリー
  - CPU 時間
  - システム上のアクティブなプロセスやファイルの最大数
- システム管理者は、全てのユーザーに影響するようなシステム全体の制限をかけることができる

システム管理者は、以下のどちらかの方向に向けていくつかの値を変更する必要があるでしょう。

- 機能を **制限** して、個々のユーザーやプロセスがメモリー、CPU 時間、システム上のプロセスの最大数などのシステムリソースを使い果たすことがないようにする
- 機能を **拡張** して、プロセスがリソースの限界に達しないようにする。たとえば多くのクライアントを処理するサーバーでは、デフォルトのファイルオープン数の制限 1024 のままではサーバーとして機能しない場合があります。

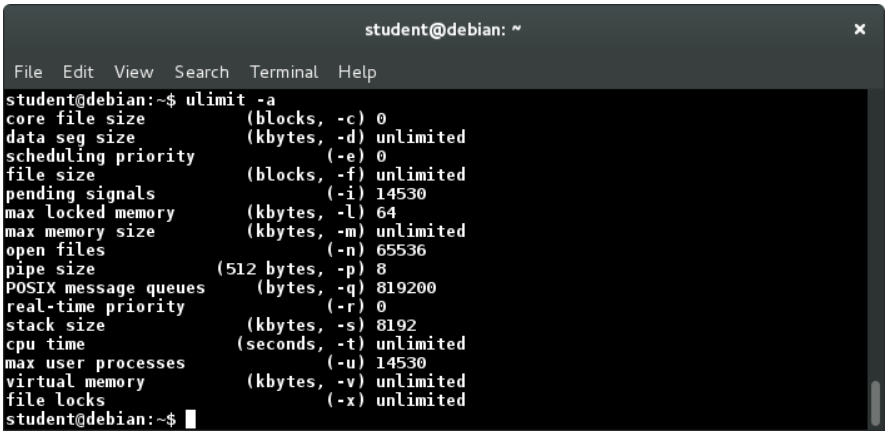


図 3.1: **ulimit**

リミットの設定

- 以下を実行することにより特定の制限値を設定できる
  `$ ulimit [options] [limit]`
  以下のように
  `$ ulimit -n 1600`
  同時に開くことができるファイルの最大数が 1600 になる
- `/etc/security/limits.conf` の設定値を修正してリブートすると設定が永続化する

変更は、現在のシェルにのみ影響することに注意してください。ログインしているすべてのユーザーに有効な変更を加えるには `/etc/security/limits.conf`（とても良いドキュメントが付属しています）を変更してから再起動する必要があります。

ソフトリミット と ハードリミット

リミットには次の２種類があります。

- **ハード**: ユーザーが引き上げることができるリソース制限の最大値、root ユーザーのみが設定可能
- **ソフト**: 現在の制限値。一般ユーザーが変更できますがハードリミットを超えることはできない

```
$ ulimit -H -n
4096
```

```
$ ulimit -S -n
1024
```

3.3 プロセスの生成

プロセスの生成

- 最初のユーザープロセスは **init** と呼ばれ pid 1 を持つ
- 後続プロセスは **init** や他の実行中プロセスから **派生 (fork)** して作られる
- **親プロセス**から派生された**子プロセス**は、親プロセスの完全なクローンとしての特徴を持つ
- 親プロセスが死んだ場合には子プロセスは **init** の養子になる
- カーネルもプロセスを生成する。**ps** コマンドで見るとそれらは [..] で囲まれた名前を持つ

注目

systemd ベースのシステムでは pid=2 の **kthreadd** という孤児となったプロセスを養子にするための特別なプロセスがある。養子となったプロセスの親プロセス ID は ppid=2 となる

通常の **Linux** システムは常に新しいプロセスを生成します。新しくプロセスを生成することをしばしば **派生 (fork)** と呼びます。新しい子プロセスがスタートしても元の親プロセスも動き続けます。

しばしば fork だけで終わらずに後に **exec** が続きます。この時親プロセスは終了し、子プロセスが親プロセスのプロセス ID を継承します。しばしば **fork and exec** とペアで呼ばれるのでこれが一つの命令だと考えている人も多いのです。

古い **UNIX** システムでは **spawn** と呼ばれるプログラムが多用されました。これは多くの点で fork and exec と似た動作をするものですが細部で違いがあります。**spawn** は **POSIX** との互換性がなく標準的な **Linux** には使われていません。

どうやって新しいプロセスが起動するかを見るのに、多くのクライアントが接続された Web サーバーを考えてみましょう。クライアントから新しい接続がある毎に新しいプロセスが立ち上がります。または、同じプロセスの中で新しい **thread** をスタートさせるだけの場合もあります。**Linux** では、技術的にみれば、完全に新しいプロセスを作るのと単にスレッドを作るのには大きな違いは無いのです。どちらの方法でも起動時間はほぼ同じで消費するリソースも大差ありません。

別の、**init** プロセスが **sshd** init スクリプトを実行して **sshd** デーモンを起動するケースでは、**init** プロセスの役割は **sshd daemon** を起動するまでです。以降はデーモンがリモートユーザーからの **ssh** 接続要求を待ち受けます。

接続要求が届くと **sshd** は接続リクエストに応答するために自分自身のコピーを作成します。リモートから接続する各ユーザーは自分専用の **sshd** デーモンを使ってリモートログインを処理できるようになります。ログインプログラムがリモートユーザーを確認すると **sshd** がスタートします。認証が成功するとログインプロセスが (**bash** と呼ばれる) シェルを fork してユーザーのコマンドを解釈できるようにします。などなど

カーネルの内部プロセスは色々なメンテナンス業務を担っています。例えば、バッファーされたデータをディスクに書き出すとか、各 CPU の負荷がバランスするようにワークロードを調整するとか、デバイスドライバーに要求されているデバイス操作を実行させるといった仕事です。これらのプロセスはシステム稼働中ずっと何かやることが発生するまではスリープした状態で残り続けます。

コマンドシェル 内で プロセス を生成

ユーザーが **bash** などのコマンドシェルインタープリター上でコマンドを実行すると:

- 新しいプロセスが生成 (ユーザーのログインシェルから fork される)
- システムコールによって親プロセスがスリープするのを待つ
- **exec** システムコールによってコマンドが子プロセスにロードされ **bash** を置き換える
- コマンド完了時には子プロセスは exit システムコールを発行して死ぬ
- 子プロセスが死ぬと親プロセスが再開、新たなシェルプロンプトを出す
- 親プロセスはユーザーの次のコマンド入力を待つ、このサイクルが繰り返される

もしコマンドが（コマンドラインの最後にアンパーサント (&) を付加することによって）**バックグラウンド** 処理するように起動された場合には、親プロセスは wait リクエストをスキップしてすぐにシェルプロンプトを出します。この時バックグラウンドに回ったプロセスも並列に実行されます。そうではなく **フォアグランド** 実行の場合シェルは子プロセスの処理が完了するかシグナルによって停止するまで待機します。

いくつかのシェルコマンド（**echo** や **kill** など）はシェルのビルトインコマンドなので、実行時に外部のプログラムをロードしません。これらのコマンドを実行する時には **fork** も **exec** も発行されません。

3.4 プロセスの状態

プロセスの状態

- プロセス次の何れかの状態になる
  - **実行 (Running)**: CPU 上で実行中、または OS が許可すれば実行できる状態
  - **待機 (Waiting)**: ディスクないし何かの外部リソースからの情報を待っている状態。**スリープ** とも呼ぶ
  - **中断 (Stopped)**: デバッガー、または Ctrl-Z によって実行を止められた状態
  - **ゾンビ (Zombie)**: 終了済みだが正しく親プロセスから正しく切り離されていない状態

プロセスは、現在 CPU または CPU コアで **実行中** か、実行キューにいて新しいタイムスライスの割り当てを待っています。スケジューラが CPU を占有するべきと判断した場合、または別の CPU がアイドル状態になりスケジューラがその CPU にプロセスを移行した場合、実行を再開します。

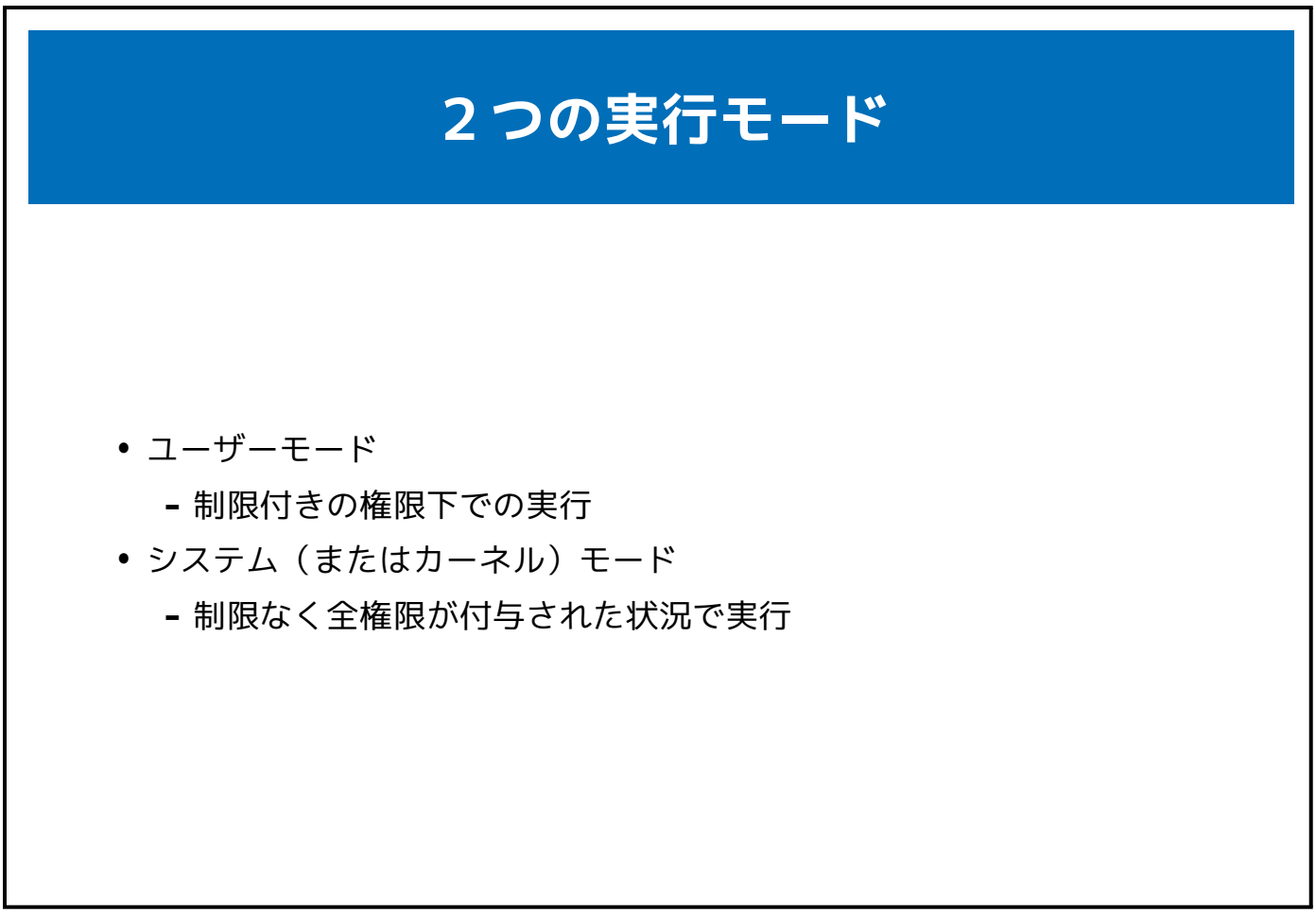
待機 (または **スリープ**) プロセスは、リクエスト（通常は I/O）が実行されるのを待っている状態です。リクエストが完了するまで待ち続けます。割り込みイベントによってリクエストが完了すると OS は待機していたプロセスに再び CPU 資源の最割り当てを許可しプロセスを続行させます。

プロセスは **中断** されています。この状態はプログラマが実行中のプログラムのメモリー、CPU レジスタ、フラグ、またはその他の属性を調べたい場合によく発生します。この調査が完了するとプロセスを再開できます。

プロセスが終了すると **ゾンビ** 状態になります。ゾンビ状態となったプロセスも OS のプロセステーブルに残っている事は重要です。プロセステーブルにはシステム内の全てのアクティブプロセスのエントリーが登録されています。ゾンビ状態となるとプロセステーブルへのエントリー以外の全てのリソースが開放されます。

スケジューラはすべてのプロセスを管理します。プロセスの状態はプロセスリストによって知ることができます。

3.5 実行モード



いつでもプロセス（またはマルチスレッドプロセスの特定のスレッド）は **ユーザーモード** または **システムモード** で実行できます。カーネル開発者は、通常、システムモードを **カーネルモード** と呼びます。

実行できる命令はモードによって異なり、ソフトウェアではなくハードウェアレベルで実行されます。

モードはシステムの状態ではありません。これはプロセッサの状態です。マルチコアまたはマルチ CPU システムでは各ユニットが個別に独自の状態になることができます。

Intel はユーザーモードを **Ring 3** 、システムモードを **Ring 0** と呼びます。

ユーザーモード

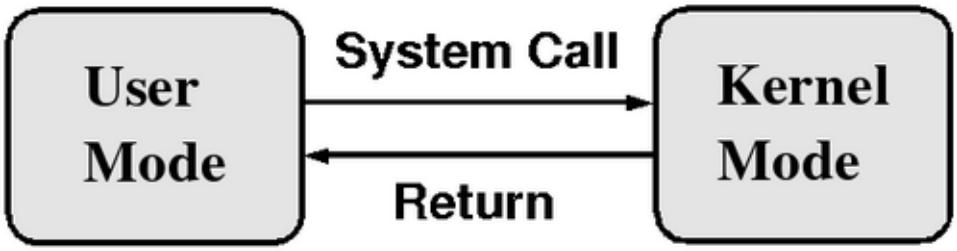


図 3.2: **ユーザーとカーネルモードとシステムコール**

- **ユーザーモード** は **カーネルモード** より権限が限定的
- 通常プログラムはこのモードで実行

システムコール（次セクションで説明します）を実行する場合を除きプロセスは **ユーザーモード** で実行されます。ユーザーモードはカーネルモードよりも権限が低くなります。

プロセスが開始されると、そのプロセスは他のプロセスから保護するために独自のユーザー空間を持ちます。これによりセキュリティが向上し安定性が向上します。これを **プロセスリソース分離** と呼ぶことがあります。

ユーザーモードで実行する各プロセスには独自のメモリー空間があり、その一部は他のプロセスと共有できます。共有メモリーセグメントを除きユーザープロセスは他のプロセスのメモリー空間の読み書きはできません。

root ユーザーによって実行される、または setuid プログラムとして実行されるプロセスもユーザーモードで実行されます。システムコールにジャンプする場合を除きハードウェアにアクセスは限られています。

システム（カーネル）モード

- プログラムによるシステムコール発行で起動
- システムコールの完了まで継続
- ユーザーモードより高い権限が付与
- システム内でカーネルコードを実行するプロセス
  - ディスクアクセス（読み込み、書き込み）
  - 追加メモリーの要求 (brk, sbrk)
  - カレントディレクトリーの変更 (cd)

システム（カーネル）モードでは CPU は、周辺機器、メモリー、ディスクなど、システム上のすべてのハードウェアに完全にアクセスできます。アプリケーションがこれらのリソースにアクセスする必要がある場合、**システムコール** を発行する必要があります。これにより、ユーザーモードからカーネルモードへの **コンテキスト切り替え** が発生します。ファイルの読み取りや書き込み、新しいプロセスの生成などを行う場合はこの手順に従う必要があります。

アプリケーションコードはカーネルモードで実行されることはなく、カーネルコードであるシステムコールのみが実行されます。システムコールが終了すると戻り値が生成されプロセスは逆コンテキストスイッチでユーザーモードに戻ります。

ハードウェア割り込みを処理したり、システムのスケジューリングルーチンやその他の管理タスクを実行したりするときなど、プロセスとは関係なくシステムがカーネルモードにある場合もあります。

3.6 デーモン

デーモン

- **デーモン** プロセスはユーザーにシステムの特定サービスを提供することが唯一の目的
- 必要なときにのみ動作するため非常に効率的
- 多くのデーモンはブート時に開始される
- デーモンの名前の多くは（常にではありませんが）最後に **d** が付く。例えば **httpd** や **systemd-udevd**
- 外部イベント (**systemd-udevd**) や経過時間 (**crond**) の影響を受ける場合がある
- 通常デーモンには制御用端末や標準入出力デバイスはない
- より優れたセキュリティ制御を提供できることがある

デーモンプロセスは、ユーザーにシステムの特定サービスを提供することを唯一の目的とするバックグラウンドプロセスです。例えば **xinetd**、**httpd**、**lpd**、**vsftpd** があります。

3.7 nice 値

nice を使用した優先順位の設定

- 全てのプロセスには **nice** 値が設定され、それによってプロセス実行時の優先度が決定される
- **nice** 値の調整によって、nice がプロセスの優先度を上げたり下げたりできる
- nice 値が大きいほど優先度は低くなる
- nice 値が小さいほど優先度は高くなる
- 非特権ユーザーができるのは nice 値を上げることだけ
- スーパーユーザーだけが nice 値を下げることができる

nice 値の範囲は -20（最少の nice 値、最高の優先順位）から +19（最高の nice 値、最低の優先順位）までです。スーパーユーザーだけが nice 値を下げられますが、一般ユーザーも nice 値をあげることができる点に注意してください。プロセスは生成時にシェルから **nice** 値 **0** を引き継ぎます。

nice の実行方法は次のとおりです。

nice はプロセスを特定の **nice** 値で実行させるために使います。

```
$ nice -n 10 myprog
```

myprog は nice 値 10 で実行されます。

```
$ nice -n 19 myprog
```

myprog は nice 値 19 を参照し最低優先度で実行されます。

```
$ sudo nice -n -20 myprog
```

myprog は nice 値 -20 を参照し最高優先度で実行されます。

nice コマンドで nice 値を指定しないとデフォルトで nice 値は 10 加算されます。引数を指定しない場合は、現在の nice 値が報告されます。

nice 値を上げてもそのプロセスが実行されないわけではありません。競合するものが他に何もない場合はすべての CPU 時間を取得することもあります。

nice 値の変更

- **renice** を使って実行中のプロセスの nice 値の変更ができる

```
$ renice --help

Usage:
 renice [-n] <priority> [-p|--pid] <pid>...
 renice [-n] <priority>  -g|--pgrp <pgid>...
 renice [-n] <priority>  -u|--user <user>...
```

- pid 20003 の nice 値を 5 にあげる

```
$ renice +5 -p 20003
```

- スーパーユーザーだけがプロセスの nice 値を下げることができる

renice を使うことで既に実行中のプロセスの nice 値の上げ下げができます。これによって稼働中に nice 値の変更ができるようになっています。

デフォルトではスーパーユーザーだけが nice 値を下げる（すなわち優先度を上げる）ことができます。しかし、通常ユーザーもあらかじめ `/etc/security/limits.conf` で設定された範囲内であれば自分のプロセスの nice 値を下げることができます。

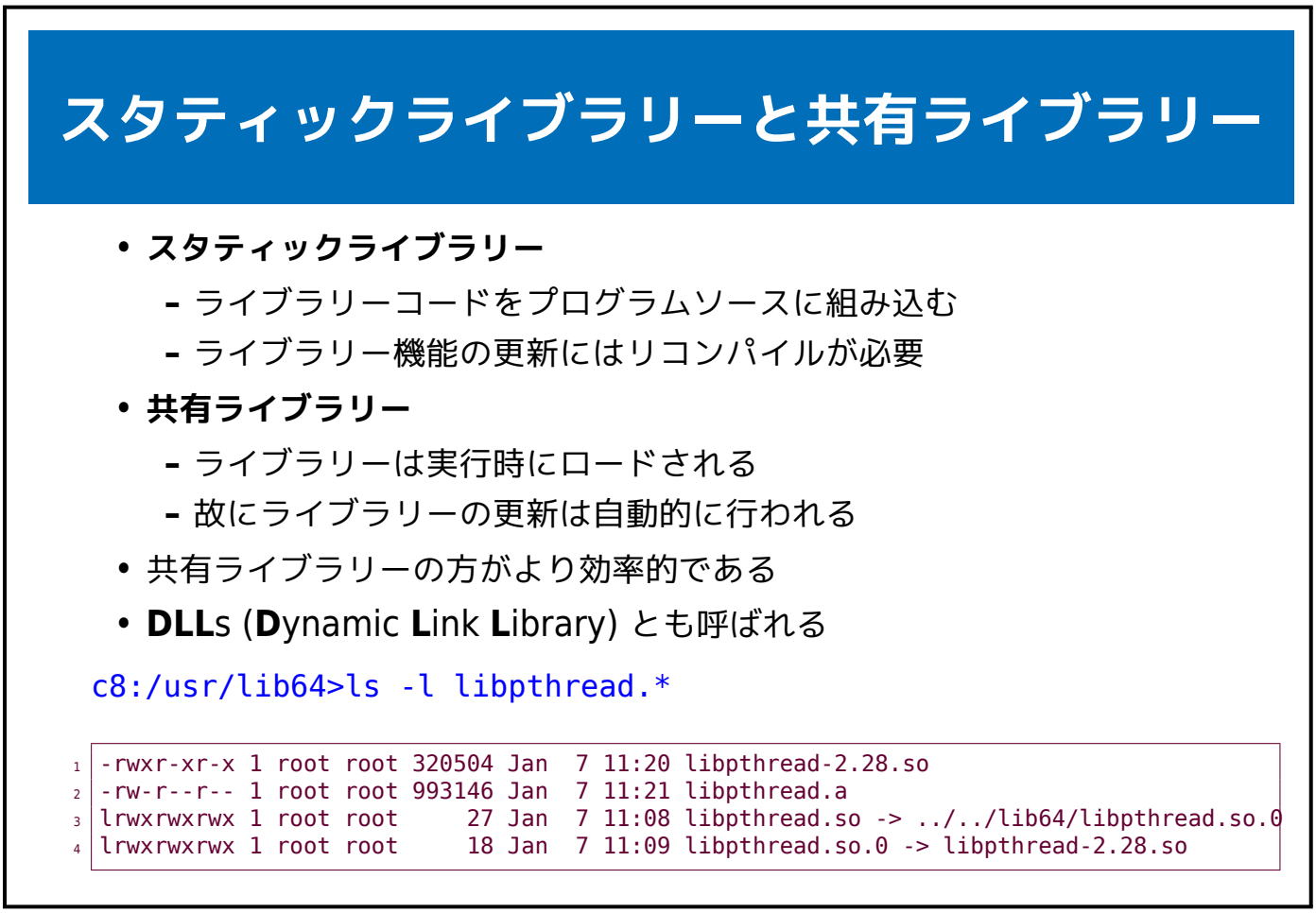
非特権ユーザーが nice 値を上げた場合、それを元の値に下げることが出来るのはスーパーユーザーだけです。

実行中のプロセスの nice 値変更は **renice** コマンドの利用が簡単です。

```
$ renice +3 13848
```

このコマンドで **pid = 13848** の nice 値は 3 に再設定されます。１つ以上のプロセスの値を同時に変更することや、その他のオプションもあります。詳細については **man renice** を参照してください。

3.8 ライブラリー



プログラムは、色々な用途向けに開発され多くの場面で利用／再利用されている **ライブラリー** コードを利用してビルドされます。
ライブラリーに２つの種類があります。

- **スタティック**：ライブラリー機能のコードは **コンパイル時** にプログラムに挿入され、以降はライブラリーが更新されたとしても変更されることはありません。
- **シェアード（共有）**：ライブラリー機能のコードは **実行時** にプログラムにロードされます。ライブラリーが後に変更された場合には、プログラムは更新された新しいライブラリーを利用します

シェアード（共有）ライブラリーは、多くのアプリケ-ションで同時利用、メモリー消費、サイズ、アプリケーションのローディング時間など多くの理由からより効率的です。
シェアード（共有）ライブラリーは **DLL**s (**D**ynamic **L**ink **L**ibrary) とも呼ばれます。

共有ライブラリーのバージョン

- バージョン付与には特段の配慮が必要
- **DLL 地獄** を回避するために
- プログラムは特定の **メジャー** ないし **マイナー** ライブラリーバージョンを要求できる
- アプリケーションの中には問題回避のためにスタティックライブラリーだけを利用するものがあるが、バグ修正やセキュリティ対策が遅れる懸念もある

共有ライブラリーに対するバージョン番号の付与には特段の配慮が必要です。ライブラリーが大幅に変更され、プログラムがそれに適切に対応できていないと深刻な問題が発生します。この問題は **DDL 地獄** と呼ばれます。

このため、プログラムはシステム上にある最新版とは別の特定の **メジャー** ライブラリーバージョンを要求することができます。しかしながら、通常はプログラムは利用可能な最新の **メジャー** バージョンを利用します。

このような問題を回避するために、いくつかのアプリケーションはプログラムにスタティックライブラリーとして組み込んでいます。しかしながら、ライブラリーにバグ対策やセキュリティフィックの改善がされても、それらを速やかにアプリケーションで利用することができません。

共有ライブラリーは **.so** という拡張子をもっています。通常ライブラリー名は `libc.so.N` のような名前になり、`N` にはメジャーバージョン番号が入ります。

Linux は共有ライブラリーのバージョン番号に配慮しています。具体例は

```
c8:/usr/lib64>ls -l libcrypt.*
```

```
-rw-r--r-- 1 root root 233788 May 10  2019 libcrypt.a
lrwxrwxrwx 1 root root     17 Aug 12  2018 libcrypt.so -> libcrypt.so.1.1.0
lrwxrwxrwx 1 root root     17 Aug 12  2018 libcrypt.so.1 -> libcrypt.so.1.1.0
-rwxr-xr-x 1 root root 142712 Aug 12  2018 libcrypt.so.1.1.0
```

単純に `libcrypt` を要求するプログラムは `libcrypt.so` が割り当てられ、メジャーないしマイナーバージョンを指定した場合は該当のライブラリーが使われます。また、スタティックライブラリーを要求するプログラムのために `libycrypt.a` も用意されています。

共有ライブラリーの検索

- 共有ライブラリーは実行時に見つからなければならない
- **ldd** はプログラムが利用している共有ライブラリーを示す
- **ldconfig** は `/etc/ld.so.conf` にライブラリー検索場所を保存
  - 起動時、または後で実行される
- デフォルトの検索対象ディレクトリーを LD_LIBRARY_PATH で上書きできる

ldd を使って、ライブラリーの **soname** と、どのファイルがポイントされているかを見ることによって、どの共有ライブラリーが要求されたを突き止めることができます。

```
student@ubuntu: $ ldd /usr/bin/vi
```

```
        linux-vdso.so.1 (0x00007ffd2ae93000)
        libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f9c9275a000)
        libtinfo.so.6 => /lib/x86_64-linux-gnu/libtinfo.so.6 (0x00007f9c9272a000)
        libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f9c926ff000)
        libcanberra.so.0 => /lib/x86_64-linux-gnu/libcanberra.so.0 (0x00007f9c926ec000)
        libacl.so.1 => /lib/x86_64-linux-gnu/libacl.so.1 (0x00007f9c926e1000)
        libgpm.so.2 => /lib/x86_64-linux-gnu/libgpm.so.2 (0x00007f9c924db000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f9c924d3000)
        libpython3.8.so.1.0 => /lib/x86_64-linux-gnu/libpython3.8.so.1.0 (0x00007f9c91f7d000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f9c91f5a000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f9c91d68000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f9c92b98000)
        libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x00007f9c91cd8000)
        libvorbisfile.so.3 => /lib/x86_64-linux-gnu/libvorbisfile.so.3 (0x00007f9c91ccd000)
        libtdb.so.1 => /lib/x86_64-linux-gnu/libtdb.so.1 (0x00007f9c91cb1000)
        libltdl.so.7 => /lib/x86_64-linux-gnu/libltdl.so.7 (0x00007f9c91ca6000)
        libexpat.so.1 => /lib/x86_64-linux-gnu/libexpat.so.1 (0x00007f9c91c78000)
        libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007f9c91c5c000)
        libutil.so.1 => /lib/x86_64-linux-gnu/libutil.so.1 (0x00007f9c91c57000)
        libvorbis.so.0 => /lib/x86_64-linux-gnu/libvorbis.so.0 (0x00007f9c91c27000)
        libogg.so.0 => /lib/x86_64-linux-gnu/libogg.so.0 (0x00007f9c91c1a000)
```

ldconfig は通常 `/etc/ld.so.conf` を使用してブート時に実行されます（それ以外にもいつでも実行できます）。**ldconfig** には共有ライブラリを検索するためのディレクトリのリストがあります。**ldconfig** はルート権限で実行する必要があります。

リンカーは **ldconfig** によって構築されたデータベースを検索する以外に、PATH 変数のようにコロンで区切られた、ディレクトリのリストの環境変数 LD_LIBRARY_PATH で指定されたディレクトリを最初に検索します。たとえば次のようにできます。

```
$ LD_LIBRARY_PATH=$HOME/foo/lib ; foo [args]
$ LD_LIBRARY_PATH=$HOME/foo/lib   foo [args]
```

3.9 演習

デモ教材ビデオ

using_renice_demo.mp4

課題 3.1: ulimit でプロセスを制御する

以下を実行してください:

```
$ help ulimit
```

次に行く前に、/etc/security/limits.conf を読んでください。

1. **bash** コマンドを実行して（または、新しい端末を開いて）、新しいシェルを開始します。以降の変更は新たに開いたシェルのみに有効となります。オープンできるファイル数の制限の、現在値とハードとソフトリミットを見ます。
2. 制限値をハードリミット値に設定し、動くことを確認します。
3. ハードリミットを 2048 に設定し、動くことを確認します。
4. リミットを元の値に戻します。うまく動きましたか？

解 3.1

1.
```
$ bash
$ ulimit -n
```
```
1024
```
```
$ ulimit -S -n
```
```
1024
```
```
$ ulimit -H -n
```
```
4096
```

2.
```
$ ulimit -n hard
$ ulimit -n
```
```
4096
```

3.
```
$ ulimit -n 2048
$ ulimit -n
```
```
2048
```

4.
```
$ ulimit -n 4096
```
```
bash: ulimit: open files: cannot modify limit: Operation not permitted
```
```
$ ulimit -n
```
```
2048
```

もはやリミット値を元に戻すことは出来ません！

もし別のリミット、たとえばスタックサイズ（-s）を選んだとしたら、ハードリミットは 無制限 なので、もとに戻すことができます。

課題 3.2: System V の IPC の動作を確認する

System V IPC は、**UNIX** の初期からある、古めの **I**nter **P**rocess **C**ommunication です。それは 3 つの機構を持っています:

1. **Shared Memory Segments**
2. **Semaphores**
3. **Message Queues**

POSIX IPC を使いこれら 3 つの機構を実現する新しいプログラムもありますが、**System V IPC** を利用するアプリケーションも多数残っています。

以下のようにして、**System V IPC** の状況の概況をみます:

```
$ ipcs
```

```
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x01114703 0          root       600        1000       6
0x00000000 98305      coop       600        4194304    2          dest
0x00000000 196610     coop       600        4194304    2          dest
0x00000000 23068675   coop       700        1138176    2          dest
0x00000000 23101444   coop       600        393216     2          dest
0x00000000 23134213   coop       600        524288     2          dest
0x00000000 24051718   coop       600        393216     2          dest
0x00000000 23756807   coop       600        524288     2          dest
0x00000000 24018952   coop       600        67108864   2          dest
0x00000000 23363593   coop       700        95408      2          dest
0x00000000 1441811    coop       600        2097152    2          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0x00000000 98304      apache     600        1
0x00000000 131073     apache     600        1
0x00000000 163842     apache     600        1
0x00000000 196611     apache     600        1
0x00000000 229380     apache     600        1
```

現在実行中の共有メモリーセグメントのほとんど全てが、キー値 0 （IPC_PRIVATE）を持っています。これは、それらが親子関係にあるプロセスで共有されていることを意味します。さらに、ひとつのセグメントを除きアタッチ終了後に破棄されることを意味する (status 欄に dest が表示されている) マークがついています。

生成した共有セグメントに最後にアタッチされたプロセスに関する情報を集めることができます:

```
$ ipcs -p
```

```
------ Message Queues PIDs --------
msqid      owner      lspid      lrpid

------ Shared Memory Creator/Last-op PIDs --------
shmid      owner      cpid       lpid
0          root       1023       1023
98305      coop       2265       18780
```

```
196610     coop        2138       18775
23068675   coop        989        1663
23101444   coop        989        1663
23134213   coop        989        1663
24051718   coop        20573      1663
23756807   coop        10735      1663
24018952   coop        17875      1663
23363593   coop        989        1663
1441811    coop        2048       20573
```

この表示結果から cpid と lpid に相当するプロセスを検索するため、つぎを実行します：

```
$ ps aux |grep -e 20573 -e 2048
```

```
coop      2048  5.3  3.7 1922996 305660 ?      Rl   Oct27  77:07 /usr/bin/gnome-shell
coop     20573  1.9  1.7 807944 141688 ?       Sl   09:56   0:01 /usr/lib64/thunderbird/thunderbird
coop     20710  0.0  0.0 112652  2312 pts/0    S+   09:57   0:00 grep --color=auto -e 20573 -e 2048
```

thunderbird が、**gnome-shell** が作成した共有セグメントを利用していることがわかります。

これらのステップを実行することで、さまざまなリソースを誰が利用しているかがわかります。たとえば、**leaks**（どのプロセスも利用していない共有資源）があるかを見るためには：

```
$ ipcs
```

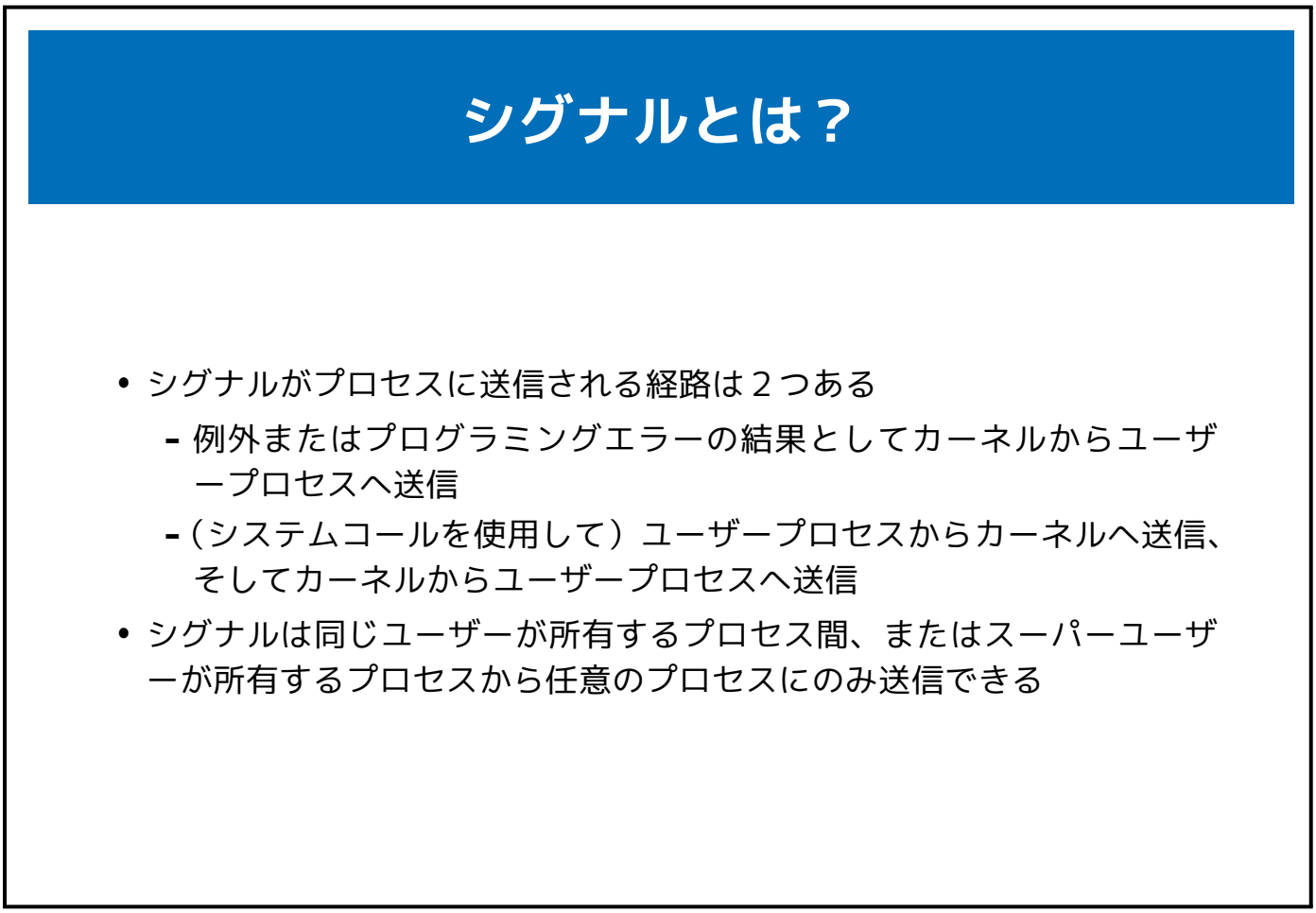
```
....
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
....
0x00000000 622601     coop       600        2097152    2          dest
0x0000001a 13303818   coop       666        8196       0
....
```

アタッチされていなく、また破棄ともマークされていない共有メモリーが表示されています。つまり、これらはこのあと何かのプロセスがアタッチしない限り、メモリーの無駄として残り続けるのです。

章 4

シグナル

4.1 シグナル



シグナルはプロセス間通信（**IPC**）の最も古い方法の 1 つであり、非同期 **asynchronous** イベント（または例外 **exceptions**）をプロセスに通知するために使用されます。

非同期とはシグナルを受信するプロセスが以下であることを意味します。

- イベントが発生することを予期していない
- イベントの受信を予期しているが、いつ発生するするかわからない

たとえばユーザーが実行中のプログラムを終了することを決定した場合、カーネルを介してプロセスにシグナルを送信しプロセスを中断して強制終了できます。

プロセスがシグナルを受信して行う処理は、プログラムに書かれた内容によって異なります。プログラムにコード化されている特定のアクションを実行して信号を処理することも、システムのデフォルトに従って応答することもできます。2 つのシグナル

- `SIGKILL` (#9)
- `SIGSTOP` (#19)

は処理はせず、常にプログラムを終了します。

4.2 シグナルの種類

シグナルの種類と利用可能なシグナル

- ハードウェアによって検出された例外（不正なメモリー参照など）
- プロセス早期終了やユーザーが発行した例外など
- `kill -l` は利用可能な全てのシグナルを表示

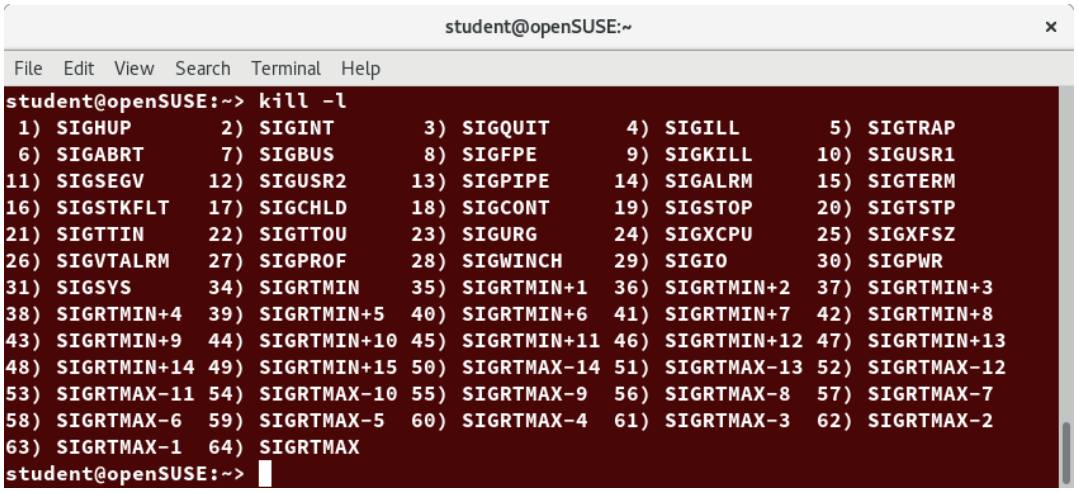


図 4.1: **利用可能なシグナル**

シグナルには多くの種類があり、カーネルから送信されたシグナルはイベント（または例外）のタイプを示します。一般的にはシグナルは次の２つをハンドルするために使われます。1) ハードウエアによって検出された例外（不正なメモリー参照など）、２）環境に起因する例外（ユーザーの端末操作による不完全なプロセスの停止など）

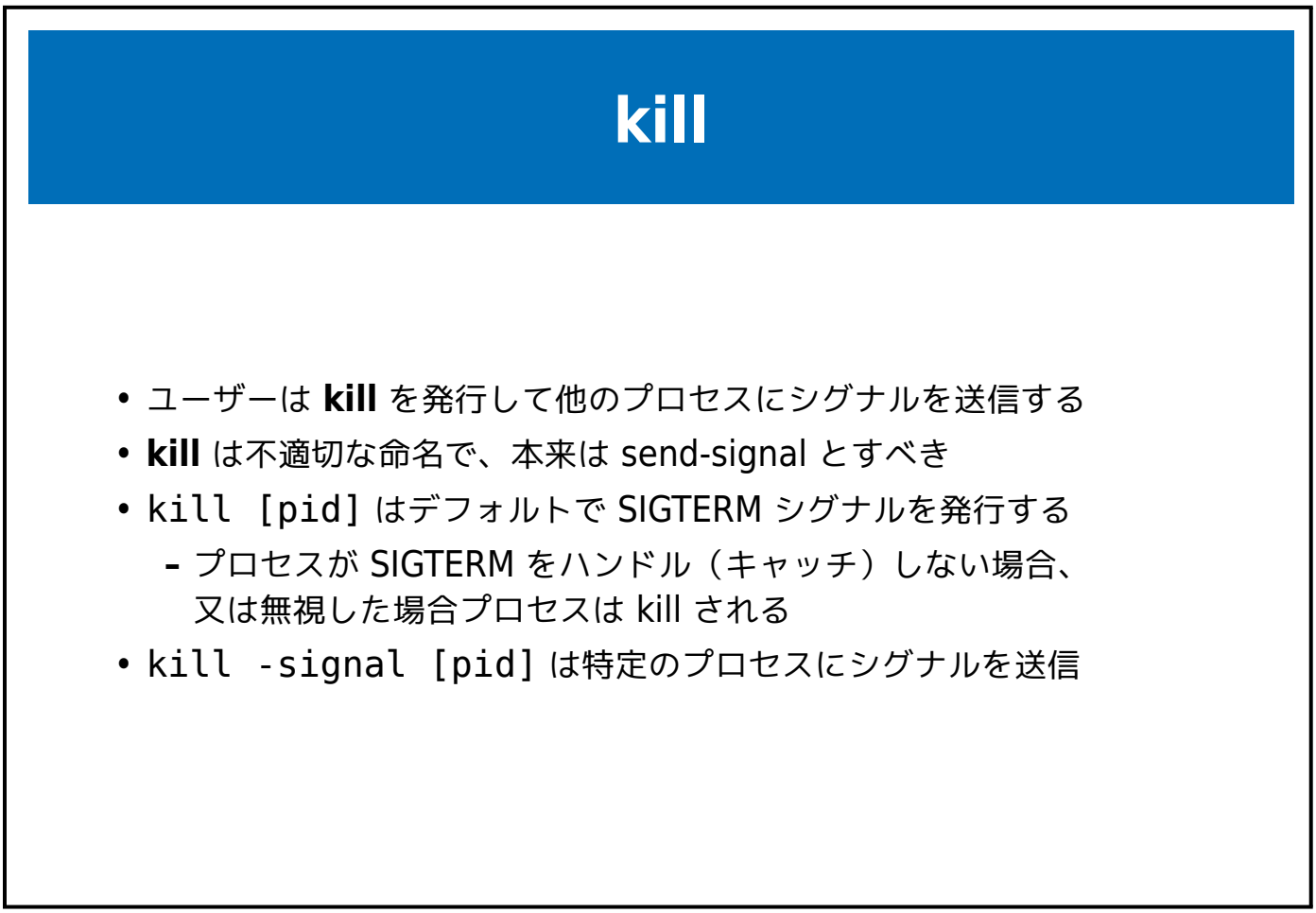
Linux のシグナルのリストとその番号を表示するには `kill -l` (これは文字の l で数字の１ではありません) を実行します。シグナルタイプはどのイベントがプロセスにシグナルを送信したか（カーネルから送信された場合）を示しています。

`SIGRTMIN` から後のシグナルはリアルタイムシグナル **real-time signals** と呼ばれ比較的最近追加されました。これらには事前に定義された目的はなく、いくつかの重要な点で通常のシグナルとは異なります。キューに入れることができ **FIFO**（First In First Out）の順序で処理されます。

シグナルの種類は（カーネルから送信されたときに）シグナルが送信される原因となったイベントを示しています。ユーザーはプロセスの 1 つに任意の種類のシグナルを明示的に送信できますが、シグナル番号やその種類が示す意味に関係なくプロセスが望む任意の方法で処理できます。

`man 7 signal` を入力すると、より多くのドキュメントが表示されます。

4.3 kill



プロセスは別のプロセスに直接シグナルを送信できないので、カーネルにシステムコールを実行してシグナルを送信要求する必要があります。（スーパーユーザーを含む）ユーザーは次のように **kill** を使用して他のプロセスにシグナルを送信できます。

kill という名前は適切な命名とはいえません。プロセスを終了させる（殺す）ために使用されることが多いは事実ですが、このコマンドの本来の機能はシグナルをプロセスに送信することです。従って **kill** という名前は実態を表していません。

デフォルトでは処理可能な終了シグナルの SIGTERM (#15) が送信されます。受信したプロセスはシグナルを受信（ハンドル）することも、プロセスが死ぬことを防ぐために無視することもできます。こうすることでプロセスに対して自分自身をクリーンアップするチャンスを与えることが出来るので望ましい使い方です。このシグナルが無視される場合はユーザーはプロセスを強制終了するために無視できない SIGKILL (#9) を送信できます。

以下のように **kill** 使ってシグナルを送信できます。（番号、名前のどちらかを指定）

```
$ kill 1234
$ kill -9 1234
$ kill -SIGTERM 1234
```

4.4 killall と pkill


```
$ killall bash
$ killall -9 bash
$ killall -SIGKILL bash
```

pkill

- 名前を使ってプロセスにシグナルを送信:
  `$ pkill [-signal] [options] [pattern]`
  - -P: ppid
  - -G: gid
  - -U: uid

kill と同様だが pid の代わりに名前を利用
rsyslog のコンフィグレーションファイルを再読込:

```
$ pkill -HUP rsyslogd
```

4.5 演習

課題 4.1: シグナルの優先順位と実行

ここで、さまざまなシグナルを処理できるシグナル ハンドラを含む **C** プログラムを提供します。ハンドラは全てのシステムコール（たとえば、I/O 実行中に発生するようなもの）を無視します。このファイルはダウンロード済の SOLUTIONS ファイルから signals.c として取り出すことができます。

signals.c

```
/*
 * Examining Signal Priorities.
 *
 * In the below, do not send or handle either of the signals SIGKILL
 * or SIGSTOP.
 *
 * Write a C program that includes a signal handler that can handle
 * any signal.  The handler should avoid making any system calls (such
 * as those that might occur doing I/O).
 *
 * The handler should simply store the sequence of signals as they
 * come in, and update a counter array for each signal that indicates
 * how many times the signal has been handled.
 *
 * The program should begin by suspending processing of all signals
 * (using sigprocmask().
 *
 * It should then install the new set of signal handlers (which can be
 * the same for all signals, registering them with the sigaction()
 * interface.
 *
 * The the program should send every possible signal to itself multiple
 * times, using the raise() function.
 *
 * Signal processing should be resumed, once again using sigprocmask).
 *
 * Before completing, the program should print out statistics
 * including:
 *
 *    The total number of times each signal was received.
 *
 *    The order in which the signals were received, noting each time the total
 *
 *    number of times that signal had been received up to that point.
 *
 * Note the following items:
 *
 *   If more than one of a given signal is raised while the process
 *   has blocked it, does the process receive it multiple times?
 *
 *   Are all signals received by the process, or are some handled
 *   before they reach it?
 *
 *   What order are the signals received in?
 *
 * If you are running KDE as your desktop environment, you may find
 * signal 32 blocked.  One signal, SIGCONT (18 on x86) may not get
 * through; can you figure out why?
 *
```

```
 * It appears that signals 32 and 33 can not be blocked and will cause
 * your program to fail.  Even though header files indicate
 * SIGRTMIN=32, the command kill -l indicates SIGRTMIN=34.
 *
 * Avoid sending these signals.
 *
 */

#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>

#define NUMSIGS 64

/* prototypes of locally-defined signal handlers */

void (sig_handler) (int);

int sig_count[NUMSIGS + 1];        /* counter for signals received */
volatile static int line;
volatile int signumbuf[6400], sigcountbuf[6400];

int main(int argc, char *argv[])
{
        sigset_t sigmask_new, sigmask_old;
        struct sigaction sigact, oldact;
        int signum, rc, i;

        /* block all possible signals */
        rc = sigfillset(&sigmask_new);
        rc = sigprocmask(SIG_SETMASK, &sigmask_new, &sigmask_old);

        /* Assign values to members of sigaction structures */
        memset(&sigact, 0, sizeof(struct sigaction));
        sigact.sa_handler = sig_handler;        /* we use a pointer to a handler */
        sigact.sa_flags = 0;        /* no flags */
        /* VERY IMPORTANT */
        sigact.sa_mask = sigmask_new;        /* block signals in the handler itself  */

        /*
         * Now, use sigaction to create references to local signal
         * handlers * and raise the signal to myself
         */

        printf
            ("\nInstalling signal handler and Raising signal for signal number:\n\n");
        for (signum = 1; signum <= NUMSIGS; signum++) {
                if (signum == SIGKILL || signum == SIGSTOP || signum == 32
                    || signum == 33) {
                        printf("  --");
                } else {
                        sigaction(signum, &sigact, &oldact);
                        /* send the signal 3 times! */
                        rc = raise(signum);
                        rc = raise(signum);
                        rc = raise(signum);
```

```
                        if (rc) {
                                printf("Failed on Signal %d\n", signum);
                        } else {
                                printf("%4d", signum);
                        }
                }
                if (signum % 16 == 0)
                        printf("\n");
        }
        fflush(stdout);

        /* restore original mask */
        rc = sigprocmask(SIG_SETMASK, &sigmask_old, NULL);

        printf("\nSignal  Number(Times Processed)\n");
        printf("-----------------------------------------\n");
        for (i = 1; i <= NUMSIGS; i++) {
                printf("%4d:%3d  ", i, sig_count[i]);
                if (i % 8 == 0)
                        printf("\n");
        }
        printf("\n");

        printf("\nHistory: Signal  Number(Count Processed)\n");
        printf("-----------------------------------------\n");
        for (i = 0; i < line; i++) {
                if (i % 8 == 0)
                        printf("\n");
                printf("%4d(%1d)", signumbuf[i], sigcountbuf[i]);
        }
        printf("\n");
        exit(EXIT_SUCCESS);
}

void sig_handler(int sig)
{
        sig_count[sig]++;
        signumbuf[line] = sig;
        sigcountbuf[line] = sig_count[sig];
        line++;
}
```

コンパイル後、次のように実行します:

```
$ gcc -o signals signals.c
$ ./signals
```

実行すると、プログラムは:

- シグナル SIGKILL や SIGSTOP を送信することはありません。これらはハンドルすることができず、また必ずプログラムを終了させてしまいます。
- シグナルの受信順序を記録するとともに、各シグナルを何回処理したかを示す配列を更新します。
- 全てのシグナルの処理をサスペンドしてから、新しいハンドラセットをインストールします。
- 全てのシグナルを複数回自分自身に送信します。そして、シグナルハンドリングのブロックを解除し待ち行列にあったハンドラを実行します。
- 以下を含む状態を出力します:

- 各シグナルを受信した回数。
- シグナルを受信した順序、各時間に受信したシグナル回数。

次のことに注目してください:

- プロセスがブロックしている間に、シグナルが **発生** したら、プロセスはそれを複数回 **受信** するのだろうか。**リアルタイム** のシグナルと通常のシグナルでは動作が異なるか。
- プロセスは全てのシグナルを受信するか。それとも、いくつかのシグナルは受信する前にハンドルされるのか。
- シグナルの受信順序は、どうなるのか？

あるシグナル SIGCONT （**x86** では 18）は処理されません。なぜかわかりますか？

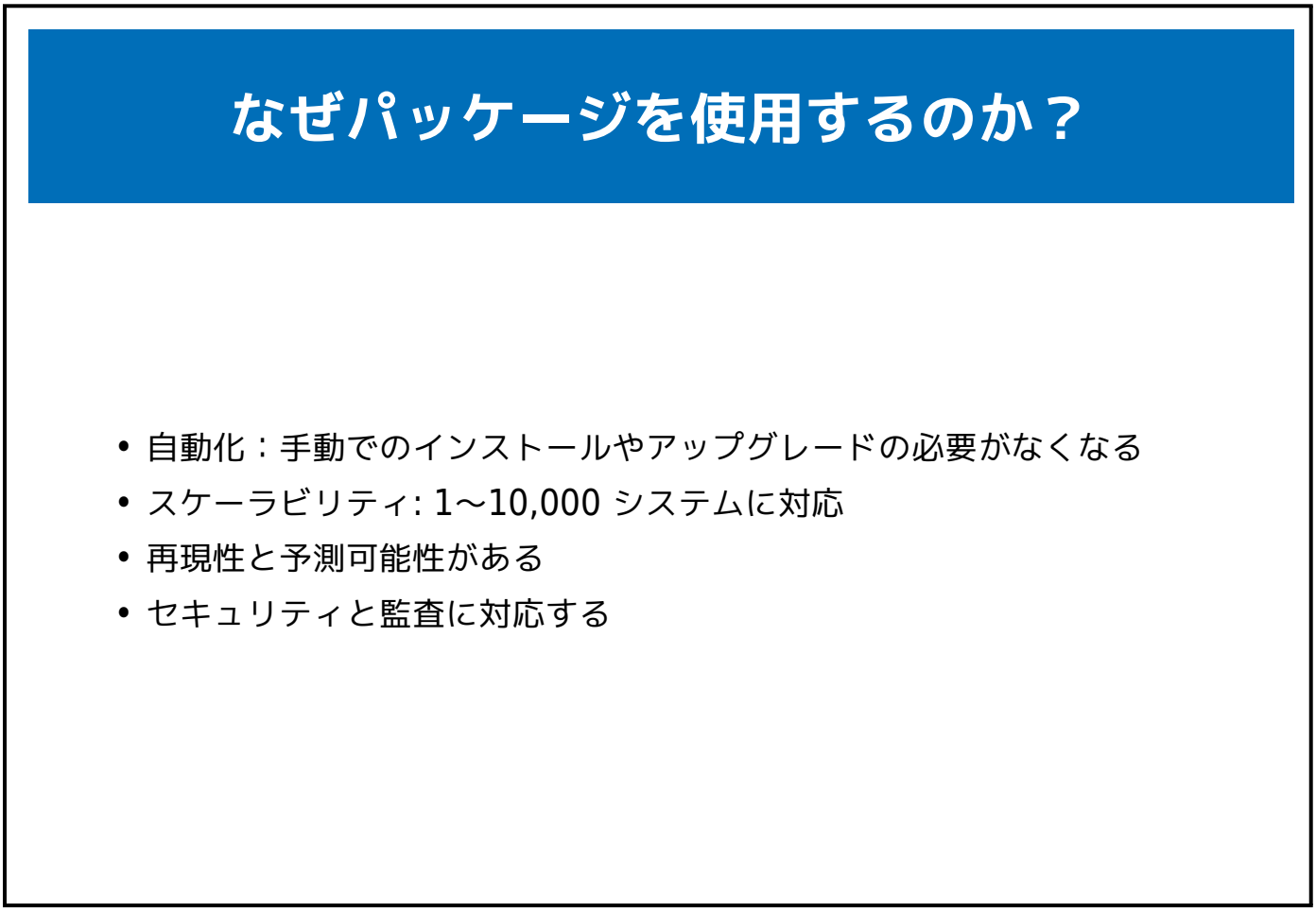
注目

いくつかの **Linux** ディストリビューションでは、シグナル 32 と 33 はブロックすることができず、プログラムは失敗します。システムヘッダーファイルには SIGRTMIN=32 となっていますが、`kill -l` コマンドでは SIGRTMIN=34 となっています。**POSIX** によれば、シグナルは名称で指定するべきで、数値での指定は非推奨とあります。これによりインプリメント非依存になります。これらのシグナルを送信することは避けるべきです。

章 5

パッケージ管理システム

5.1 なぜパッケージを使用するのか？



ソフトウェアパッケージ管理システムは **Linux** がエンタープライズ IT 環境にもたらした最大の進歩の 1 つだと言われています。システム管理者は、自動化された汎用的で信頼性の高い方法でファイルとメタデータの記録を把握することにより、パッケージ管理システムを使用して、個々のシステムを手動で操作することなく数千のシステムにインストールプロセスを拡張できます。

5.2 ソフトウェアパッケージの概念

ソフトウェアパッケージの概念

- 関連ソフトウェアを１つのパッケージ（アーカイブ）に収集して圧縮、先に別パッケージをインストールしておくことを前提にする場合もある
- 簡単なソフトウェアのインストール、または削除が可能
- 内部データベースを介してファイルの整合性を検証
- パッケージの出所証明
- アップグレードが容易
- 論理的な特徴でパッケージをグループ化
- パッケージ間の依存関係を管理

パッケージ管理システム はシステム管理者がソフトウェアパッケージのインストール、アップグレード、構成、および削除を既知で汎用的な方法で自動化できるツールを提供します。

特定のパッケージには、実行可能ファイル、データファイル、ドキュメント、インストールスクリプト、および構成ファイルが含まれる場合があります。またバージョン番号、チェックサム、ベンダー情報、依存関係、説明などの **メタデータ属性** も含まれます。

すべての情報は、インストール時に内部データベースにローカルに保存されます。これによりバージョンと更新情報が簡単に照会できます。

5.3 パッケージの種類

パッケージの種類

- **バイナリパッケージ** には実行可能ファイルやライブラリなどデプロイの準備ができているファイルが含まれる、バイナリーパッケージはアーキテクチャ依存
- **ソースパッケージ** はバイナリパッケージの生成に使用される、ソースパッケージからバイナリパッケージを常に再構築できる必要がある、ソースパッケージは複数のアーキテクチャに対応
- **アーキテクチャ非依存** パッケージにはドキュメントや構成情報と共に、インタプリターを使って動かすファイル／スクリプトが含まれる
- **メタパッケージ** はデスクトップ環境やオフィススイーツのような比較的大規模なサブシステムのインストールに必要な、関連パッケージのグループが含まれる

通常システム管理者は、バイナリーパッケージを扱います。32 ビットプログラムを実行できる 64 ビットシステムの場合、ひとつのプログラムに対して 2 つのバイナリパッケージがインストールされていることもあります。片方の名前には **x86_64** または **amd64**、もう片方の名前には **i386** または **i686** が含まれます。

ソースパッケージは、バイナリパッケージの作成に使用されたソースコードの変更履歴追跡に役立ちます。通常デフォルトではシステムにインストールされていませんが、ベンダーからいつでも入手できます。

必ずソースパッケージからバイナリパッケージが再構築できなければなりません。たとえば **RPM** ベースのシステムでは、以下を実行して **p7zip** バイナリパッケージを再構築できます。

```
# rpmbuild --rebuild -rb p7zip-16.02-16.el8.src.rpm
```

結果は `/root/rpmbuild` に置かれます。

```
# root/rpmbuild>find . -name "*rpm"
```

```
./RPMS/x86_64/p7zip-plugins-16.02-16.el8.x86_64.rpm
./RPMS/x86_64/p7zip-debugsource-16.02-16.el8.x86_64.rpm
./RPMS/x86_64/p7zip-plugins-debuginfo-16.02-16.el8.x86_64.rpm
./RPMS/x86_64/p7zip-16.02-16.el8.x86_64.rpm
./RPMS/noarch/p7zip-doc-16.02-16.el8.noarch.rpm
```

結果が置かれる場所は **Linux** ディストリビューションとバージョンに依存します。

5.4 利用可能なパッケージ管理システム

利用可能なパッケージ管理システム

図 5.1: **RPM と APT**

2 つの非常にポピュラーなパッケージ管理システム

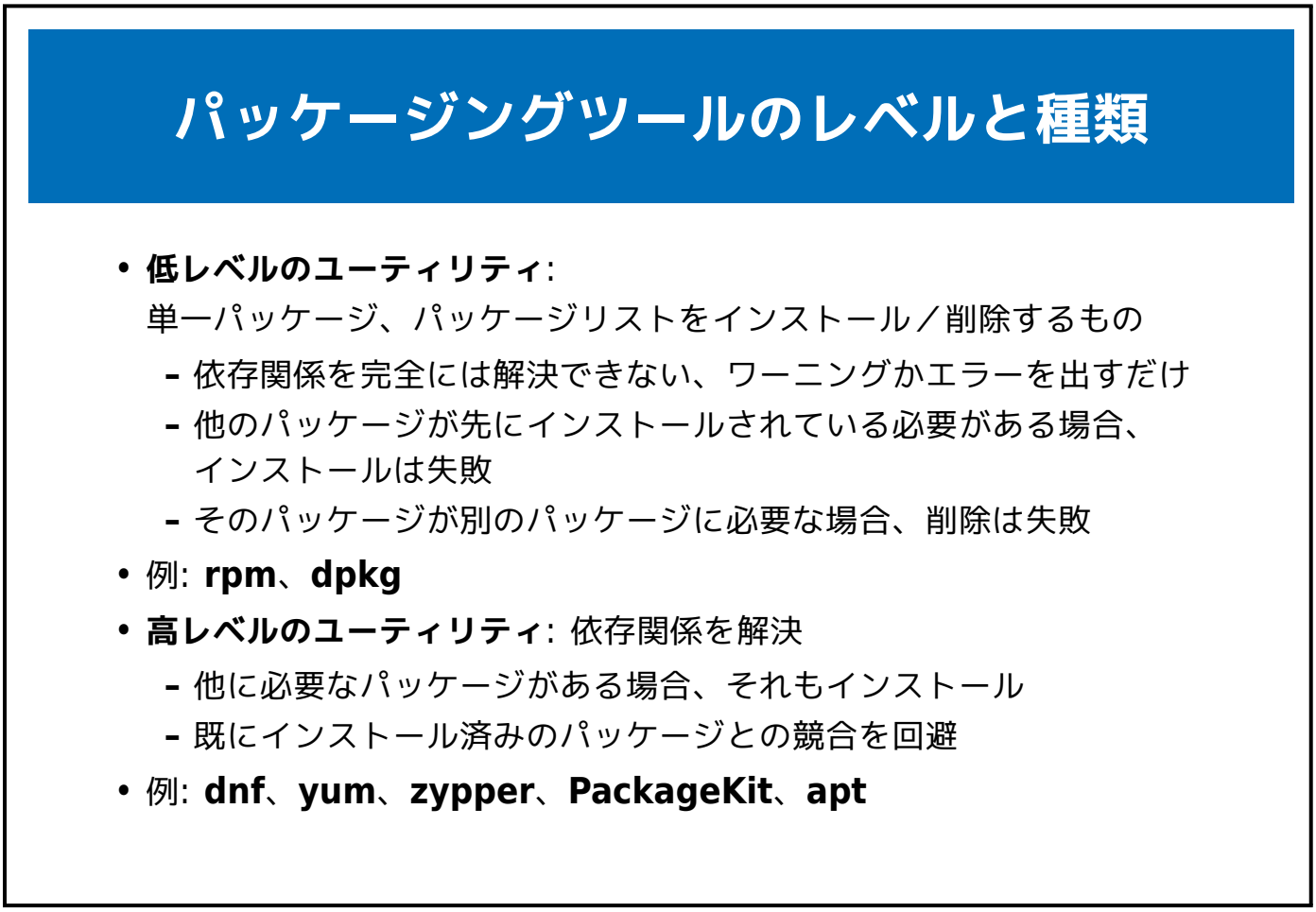
- **RPM** (**Red Hat Package Manager**)
  このシステムは、**Red Hat Enterprise Linux**、**Fedora**、**CentOS** など **Red Hat** 派生のすべてのディストリビューション、および **SUSE** とその関連コミュニティ **openSUSE** ディストリビューションで使用
- **dpkg** (**Debian Package**)
  このシステムは **Debian**、**Ubuntu**、**Linux Mint** など **Debian** から派生したすべてのディストリビューションで使用

他にも **Gentoo** が使用する **portage/emerge**、**Arch** が使用する **pacman**、および **組み込み Linux** システムと **Android** が使用する特殊なパッケージ管理システムなどがあります。

もう 1 つの古いシステムとして、実際の管理や削除の戦略なしにパッケージを **tarballs** として提供する方法があります。このアプローチは、最も古い **Linux** ディストリビューションの 1 つである **Slackware** でまだ使われています。

しかしほとんどの場合 **RPM** または **dpkg** のいずれかを使いますので、このコースではこれらを学んでいきます。

5.5 パッケージングツールのレベルと種類



パッケージングシステムには、２つのレベルがある

1. **低レベルのユーティリティ**: これは、単一のパッケージまたはパッケージのリストをインストールするか削除するだけのものであり、各パッケージには個別に具体的な名前が付けられています。依存関係は完全には処理されず警告のみが行われるかエラーになります。

   - 他のパッケージが先にインストールされている必要がある場合、インストールは失敗
   - そのパッケージが別のパッケージに必要な場合、削除は失敗

   rpm と **dpkg** ユーティリティがパッケージングシステムに対してこの役割を果たします。

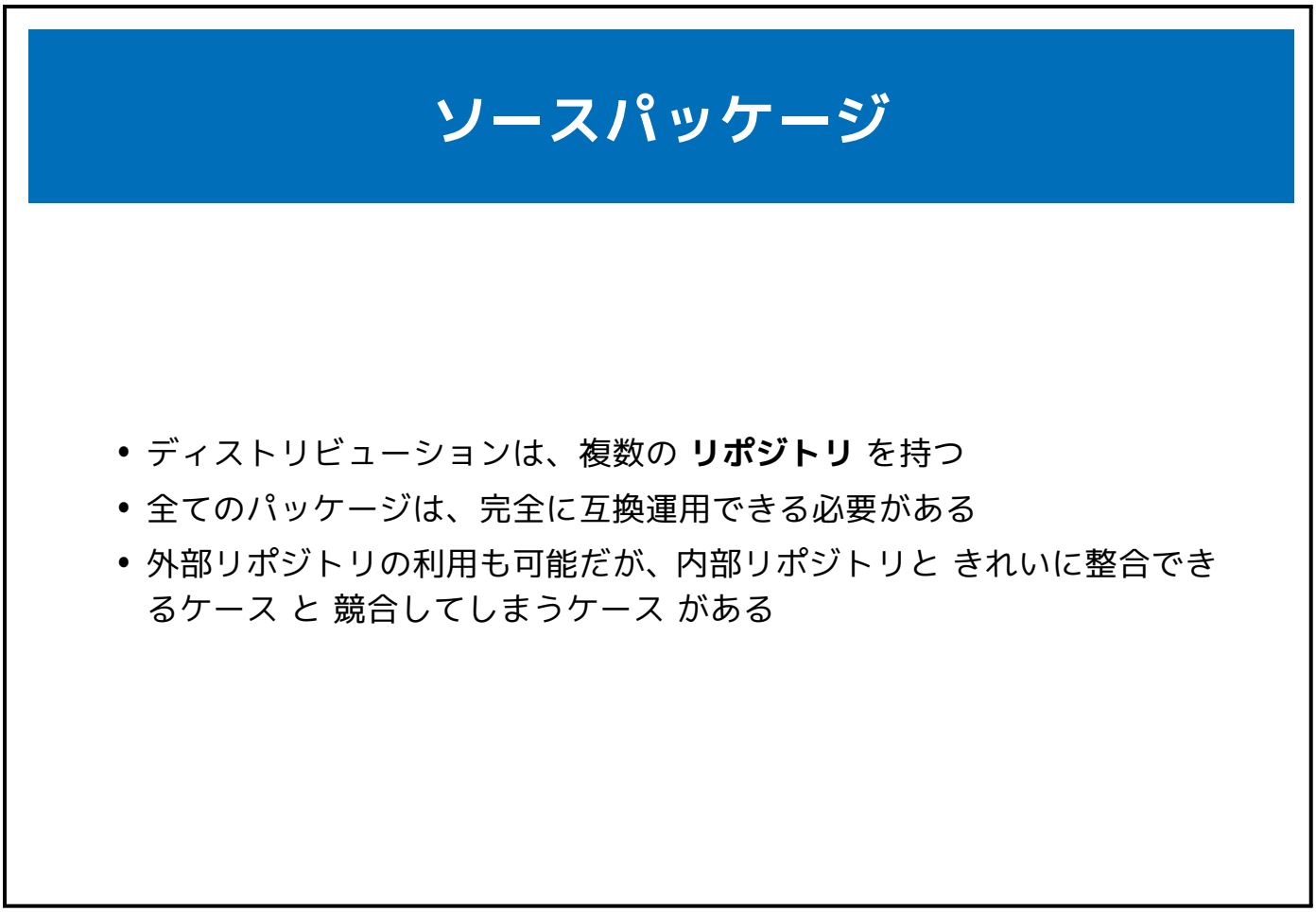
2. **高レベルのユーティリティ**: これは依存関係を解決

   - 他に必要なパッケージないしパッケージグループがある場合、依存関係の解決の為にそれらを先にインストール
   - インストールされている別のパッケージがパッケージの削除を妨げる場合、管理者は影響を受けるソフトウェアをすべて中止するか削除するかを選択できます。

   dnf と **zypper** ユーティリティ（と、以前の **yum**）が **rpm** システムの依存関係を解決し、**apt-get** と **apt-cache** やその他のユーティリティが **dpkg** システムの依存関係を解決します。

このコースでは、コマンドラインインターフェイスを使ったパッケージングシステムについてのみ説明します。各 **Linux** ディストリビューションで使用されるグラフィカルなフロントエンドは便利ですが、ここでは特定のインターフェイスに縛られない柔軟性を持ちたいと考えています。

5.6 ソースパッケージ



すべてのディストリビューションには、システムユーティリティがソフトウェアを取得し新しいバージョンに更新するための、１つ以上のパッケージ **リポジトリ** があります。ディストリビューションは、リポジトリ内のすべてのパッケージが問題なく同時動作することを確認します。

また、ディストリビューションの標準サポートリストに、いつでも外部リポジトリを追加することができます。これらはディストリビューションと密接に連携しているので、追加しても重要な問題が発生することはほとんどありません。例としてはバージョン依存のリポジトリ **EPEL** (**E**xtra **P**ackages for **E**nterprise **L**inux) セットがあります。このソースは **Fedora** であり、メンテナ達も **Red Hat** と親密なため、**RHEL** に適合するように作られています。

ただし、一部の外部リポジトリにはあまり構築も保守もされていないものがあります。たとえば、メインリポジトリでパッケージが更新されても外部パッケージの依存パッケージが更新されない可能性があり、**依存関係地獄** につながる可能性があるのです。

5.7 ソフトウェアパッケージの作成

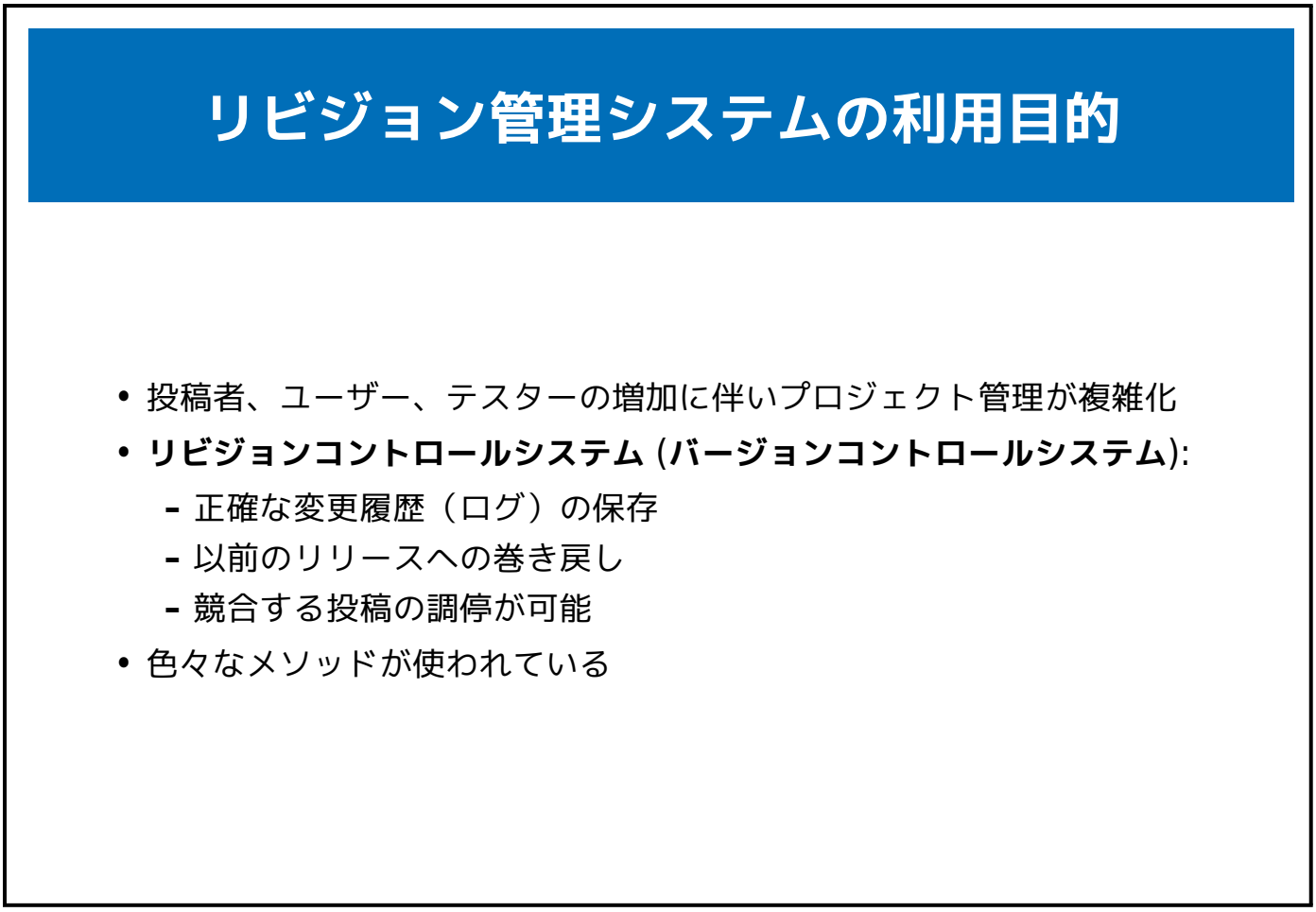
ソフトウェアパッケージの作成

- カスタムソフトウエアのパッケージインストールを簡単にする
- インストール中に、パッケージに追加タスクの実行を許す
  - 必要なシンボリックリンクの作成
  - 必要に応じたディレクトリの作成
  - パーミッション設定
  - スクリプト化できるものすべて
- 色々なフォーマットが利用できる
  - **RPM** は **Red Hat** と **SUSE** ベースのシステムが利用
  - **DEB** は **Debian** for **Debian** ベースのシステムが利用
  - **TGZ** は **Slackware** が利用
  - **APK** は **Android** が利用

独自のカスタムソフトウェアパッケージを構築すると、独自のソフトウェアを簡単に配布およびインストールできます。**Linux** のほとんどすべてのバージョンにこれを行うためのメカニズムがあります。

独自のパッケージを作成すると、ソフトウェアの内容とインストール方法を正確に制御できます。新しいソフトウェアのインストールや、古いソフトウェアの削除に必要な全タスクを行うスクリプトを実行するために、パッケージを作成することができます。

5.8 リビジョン管理システム



ソフトウェアプロジェクトでは、プロジェクトサイズが大きくなったり貢献する開発者の数が増えたりするにつれ、管理がより複雑になります。

更新を整理し連携を促進するために、ソース管理にはさまざまなスキームが利用できます。このようなプログラムの標準機能には、変更の正確な履歴やログの保持、以前のリリースへのバックアップ、複数の開発者からの競合する可能性のある更新の調整などが含まれます。

ソースコントロールシステム (または **リビジョンコントロールシステム** とも一般に呼ばれる) は、共同開発を調整する役割を果たします。

5.9 利用可能なソースコントロールシステム

利用可能なソースコントロールシステム

Table 5.1: **利用可能なソースコントロールシステム**

製品	URL
RCS	https://www.gnu.org/software/rcs/
CVS	https://www.nongnu.org/cvs
Subversion	https://subversion.apache.org
git	https://www.kernel.org/pub/software/scm/git/
GNU Arch	https://www.gnu.org/software/gnu-arch
Monotone	https://www.monotone.ca
Mercurial	https://www.mercurial-scm.org

利用可能な製品にはプロプライエタリ、オープンそれぞれに多くの選択肢があります。上のリストは **GPL** ライセンスでリリースされた製品の代表例です。

ここでは、Linux カーネル開発コミュニティから生まれ広く使用されている製品である **git** のみに焦点を当てます。**git** はあっという間にオープンソースプロジェクトの主流となったもので、クローズドソース環境でもよく使用されています。

5.10 Linux カーネルと git の誕生

Linux カーネルと git の誕生

- **Linux** カーネル開発には、特別なシステム要件がある
  - 開発者が全世界に幅広く分散
  - 数千人規模の開発者
  - **GPL** ライセンスの下で完全に公開
- **git** はこれらのニーズを満たし **Linux** の効率的な成長を促した

Linux カーネル開発システムには、文字どおり何千人もの開発者が関与し、世界中に広く配布されるという点で特別なニーズがあります。さらにそれはすべて **GPL** ライセンスの下で公開されています。

Linux には長い間実質的なソースリビジョン管理システムはありませんでした。そこで主要なカーネル開発者は使用制限付きライセンスを付与した商用プロジェクトの **BitKeeper** (https://www.bitkeeper.com 参照) を **Linux** カーネル開発に使用しました。

しかし、2005 年春のライセンス制限に関する公開論争の結果、**Linux** カーネルの開発に **BitKeeper** を無料で使用できなくなりました。

そこで求められたのが **git** の開発です。開発者は Linus Torvalds でした。**git** のソースコードは https://www.kernel.org/pub/software/scm/git/ のインデックスから入手でき、完全なドキュメントも https://www.kernel.org/pub/software/scm/git/docs/ から入手できます。

git の仕組み

- 伝統的なバージョンコントロールシステムとは違う
- 管理の基本単位は **ファイル** ではない
- **オブジェクトデータベース** には:
  - **Blobs（ブロブ）**
  - **Trees（ツリー）**
  - **Commits（コミット）**
- **ディレクトリキャッシュ** はディレクトリツリーの状態を取得
- グラフィカルインターフェースもある
- **github** などのサイトに数百万の **git** リポジトリ

技術的にみて **git** は普通の意味でのソース管理システムでは **ありません**。管理の基本単位は、ファイルではありません。**オブジェクト**データベースとディレクトリキャッシュという２つの重要なデータ構造を持っています。

オブジェクトデータベースには３種類のオブジェクトが含まれます。

- **Blobs（ブロブ）：** ファイルの内容を含むバイナリデータの塊
- **Trees（ツリー）：** ファイル名と属性を含むブロブの集合。ディレクトリ構造を提供
- **Commits（コミット）：** ツリーのスナップショットを記述するチェンジセット

ディレクトリキャッシュは、ディレクトリツリーの状態をファイルに保存します。

ファイルをベースにしたシステムから管理システムを解放することにより、多くのファイルを含むチェンジセットを適切に処理できるようになりました。

git は早いペースで開発されており、グラフィカルインターフェイスも迅速に構築されています。たとえば https://www.kernel.org/git/ を参照してください。特定の変更とソースのツリーを簡単に参照できます。

GitHub https://www.github.com は現在文字どおり何百万もの **git** リポジトリを公開／非公開にかかわらずホストしています。**git** を有効に使用する方法に関しては、見やすい記事、書籍、オンラインチュートリアルなどが多数あります。

5.11 演習

デモ教材ビデオ

using_package_management_demo.mp4

課題 5.1: git によるバージョン コントロール

git がインストール済であることを確認する

おそらく **git** はインストールされていると思います。`which git` コマンドでインストールされているかどうかがわかります。もしインストールされていなければ、ソースを入手、コンパイル、インストールする手もありますが、通常適切なコンパイル済のバイナリパッケージをインストールするのが容易です。正確なパッケージ名称はいろいろありますが、ディストリビューションにより以下のどれかが使えるはずです。

```
$ sudo apt-get install git*  # Debian /Ubuntu
$ sudo zypper install git*   # openSUSE
$ sudo dnf install git*      # Fedora / RHEL 8 / CentOS 8
$ sudo yum install git*      # RHEL 7 / CentOS 7
```

使用しているディストリビューションによります。

どのように **git が使えるか** の感触を確かめて、その使いやすさを実感しましょう。ローカルのプロジェクトを作成します。

1. 作業用ディレクトリを作成し **git** を使うための準備をします:

   ```
   $ mkdir git-test
   $ cd git-test
   $ git init
   ```

2. プロジェクトの初期化とは、その下にバージョンコントロールに必要な情報を格納する `.git` ディレクトリを作成することです; プロジェクトの主ディレクトリには触りません。このディレクトリの初期値は以下のような内容です:

   ```
   $ ls -l .git
   ```

   ```
   total 40
   drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 branches/
   -rw-rw-r-- 1 coop coop   92 Dec 30 13:59 config
   -rw-rw-r-- 1 coop coop   58 Dec 30 13:59 description
   -rw-rw-r-- 1 coop coop   23 Dec 30 13:59 HEAD
   drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 hooks/
   drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 info/
   drwxrwxr-x 4 coop coop 4096 Dec 30 13:59 objects/
   drwxrwxr-x 4 coop coop 4096 Dec 30 13:59 refs/
   ```

   あとで、このディレクトリとサブディレクトリの内容について説明します; 開始時点ではほとんど空の状態です。

3. ファイルを作成し、プロジェクトに追加します:

   ```
   $ echo some junk > somejunkfile
   $ git add somejunkfile
   ```

4. 以下のようにしてプロジェクトの状態を見ることができます:

   ```
   $ git status
   ```

```
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

      new file: somejunkfile

```

これは、ファイルが **staged** 状態にあり、いまだ **commit** されていないことを意味します。

5. プロジェクトの責任者を **git** に登録します:

```
$ git config user.name "Another Genius"
$ git config user.email "b_genius@linux.com"
```

グローバルな構成ファイルで指定しない限り、全ての新プロジェクトで指定する必要があります。

6. それでは、ファイルを修正して違いを見ましょう:

```
$ echo another line >> somejunkfile
$ git diff
```

```
diff --git a/somejunkfile b/somejunkfile
index 9638122..6023331 100644
--- a/somejunkfile
+++ b/somejunkfile
@@ -1 +1,2 @@
 some junk
+another line
```

7. 変更をレポジトリにコミットします:

```
$  git commit -m "My initial commit"
```

```
Created initial commit eafad66: My initial commit
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 somejunkfile
```

-mオプションで識別用のメッセージを指定しなかった場合、エディタが起動しメッセージを入力できるようになります。コミットメッセージの入力は **絶対に** する必要があり、しないとコミットは拒否されます。エディタはEDITOR 環境変数で指定したものが使われますが、GIT_EDITOR で上書きすることができます。

8. 履歴を見ることができます:

```
$ git log
```

```
commit eafad66304ebbcd6acfe69843d246de3d8f6b9cc
Author: A Genius <a_genius@linux.com>
Date:   Wed Dec 30 11:07:19 2009 -0600

    My initial commit
```

ここで、いくつかの情報が表示されています。長い 16 進数の文字列は **コミット番号** です; それは 160 ビット、40 桁のユニークな識別子です。**git** はファイル名称ではなくこの面倒な情報を利用します。

9. これで、`git add` を使って、自由にファイルの修正、新規ファイルの追加ができます。しかし、新たに`git commit` するまではステージングの状態です。

10. 入門的な内容を実施しただけですが、これで終了します。

章 6

RPM

6.1 RPM (Red Hat Package Manager)

RPM を使用する利点

- ソフトウエアパッケージの管理
  - タスク、ないしサブシステムに関連したファイルを集約したもの
  - プログラム、データー、ドキュメント、構成情報が含まれる
  - 依存関係に関する情報が含まれることもある
  - パッケージの検索、取得は行わない
- 管理者に対するアドバンテージ
  - パッケージのインストールとアンインストール（消去）が容易になる
  - パッケージが正常にインストールされたかの検証が容易
  - パッケージのインストールに **FTP** や **HTTP** が利用できる
- 開発者に対するアドバンテージ
  - オリジナルの’純正’ソースを利用できる
  - 異なるアーキテクチャ向けの **RPM** を生成できる

RPM は **Red Hat** が開発したパッケージ管理ユーティリティです。(名前はもともと **R**edhat **P**ackage **M**anager を表していました。) 特定のタスクまたはサブシステムに関連するすべてのファイルは、1 つのファイルにパッケージ化されます。このファイルには、ファイルをインストールおよびアンインストールする方法とその場所に関する情報も含まれています。開発者がプログラムの新しいバージョンを作成すると、通常新しい **RPM** パッケージがリリースされます。これらのファイルは、他の **Linux** ディストリビューションでは使用できない可能性があることに注意してください。

RPM によってシステム管理者はソフトウェアパッケージの管理が容易になります。特定のファイルがどのパッケージのものであるか、どのバージョンのパッケージがインストールされているか、それが正しくインストールされたかどうかを簡単に判断することができます。またパッケージをすべて削除してディスク領域を解放することも簡単です。**RPM** はドキュメントファイルをパッケージの他の部分と区別し、システムにドキュメントをインストールするかどうか選択できるようにしています。

RPM によりソフトウェア開発者の仕事が楽になります。多くの場合、開発者は別のオペレーティングシステム上で正しくコンパイルして実行できるようにコードを変更する必要があります。しかし **RPM** を使用すると、構築する人は **Linux** での構築に必要な変更を元のソースとは別に持つことができます。これにより、構築関係の変更がすべて 1 か所に集約され新しいバージョンのコードの組み込みが容易になります。また、さまざまなアーキテクチャ向けの **Linux** バージョン構築も容易になります。

ネットワーク経由のインストレーション

特定の URL が明示的に示されない限り **rpm** はネットワークからパッケージを取得しません。絶対または相対パスを使ってローカルマシンからインストールします。

しかし **HTTP** や非推奨となっている **FTP** プロトコルを使って離れた場所から直接インストールすることも可能です。

6.2 パッケージファイル名

パッケージファイル名

- **RPM** の標準バイナリパッケージの命名形式:

  `<name>-<version>-<release>.<distro>.<architecture>.rpm`
  `sed-4.5-2.el8.x86_64.rpm`
- **RPM** の標準ソースパッケージの命名形式:

  `<name>-<version>-<release>.<distro>.src.rpm`
  `sed-4.5-2.el8.src.rpm`

RPM 標準 (http://www.rpm.org/) に記載されているように **RPM** パッケージファイル名のフィールドには、特別な意味を持たせています。

RPM で動作する **dnf**、**yum**、**zypper** のときに詳しく説明しますが、特定のインストールでは、複数の異なるパッケージリポジトリを使用する可能性があるため、`distro` フィールドはパッケージの作成元のリポジトリを指定することに留意してください。

6.3 RPM データベースと補助プログラム

/var/lib/rpm は、**Berkeley DB** ハッシュファイル形式で **RPM** データベースファイルを保持する、デフォルトのシステムディレクトリです。データベースファイルを手動で変更しないでください。更新は **rpm** プログラムでのみ行う必要があります。

代替データベースディレクトリは、**rpm** プログラムの --dbpath オプションで指定できます。たとえば、別のシステムからコピーされた **RPM** データベースを調べるためにこれを使うことができます。

--rebuilddb オプションを使用すると、インストールされたパッケージヘッダーからデータベースインデックスを再構築できます。これはどちらかというと修復目的のものであり、ゼロからの再構築を意図したものではありません。

RPM で使用される補助プログラムとスクリプトは /usr/lib/rpm にあります。かなりの数があります。たとえば **RHEL 8** システムの場合：

```
$ ls /usr/lib/rpm | wc -l
74
```

ここで使っている **wc** は、出力の行数を表示するものです。

rpmrc ファイルを作成し、**rpm** のデフォルトに指定することができます。デフォルトでは **rpm** は以下の順番で探します。

1. /usr/lib/rpm/rpmrc
2. /etc/rpmrc
3. ~/.rpmrc

これらのファイルをすべて読むことに注意してください。**rpm** はパッケージを検出してもすぐに停止しません。

--rcfile オプションを使用して他の rpmrc ファイルを指定することもできます。

6.4 クエリ

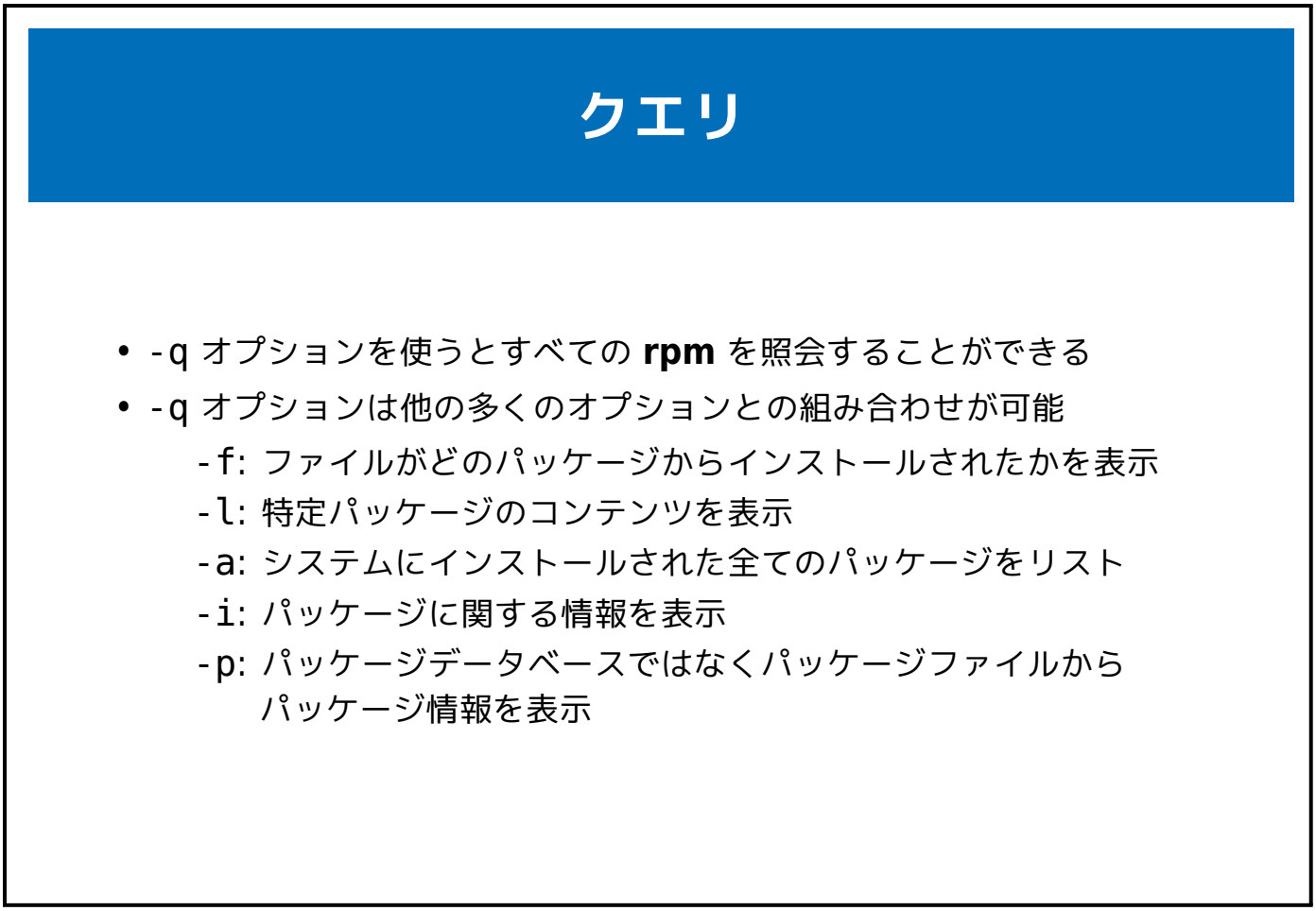


Table 6.1: **rpm 照会コマンドの例**

どのバージョンのパッケージがインストールされているか？	`rpm -q bash`
ファイルがどのパッケージから来たものか？	`rpm -qf /bin/bash`
パッケージによってどのファイルがインストールされたか？	`rpm -ql bash`
パッケージに関する情報を表示	`rpm -qi bash`
パッケージデータベースではなくパッケージファイルからパッケージ情報を表示	`rpm -qip foo-1.0.0.1.noarch.rpm`
システムにインストールされているすべてのパッケージをリスト	`rpm -qa`

その他の便利なオプションとして `--requires` と `--whatprovides` があります。

`--requires` オプションはパッケージの前提条件のリストを返します。`--whatprovides` オプションは、インストールされたどのパッケージが特定の必須パッケージを提供しているかどうかを表示します。

```
$ rpm -q  --requires bash
$ rpm -qp --requires foo-1.0.0-1.noarch.rpm
$ rpm -q  --whatprovides libc.so.6
```

6.5 パッケージの検証

パッケージの検証

- パッケージの検証には rpm -V コマンドを利用する
- 全てが正常な場合は、何も出力されない
  - S: ファイルサイズが異なる
  - M: ファイルのパーミッションや種類が異なる
  - 5: MD5 チェックサムが異なる
  - D: デバイスのメジャー/マイナー番号の不一致
  - L: シンボリックリンクパスの不一致
  - U: ユーザーの所有権が異なる
  - G: グループ所有権が異なる
  - T: 修正時刻が異なる
- システム上のすべてのパッケージを確認するには

```
$ sudo rpm -Va
```

rpm の -V オプションを使用して、特定パッケージのファイルがシステムの RPM データベースと一致しているかを確認できます。rpm -Va を使ってシステム上のすべてのパッケージを確認することができます。

問題がある場合にのみ出力が表示されます。出力では各文字は、ファイルの属性をデータベースに記録された属性と比較した結果を示しています。単体の「.」（ピリオド）はテストを通過したことを、単体の「?」（疑問符）はテストが実行できなかったことを意味しています（たとえば、ファイルのパーミッションが読み取り禁止だった場合など）。それ以外の文字は対応する --verify テストの失敗を意味しています。

すべてが正常な場合、何も出力されません。

```
$ rpm -V bash
```

ファイルのサイズ、チェックサム、および修正時刻が変更されたことを出力する例です。

```
$ rpm -V logrotate
```

```
S.5....T.  c /etc/logrotate.conf
```

ファイルが欠落していることを出力する例です。

```
$ sudo mv /sbin/logrotate /sbin/logrotate_KEEP
$ rpm -V logrotate
```

```
S.5....T.  c /etc/logrotate.conf
missing     /usr/sbin/logrotate
```

6.6 パッケージのインストールと削除

パッケージのインストール

- **RPM** はパッケージのインストールに関わる色々なタスクを実行する
  - 依存関係のチェック
  - 競合のチェック
  - インストール前に要求されるコマンドの実行
  - 構成ファイルの知的な処理
  - パッケージからファイルを解凍し、正しい属性でインストール
  - インストール後に必要なコマンドの実行
  - システムの **RPM** データベースの更新
- パッケージのインストールは `rpm -i`

```
$ sudo rpm -ivh bash-4.4.19-12.el8_0.x86_64
```

`-i` はインストール、`-v` は詳細表示、`-h` は進捗に伴ってハッシュマークを表示させるオプション指定

一部のパッケージは、他のパッケージがインストールされていないと適切に動作しないため、依存性のチェックが必要です。

インストール済のパッケージを上書きインストールしようとした場合や、新しいバージョンの上に古いバージョンをインストールする場合には、競合が発生します。

パッケージを作成する開発者は、インストールの前後に特定のタスクを実行するように指定できます。

構成ファイルをインストールするとき、ファイルが存在し以前のバージョンのパッケージがインストールされてから変更された **RPM** はサフィックス `.rpmsave` を使って古いバージョンを保存します。これにより、古い構成ファイルに加えた変更を新しいバージョンのファイルに統合できます。この機能を使うには RPM パッケージが適切に作成されている必要があります。

RPM は適切な場所にファイルをインストールすることに加えて、パーミッション、所有権、変更（ビルド）時刻などの属性も設定します。

RPM はパッケージをインストールするたびにシステムデータベース内の情報を更新します。この情報は、競合をチェックに使用されます。

パッケージのアンインストール

- 通常アンインストールが成功した場合は、何も出力されません

```
$ sudo rpm -e system-config-lvm
package system-config-lvm is not installed
```

- 依存関係によるエラーの例

```
$ sudo rpm --test -e xz
error: Failed dependencies:
        xz is needed by (installed) pcp-5.1.1-4.el8_3.x86_64
        xz is needed by (installed) sos-3.9.1-6.el8.noarch
        xz is needed by (installed) gettext-devel-0.19.8.1-17.el8.x86_64
        xz is needed by (installed) libreport-2.9.5-15.el8.x86_64
        xz is needed by (installed) dracut-049-95.git20200804.el8_3.4.x86_64
        xz is needed by (installed) libvirt-daemon-driver-qemu-6.0.0-28.module+el8.3.0+7827+5e65edd7.x86_64
        xz is needed by (installed) rpm-build-4.14.3-4.el8.x86_64
        xz is needed by (installed) libguestfs-1:1.40.2-25.module+el8.3.0+7421+642fe24f.x86_64
        xz is needed by (installed) sysstat-11.7.3-5.el8.x86_64
        /usr/bin/xz is needed by (installed) file-roller-3.28.1-3.el8.x86_64
```

`-e` オプションにより rpm はパッケージをアンインストール（消去）します。通常、アンインストールしようとしているパッケージがシステム上の他のパッケージで必要な場合、`rpm -e` はエラーメッセージを出力して失敗の形で終了します。

`-e` オプションとともに `--test` オプションを使用すると、実際にアンインストールを行うことなくアンインストールが成功するか失敗するかを確認できます。操作が成功すると `rpm` は何も出力しません。`-vv` オプションを追加すれば、更に詳細な情報を取得できます。

削除する時のパッケージ引数はパッケージ名です。**rpm** ファイル名ではないことに注意してください。

重要な注意事項: 決して **rpm** パッケージそのものを削除（消去／アンインストール）しないでください。もし削除してしまった場合、オペレーティングシステムの再インストールか、レスキュー環境の起動以外に問題を解決する手段はありません。

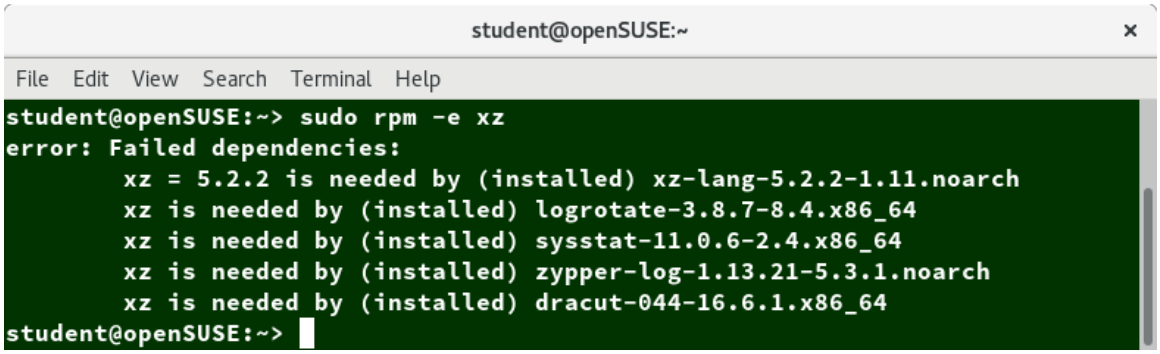


図 6.1: **Uninstalling RPM Packages**

openSUSE で取得されたスクリーンショットは、スライドの **RHEL** で取得されたものと依存関係が多少異なる場合があります。

6.7 アップデート、アップグレード、RPM パッケージの更新

パッケージのアップデート

- 元のパッケージが置き換えられる（インストールされている場合）：
  `$ rpm -Uvh bash-4.4.19-12.el8.x86_64.rpm`
- 現在インストールされていなければパッケージをインストール
- 同一ファイルを含む２つのパッケージのインストールは出来ない
- `--oldpackage` オプションを使ってダウングレードが可能

アップグレードする場合、新しいバージョンがインストールされると、既にインストールされているパッケージは削除されます。唯一の例外は元のインストールの構成ファイルで、拡張子が `.rpmsave` に変更されて保存されます。

`-U` オプションを使用し、かつパッケージがまだインストールされていない場合、単純にインストールされエラーはありません。

`-i` オプションはアップグレード用として設計されていません。古い **RPM** パッケージの上に新しい **RPM** パッケージをインストールしようとすると、既存システムファイルの上書きというエラーメッセージを表示して失敗します。

ただし、パッケージの各バージョンに同じファイルが含まれていない場合、同じパッケージの異なるバージョンがインストールできる場合があります。カーネルパッケージとライブラリパッケージは、代替アーキテクチャから複数回インストールされる唯一のパッケージです。

rpm -U でダウングレードする（つまり現在のバージョンを以前のバージョンに置き換える）場合は、コマンドラインに `--oldpackage` オプションを追加する必要があります。

パッケージの更新

- カレントディレクトリの全てのパッケージを **更新**:
  ```
  $ sudo rpm -Fvh *.rpm
  ```
  - 古いバージョンのパッケージがインストールされている場合、ディレクトリ内の新しいバージョンにアップグレード
  - システム上のバージョンがディレクトリ内のバージョンと同じ場合、何もしない
  - パッケージのバージョンがインストールされていない場合、ディレクトリ内のパッケージは無視される
- 更新（-F）は多数のパッチ（すなわちアップグレードされたパッケージ）を一度に適用するのに便利です。

更新とアップグレードは似ていますが、ひとつ違いがあります。アップグレード時にパッケージが無い場合、そのパッケージが追加ローディングされます。更新の場合、パッケージがロードされていなければ単純に無視されます。オリジナルのパッケージが既にローディングされている場合は、アップグレードも更新も新しいパッケージをインストールします。

-F オプションは、いくつかの新しいパッチをダウンロードしてすでにインストールされているパッケージをアップグレードし、新しいパッケージをインストールしない時に便利です。

6.8 Linux カーネルのアップグレード

Linux カーネルのアップグレード

- **Red Hat** ベースのシステムに常に新しいカーネルをインストール（アップグレードではない）：

```
$ sudo rpm -ivh kernel-{version}.{arch}.rpm
```

- 問題があったら古いカーネルに戻せる
- 新カーネルのテスト終了後に古いカーネルを削除できるが、動作確認済の古いカーネルも、１つ以上残して置くことを推奨

システムに新しいカーネルをインストールした場合、それを有効にするためには再起動（数少ないアップデートの 1 つ）が必要です。カーネルのアップグレード (`-U`) を実行しないでください。アップグレードを行うと現在実行中の古いカーネルが削除されます。

アップグレードしてもシステムは停止しませんが、再起動後に問題が発生した場合古いカーネルがシステムから削除されているため、再起動ができなくなります。ただしインストール (`-i`) を使った場合両方のカーネルは共存しどちらを起動するかを選択できます。つまり、必要に応じて古いカーネルにも戻せます。

これを行うと、新しいバージョンのカーネルが含まれるように **GRUB** 構成ファイルが自動更新されます。他の目的でシステムを再構成していない限り、ブート時のデフォルト選択になります。

新しいカーネルバージョンをテストしたら、必要なら古いバージョンを削除できますが、これは必須ではありません。スペースが不足していない限り、１つ以上古いカーネルも使用可能にしておくことをお勧めします。

6.9 rpm2archive と rpm2cpio

rpm2archive と rpm2cpio の利用

- RPM パッケージのアーカイブ:

```
$ rpm2archive bash-XXXX.rpm
```

`bash-XXXX.rpm.tgz` を作る

- １ステップで展開:

```
$ cat bash-XXXX.rpm | rpm2archive - | tar -xvz
```

- RPM パッケージファイルを cpio にアーカイブに変換:

```
$ rpm2cpio bash-XXXX.rpm > bash.cpio
```

- 一つ以上のファイルの展開:

```
$ rpm2cpio bash-XXXX.rpm | cpio -ivd bin/bash
$ rpm2cpio logrotate-XXXX.rpm | cpio --extract --make-directories
```

- パッケージ内のファイルのリスト:

```
$ rpm -qilp bash-XXXX.rpm
```

rpm2archive を使って RPM パッケージファイルを **tar** アーカイブに変換できます。コマンドに - が与えられた場合には入力と出力はそれぞれ `stdin` と `stdout` になります。

rpm2archive は、以前のパッケージファイルを **cpio** アーカイブに変換する、またはパッケージファイルからファイルを解凍する **rpm2cpio** よりも直接的で効率的です。これを行うと現在のディレクトリの相対パスに展開されます。もし `/home/student` ディレクトリにいるユーザーが `bin/bash` ファイルを展開した場合、上の例のようにファイルが `/home/student/bin/bash` に展開されます。

パッケージ内のファイルを一覧表示したいだけなら、一番簡単な方法は **rpm** コマンドを使うことです。

```
$ rpm -qilp package.rpm
```

6.10 演習

デモ教材ビデオ

[ビデオによるデモ教材]
using_rpm_demo.mp4

課題 6.0: RPM が必要です

RHEL、CentOS、Fedora、SUSE、openSUSE

本演習を実行するためには、**RHEL**、**CentOS**、**Fedora**、**SUSE** や **openSUSE** などの **RPM** ベースのシステムにアクセスできる必要があります。

課題 6.1: RPM を使う

rpm パッケージを扱い、調べるためのいくつかの簡単な操作を行います。

本演習は **Red Hat** と **SUSE** ベースのシステム両方で実施できます。

1. /etc/logrotate.confというファイルを有するパッケージの検索
2. パッケージが含む全てのファイルとパッケージ自身の情報の表示
3. パッケージのインストール状況の確認
4. パッケージの削除

解 6.1

1. `$ rpm -qf /etc/logrotate.conf`

```
logrotate-3.14.0-3.el8.x86_64
```

2. `$ rpm -qil logrotate`

```
...
```

この２つのステップを一度にやる方法もあります:

```
$ rpm -qil $(rpm -qf /etc/logrotate.conf)
```

3. `$ rpm -V logrotate`

```
..?......    /etc/cron.daily/logrotate
S.5....T.  c /etc/logrotate.conf
```

4. `$ sudo  rpm -e logrotate`

On RedHat RHEL 8 では

```
error: Failed dependencies:
        logrotate is needed by (installed) vsftpd-3.0.3-28.el8.x86_64
        logrotate >= 3.5.2 is needed by (installed) rsyslog-8.37.0-13.el8.x86_64
```

On openSUSE openSUSE-Leap 15.1 では

```
error: Failed dependencies:
        logrotate is needed by (installed) xdm-1.1.11-lp151.13.2.x86_64
        logrotate is needed by (installed) wpa_supplicant-2.6-lp151.4.4.x86_64
        logrotate is needed by (installed) chrony-3.2-lp151.8.6.x86_64
        logrotate is needed by (installed) net-snmp-5.7.3-lp151.7.5.x86_64
        logrotate is needed by (installed) syslog-service-2.0-lp151.3.3.noarch
        logrotate is needed by (installed) vsftpd-3.0.3-lp151.6.3.x86_64
        logrotate is needed by (installed) libvirt-daemon-5.1.0-lp151.7.6.1.x86_64
        logrotate is needed by (installed) iscsiuio-0.7.8.2-lp151.13.6.1.x86_64
        logrotate is needed by (installed) mcelog-1.60-lp151.2.3.1.x86_64
```

正確なパッケージの依存関係は、ディストリビューションやインストール済のソフトウェアに関係していることに注意してください。

課題 6.2: RPM データベースの再構築

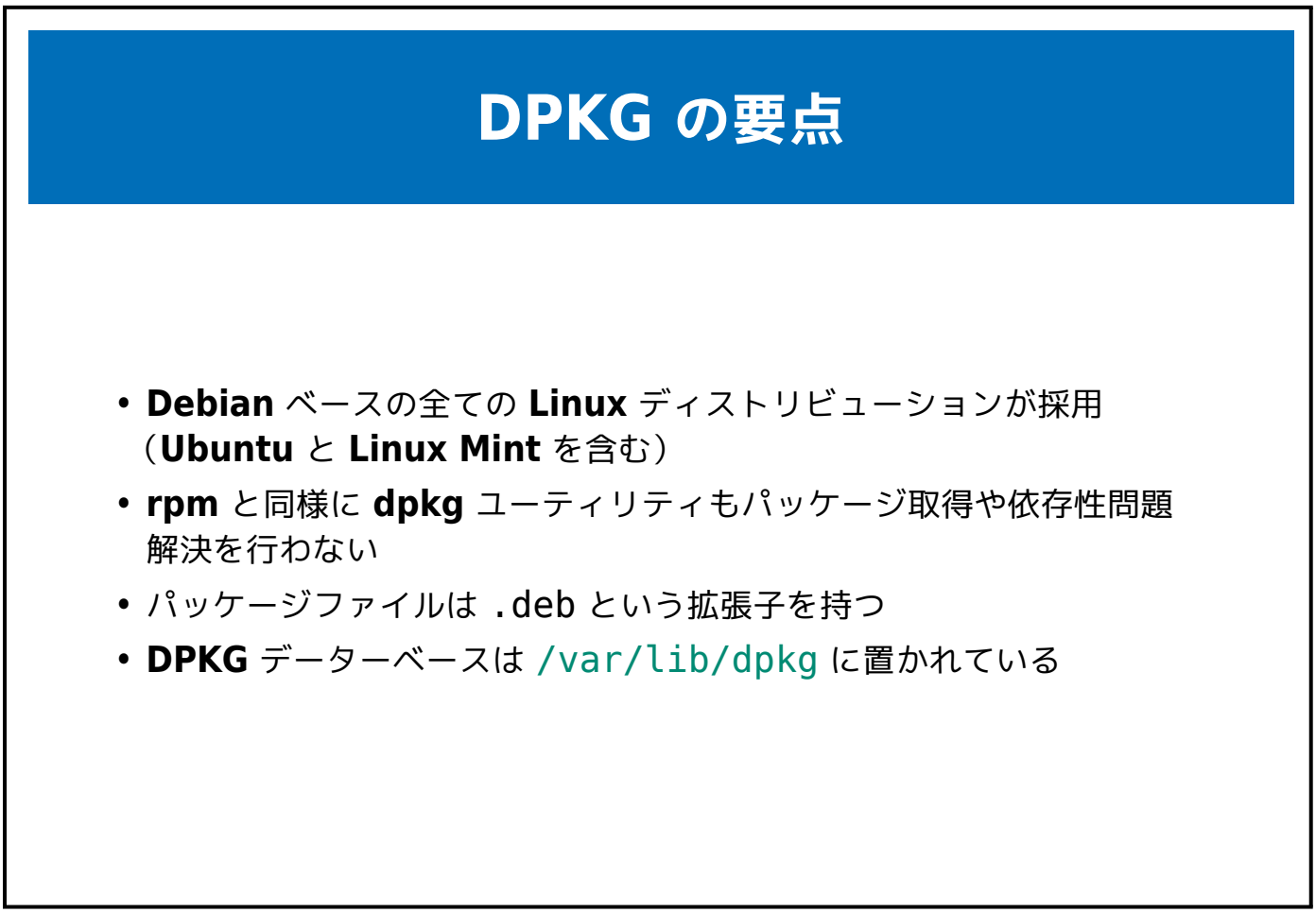
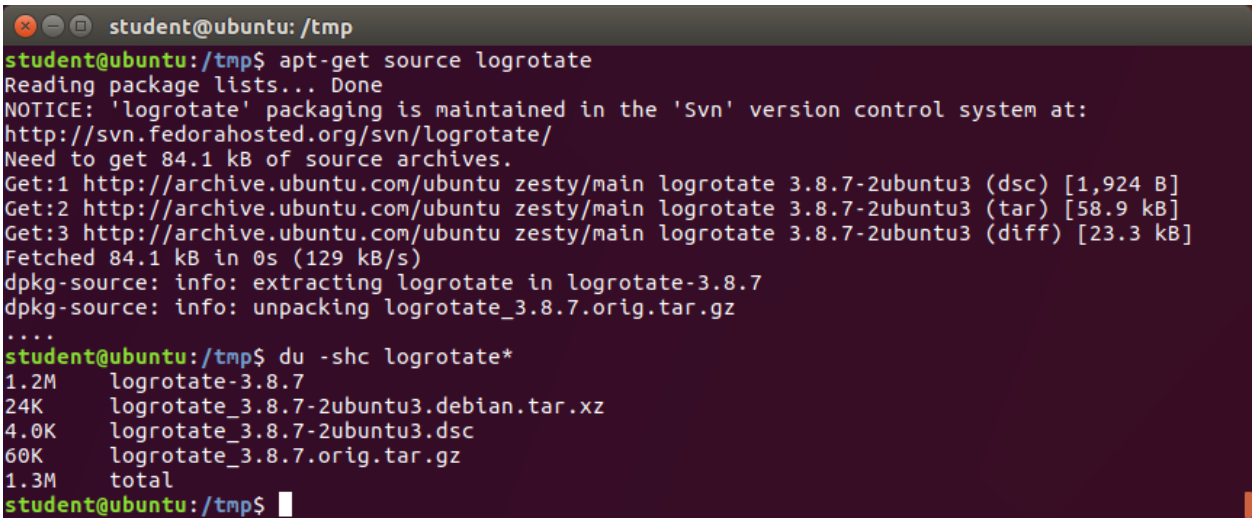
/var/lib/rpmに格納されている **RPM** データベースが破壊されてしまう場合があります。本演習では、新しいデータベースを構築しその整合性を確認します。

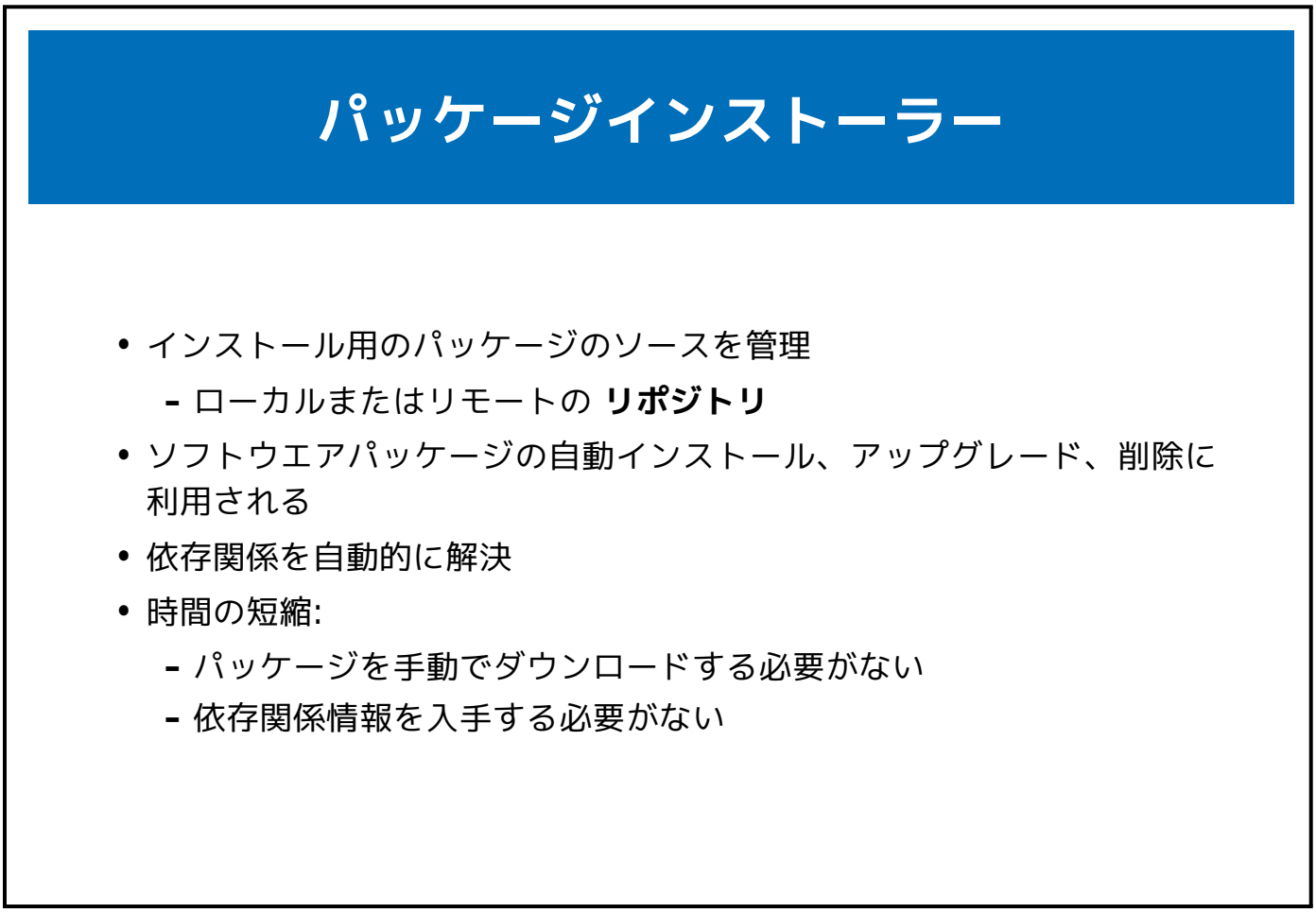
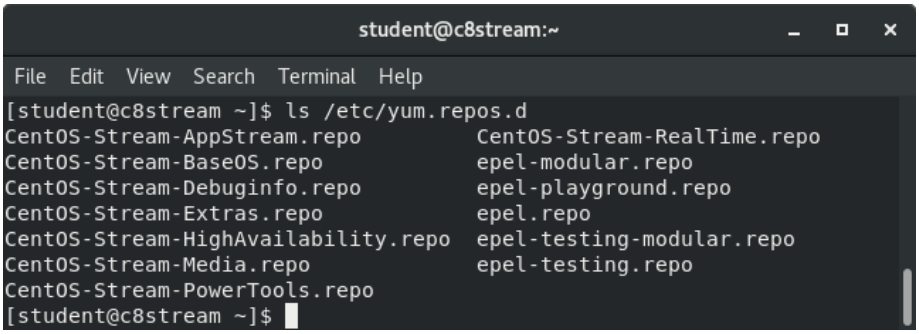
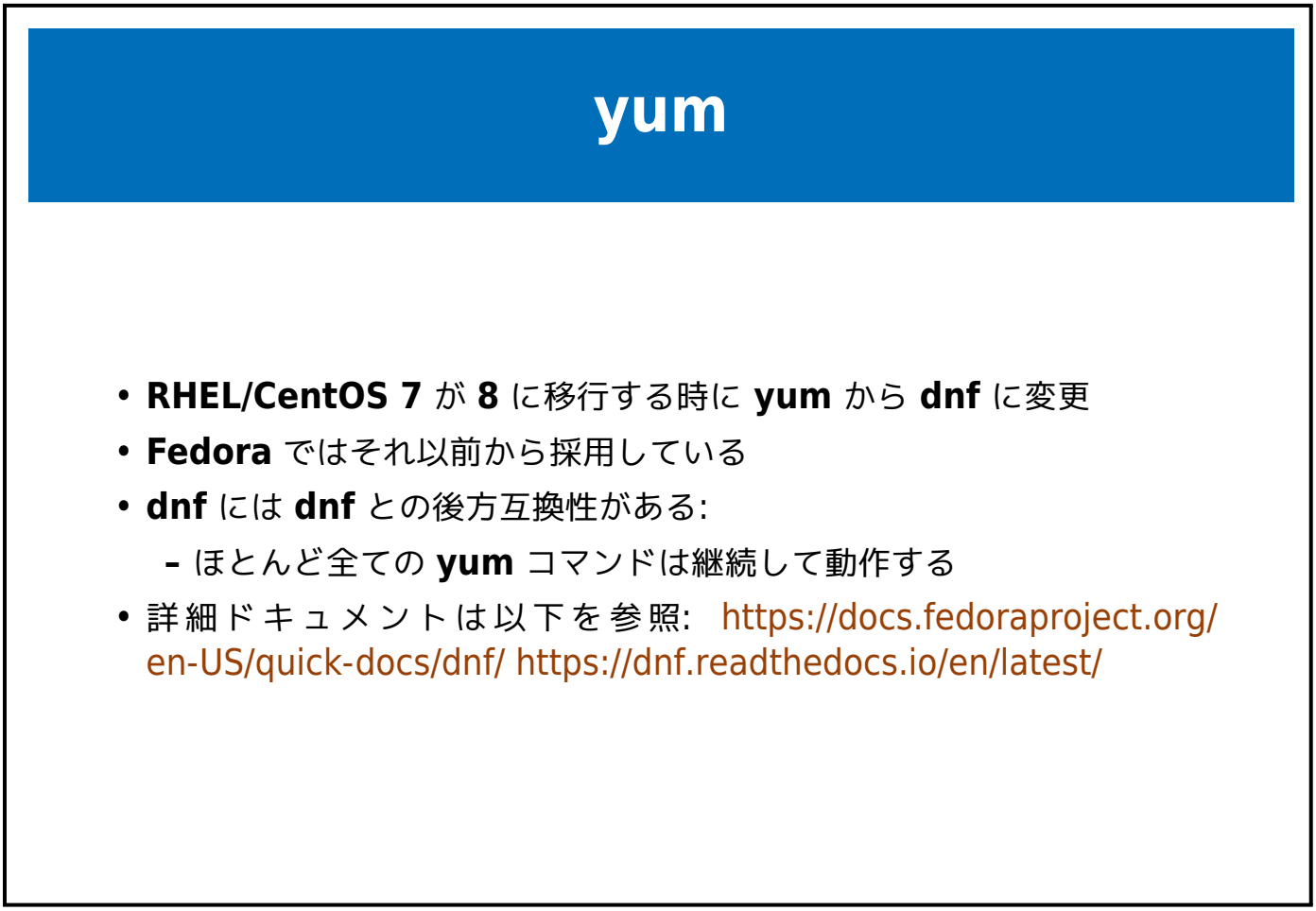
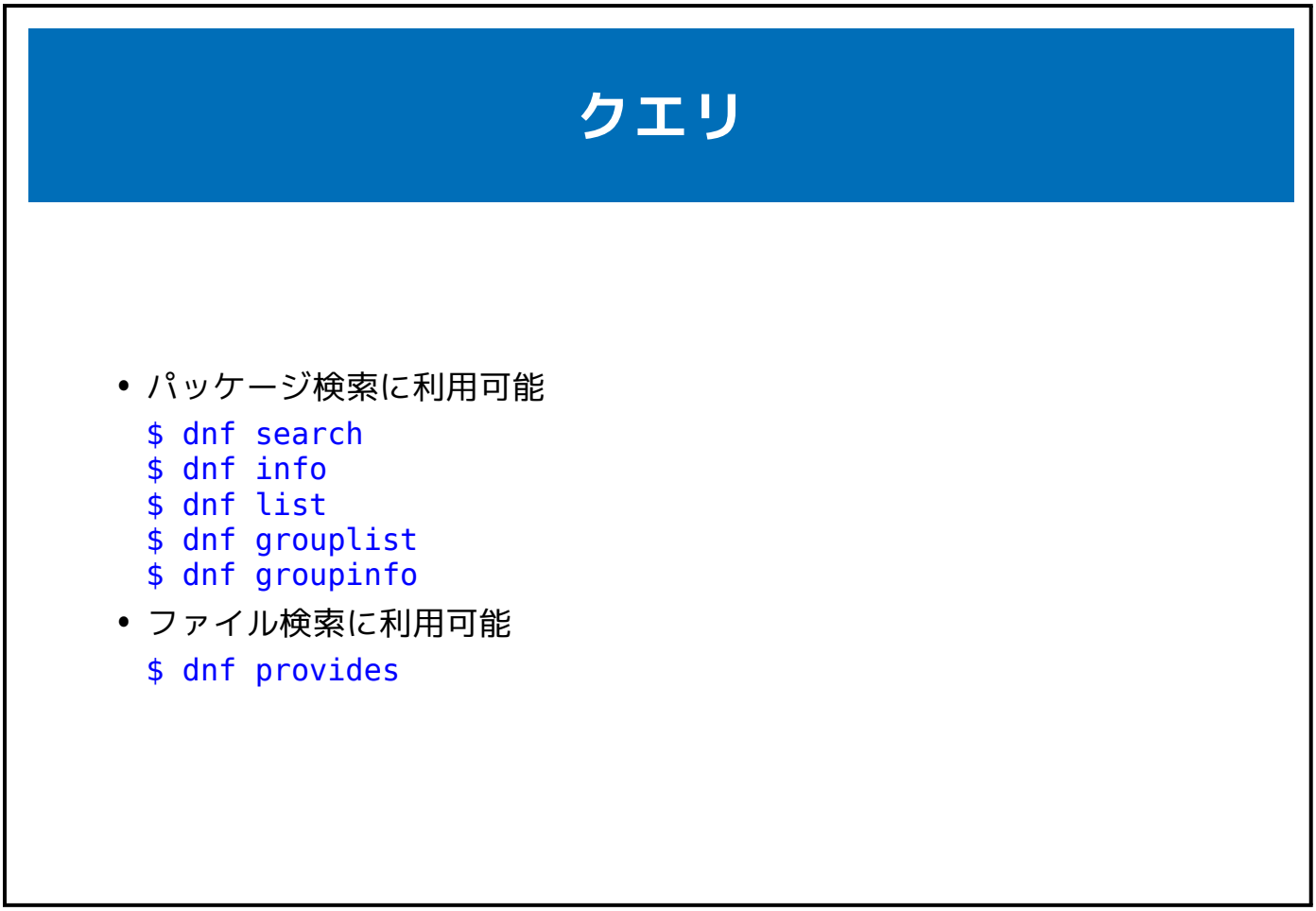
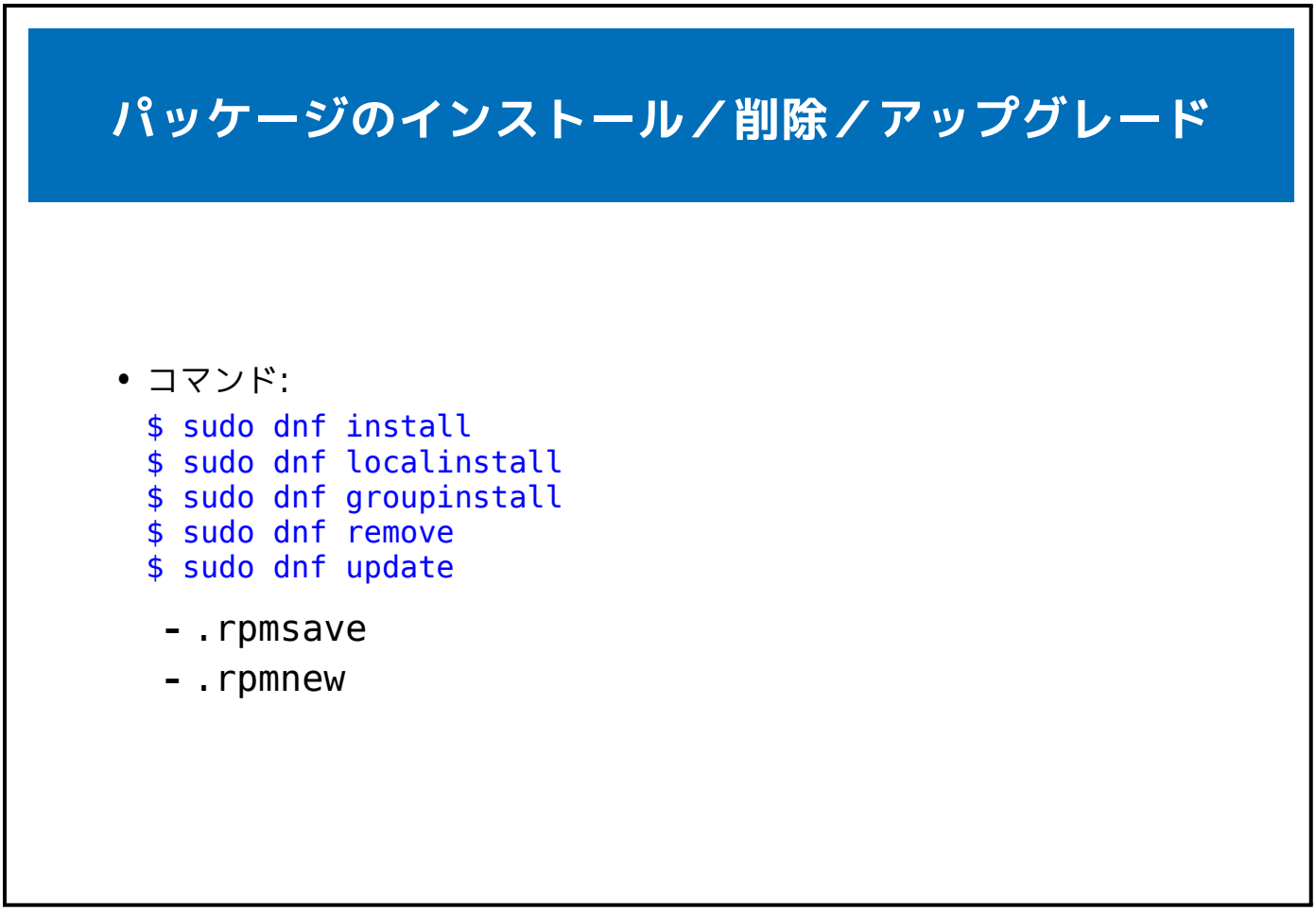
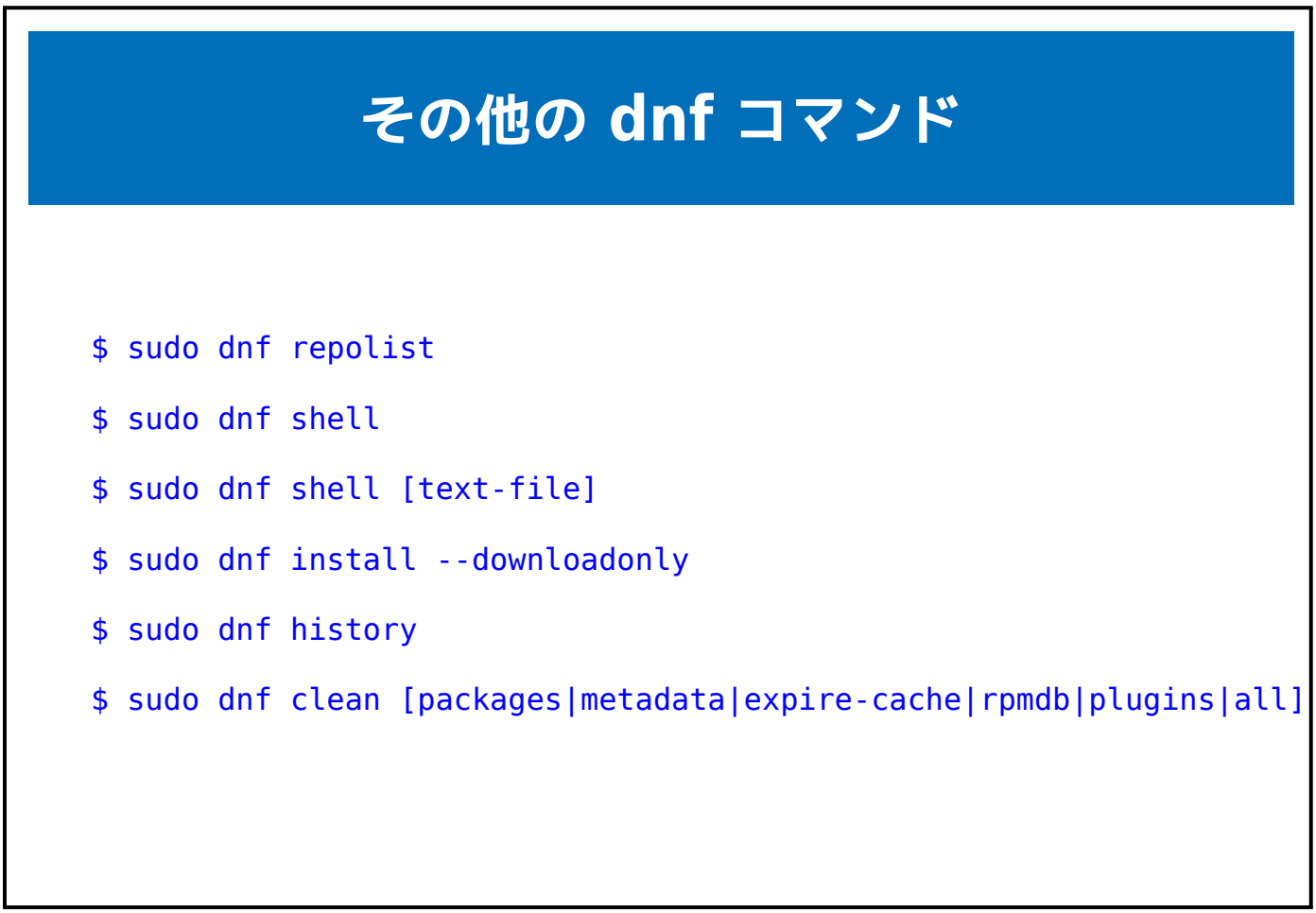
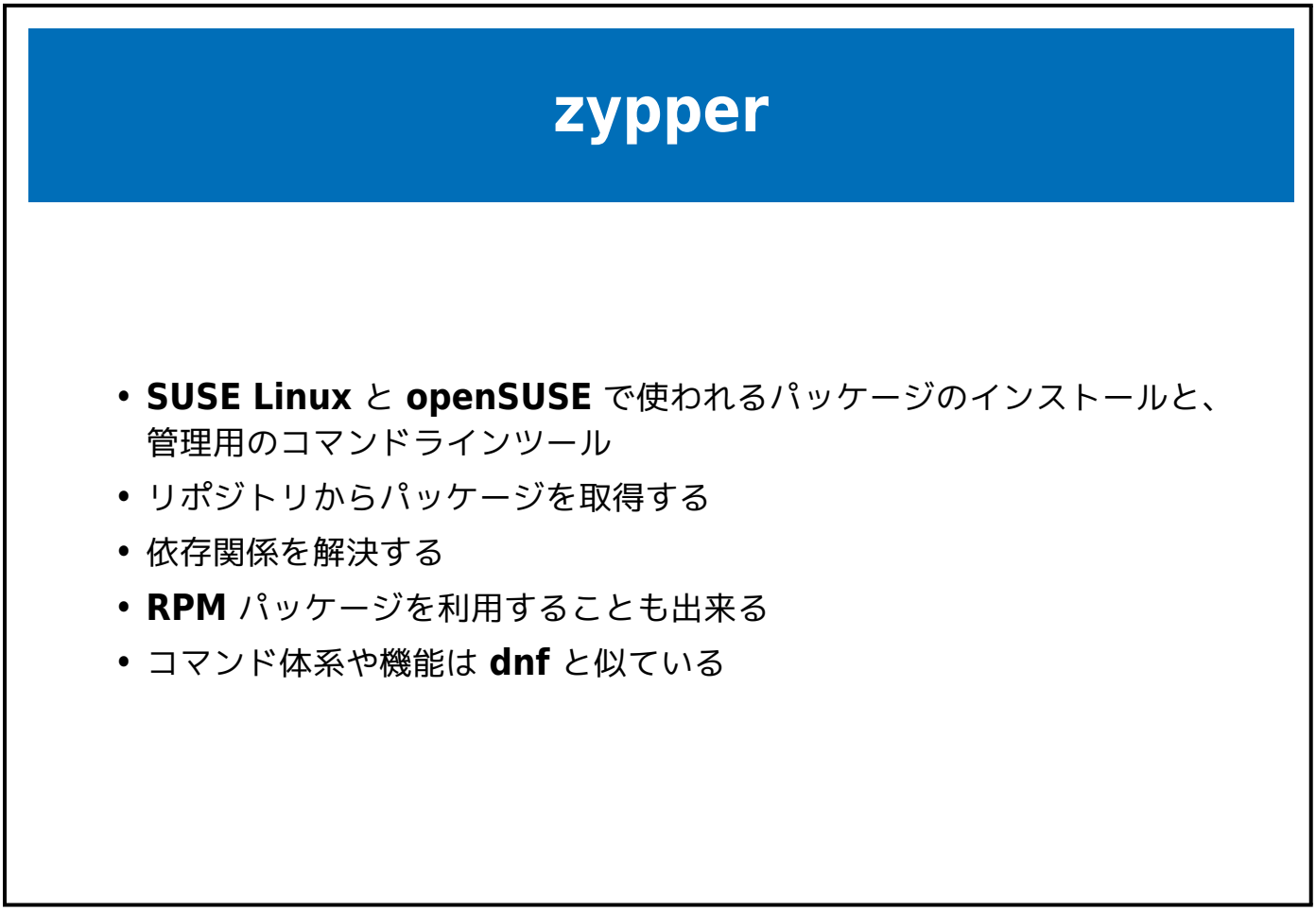
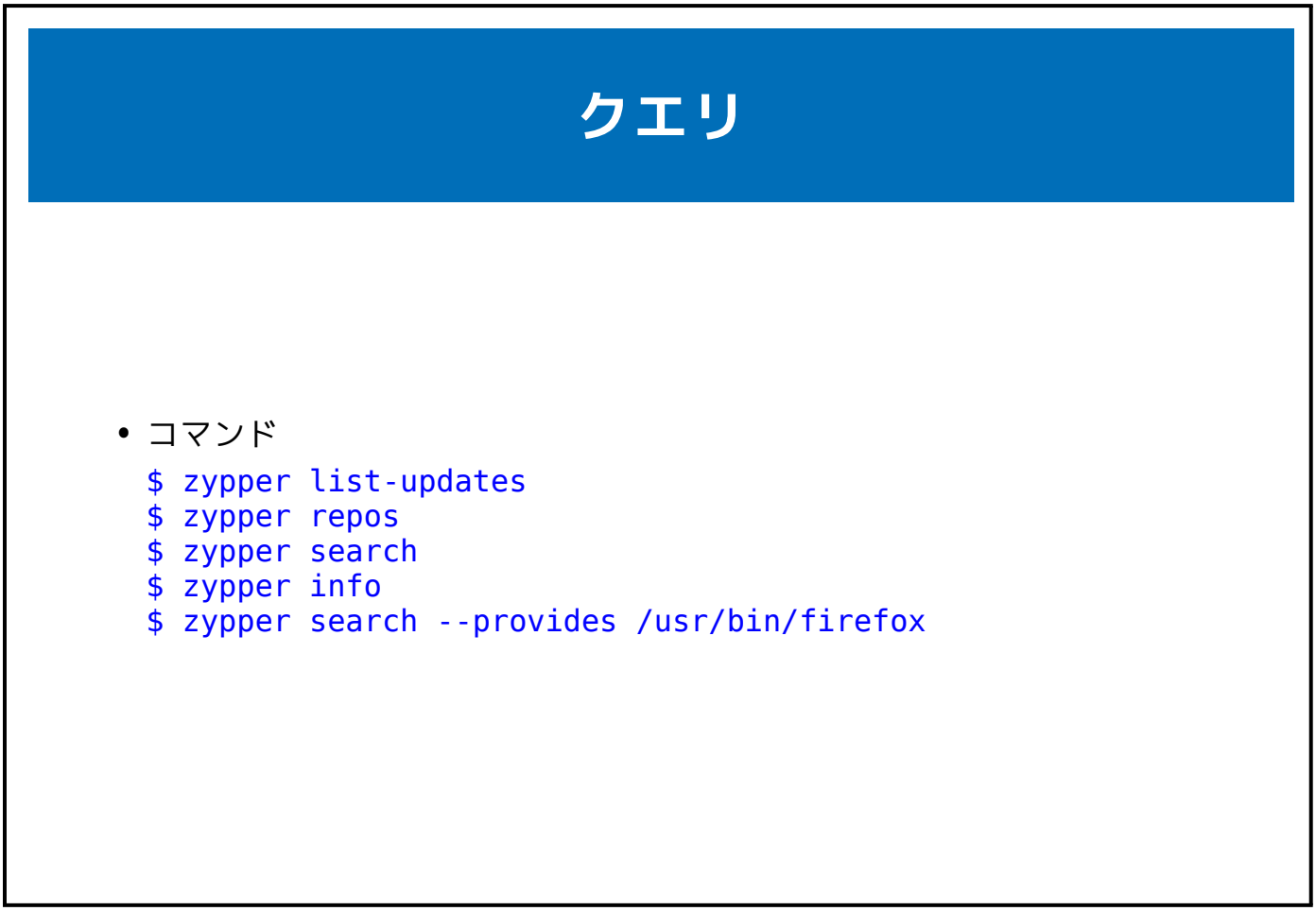
本演習は **Red Hat** と **SUSE** ベースのシステム両方で実施できます。

1. 再構築の過程で上書きするので、/var/lib/rpm の内容をバックアップします。もし、バックアップをしないで何かうまくできないことが起きたら、深刻な状況に陥ります。
2. データベースを再構築します。
3. 新しいディレクトリの内容とバックアップした内容を比較します。バイナリデータのため、直接比較しても駄目です。ファイルの数と名前を比較します。
4. システムにインストールされている全ての **rpms** のリストを表示します。再構築の手順を実行する前にリストした内容と比較します。正しくクエリコマンドが動いていれば、データベースファイルは問題ありません。
5. 両方のディレクトリ内容を比較します。同じファイルを含んでいますか？
6. バックアップを削除しても構いません（おそらく 100MB 程度でしょう）。しかし、破棄する前にシステムがきちっと動作するかをしばらく監視したほうが良いと思います。

解 6.2

1. ```
   $ cd /var/lib
   $ sudo cp -a rpm rpm_BACKUP
   ```
2. ```
   $ sudo rpm --rebuilddb
   ```
3. ```
   $ ls -l rpm rpm_BACKUP
   ```
4. ```
   $ rpm -qa | tee /tmp/rpm-qa.output
   ```
5. ```
   $ ls -l rpm rpm_BACKUP
   ```
6. システムに問題が無いことを確認してから、実行してください！

   ```
   $ sudo rm -rf rpm_BACKUP
   ```
   ```

章 7

dpkg

7.1 DPKG (Debian パッケージ)



DPKG (**Debian Package**) は、**Debian Linux** およびそれから派生した他のディストリビューションが採用する、ソフトウェアパッケージをインストール、削除、そして管理するためのパッケージングシステムです。**RPM** と同様に日常的に使用するパッケージを直接取得する設計にはなっておらず、ローカルにインストールしたり削除したりします。

パッケージファイルの拡張子は、.deb で **DPKG** データベースは /var/lib/dpkg ディレクトリにあります。

rpm と同様に、**dpkg** プログラムは全体の一部分だけを見ているにすぎません。システムにインストールされているものと、コマンドラインで与えられたものは把握していますが、システム上のディレクトリやインターネット上のディレクトリに他のパッケージがあるのかどうかは掌握していません。そのため、依存関係が満たされていない場合や、他のインストール済みパッケージに必要なパッケージを削除しようとした場合には失敗します。

7.2 パッケージのファイル名とソース

パッケージのファイル名とソース

- バイナリパッケージの標準的な命名形式:
  `<name>_<version>-<revision_number>_<architecture>.deb`
  以下に見られるように:
  `logrotate_3.14.0-4_amd64.deb`
  `logrotate_3.14.0-4ubuntu3_amd64.deb`
- ソースパッケージは少なくとも３つのファイルで構成される:
  - **アップストリーム tarball** (ファイル名の最後が `.tar.gz`)
  - **説明ファイル** (ファイル名の最後が `.dsc`)
  - 第二の tarball (ファイル名の最後が `.diff.gz` または `.debian.tar.gz`) にはアップストリームソースへのパッチとパッケージ用に作成された追加ファイルが含まれる

Debian パッケージのファイル名は、特定の情報を表すフィールドに基づいています。

歴史的な理由から、64 ビットの **x86** プラットフォームは **x86_64** ではなく **amd64** と呼ばれています。**Ubuntu** などのディストリビュータは、パッケージ名に自分の名前を入れています。

Ubuntu システムでは、次に示すようにソースパッケージをダウンロードしてダウンロードまたは作成されたファイルを確認できます:



図 7.1: **Ubuntu でソースパッケージを取得**

7.3 DPKG クエリ

DPKG クエリ

- インストールされているすべてのパッケージをリスト:
  ```
  $ dpkg -l
  ```
- wget パッケージにインストールされているファイルをリスト:
  ```
  $ dpkg -L wget
  ```
- インストール済パッケージとパッケージファイルに関する情報を表示:
  ```
  $ dpkg -s wget
  $ dpkg -I webfs_1.21+ds1-8_amd64.deb
  ```
- パッケージファイルに関する情報を表示:
  ```
  $ dpkg -c  webfs_1.21+ds1-8_amd64.deb
  ```
- /etc/init/networking.conf を所有しているパッケージを表示:
  ```
  $ dpkg -S /etc/init/networking.conf
  ```
- インストールされたパッケージの整合性を確認:
  ```
  $ dpkg -V package
  ```

dpkg 1.17 以降のバージョンのみが-V オプションをサポートします。引数がない場合は、システム上のすべてのパッケージを検証します。出力の意味を知りたい方は **man** ページを参照してください。

7.4 インストール／アップグレード／アンインストール

インストール／アップグレード／アンインストール

- **foobar** パッケージのインストールまたはアップグレード:
  `$ sudo dpkg -i foobar.deb`
- 構成ファイルを除くインストール済みのパッケージをすべて削除:
  `$ sudo dpkg -r package`
- 構成ファイルを含めインストールされている全パッケージを削除:
  `$ sudo dpkg -P package`

`-i` オプションを使うときは (インストールのため):

- パッケージが現在インストールされていない場合は、インストールします。
- パッケージが現在インストールされているパッケージよりも新しい場合は、アップグレードされます。

`-P` オプションは **パージ (purge)** を表すことに注意してください。

7.5 演習

デモ教材ビデオ

[ビデオによるデモ教材]
using_dpkg_demo.mp4

課題 7.0: dpkg が必要です

Debian、Ubuntu、Linux Mint

本演習を実行するためには、**Debian**、**Ubuntu** や **Linux Mint** などの **Debian** ベースのシステムにアクセスできる必要があります。

課題 7.1: dpkg を使う

Debian パッケージを検索したり、検証したりする簡単な操作を学びます。

1. /etc/logrotate.conf が、どのパッケージに含まれているかを調べます
2. パッケージに含まれている全てのファイルを参照しているパッケージに関する情報の表示
3. パッケージのインストールの検証
4. パッケージの削除

解 7.1

1. `$ dpkg -S logrotate.conf`

```
logrotate: /usr/share/man/man5/logrotate.conf.5.gz
logrotate: /etc/logrotate.conf
rsync: /usr/share/doc/rsync/examples/logrotate.conf.rsync
```

2. `$ dpkg -L logrotate`

```
...
```

3. `$ dpkg -V logrotate`

4. `$ sudo dpkg -r logrotate`

```
dpkg: dependency problems prevent removal of logrotate:
 ubuntu-standard depends on logrotate; however:
  Package logrotate is to be removed.
 libvirt-daemon-system depends on logrotate.

dpkg: error processing package logrotate (--remove):
 dependency problems - not removing
Errors were encountered while processing:
 logrotate
```

章 8

dnf と yum

8.1 パッケージインストーラー



rpm や **dpkg** などの低レベルのパッケージ **ユーティリティ** は、特定のソフトウェアパッケージファイルのインストールとインストール済みのソフトウェアを管理します。

高レベルのパッケージ **管理システム**（**dnf**、**yum**、**apt**、**zypper**）には、利用可能なソフトウェアのデータベースと連携し非常に洗練された方法でソフトウェアを検索、インストール、更新、アンインストールするツールが組み込まれています。

ソフトウェアリポジトリは、ディストリビューションとその他の独立したソフトウェアプロバイダによって提供されます。パッケージインストーラーは、リポジトリが保持しているカタログから派生した、利用可能なソフトウェアのデータベースを維持します。低レベルのパッケージツールとは異なり、依存関係を自動検索してインストールする機能があります。これは重要な機能です。

このセクションでは **dnf** と **yum** について説明します。**zypper** と **apt** はこの後のセクションで説明します。

8.2 dnf と yum

dnf とは何か？

- RPM のフロントエンド
- リモートリポジトリからパッケージをフェッチ
  - 複数の別々のリポジトリを利用可能
  - リポジトリ構成ファイルは `/etc/yum.repos.d/`
- 依存関係の解決に対応
- **RHEL**、**CentOS**、**Fedora** などが採用



図 8.1: **rpm リポジトリ**

dnf にはパッケージマネージメントに役立つ機能が多く含まれます。**RPM** のフロントエンドであるだけでなく、一つないし複数のリモートリポジトリからパッケージを取得する機能も有しています。特に優れた機能の一つは、依存関係解決ができることです。

リポジトリの構成ファイルは `/etc/yum.repos.d` に置かれており、`.repo` という拡張子を持っています。

repo ファイルの基本は以下のようなものです:

Repo の例

```
[repo-name]
    name=Description of the repository
    baseurl=http://somesystem.com/path/to/repo
    enabled=1
    gpgcheck=1
```

dnf は、処理性能を高める為にキャッシュ情報を持ちます。以下のコマンドで一部、ないし全部のキャッシュを削除できます:

```
$ dnf clean [ packages | metadata | expire-cache | rpmdb | plugins | all ]
```

`enabled` の値を変更するか、`--disablerepo somerepo` と `--enablerepo somerepo` を使うと、特定の repo を使うか使わないかを切り替えることができます。

dnf の操作には、ディストリビューションが提供するグラフィカルインターフェースを使わず、コマンドラインだけを利用します。

8.3 yum



dnf のバージョンによっては、使おうとした **yum** コマンドが廃止予定と警告され、正しいコマンドが示されるかもしれません。

あなたが既に **yum** を使い込んでいるなら、日々の業務で利用する大部分の **yum** のコマンドのサブセットは **dnf** でも動作するので、**dnf** を少しづつ覚えていくことができます。

8.4 クエリ




```
$ dnf search keyword
```

パッケージをキーワードで検索することが可能です。

```
$ dnf info package-name
```

パッケージ情報を表示します。

```
$ dnf list [installed | updates | available]
```

インストール済み、利用可能、更新済みのパッケージをリストします。

```
$ dnf grouplist
```

インストール済み、利用可能、更新済みのパッケージグループをリストします。

```
$ dnf groupinfo packagegroup
```

パッケージグループ情報を表示します。

```
$ dnf provides /path/to/file
```

ファイル名からパッケージのオーナーを表示します。(紛らわしいのは、ファイル名に少なくとも一つの / が必要なことです。)

8.5 パッケージのインストール／削除／アップグレード




```
$ sudo dnf install package
```

リポジトリから、パッケージをインストールします。同時に、インストール依存関係も解決します。

```
$ sudo dnf localinstall package-file
```

ローカルな rpm ファイルからパッケージをインストールします。

```
$ sudo dnf groupinstall 'group-name'
```

リポジトリから、特定ソフトウエアグループをインストールします。同時に、グループ内の各パッケージのインストール依存関係を解決します。

```
$ sudo dnf remove package
```

システムからパッケージを削除します。

```
$ sudo dnf update [package]
```

リポジトリからパッケージを更新します。パッケージ名が指定されていない場合は、全てのパッケージを更新します。

インストール（または更新）中にパッケージに更新された構成ファイルを見つけると、古い構成ファイルの名前を `.rpmsave` 拡張子に変更します。古い構成ファイルを新しいソフトウェアで引き続き使用する場合は、新しい構成ファイルの名前に `.rpmnew` 拡張子を付けます。これらのファイル名拡張子を検索して、何らかの調整を行う必要があるかを調査できます。

8.6 その他の dnf コマンド




```
$ sudo dnf list "dnf-plugin*"
```

追加的な **dnf** プラグインをリストします。

```
$ sudo dnf repolist
```

利用可能なリポジトリを表示します。

```
$ sudo dnf shell
$ sudo dnf shell file.txt
```

複数の dnf コマンドを利用できる対話型シェルを提供します。２番目では `file.txt` からコマンドを実行します。

```
$ sudo dnf install --downloadonly package
```

パッケージのダウンロードのみを行います。ファイルは `/var/cache/dnf` に保存されます。

```
$ sudo dnf history
```

dnf コマンドの履歴の閲覧、修正オプションを付加して取り消し（undo）や再実行（redo）ができます。

```
$ sudo dnf clean [packages|metadata|expire-cache|rpmdb|plugins|all]
```

ローカルに保存されたファイルと `/var/cache/dnf` に保存されたメタデータを削除します。陳腐化したデータを削除してスペースを空けることができます。

8.7 演習

デモ教材ビデオ

[ビデオによるデモ教材]
using_yum_and_dnf_demo.mp4

課題 8.0: dnf か yum が必要

RHEL、CentOS、Fedora、SUSE、openSUSE

以下の演習を実行するには **RHEL/CentOS** や **fedora** など、**dnf** ベースのシステムにアクセスできる必要があります。**RHEL/CentOS 7** では **dnf** を **yum** に置き換えて、以下の演習を行ってください。**dnf** には後方互換性があります。

課題 8.1: dnf 基本コマンド

1. システムに最新のアップデートがあるか確認します。
2. 特定のパッケージを更新します。
3. インストール済のカーネル関係のパッケージをリストします。そして全てのインストール済または利用可能な Kernel パッケージのリストを表示します。
4. **httpd-devel** パッケージ、または他のまだインストールしていないパッケージのインストール。以下のように入力すると:

   ```
   $ sudo dnf list
   ```

   これで完全なリストが表示されます; 引数にワイルドカードを指定することで、リストを絞ることができます。

解 8.1

1. ```
   $ sudo dnf update
   $ sudo dnf check-update
   $ sudo dnf list updates
   ```

   1 番目のコマンドだけが、インストールを試します。
2. ```
   $ sudo dnf update bash
   ```
3. ```
   $ sudo dnf list installed "kernel*"
   $ sudo dnf list "kernel*"
   ```
4. ```
   $ sudo dnf install httpd-devel
   ```

課題 8.2: パッケージの情報を探す

（直接 **rpm** を使わず）**dnf** を使い、以下の情報を得ます:

1. 名称や説明に **bash** に関する内容を含む全てのパッケージ
2. インストール済で利用可能な **bash** パッケージ
3. **bash** のパッケージ情報
4. **bash** パッケージの依存情報

これらのコマンドを、root と一般のユーザーの両方で試してください。違いがわかりますか？

解 8.2

注目

ディストリビューションのバージョンによっては、**sudo** を使わず以下のコマンドを実行すると、情報を得るだけにもかかわらずパーミッションエラーが出ます。

```
$ sudo dnf search bash
$ sudo dnf list bash
$ sudo dnf info bash
$ sudo dnf deplist bash
```

課題 8.3: パッケージ グループを管理する

注目

RHEL/CentOS 上で、**sudo** を使わず以下のコマンドを実行すると、情報を得るだけにもかかわらずパーミッションエラーが出ます。

dnf は、パッケージグループ管理機能を提供しています。

1. システムで利用可能な全てのパッケージグループをリストするには、以下のようにします:

   ```
   $ dnf grouplist
   ```

2. Virtualization Host グループを指定して、その情報を得るには、以下のようにします。

   ```
   $ dnf groupinfo "Virtualization Host"
   ```

3. 次のようにしてインストールします:

   ```
   $ sudo dnf groupinstall "Virtualization Host"
   ```

4. システムにインストール済で不要になったパッケージグループがあります。dnf groupremove を使い、以下のように削除します:

   ```
   $ sudo dnf groupremove "Virtualization Host"
   ```

   削除の確認プロンプトがでます。これにより、安全にこのコマンドの動作を確かめることができます。

   groupremove が、インストール済のものを全ては削除するわけでは **ない** ことに注意してください; これがバグなのか、機能なのかは議論があるところです。

課題 8.4: 新しいレポジトリの追加

著者によれば（http://www.webmin.com/index.htm）:

“**Webmin** は、Unix の管理者向け Web ベースのインターフェースを提供します。最新のブラウザを使って、ユーザー アカウント、Apache、DNS、ファイル共有、その他の設定が行えます。Webmin を使えば、/etc/passwd などのシステム構成ファイルを手動編集する必要がなくなります。コンソールやリモートで、システムを管理できるようになるのです。”

インストールとアップグレード用のレポジトリを作成します。ダウンロードと最新の **rpm** を入手できるページを作成します。自動的な更新機能は提供しません。

1. /etc/yum.repos.d中にwebmin.repoというレポジトリを作成します。内容は以下の通りです：

webmin.repo の内容

```
[Webmin]
name=Webmin Distribution Neutral
baseurl=http://download.webmin.com/download/yum
mirrorlist=http://download.webmin.com/download/yum/mirrorlist
enabled=1
gpgcheck=0
```

2. webmin パッケージをインストールします。

```
$ sudo dnf install webmin
```

章 9

zypper

9.1 zypper



zypper は **SUSE** ベースのシステム向けに、**rpm** プログラムの下回りを利用するためのハイレベルのインテリジェントサービスを提供するもので、**Red Hat** ベースのシステムにおける **dnf/yum** と同様な役割を果たします。インストール時には、依存関係を自動解決する機能を持っています。外部のソフトウエア **リポジトリ** にアクセスし、必要に応じてリポジトリの同期、パッケージの取得、インストールを行います。

zypper は **SUSE Linux** と **openSUSE** でパッケージをインストールおよび管理するためのコマンドラインツールです。
機能と基本的なコマンド構文は **dnf** と非常によく似ており **rpm** パッケージも利用できます。

9.2 クエリ




```
$ zypper list-updates
```

利用可能な更新をリストします。

```
$ zypper repos
```

利用可能なリポジトリをリストします。

```
$ zypper search <string>
```

文字列に該当するリポジトリを検索します。

```
$ zypper info firefox
```

パッケージの情報を表示します。

```
$ zypper search --provides /usr/bin/firefox
```

パッケージに含まれるファイルを表示するために、リポジトリを検索します。

9.3 パッケージのインストール／削除／アップグレード

パッケージのインストール／削除／アップグレード

- zypper インストール
  ```
  $ sudo zypper install firefox
  ```
- zypper 更新
  ```
  $ sudo zypper --non-interactive update
  $ sudo zypper update firefox
  ```
- zypper 削除
  ```
  $ sudo zypper remove firefox
  ```
- --non-interactive (または -n) を付与するとコマンド実行時に確認を求められない

```
$ sudo zypper install firefox
```

システムのパッケージをインストール、ないし更新します。

```
$ sudo zypper --non-interactive install firefox
```

インストールや更新時に確認を求められないので、スクリプトで使うのに便利です。

```
$ sudo zypper update
```

リポジトリから全てのパッケージを更新します。

```
$ sudo zypper --non-interactive update
```

リポジトリから全てのパッケージを更新します。実行時に確認を要求されないので、スクリプト内で使う時に便利です。

```
$ sudo zypper remove firefox
```

システムからパッケージを削除します。
dnf 同様パッケージ削除コマンドを実行すると、一緒に使われている他のパッケージも同時に削除される点に注意してください。

9.4 その他の zypper コマンド

その他の zypper コマンド

- **zypper** のコマンドラインシェルを起動:
  `$ sudo zypper shell`
- 新しいリポジトリの追加:
  `$ sudo zypper addrepo`
- リポジトリの削除:
  `$ sudo zypper removerepo`
- `/var/cache/zypp` をクリーンアップして、空き領域を拡大:
  `$ sudo zypper clean [--all]`

いくつかの **zypper** コマンドは、続けて実行する必要があります。
コマンド発行の都度データベース全体を再読み取りしないように、次のように **シェルモード** で **zypper** を実行できます。

```
$ sudo zypper shell
> install bash
...
> exit
```

bash shell. **zypper** は **readline** ライブラリをサポートしているため、**zypper** シェルでも **bash** シェルと同じコマンドライン編集機能を使用できます。

新しいリポジトリを追加するには:

```
$ sudo zypper addrepo URI alias
```

これは指定された `URI` にあり、追加後は指定された 別名（エイリアス）を使用します。

リストからリポジトリを削除するには:

```
$ sudo zypper removerepo alias
```

`alias` を使った削除 repo の指定

9.5 演習

デモ教材ビデオ

using_yast_demo.mp4

課題 9.0: zypper が必要です

On openSUSE openSUSE 環境を利用

本演習を行うには **SUSE**、**opeSUSE** など **zypper** ベースのシステムへのアクセスが必要です。

課題 9.1: zypper 基本コマンド

1. 最新のアップデートの確認。
2. あるパッケージの更新。
3. 有効かどうかに関係なく、システムが認識しているすべてのリポジトリのリスト。
4. インストール済のカーネル関係のパッケージのリスト。そして、全てのインストール済または利用可能なパッケージのリスト。
5. **apache2-devel** パッケージ、または、まだインストールしていないパッケージのインストール。(**httpd** は **SUSE** システムにおける **apache2** である)。以下のように入力します:

   ```
   $ sudo zypper search
   ```

   完全なリストが表示される; 引数にワイルドカードを指定することで、リストを絞ることができます。

解 9.1

1. `$ zypper list-updates`
2. `$ sudo zypper update bash`
3. `$ zypper repos`
4. ```
   $ zypper search -i kernel
   $ zypper search kernel
   ```
5. `$ sudo zypper install apache2-devel`

課題 9.2: zypper を使って、パッケージの情報を得る

(直接 **rpm** を使わず)、**zypper** を使い以下の情報を得ます:

1. 名称や説明に **bash** に関する内容を含む全てのパッケージ
2. インストール済で利用可能な **bash** パッケージ
3. **bash** のパッケージ情報
4. **bash** パッケージの依存情報
   ```

これらのコマンドを root と一般のユーザーの両方で試してください。違いがわかりますか？

解 9.2

1. `$ zypper search -d bash`

   -dオプションを付けないと、実際の名称に bash を含むパッケージのみが表示されます。どこに **bash** の記載があるかを見るためには、zypper info をパッケージに対して実行する必要があります。

2. `$ zypper search bash`

3. `$ zypper info  bash`

4. `$ zypper info--requires  bash`

   このコマンドは **bash** が要求するファイルをリストします。**bash** をインストールするときに依存関係を確認する簡単な方法です。

   ```
   $ sudo zypper remove --dry-run bash
   ```

   bash はシステムに統合されているため、この演習の例題としては良い例とは言えません; 削除はできません。

章 10

APT

10.1 APT

APT とは何か？

- **A**dvanced **P**ackaging **T**ool の略
- **apt-get** や **apt-cache** 等のユーティリティを含む (**dpkg** プログラムを下回りとして利用)
- **Debian** パッケージを利用する (ファイル拡張子は .deb)
- いくつかの **Debian** ベースの **Linux** ディストリビューションがひとつのリポジトリを利用するのも珍しくない

apt と apt-get

ほとんどすべてのインタラクティブな操作を行うには **apt-get** を使用する必要はなくなりました。**apt** という短い名前を使用できます。ただし **apt-get** の使用は習慣化されており、スクリプトでは使いやすいのです。このコースでは、コマンドラインでもスクリプトでも使えるコマンドの参考にするために **apt-get** を使います。

Debian ベースのシステムではプログラムの **APT** (**A**dvanced **P**ackaging **T**ool) は、**dpkg** プログラムの下回りを利用するための高度なインテリジェントサービスを提供し、**Red Hat** ベースのシステムにおける **dnf** と同様な役割を果たします。主なユーティリティは **apt** と **apt-cache** です。パッケージのインストール、更新、削除時に依存関係を自動的に解決できます。外部ソフトウェア **リポジトリ** にアクセスして同期を取り、必要に応じてソフトウェアを取得してインストールします。

繰り返しになりますが **synaptic** や **Ubuntu Software Center** のような（コンピュータ上の）グラフィカルインターフェイス、もしくは **aptitude** などの **APT** の古いフロントエンドはここでは扱いません。

10.2 APT ユーティリティ

apt、apt-get、apt-cache など

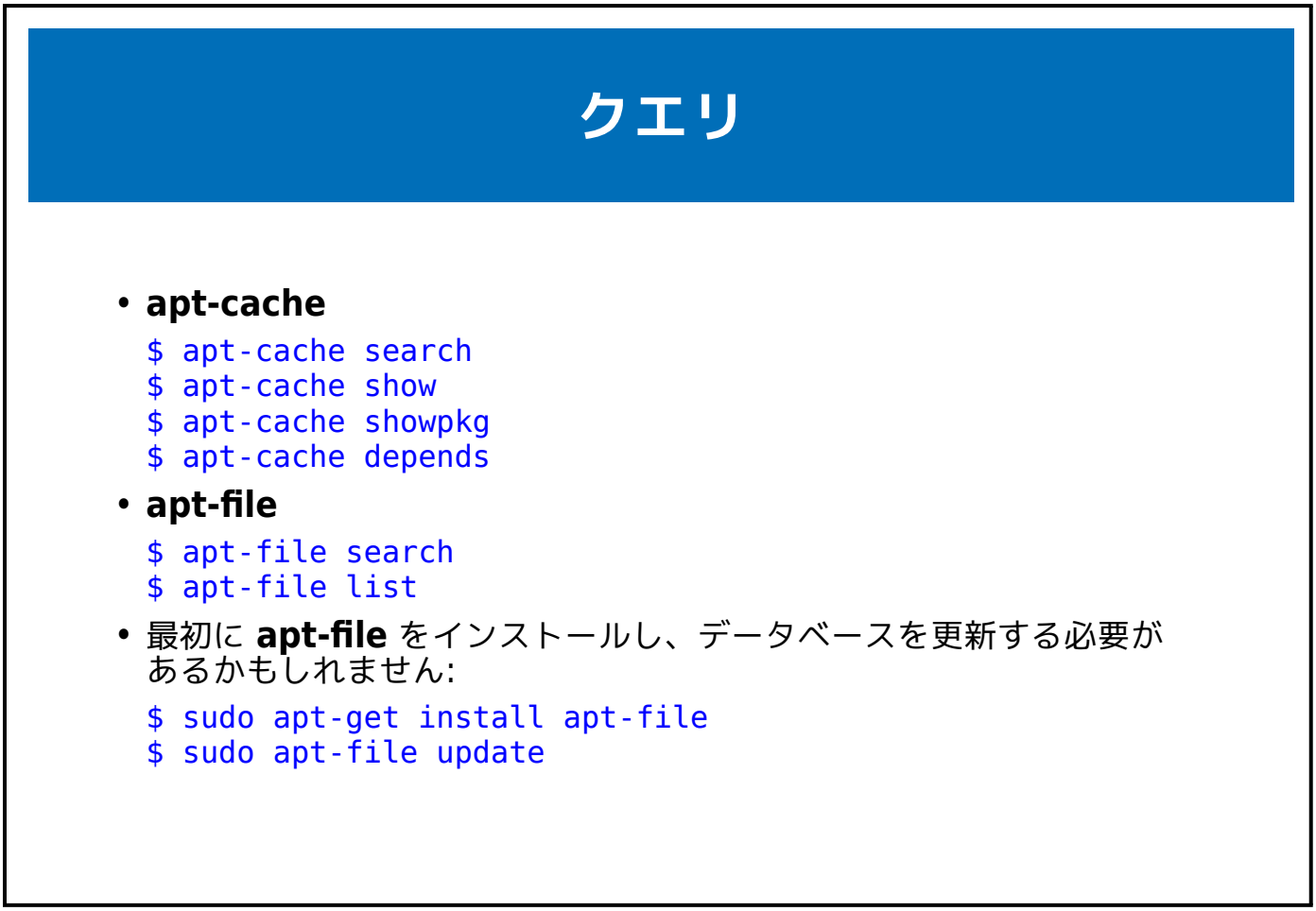
- **Debian Linux** のためのパッケージ操作用コマンドラインツール
  - 個々のパッケージのインストールと管理を行う
  - パッケージのアップグレードを行う
  - ディストリビューションのアップグレードを行う
- **dpkg** のフロントエンドである
- リポジトリからパッケージを取得することもできる
- 秀抜なリソース検索、調査、ダウンロードの仕組み:
  https://www.debian.org/distrib/packages
  https://packages.ubuntu.com

これらはパッケージ管理用の主要な **APT** コマンドラインツールです。個々のパッケージやシステム全体のインストール、管理、アップグレードに使用できます。ディストリビューションを完全に新しいリリースにアップグレードすることもできますが、これは難しい作業かもしれません。

apt が **rpm** ファイルを操作できるようにする（不完全な）拡張機能もあります。

dnf や **zypper** と同様に **APT** は複数のリモートリポジトリを扱うことができます。

10.3 クエリ




```
$ apt-cache search apache2
```

apache2 という名前のパッケージが含まれるリポジトリを検索します。

```
$ apt-cache show apache2
```

apache2 パッケージに関する基本的な情報を表示します。

```
$ apt-cache showpkg apache2
```

apache2 パッケージに関する詳細な情報を表示します。

```
$ apt-cache depends apache2
```

apache2 に依存関係のあるパッケージの完全なリストを表示します。

```
$ apt-file search apache2.conf
```

`apache2.conf` という名前のパッケージが含まれるリポジトリを検索します。

```
$ apt-file list apache2
```

apache2 パッケージに含まれる全てのファイルをリストします。

10.4 パッケージのインストール／削除／アップグレード

パッケージのインストール／削除／アップグレード

- パッケージのインストールと削除:

```
$ sudo apt-get install glibc
$ sudo apt-get remove glibc
$ sudo apt-get --purge remove glibc
```

- リポジトリデーターベースの **更新** とシステムないしパッケージの **アップグレード**:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get dist-upgrade
```

個別パッケージ単位の更新は行わない

dnf や **zypper** とは違って個別パッケージ単位での更新／アップグレードは行いません。パッケージ全体がひとつのユニットとして扱われます。

```
$ sudo apt-get install [package]
```

新しいパッケージをインストールします。または、既にインストールされているパッケージを更新します。

```
$ sudo apt-get remove [package]
```

構成ファイルを削除せずに、パッケージをシステムから削除します。

```
$ sudo apt-get --purge remove [package]
```

システムから、パッケージとその構成ファイルを削除します。

```
$ sudo apt-get update
```

パッケージインデックスファイルをリポジトリソースと同期させます。利用可能なパッケージのインデックスを `/etc/apt/sources.list` で指定された（ひとつまたは複数の）場所からフェッチします。

```
$ sudo apt-get upgrade
$ sudo apt-get dist-upgrade
```

`upgrade` は既にインストールされているパッケージに、利用可能なすべての更新を適用します。よく誤解されるようですが `dist-upgrade` はディストリビューションを新しいバージョンに更新するわけではありません。

dnf や **zypper** とは違って **upgrade** を実行する前には **update** を実行する必要があります。**Red Hat** 環境に慣れている人はここに混乱するかもしれません。

10.5 クリーンアップ

クリーンアップ

- 古いバージョンの **Linux** カーネルなど使わなくなったパッケージを取り除く:

```
$ sudo apt-get autoremove
```

- インストール済パッケージのキャッシュファイルやアーカイブを消去

```
$ sudo apt-get clean
```

これにより、多くのスペースが開放される

10.6 演習

デモ教材ビデオ

using_package_gui_ubuntu_demo.mp4

課題 10.0: apt が必要

Debian、Ubuntu、Linux Mint 環境を利用

本演習を実行するためには **Debian**、**Ubuntu**、**Linux Mint** など **Debian** ベースのシステムへのアクセスが必要です。

課題 10.1: APT の基本コマンド

1. アップデートの有無の確認。
2. インストール済のカーネル関係のパッケージのリスト、インストール済もしくはインストール可能なパッケージのリスト
3. **apache2-dev** パッケージやその他の未インストール パッケージをインストールする場合、以下のようにします。

   ```
   $ apt-cache pkgnames
   ```

   本コマンドは全リストを表示します。ワイルドカードを使うことで、リストを短くすることも可能です。

解 10.1

1. パッケージインデックスファイルをリモートのレポジトリと同期させます。

   ```
   $ sudo apt-get update
   ```

   実際にアップグレードします。

   ```
   $ sudo apt-get upgrade
   $ sudo apt-get -u upgrade
   ```

   (以前に説明したように、`dist-upgrade` を用いることもできます)。最初の形式はインストールのみ実行します。

2. ```
   $ apt-cache search "kernel"
   $ apt-cache search -n "kernel"
   $ apt-cache pkgnames "kernel"
   ```

   2 番目と 3 番目のコマンドは、名称に `kernel` を持つものを表示します。

   ```
   $ dpkg --get-selections "*kernel*"
   ```

   本コマンドは、インストールされたパッケージのみを表示します。**Debian** ベースのシステムでは、カーネル関係のパッケージを表示するために `kernel` ではなく `linux` を用います。なぜなら、パッケージの名称が `kernel` を含まないことが多々あるからです。

3. ```
   $ sudo apt-get install apache2-dev
   ```

課題 10.2: パッケージの情報を参照するために apt コマンドを使う

以下の情報を参照するために、**apt-cache** と **apt-get** を使用します。(**dpkg** は使いません):

1. 名称や説明中に **bash** を参照するすべてのパッケージ。

2. インストール済またはインストール可能な **bash** パッケージ。
3. **bash** に関するパッケージ情報。
4. **bash** パッケージの依存情報。

root と一般のユーザーの両方で上記コマンドを実行し、その違いを確認してください。

解 10.2

1. `$ apt-cache search bash`
2. `$ apt-cache search -n bash`
3. `$ apt-cache show bash`
4. `$ apt-cache depends bash`
   `$ apt-cache rdepends bash`

課題 10.3: APT で、パッケージグループを管理する

metapackages を利用することで、**APT** は **dnf** が提供するものと同様のパッケージグループの管理機能を提供します。グループとして同時にインストール、削除する必要がある関連する **virtual packages** と考えることができます。

利用可能な **metapackages** のリストを表示します:

```
$ apt-cache search metapackage
```

```
bacula - network backup service - metapackage
bacula-client - network backup service - client metapackage
bacula-server - network backup service - server metapackage
cloud-utils - metapackage for installation of upstream cloud-utils source
compiz - OpenGL window and compositing manager
emacs - GNU Emacs editor (metapackage)
....
```

以下のようにして、１つのパッケージをインストールすると同様に、パッケージ群をインストールできます。

```
$ sudo apt-get install bacula-client
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  bacula-common bacula-console bacula-fd bacula-traymonitor
Suggested packages:
  bacula-doc kde gnome-desktop-environment
The following NEW packages will be installed:
  bacula-client bacula-common bacula-console bacula-fd bacula-traymonitor
0 upgraded, 5 newly installed, 0 to remove and 0 not upgraded.
Need to get 742 kB of archives.
After this operation, 1,965 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

未インストールのメタパッケージを選択して、削除します。

章 11

システム監視

注目

** これらのセクションの一部または全体をオプション扱いとする場合があります。これらには補足資料、専門トピック、または高度な話題が含まれインストラクターが教室の状況や時間制約に応じてこれらの内容を紹介するかを判断します。

11.1 システムとネットワークの監視

監視ツール

- 多くのコマンドラインユーティリティがある
  - 各サブシステムについては、以降の章で紹介する
- ほとんどは `/proc` や `/sys` から情報を取得
- 全ての **Linux** ディストリビューションがグラフィカルなシステムモニターを提供:
  - **gnome-system-monitor**
  - **ksysguard**

大部分の詳細情報を非表示とした、表示がシンプルなグラフィカルシステムモニタも多くありますが、このコースではコマンドラインツールだけを紹介します。

Linux ディストリビューションには、パフォーマンスとプロファイリングに関する多くの標準ツールがインストールされています。それらの多くは、他の **UNIX** 系のオペレーティングシステムでもよく知られていますが **Linux** 専用に開発されたものもあります。

これらのツールのほとんどは、マウントされた **疑似ファイルシステム**、特に `/proc` と二次的な `/sys` を使用します。どちらもファイルシステムとカーネル構成を調べる章ですでに説明しています。ここでも、この 2 つを見ていきます。

疑似ファイルシステムである `/proc` と `/sys` には、システムに関する多くの情報が含まれています。さらに、これらのディレクトリツリーのエントリの多くは書き込み可能となっており、システムの動作を変更するためにも使用されます。大部分の疑似ファイルへの書き込みには **root** ユーザー権限が必要です。

`/dev` などはすべてメモリー内に存在する疑似ファイルシステムです。システムが実行されていないときにディスクパーティションを見ると、マウントポイントとして使用される空のディレクトリだけが存在します。

情報は表示を更新する時にのみ収集されます。定期的なポーリングで更新されているわけではありません。

ネットワーク監視ツール

Table 11.1: **ネットワーク監視ユーティリティ**

ユーティリティ	目　的	パッケージ
netstat	詳細なネットワーク統計	netstat
iptraf	ネットワークインターフェースの情報を収集	iptraf
tcpdump	ネットワークパケットを通信量の詳細分析	tcpdump
wireshark	ネットワーク通信の詳細分析	wireshark

時間制約から、本コースではネットワーク監視の詳細まで紹介出来ませんが、ネットワーク設定ついては後ほど説明します。
このトピックは **LFS311: Linux for System Engineers** で紹介する予定です。

11.2 sar **

sar

- **sar** は **S**ystems **A**ctivity **R**eporter の略である
- データを収集し、レポートを生成する多目的ツール
- **sadc** は統計情報を収集し、定期的に `/var/log/sa` に保存する
- **sar** は以下のコマンドで起動する:

  `$ sar [ options ] [ interval ] [ count ]`

  レポートは `interval` 秒後に合計 `count` 回繰り返される
- オプションを付けないと CPU 利用率をレポートする

sar は **S**ystems **A**ctivity **R**eporterSystems の略です。システムの稼働状況とパフォーマンスデータを収集し、人が読める形のレポートを作成するための汎用ツールです。

Linux システムでは **sar** のバックエンドは **sadc**（system activity data collector）で、実際に統計情報を `/var/log/sa` ディレクトリに情報を保存します。デフォルトの実行頻度は毎日ですが、変更可能です。データ収集はコマンドラインから起動できます。通常の定期的な収集は `/etc/cron.d/sysstat` に保存されている **cron** ジョブとして開始されます。

sar は（デフォルトの場所から、または `-f` オプションで指定されたファイルから）データを読み込みレポートを作成します。

```
student@ubuntu:~$ sudo sar 3 3
Linux 5.11.0 (ubuntu)   03/22/2021      _x86_64_        (4 CPU)

12:37:23 PM     CPU     %user     %nice   %system   %iowait    %steal     %idle
12:37:26 PM     all      0.08      0.00      0.00      0.08      0.00     99.83
12:37:29 PM     all      0.00      0.00      0.00      0.00      0.00    100.00
12:37:32 PM     all      0.00      0.00      0.00      0.00      0.00    100.00
Average:        all      0.03      0.00      0.00      0.03      0.00     99.94
student@ubuntu:~$
```

図 11.1: **sar の利用**

sar の例

- ページング統計情報と I/O と転送速度の統計情報のレポート

```
c8:/tmp># GETTING PAGING STATISTICS
c8:/tmp>sar -B  3 3
Linux 5.11.8 (c8)       03/22/2021      _x86_64_        (8 CPU)

02:49:22 PM  pgpgin/s pgpgout/s   fault/s  majflt/s  pgfree/s pgscank/s pgscand/s pgsteal/s    %vmeff
02:49:25 PM      0.00    974.67    109.33      0.00   7483.67      0.00      0.00      0.00      0.00
02:49:28 PM      0.00    169.33     94.33      0.00   1268.00      0.00      0.00      0.00      0.00
02:49:31 PM      0.00      9.33    106.33      0.00   1095.00      0.00      0.00      0.00      0.00
Average:         0.00    384.44    103.33      0.00   3282.22      0.00      0.00      0.00      0.00
c8:/tmp># # GETTING I/O AND TRANSFER RATE STATISTICS
c8:/tmp>sar -b 3 3
Linux 5.11.8 (c8)       03/22/2021      _x86_64_        (8 CPU)

02:50:09 PM       tps      rtps      wtps   bread/s   bwrtn/s
02:50:12 PM    174.67      0.00    174.67      0.00   2002.67
02:50:15 PM      0.00      0.00      0.00      0.00      0.00
02:50:18 PM      1.00      0.00      1.00      0.00     13.33
Average:        58.56      0.00     58.56      0.00    672.00
c8:/tmp>
```

図 11.2: **sar で I/O 統計情報を取得**

以下は **sar** の主要なオプション、モードのリスト。サブオプションがあるものもある:

Table 11.2: **sar オプション**

オプション	意　味
-A	ほぼすべての情報を取得
-b	I/O と転送速度の統計情報の取得（iostat と同じ）
-B	ページフォールトを含むページング統計情報の取得
-d	ブロックデバイスのアクティビティ情報の取得（iostat -x と同じ）
-n	ネットワーク統計情報の取得
-P	CPU ごとの統計情報の取得（sar -P ALL 3 と同じ）
-q	キューの長さ（実行キュー、プロセス、スレッド）の取得
-r	メモリー使用の統計情報の取得
-S	スワップ使用の統計情報の取得
-u	CPU 使用率の取得（デフォルト）
-v	inode とファイルおよびファイルハンドルに関する統計情報の取得
-w	コンテキストスイッチの統計情報の取得
-W	スワップ統計情報、1 秒あたりに入出力したページ情報の取得
-f	-o オプションで作成された指定ファイルから情報を抽出
-o	読み取り値を保存するファイルを指定。後で -f オプションで読み取る

11.3 システムログファイル

ログファイル

- 大部分のファイルは /var/log にある
- **Linux** ディストリビューションによりファイル名は異なる　例えば
  - **RHEL** では /var/log/messages
  - **Ubuntu** では /var/log/syslog

  新しく出力される行を連続的に表示する

```
$ sudo tail -f /var/log/messages
```

  または

```
$ dmesg -w
```

  カーネルに関連したメッセージだけを表示する

システムログファイルは、監視とトラブルシューティングに不可欠です。**Linux** ではこれらのメッセージを /var/log の下のさまざまなファイルで見ることができます。

メッセージ処理の基本的な制御は、多くの **UNIX** 系のオペレーティングシステムに共通の **syslogd**（最近のシステムでは通常 **rsyslogd**）デーモンによって制御されます。新しい **systemd** ベースのシステムでは syslogd の代わりに **journalctl** を使用できますが、通常は **syslogd** を保持しておいて **journalctl** と連携します

重要なメッセージは、ログファイルだけでなくシステム **コンソール** ウィンドウにも送信されます。グラフィカルインターフェースを実行していない場合、または仮想端末にいる場合は、直接そこに表示されます。さらにこれらのメッセージは /var/log/messages（または **Ubuntu** の /var/log/syslog）にコピーされますが、**X** か **Wayland** を実行している場合はメッセージを直ちに表示するにはいくつかの手順を実行する必要があります。

重要なログファイル

/var/log の下には、いくつかの重要なログファイルがある

Table 11.3: **システムログファイル**

ファイル	目　的
boot.log	システムブートメッセージ
dmesg	ブート以降に保存されるカーネルメッセージ （カーネルメッセージバッファの現在の内容を表示するには **dmesg** と入力）
messages or syslog	すべて重要なシステムメッセージ
secure	セキュリティ関連のメッセージ

ログメッセージを見るには、端末ウィンドウを開いて `tail -f /var/log/messages` と打つのが良いでしょう。**GNOME** デスクトップでは `System->Administration->System Log` または `Applications->System Tools->Log File Viewer` をクリックしてメッセージにアクセスすることもできます。デスクトップメニューか、その他のデスクトップに利用可能な類似のリンクがあります。

ログファイルが際限なく肥大化するのを防ぐため **logrotate** プログラムが周期的に実行され（デフォルトでは）4 世代の記録を残します。（ログファイルを圧縮するオプションもあります）この動作は /etc/logrotate.conf によって制御されます。

11.4 演習

デモ教材ビデオ

using_system_monitoring_demo.mp4

課題 11.1: stress または stress-ng を使う

stress は the University of Oklahoma の Amos Waterland が **C** 言語で書き **GPL v2** でライセンスされています。さまざまなワークロードを生成できるので、自由にストレスを設定してシステムにかけることができます。

stress-ng は基本的に **stress** のやり方やオプションを変えずにエンハンスされています。アクティブにメンテナンスされています: https://wiki.ubuntu.com/Kernel/Reference/stress-ng を見てください。

ほとんどのディストリビューションは **stress-ng** をパッケージとして持っています。しかし **RHEL/CentOS** にはパッケージが無いため、EPEL レポジトリから入手する必要があります。現時点では EPEL 8 にはありませんが EPEL 7 から問題無くインストールできます。

ソースからインストールする

運が良ければ、ディストリビューションのパッケージシステムを使って **stress** や **stress-ng** をインストールできます。

または **git** を使って、https://kernel.ubuntu.com/git/cking/stress-ng.git/ から **stress-ng** のソースを入手できます。（tar ボールをダウンロードして使うこともできます。）ダウンロード、コンパイル、インストールの手順を示します:

```
$ git clone git://kernel.ubuntu.com/cking/stress-ng.git
$ cd stress-ng
$ make
$ sudo make install
```

インストールできたら、次のコマンドでオプションの一覧を表示します:

```
$ stress-ng --help
```

また、詳細なドキュメントがみたいときには:

```
$ info stress-ng
```

コマンド実行例:

```
$ stress-ng -c 8 -i 4 -m 6 -t 20s
```

このコマンドは:

- CPU を使う（CPU インテンシブ）プロセスを 8 個生成します。それぞれ、`sqrt()` の計算を実行します。
- I/O を使う（I/O インテンシブ）プロセスを 4 個生成します。それぞれ、`sync()` を実行します。
- メモリーを使う（メモリーインテンシブ）プロセスを 6 個生成します。それぞれ、`malloc()` を実行して、デフォルトで 256MB を割り当てます。大きさは、`--vm-bytes 128M` の範囲で変更可能です。
- 20 秒ごとにストレス テストを実行します。

stress-ng をインストールしたらアプリケーションメニューか、コマンドラインからグラフィカルなシステムモニターを実行します。通常、それは **gnome-system-monitor** か **ksysguard** です。

それでは、システムにストレスをかけましょう。どのくらいの負荷にするかは、CPU の数、**RAM** の大きさなどのシステムが持つリソースに依存します。

実行例です。

```
$ stress-ng -m 4 -t 20s
```

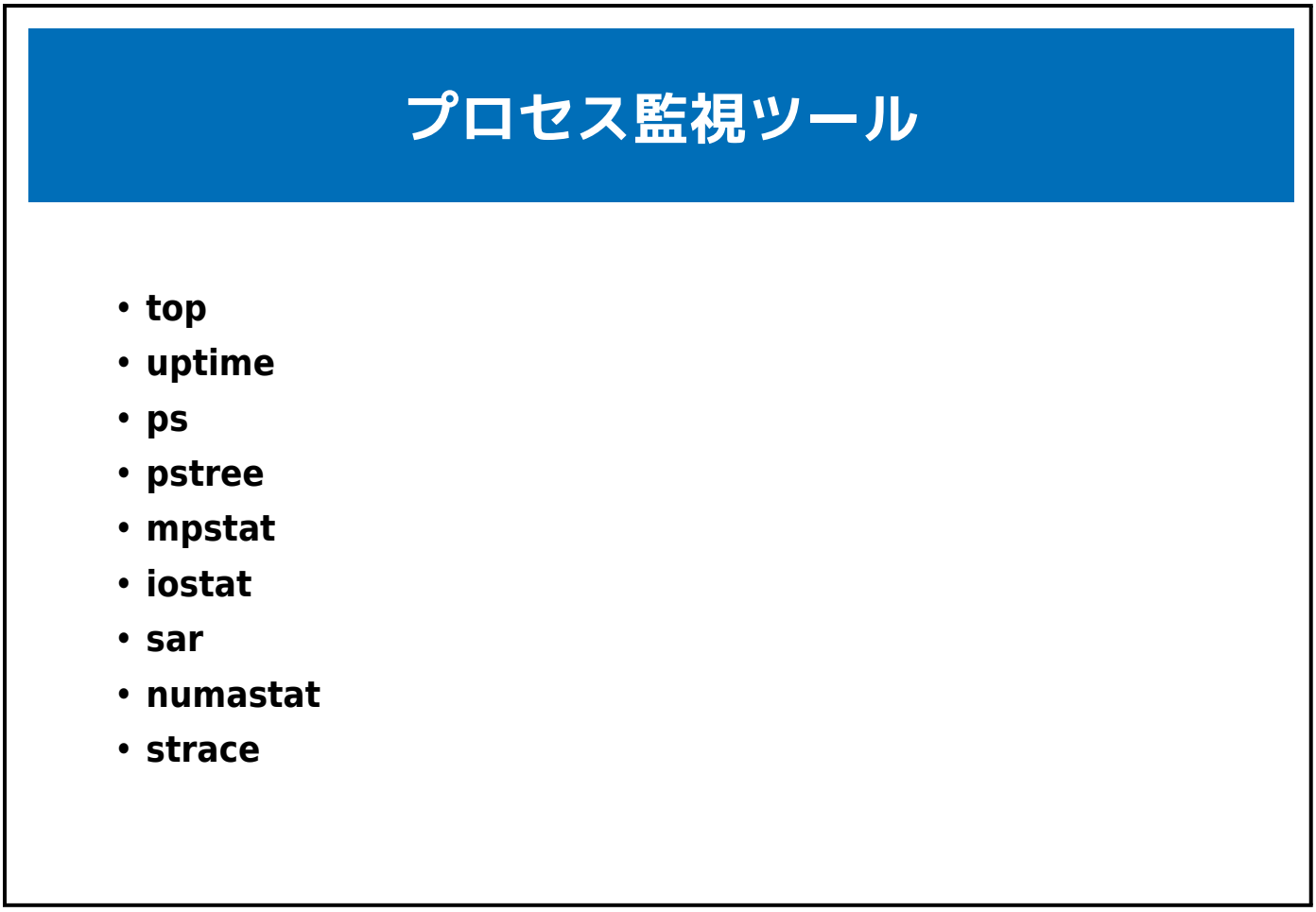
これは、システムにメモリーストレスのみをかけます。

スイッチの組み合わせを試して、それぞれのインパクトを見てください。高負荷の状態を作り出すのに **stress-ng** が適していることがわかるでしょう。

章 12

プロセス監視

12.1 プロセス監視



Linux 管理者は、プロセス監視に **ps**、**pstree**、**top** などの多くのユーティリティを使用します。これらはすべて **UNIX** 系のオペレーティングシステムで長く使われてきたものです。

プロセス監視の主要ツールのリストを確認してみましょう:

Table 12.1: **プロセスと負荷の監視ユーティリティ**

ツール	目　的
top	動的に更新、プロセスの活動状況の表示
uptime	システムの稼働時間と平均負荷の表示
ps	プロセスに関する詳細情報の表示
pstree	プロセスとその親子関係のツリー表示
mpstat	マルチプロセッサの使用量の表示
iostat	CPU 使用率と I/O 統計情報の表示
sar	システムのアクティビティに関する情報の収集・表示
numastat	NUMA（Non-Uniform Memory Architecture）に関する情報の表示
strace	プロセスが行うすべてのシステムコールに関する情報を表示

`/proc` もシステム上のプロセスや他のアイテムの監視に役立ちます。

12.2 ps

ps を使用したプロセス状態の表示

- /proc から情報を収集
- 何をどう表示するかは、設定次第でフレキシブルに調整可能
- 代表的なオプションの使用例:

```
$ ps aux
$ ps -elf
$ ps -eL
$ ps -C "bash"
```

ps は、プロセスに関連付けられた特性と統計情報を表示する有用なツールです。これらの情報はすべて、プロセスに関連付けられた /proc ディレクトリから収集されます。

このコマンドユーティリティは、すべての **UNIX** 系のオペレーティングシステムに異なる形で存在します。その多様性は **Linux** バージョンの **ps** のオプションに現れており、次の３つのカテゴリに分類されます。

1. **UNIX** では前に「-」を付ける必要があり、グループ化ができます。
2. **BSD** では前に「-」を付けることはできません。グループ化はできます。
3. **GNU** の長いオプションは、それぞれの前に「--」が必要です。

オプション付け方の違いは混乱を生みます。ほとんどのシステム管理者は、通常は１つか２つの標準的な組み合わせを使用します。

```
c8:/tmp>ps auxf
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           2  0.0  0.0      0     0 ?        S    06:43   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   06:43   0:00  \_ [rcu_gp]
....
coop        3068  0.0  0.2 846052 38012 ?        Ssl  06:53   0:11  \_ /usr/libexec/gnome-terminal-server
coop        3103  0.0  0.0 237040  5648 pts/0    Ss   06:53   0:00  |   \_ bash
coop       51808  0.0  0.0 248280  6272 pts/0    S+   13:41   0:00  |   |   \_ git log
coop       51809  0.0  0.0 219456  2496 pts/0    S+   13:41   0:00  |   |       \_ /usr/bin/less
coop        3158  0.0  0.0 237512  5980 pts/1    Ss+  06:53   0:00  |   \_ bash
c8:/tmp>
```

図 12.1: **BSD オプションでの ps の使用例**

ps 出力のカスタマイズ

- -o オプションの後にコンマで区切られたフィールド識別子のリストを指定すると、カスタマイズされた ps フィールドのリストを出力できる
  - pid: プロセス ID 番号
  - uid: プロセスオーナーのユーザー ID 番号
  - cmd: すべての引数を指定したコマンド
  - cputime: CPU 時間の累積
  - pmem: プロセスのマシン上の物理メモリー利用率、パーセント表示
- その他の多くの出力オプションは **ps** の **man** ページを参照

```
c8:/tmp>ps -o pid,user,uid,priority,cputime,pmem,size,command
    PID USER       UID PRI     TIME %MEM  SIZE COMMAND
  47619 coop      1000  20 00:00:00  0.0  2656 bash
  49543 coop      1000  20 00:00:00  0.0  1100 ps -o pid,user,uid,priority,cputime,pmem,size,command
c8:/tmp>ps -o pid,user,uid,priority,cputime,pmem,size,command
```

図 12.2: **ps 出力のカスタマイズ**

これ以外にオプション -elf も良く利用されます:

```
c8:/tmp>ps -elf
F S UID          PID    PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S root           1       0  0  80   0 - 60929 -      06:45 ?        00:00:02 /usr/lib/systemd/systemd --swit
ched-root --system --deserialize 18
1 S root           2       0  0  80   0 -     0 -      06:45 ?        00:00:00 [kthreadd]
1 I root           3       2  0  60 -20 -     0 -      06:45 ?        00:00:00 [rcu_gp]
....
0 S coop        6302    2365  1  80   0 - 182213 -     06:50 tty2     00:02:33 gnome-system-monitor
0 S coop       47591   37378  0  80   0 - 288155 -     07:55 pts/2    00:00:22 evince LFS301.pdf
0 S coop       47602    2089  0  80   0 - 53719 -      07:55 ?        00:00:00 /usr/libexec/evinced
....
0 S coop       54206    2715  0  80   0 - 58937 -      09:23 pts/4    00:00:00 bash
0 S root       54295    1140  0  80   0 - 54263 -      09:23 ?        00:00:00 sleep 60
4 R coop       54296   54206  0  80   0 - 67123 -      09:23 pts/4    00:00:00 ps -elf
c8:/tmp>
```

図 12.3: **UNIX オプションでの ps の使用例**

12.3 pstree

pstree の使用方法

- ユーザーにプロセスの親子関係を表示
- -p でプロセス ID を表示
- -H [pid] で [pid] とその祖先（分岐元）を表示

```
$ pstree -aAps 31478
systemd,1 --switched-root --system --deserialize 21
  `-vmplayer,31478
      |-{vmplayer},31570
      |-{vmplayer},31593
      |-{vmplayer},32572
      `-{vmplayer},32698
```

pstree はプロセスの親子関係と、マルチスレッドアプリケーションについて視覚的に示します:

```
$ pstree -aAp 2408
```

```
1  bash,2408
2  |-emacs,24998 pmonitor.tex
3  |   |-{emacs},25002
4  |   `-{emacs},25003
5  |-evince,18036 LFS201-SLIDES.pdf
6  |   |-{evince},18040
7  |   |-{evince},18046
8  |   `-{evince},18047
```

オプションの説明については **pstree** の **man** ページを参照してください。上記は pid=2408 の情報表示を選択しています。
子プロセスの１つ (**evince**、pid=18036) は、子プロセスが３つあることに注目してください。これを別の方法で確認するには:

```
$ ls -l /proc/18036/task
```

```
1  total 0
2  dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18036
3  dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18040
4  dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18046
5  dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18047
```

12.4 top

top

- CPU 使用率の高いプロセスを表示する目的で利用
- プロセスは CPU 利用率の高い順に並べられる
- セキュアモード (top s) でない場合はプロセスにシグナルを送信可能
  - k キーを押す
  - プロンプトが出たら PID を入力
  - プロンプトにシグナル番号を入力
- 古いユーティリティなので多数のオプションを使って対話的操作が可能
- h を押すとキー割り付けを表示
- 並べ替えオプションなどは使いやすい

h でヘルプ表示が出ます。さまざまな条件で、表示を並べ替えることが可能です。ユーザーが `Ctrl-C` で中断するまで 5 秒間隔で更新します。(時間は設定可能です)

```
top - 09:42:28 up  2:56,  1 user,  load average: 1.20, 1.16, 1.09
Tasks: 339 total,   1 running, 338 sleeping,   0 stopped,   0 zombie
%Cpu0  :  1.0 us,  0.3 sy,  0.0 ni, 98.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  1.3 us,  0.3 sy,  0.0 ni, 98.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.7 us,  0.3 sy,  0.0 ni, 99.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.7 us,  0.3 sy,  0.0 ni, 99.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.3 us,  0.3 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  2.0 us,  0.7 sy,  0.0 ni, 97.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  1.7 us,  0.3 sy,  0.0 ni, 98.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  : 17.7 us,  1.3 sy,  0.0 ni, 81.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  15874.0 total,   3964.8 free,   3177.2 used,   8732.0 buff/cache
MiB Swap:   8096.0 total,   8086.7 free,      9.3 used.  11455.8 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   3605 coop      20   0 3793748 484588 123312 S  18.6   3.0   1:07.44 thunderbird
   2131 coop      20   0 1086964  83868  50788 S   3.3   0.5   8:21.00 Xorg
   2107 coop       9 -11 2729100  20408  13368 S   2.7   0.1   2:10.98 pulseaudio
   6302 coop      20   0  728788  49084  36336 S   1.7   0.3   2:52.34 gnome-system-mo
  44118 coop      20   0 3472692 219620 116372 S   1.3   1.4   0:55.83 spotify
   2365 coop      20   0 4663756 432076  73556 S   1.0   2.7   7:15.12 gnome-shell
   3691 coop      20   0   36.8g 244560  89984 S   1.0   1.5   1:54.33 slack
  50372 coop      20   0 1225676  67700  58684 S   0.7   0.4   0:25.04 slack
    405 root       0 -20       0      0      0 I   0.3   0.0   0:09.96 kworker/u17:2-i915_flip
   2499 root      20   0  414964  31624   9704 S   0.3   0.2   0:08.90 sssd_kcm
  44149 coop      20   0 2183944 226052  78704 S   0.3   1.4   0:23.67 spotify
  56244 coop      20   0  275384   5384   4544 S   0.3   0.0   0:00.21 top
      1 root      20   0  243716  12144   8300 S   0.0   0.1   0:02.18 systemd
```

図 12.4: **top の利用**

/proc の応用

- /proc ファイルシステムはカーネルデータ構造へのインターフェース
- /proc には、各アクティブなプロセスに対応したプロセス ID 番号（PID）名のサブディレクトリがある
- /proc/self は現在実行中のプロセス
- /proc ディレクトリ内のいくつかのパラメータは変更できる
- 詳細は proc の man ページを参照

下記のサンプル画面に示した /proc のサブディレクトリには、数字の名前のサブディレクトリが含まれています。サブディレクトリ名の数字は、システム上の全てのアクティブなプロセスの PID を表しています。それぞれのサブディレクトリには、そのプロセスに関する情報が格納されたファイルがあります。

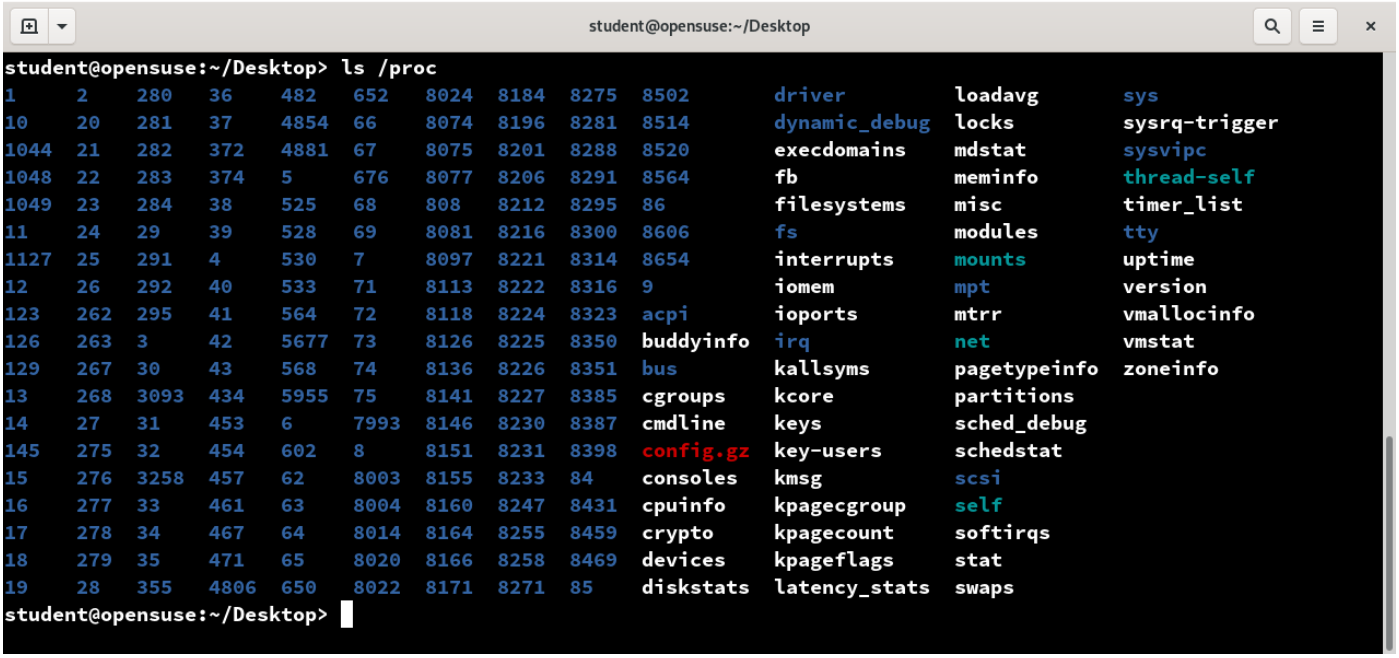


図 12.5: **/proc の中身**

12.5 演習

課題 12.1: プロセス

1. オプション -ef を付けて **ps** を実行してください。その次にオプション aux で実行します。出力に違いに注意してください。
2. **ps** を実行すると、プロセス ID、プライオリティ、nice 値とプロセスのコマンドラインが表示されます。
3. コマンド入力行で **bash** を入力し、**bash** の新しいセッションを開始します。nice 値を10 に指定した **nice** コマンドを使い、もうひとつ **bash** セッションを開始します。
4. ステップ 2 と同様に **ps** を実行し、プライオリティと nice 値の違いに注目してください。2 つの **bash** セッションのプロセス ID にも注意してください。
5. **renice** を使ってどちらかの **bash** セッションの nice 値を変更します。もう一度プライオリティと nice 値の変化を監視してください。
6. **top** を実行して、出力の変化を見てください。終了するには q を押下してください。

解 12.1

1. ```
   $ ps -ef
   $ ps aux
   ```

2. ```
   $ ps -o pid,pri,ni,cmd
   ```

   ```
     PID PRI  NI CMD
    2389  19   0 bash
   22079  19   0 ps -o pid,pri,ni,cmd
   ```

   （注意：パラメータの間には空白は不要です。）

3. ```
   $ bash
   $ nice -n 10 bash
   $ ps -o pid,pri,ni,cmd
   ```

   ```
    2389  19   0 bash
   22115  19   0 bash
   22171   9  10 bash
   22227   9  10 ps -o pid,pri,ni,cmd
   ```

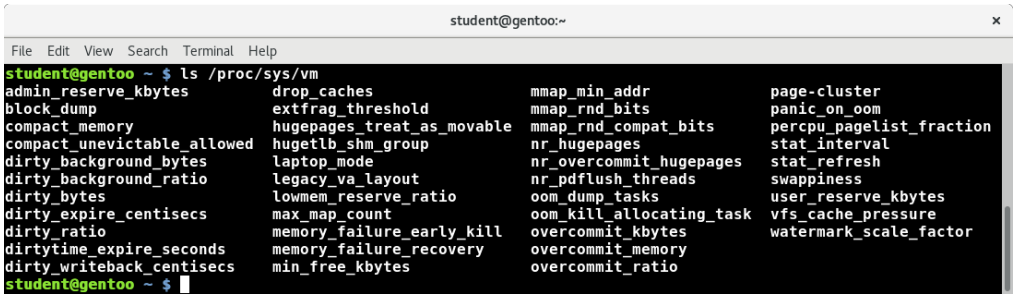
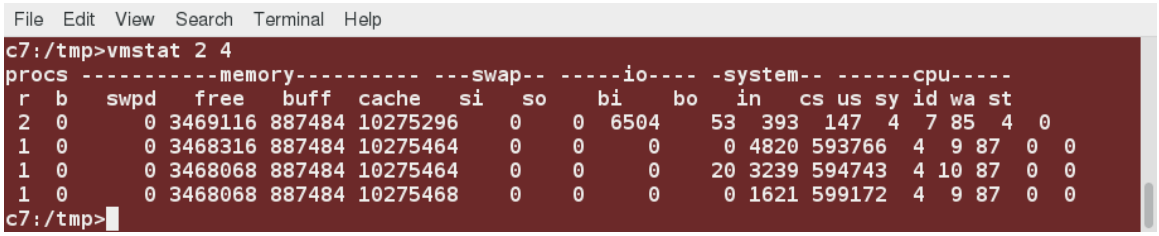
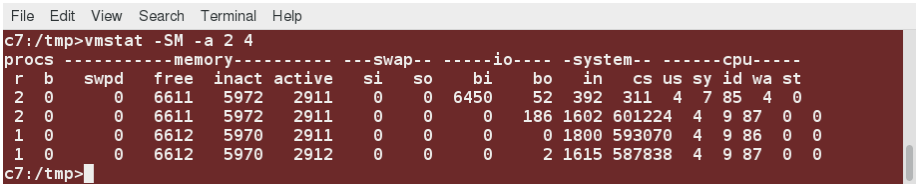
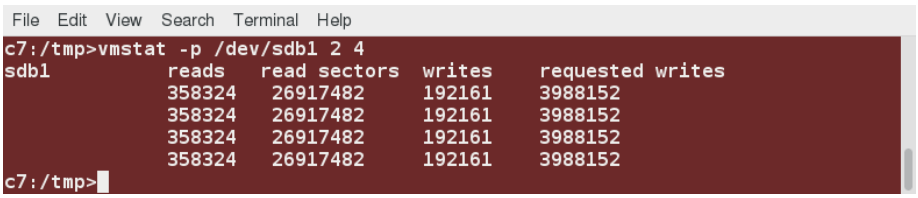
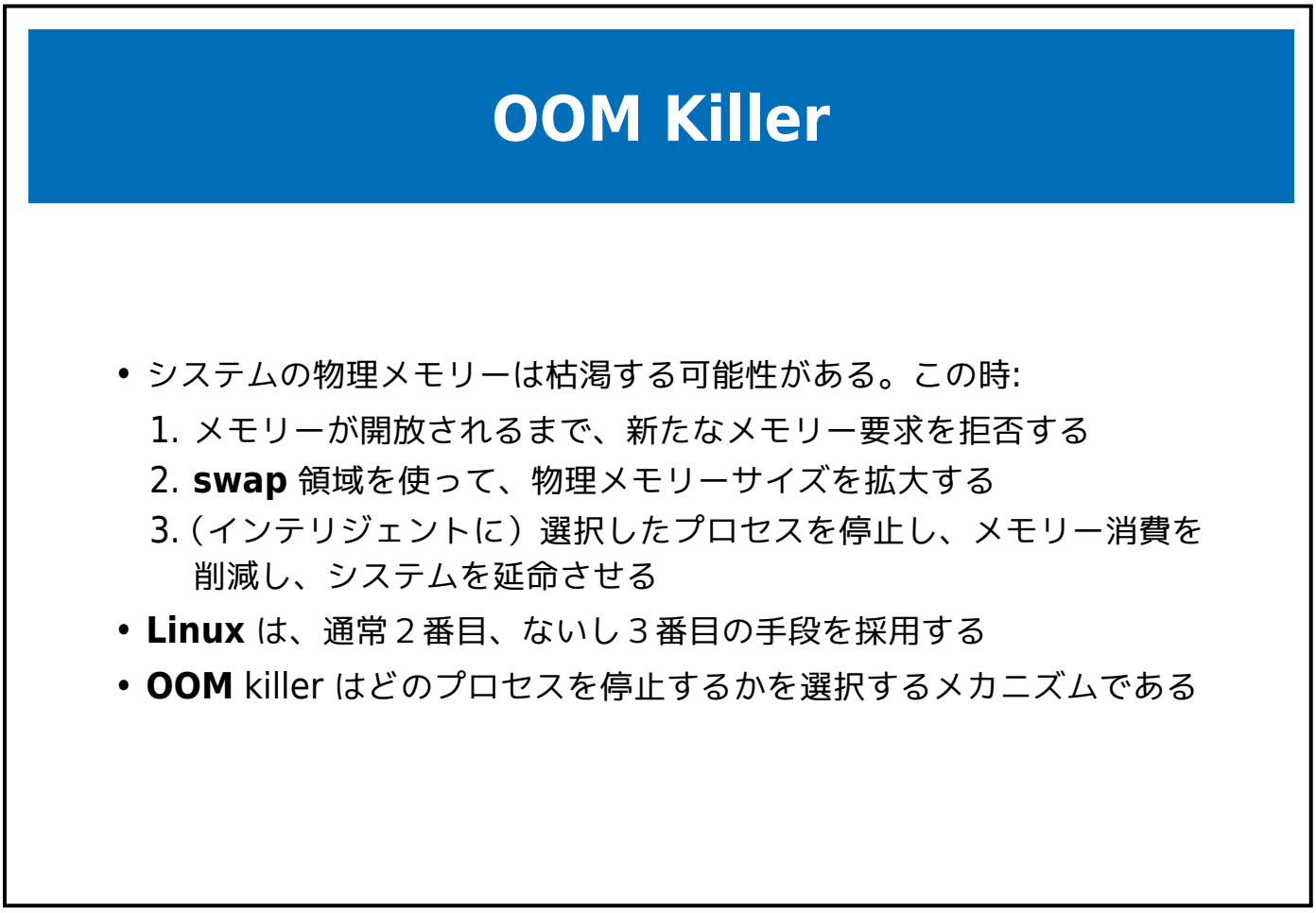
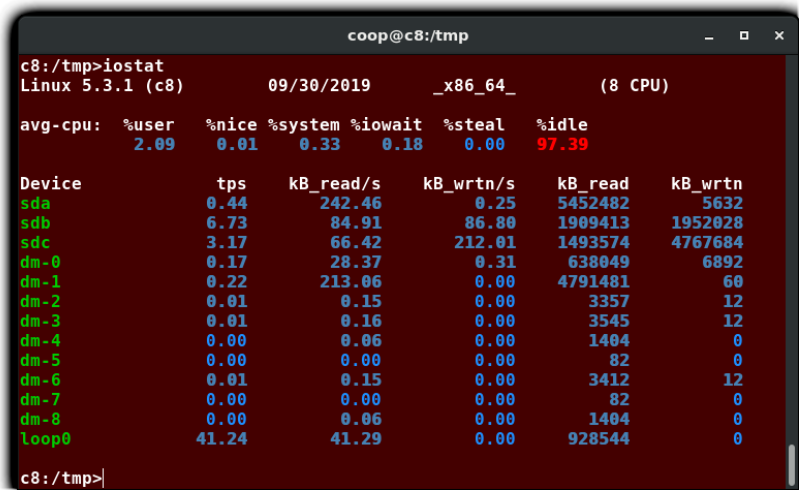
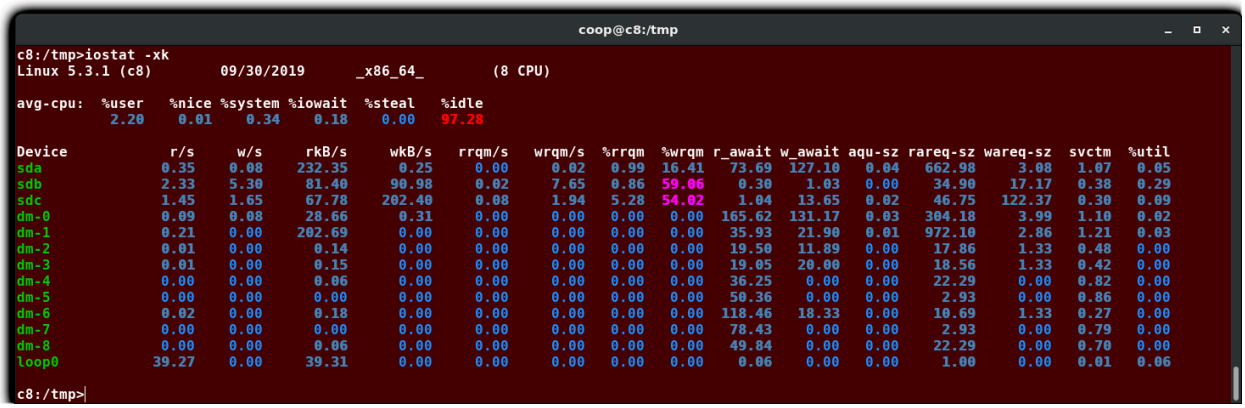
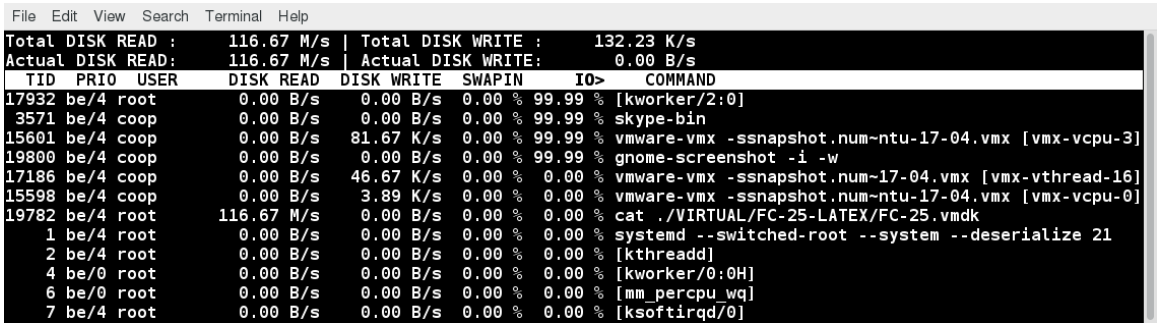
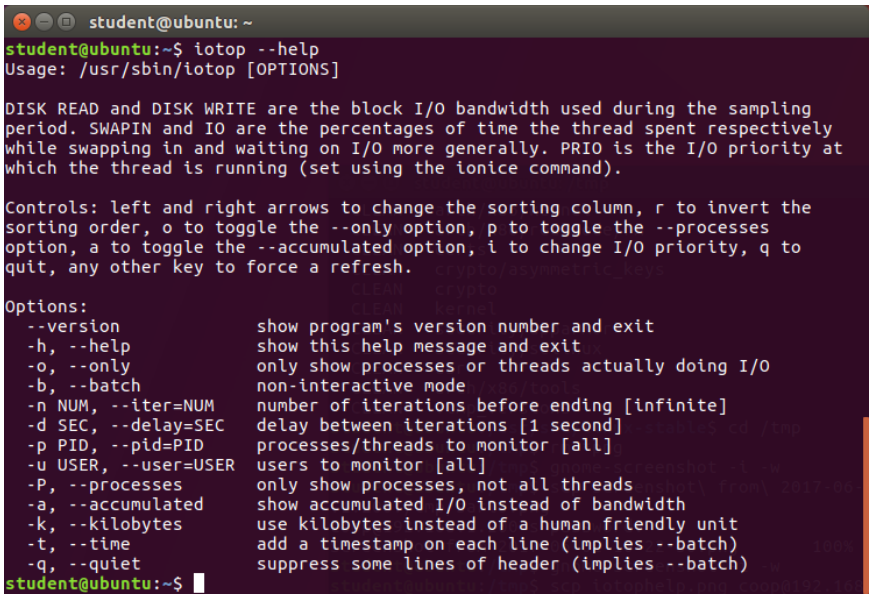
4. ```
   $ renice 15 -p 22171
   $ ps -o pid,pri,ni,cmd
   ```

   ```
     PID PRI  NI CMD
    2389  19   0 bash
   22115  19   0 bash
   22171   4  15 bash
   22246   4  15 ps -o pid,pri,ni,cmd
   ```

5. ```
   $ top
   ```

課題 12.2: プロセスの状態を監視する

1. /dev/urandom から読み出し、/dev/null へ書き込む **dd** をバックグラウンドで実行します。
2. プロセスの状態をチェックします。どうなっているべきでしょうか？
   ```

3. **fg** コマンドで、プロセスをフォアグラウンドに持ってきます。次に Ctrl-Z を入力します。何が起こるでしょうか。プロセスの状態を見てみます。どうですか。
4. **jobs** プログラムを実行します。何が表示されますか。
5. ジョブをフォアグラウンドの持ってきます。次に他のウィンドウから **kill** を使って終了させます。

解 12.2

1. `$ dd if=/dev/urandom of=/dev/null &`
2. `$ ps -C dd -o pid,cmd,stat`

```
25899 dd if=/dev/urandom of=/dev/ R
```

S か R になっているはずです。

3.
```
$ fg
$ ^Z
$ ps -C dd -o pid,cmd,stat
```

```
  PID CMD                         STAT
25899 dd if=/dev/urandom of=/dev/ T
```

状態は T になっているはずです。

4. **jobs** コマンドを入力します。出力は、どうなっていますか？

```
$ jobs
```

```
[1]+  Stopped                 dd if=/dev/urandom of=/dev/null
```

5. ジョブをフォアグラウンドに戻します。他の端末から **kill** コマンドで、終了させます。

```
$ fg
$ kill  25899
```

章 13

メモリーの監視と利用

13.1 メモリー監視とチューニング

メモリー監視

Linux におけるメモリー監視とチューニングを担う重要なツール

Table 13.1: **メモリー監視ユーティリティ**

ユーティリティ	目　的	パッケージ
free	メモリー使用量の概要の表示	procps
vmstat	動的に更新される仮想メモリー統計とブロック I/O の詳細情報	procps
pmap	プロセスのメモリーマップの表示	procps

```
$ free -m

              total        used        free      shared  buff/cache   available
Mem:          15901        2080        9317         992        4502       12457
Swap:          8095           0        8095
```

時代の進展に伴ってシステムのメモリーリソース要求は肥大化しましたが、同時に RAM の価格も下がったため、結果的にはパフォーマンスが向上ししました。

それでも、メモリー不足がシステム全体のパフォーマンスとスループットのボトルネックとなることはよくあります。
CPU と I/O サブシステムが、メモリーからのデータの取得や、メモリーへのデータの書き込みで待たされることがあるからです。

メモリーに関連したシステムの挙動監視、デバッグ、調整ツールは多数あります。

メモリーサブシステムのチューニングは複雑な作業です。まずメモリー使用量と I/O スループットが本質的に関連していることを理解する必要があります。大部分のメモリーは、ディスク上のファイルのコンテンツをキャッシュするために使用されます。

したがって、メモリーパラメータの変更が、I/O パフォーマンスに大きな影響を与える可能性があります。
逆に I/O パラメータを変更した結果、仮想メモリーサブシステムの効果が大きく影響を受ける可能性もあります。

/proc/meminfo

```
$ cat /proc/meminfo
```

```
MemTotal:       16265960 kB Writeback:             0 kB VmallocChunk:          0 kB
MemFree:        10481080 kB AnonPages:       2248272 kB Percpu:             6688 kB
MemAvailable:   12596456 kB Mapped:           811700 kB HardwareCorrupted:     0 kB
Buffers:          177824 kB Shmem:            659452 kB AnonHugePages:    716800 kB
Cached:          2768600 kB KReclaimable:     159792 kB ShmemHugePages:        0 kB
SwapCached:            0 kB Slab:             307548 kB ShmemPmdMapped:        0 kB
Active:          3320564 kB SReclaimable:     159792 kB FileHugePages:         0 kB
Inactive:        1566504 kB SUnreclaim:       147756 kB FilePmdMapped:         0 kB
Active(anon):    2381096 kB KernelStack:       15680 kB HugePages_Total:       0
Inactive(anon):   218792 kB PageTables:        74940 kB HugePages_Free:        0
Active(file):     939468 kB NFS_Unstable:          0 kB HugePages_Rsvd:        0
Inactive(file):  1347712 kB Bounce:              0 kB HugePages_Surp:        0
Unevictable:      437372 kB WritebackTmp:        0 kB Hugepagesize:       2048 kB
Mlocked:              32 kB CommitLimit:    29713776 kB Hugetlb:              0 kB
SwapTotal:      21580796 kB Committed_AS:   10239968 kB DirectMap4k:      258124 kB
SwapFree:       21580796 kB VmallocTotal:   34359738367 kB DirectMap2M:     9052160 kB
Dirty:              288 kB VmallocUsed:       24592 kB DirectMap1G:     8388608 kB
```

疑似ファイル /proc/meminfo には、どのようにメモリーが使用されているかについての数多くの情報が含まれます。

13.2 /proc/sys/vm

/proc/sys/vm

- 仮想メモリーの動作を設定するための、多くの調整つまみが含まれる
- ほとんど全ての項目は（root によって）書き込み可能
- このディレクトリーに具体的にどんな項目が見えるかは、カーネルバージョンなどに依存する
- 設定値は、直接書き込むことも **sysctl** 経由して変更することもできる

`/proc/sys/vm` のパラメータを調整するときのベストプラクティスは、一度には１項目だけを調整して効果を確認することです。重要な（相互に関連する）タスクは次のとおり:

- フラッシングパラメータ制御; すなわち、ダーティにできるページの数とそれをディスクに書き出す頻度の設定です。
- スワップ制御; ファイルコンテンツをどれだけオンメモリーに残すか、逆に領域不足時にスワップアウトさせるかの設定です。
- メモリーオーバーコミット量の制御です。多くのプログラムは、コピーオンライト（COW）により要求メモリーの全ては使用しません。

メモリーのチューニングは繊細です。特定のシステム状況／負荷下では有効な設定でも、他では全く効果が無いこともあります。



図 13.1: **/proc/sys/vm**

13.3 vmstat

vmstat

- メモリー、ページング、I/O、プロセッサアクティビティ、プロセスに関する情報を表示
- 多数のオプションがある

```
$ vmstat [options] [delay] [count]
```

- **サンプリング周期** [delay] が秒単位で指定されている場合、その間隔で count 回データが収集される
- 最初の行は、再起動以降の平均値
- ２行目以降は、指定間隔でのアクティビティを表示



図 13.2: **vmstat**

オプション -S m が指定された場合、メモリー統計の単位は KB ではなく MB になります。

-a オプションを使用すると **vmstat** はアクティブおよび非アクティブなメモリーに関する情報を表示します。アクティブページは最近使用されたページです。ページはクリーン（ディスクの内容が最新）またはダーティ（最終的にディスクに書き出す必要がある）のどちらかです。対照的に、非アクティブページは最近使用されていないのでクリーンである可能性が高く、メモリーに負荷がかかるとすぐに解放されます。

メモリーは、アクティブリストと非アクティブリストの間を行ったり来たりします。これは、新しく参照されたり、使用と使用の間隔が長くなったりするためです。



図 13.3: **vmstat の使用例**

一つのパーティションの簡単な統計を取りたいなら -p が利用できます。



図 13.4: **1 つのディスクに対して vmstat を使用する例**

13.4 Out of Memory Killer (OOM)



メモリーの負荷に対処する最も簡単な方法は、空きメモリーがある限りメモリー割り当てを成功させ、すべてのメモリーを使い切ったら失敗させることです。

２番目の方法は、ディスク上のスワップ領域を使用して、常駐メモリーの一部を書き出すことです。この場合（少なくとも理論的には)、使用可能なメモリーサイズの合計は実際の RAM のサイズにスワップ領域のサイズを加えたものになります。この戦略では、負荷がかかった時にどのメモリーのページをスワップアウトするの決定方法が課題となります。このアプローチでは、もしスワップ領域自体が一杯になったら、以降のメモリー要求は必ず失敗させる必要があります。

しかし Linux はもう少し賢く動きます。Linux はシステムにメモリーのオーバーコミットを許すのです。そのため RAM とスワップの合計サイズを超えるメモリー要求であっても許可するのです。これは無謀に思えるかもしれませんが（全てではないにしても）多くのプロセスは要求したメモリーの全ては使用しないのです。

例としては 1MB のバッファを確保するが、実際には数ページ分のメモリーのみを使用するプログラムがあります。また別の例として、子プロセスは fork されるたびに親のメモリー空間全体のコピーを受け取ります。Linux は COW（copy on write）を採用しているので、プロセスが実際にメモリーを変更するまで実際のコピーは発生しないのです。ただし、カーネルはコピーが発生する可能性は想定しておかなければなりません。

このように、カーネルはメモリーのオーバーコミットを許しますが、これはユーザープロセス用のページに対してのみです。カーネルが使用するページは、スワップさせることはできないので、要求時に常に割り当てられます。

OOM Killer のアルゴリズム

- (OOM killer が採用する) 推論アルゴリズムは、通常は動作には影響を与えないように考えられている。これより穏便な停止、または削減の//方法はあるだろうか？
- 各プロセスの `/proc/[pid]/oom_score` ファイルに格納された **badness** に基づいて、どのプロセスを停止させるか選択される
- 同ディレクトリにあるプロセスの `oom_adj_score` を利用して、各タスクの調整が可能

`/proc/sys/vm/overcommit_memory` の値を設定により、オーバーコミット動作を変更したり無効にすることもできます:

- 0:（デフォルト設定）オーバーコミットを許可しますが、明らかなオーバーコミットは拒否し root ユーザーには一般のユーザーよりもいくらか多くのメモリーを割り当てます。
- 1: すべてのメモリーリクエストのオーバーコミットを認めます。
- 2: オーバーコミットをオフにします。メモリーコミットの合計が、スワップ領域のサイズと構成変更可能なパーセンテージ（デフォルトでは 50）の RAM のサイズとの合計に達すると、メモリー要求は失敗します。このパーセンテージは `/proc/sys/vm/overcommit_ratio` で変更できます。

Andries Brouwer がこれについての面白い見解を示しました。(https://lwn.net/Articles/104185/):

> 「航空機会社は、より少ない燃料で飛行機を飛ばす方が格安であることを発見しました。飛行機はより軽くなり、燃料の使用はより少なくなり、お金の節約が実現しました。しかし、まれなケースですが燃料の量が不足し飛行機が墜落することがありました。この問題は、特別な OOF（out-of-fuel：燃料切れ）メカニズムを開発した会社のエンジニアによって解決されました。緊急の場合、乗客が一人選択され飛行機から放り出されました。（必要に応じてこの手順が繰り返されました。）多数の理論が開発され、放り出される犠牲者の適切な選択方法に関して多くの発表がありました。犠牲者はランダムに選ばれるべきか？それとも最も体重が重い人を選ぶべきか？それとも年長者を選ぶべきか？乗客は追い出されないようにお金を支払うべきだがそうすると犠牲者は機内で最も貧しい人になるのか？さらに例えば、最も重い人が選ばれた場合それがパイロットだったら特別な例外があるべきか？ファーストクラスの乗客は免除されるべきか？現在も OOF メカニズムは存在しており、時々稼働しています。そして燃料不足がなくても乗客を放り出します。エンジニアは、今もこの誤動作がどのように引き起こされるかを研究しています。」

13.5 演習

課題 13.1: OOM Killer を呼び出す

`/proc/swaps` を調べて、スワップ用のパーティションファイルが存在するか確認してください。

全てのスワップを停止してください。

```
$ sudo /sbin/swapoff -a
```

全てが完了したら、もとに戻すことを忘れないでください。

```
$ sudo /sbin/swapon -a
```

ここからシステムをメモリー不足の状態にします。一つの方法として以前にインストールした **stress-ng** プログラムを使います。以下のように実行します:

```
$ stress-ng -m 12 -t 10s
```

これは 10 秒ごとに 3GB を消費します。

OOM（メモリー不足：Out of Memory）killer が立ち上がり、プロセスを kill して状態を正常に戻そうとします。**dmesg** を実行するか、`/var/log/messages` または `/var/log/syslog` を監視したり、システムログを表示するための GUI を使ったりして、何が起こっているかを見ることができます。

どのプロセスが先に kill されるでしょうか？

章 14

I/O の監視とチューニング

注目

** これらのセクションの一部または全体をオプション扱いとする場合があります。これらには補足資料、専門トピック、または高度な話題が含まれインストラクターが教室の状況や時間制約に応じてこれらの内容を紹介するかを判断します。

14.1 I/O 監視

I/O の監視とディスクのボトルネック

- 入力／出力 (I/O) の挙動監視は、ピーク性能確保のための基本となる
- システムは I/O 性能律速（i/o-bound）、CPU 性能律速（CPU bound）、メモリー性能律速（memory bound）に分類される; 測定しないとどのタイプに該当するかは判定できない
- ３つの重要なツール:
  - **iostat**
  - **iotop**
  - **ionice**

ディスクのパフォーマンス問題は、メモリー不足や不適切なネットワークハードウェア、チューニング不足などと強く結びついている可能性があります。問題の切り分けは困難です。

CPU が I/O の完了待ちでアイドル状態にあるとか、ネットワークがバッファクリアを待っているような場合、システムは I/O 性能律速（i/o-bound）であると呼びます。

極端に I/O が遅すぎるのをメモリー不足と見誤るケースがあります。もし読み書きに使用されているメモリーバッファが溢れると、それはメモリー不足に見えるでしょう。実際には、バッファーへの読み込みや書き出しに時間がかかり過ぎなのが根本原因です。同様に、ネットワーク転送でも I/O に時間がかかり過ぎだと、ネットワークのスループットが犠牲になります。

リアルタイム監視とトレースは、ディスクのボトルネックを特定して軽減するために欠かせないツールです。ただし、発生頻度の低い問題や再現性の低い問題は、解決が困難です。

I/O チューニングは、関連する調整項目が多く複雑です。**I/O scheduling** についても後ほど説明します。

14.2 iostat

iostat

- 全ての I/O アクティビティを監視する基本的なツールである
- オプションを使って様々なレポートを出力できる

```
$ iostat [OPTIONS]  [devices] [interval] [count]
```



図 14.1: **iostat**

iostat は、システム上の I/O デバイスのアクティビティを監視するための基本的なツールです。詳細なコンテンツをオプションで制御して、多くの情報を含むレポートを作成できます。

CPU 使用率の簡単な説明の後、I/O の統計情報を示します。tps（1 秒あたりの I/O トランザクション。いくつかの論理的なリクエストを 1 つのリクエストにまとめることができます）、ブロックの単位時間ごとの読み書き数、このブロックはほとんどの場合 512 バイトのセクタです。そして、読み取りと書き込みの合計ブロック数です。

情報は、ディスクパーティション（と **LVM** が使われている場合は、デバイスマッパーの論理パーティション）に分解されます。

更に詳しいレポート:



図 14.2: **iostat の例**

14.3 iotop

iotop

- **iotop** は、現在の I/O アクティビティを表示する
- 表示は **top** のように定期的に更新される
- -o オプションを使用すると I/O 処理を実施しているプロセス／スレッドのみが表示されるので、混乱を避けられる



図 14.3: **iotop の例**

もう 1 つの非常に便利なユーティリティは **iotop** です。これは root 権限で実行する必要があります。現在の I/O 使用量を表示し、定期的に表示は更新されます。

PRIO フィールド（プライオリティ）の **be** と **rt** エントリは **ionice** セクションで説明します。**be** は **ベストエフォート**、**rt** は **リアルタイム** を表しています。



図 14.4: **iotop オプション**

14.4 ionice **

ionice

- **ionice** は、I/O の **スケジューリングクラス** と **優先度** を設定

```
$ ionice [-c class] [-n priority] [-p pid ] [COMMAND [ARGS] ]
```

Table 14.1: **I/O スケジューリングクラス**

スケジューリングクラス	-c 値	意　味
None 又は Unknown	0	デフォルト値を採用
Real Time	1	ディスクに最初にアクセスでき他のプロセスに優先して処理される。優先順位によって、各プロセスが取得できるタイムスライスを定義。
Best Effort	2	すべてのプログラムは、優先順位の設定に従ってラウンドロビン方式で実行される。これがデフォルト。
Idle	3	定義された期間他のプログラムが要求しない限り、ディスク I/O にアクセスしない。

ionice と CFQ はもう使われません

ionice は **CFQ** I/O スケジューラーとだけ動作しますが **CFQ** はカーネルバージョン 5.0 で削除されました。

ionice ユーティリティを使用すると、特定のプロセスの I/O スケジューリングクラスと優先順位の両方を設定できます。

-c パラメータは、上の表に示すように I/O スケジューリングクラスを指定します。

-p 引数で **pid** を指定すると、指定されたプロセスに適用されます。それ以外の場合は、指定された引数を持つ COMMAND を実行するプロセスが対象となります。引数が指定されていない場合、**ionice** は次のように現在のシェルプロセスのスケジューリングクラスと優先度を返します。

```
$ ionice
idle: prio 7
```

ベストエフォート および **リアルタイム** のクラスは優先度を指定する -n 引数を取ります。これは 0～7 の範囲で 0 が最高の優先度です。実際の例:

```
$ ionice -c 2 -n 3 -p 30078
```

注意：**ionice** は **CFQ** I/O スケジューラを使用している場合にのみ使えます。ただし、**CFQ** I/O スケジューラはカーネルバージョン 5.0 で削除されていることに注意してください。

14.5 演習

課題 14.1: bonnie++

bonnie++ は、ドライブとファイルシステムのテストと性能測定に広く利用されているベンチマークプログラムです。初期の実装である **bonnie** の後継です。

結果を端末に表示したりファイルに出力できる他、**csv**（**c**omma **s**eparated **v**alue の略）フォーマットにも格納できます。一緒に使うプログラムとして **bon_csv2html** と **bon_csv2txt** があり、html やプレーンテキスト形式への変換に利用されます。

使用を開始する前に **bonnie++** の **man** を読んで、性能テストに関するオプションや、それがどのくらいシステムを消費しストレスをかけるかを理解してください。

```
$ bonnie++ --help
```

```
bonnie++: invalid option -- '-'
usage:
bonnie++ [-d scratch-dir] [-c concurrency] [-s size(MiB)[:chunk-size(b)]]
      [-n number-to-stat[:max-size[:min-size][:num-directories[:chunk-size]]]]
      [-m machine-name] [-r ram-size-in-MiB]
      [-x number-of-tests] [-u uid-to-use:gid-to-use] [-g gid-to-use]
      [-q] [-f] [-b] [-p processes | -y] [-z seed | -Z random-file]
      [-D]

Version: 1.98
```

簡単なテストは次のコマンドで実行できます:

```
$  time sudo bonnie++ -n 0 -u 0 -r 100 -f -b -d /mnt
```

それぞれ:

- `-n 0` ファイルサイズとして 0 を指定すると、ファイルを作成しません。
- `-u 0` ユーザー id を指定します。この場合は root 権限で実行します。
- `-r 100` RAM サイズの指定。100MB の RAM しかないと見せかけます。
- `-f` キャラクタ入出力テストを省略します。
- `-b` 書き込みのたびに fsync を実行することを意味します。これにより、キャッシュへの書き込みを行わず、強制的にディスクへ書き出します。
- `-d /mnt` 一時ファイルを格納するディレクトリを指定します;　十分な領域（ここでは 300MB）を確保してください。

メモリー サイズを指定しなかった場合、プログラムは自動的にシステムのメモリーサイズを検出し、その 2〜3 倍の大きさのテスト用ファイルを作成します。それを行った場合には時間がかかりすぎるため、今回はそのようなことはしません。

On RedHat RedHat では

RHEL/CentOS 8 システムの場合:

```
c8:/tmp>sudo bonnie++ -n 0 -u 0 -r 100 -f -b -d /mnt
```

```
Using uid:0, gid:0.
Writing intelligently...done
Rewriting...done
Reading intelligently...done
start 'em...done...done...done...done...done...
Version  1.98       ------Sequential Output------ --Sequential Input- --Random-
                    -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
```

```
Name:Size etc        /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP
c8             300M            271m  20  288m  15           +++++ +++  3916  22
Latency                          30us      20us               12us   18075us

1.98,1.98,c8,1,1616174245,300M,,8192,5,,,277062,20,294543,15,,,+++++,+++,3916,22,,,,,,,,,,,,,,,,,\
30us,20us,,12us,18075us,,,,,,
```

On Ubuntu Ubuntu では

同じ物理マシン上の **VMware** ハイパーバイザー下で仮想マシンとして動作する **Ubuntu 20.04** システムの場合:

```
$ sudo bonnie++ -n 0 -u 0 -r 100 -f -b -d /mnt
```

```
Using uid:0, gid:0.
Writing intelligently...done
Rewriting...done
Reading intelligently...done
start 'em...done...done...done...done...done...
Version  1.98       ------Sequential Output------ --Sequential Input- --Random-
                    -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
Name:Size etc        /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP
ubuntu         300M            287m  88 +++++ +++            +++++ +++ +++++ +++
Latency                        3805us    4650us              2136us    1852us

1.98,1.98,ubuntu,1,1616156268,300M,,8192,5,,,293955,88,+++++,+++,,,+++++,+++,+++++,+++,,,,,,,,,,,,,,,\
,,,,3805us,4650us,,2136us,1852us,,,,,,
```

明らかに性能低下が見られます。

bonnie++.outファイルに先の結果を書き込み、html 形式に変換します:

```
$ bon_csv2html < bonnie++.out > bonnie++.html
```

もしくは、平文で良ければ:

```
$ bon_csv2txt < bonnie++.out > bonnie++.txt
```

ドキュメントを読んで、もっと時間がかかり大きく大変なテストに挑戦してください。ここでやらなかったテストを実行してみてください。システムが問題なく動作したらその結果を残しておき、将来システムの調子が悪くなったときに比較してください。

課題 14.2: fs_mark

fs_mark ベンチマークは、ファイルシステムに低レベルのストレスをかけます。複数のディレクトリとドライブ間で、相互にまたがる重い非同期 I/O を実行したりします。Rick Wheeler が開発した少し古く、テストに時間がかかるプログラムです。

http://sourceforge.net/projects/fsmark/ からダウンロードできます。tar ボールを入手したら、解凍してコンパイルします:

```
$ tar zxvf fs_mark-3.3.tgz
$ cd fs_mark
$ make
```

ここでは簡単に説明をしていますので、詳細は README ファイルを読んでください。

もし、コンパイルが失敗したら:

```
$ make
```

```
....
/usr/bin/ld: cannot find -lc
```

glibc の**スタティック・バージョン** をインストールしていないためです。**Red Hat** ベース システムの場合、次のようにしてインストールします:

```
$ sudo dnf install glibc-static
```

SUSE 関連のシステムの場合:

```
$ sudo zypper install glibc-devel-static
```

Debian ベースシステムの場合、静的ライブラリは共有ライブラリと一緒にインストールされるため、追加でインストールする必要はありません。

RedHat、Centos、Fedora では

- **RHEL 8** では `glibc-static`は、`codeready-builder` レポに存在するので利用可能にする必要があるかもしれません。
- しかし、**CentOS 8** ではこれは **PowerTools** レポに含まれています。あなたのシステムですでに`/etc/yum.repos.d/CentOS-PowerTools.repo` が存在するのであれば、そのファイルに `enabled=1` が存在するか確認してください。無い場合くは、このコースの SOLUTIONS セクションにファイルのコピーを用意してあります。

テスト用に大きさ 10KB のファイルを 2500 個作成します。それぞれ書き込み後、ディスクへフラッシュするため **fsync** を実行します。これらを`/tmp` ディレクトリ下で行います:

```
$ fs_mark -d /tmp -n 2500 -s 10240
```

実行中に、別端末で **iostat** の拡張情報を収集します:

```
$ iostat -x -d /dev/sda 2 10
```

fs_mark がレポートした秒ごとのファイル数と、**iostat** がレポートしたバンド幅の使用率に注意してください。これが 100%に近いと I/O 性能の限界です（I/O バウンド）。

章 15

I/O スケジューリング **

注目

** これらのセクションの一部または全体をオプション扱いとする場合があります。これらには補足資料、専門トピック、または高度な話題が含まれインストラクターが教室の状況や時間制約に応じてこれらの内容を紹介するかを判断します。

15.1 I/O スケジューリング

ディスクのボトルネックと I/O スケジューリング

- I/O スケジューラは、低水準デバイスドライバーと I/O 要求を調停する
- バランスを取る必要がある:
  - ハードウエアのアクセス時間
  - レイテンシ（開始までの遅延時間）
  - デッドライン（完了までの時間）
  - 公平性
  - 効率性

重要

SSD の登場によって、スケジューリングの要件は根本的に変わっていく可能性があります

I/O スケジューラは、汎用ブロックレイヤーと低レベルの物理デバイスドライバ間のインターフェイスを提供します。VM（仮想メモリー）レイヤーと、VFS（仮想ファイルシステム）レイヤーの両方が、ブロックデバイスに I/O 要求を送ります。これらの要求をブロックデバイスに渡す前に、優先順位を付けて順序付けるのが I/O スケジューリングレイヤーの仕事です。

I/O スケジューリングアルゴリズムは特定の（場合によっては競合する）要件を満たす必要があります。

- ハードウェアのアクセス時間を最小限に抑える必要があります。つまり、ディスク上の物理的な場所に応じてリクエストを並べる必要があります。この要請に応えるため、リード要求が物理的な順番で（＝ディスクの回転待ち時間や、読み取りヘッドのシーク動作が最少となる順番で）でペンディングキューに挿入されるエレベータ方式につながります。
- リクエストは、できる限り連続した大きな領域をアクセスできるようにマージされるべきです。こうすることで、ディスクアクセス時間も最小限に抑えられます。
- ブロックデバイスアクセスは、可能な限り低レイテンシで行われる必要があります。（デッドライン担保のため）実時間保証が重要となる場合もあります。
- 通常、書き込みはキャッシュに対して行われるので、キャッシュからディスクへの書き込みが完了するまでプロセスを停止させる事はありません。しかし読み取り操作は、ほとんどの場合プロセスがデータの読み込み完了を待ってから先に進む必要があります。書き込みより読み取りを優先すると、並列性とシステムの応答性が向上します。
- プロセスは公平に、もしくは少なくとも意識的に優先順位を付けて、I/O 帯域幅を共有する必要があります。I/O レイヤーの全体的なパフォーマンスが低下する場合でも、プロセスのスループットを過度に低下させないようにすべきです。

15.2 I/O スケジューラの選択

I/O スケジューラの選択

- 利用可能な I/O スケジューラの確認（/dev/sda に対して）：

```
$ cat /sys/block/sda/queue/scheduler

noop [deadline] cfq          # Before kernel version 5.0
none bfq kyber [mq-deadline] # After  kernel version 5.0
```

- 設定値の変更は（スーパーユーザーとして）：

```
# echo bfq > /sys/block/sda/queue/scheduler
$ cat /sys/block/sda/queue/scheduler

none [bfq] kyber mq-deadline
```

- スケジューラ固有の調整が/sys/block/sda/queue/iosched にある

異なるワークロードに対しては、違った I/O スケジューラが適合する場合があります。大規模なデータベースサーバー と デスクトップシステム などが良い例です。柔軟性を持たせるために **Linux** カーネルはオブジェクト指向のスキームを採用しており、さまざまなデータ構造に求められる色々な機能を持っています。カーネルコマンドラインで実行中に選択できるものもあります。

少なくとも 1 つの I/O スケジューリングアルゴリズムをカーネルに組み込む必要があります。利用できるものは、時代とともに変化しており **Linux** カーネルのバージョン 5.0 で大幅な見直しが行われました。

デフォルトの選択は、コンパイル構成オプションで行います。カーネル 2.6.18 より前のカーネルでは **AS**（=acticipatory）で、その後 **CFQ**（=Completely Fair Queuing）になり、現在は **deadline-mq**（=deadline multi queue）ですが、ディストリビューションによっては別のものを選択している場合もあります。

デバイスごとに異なる I/O スケジューラを使用することも可能です。

15.3 演習

課題 15.1: I/O スケジューラの比較

ダウンロード済の SOLUTIONS ファイルから抜き出した lab_iosched.sh を用いて、I/O スケジューラを比較します。

lab_iosched.sh

```
#!/bin/bash

NMAX=8
NMEGS=100
[[ -n $1 ]] && NMAX=$1
[[ -n $2 ]]  && NMEGS=$2

echo Doing: $NMAX parallel read/writes on: $NMEGS MB size files

TIMEFORMAT="%R   %U   %S"

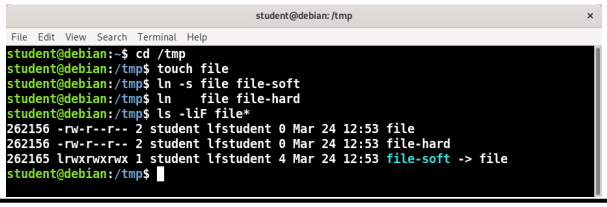
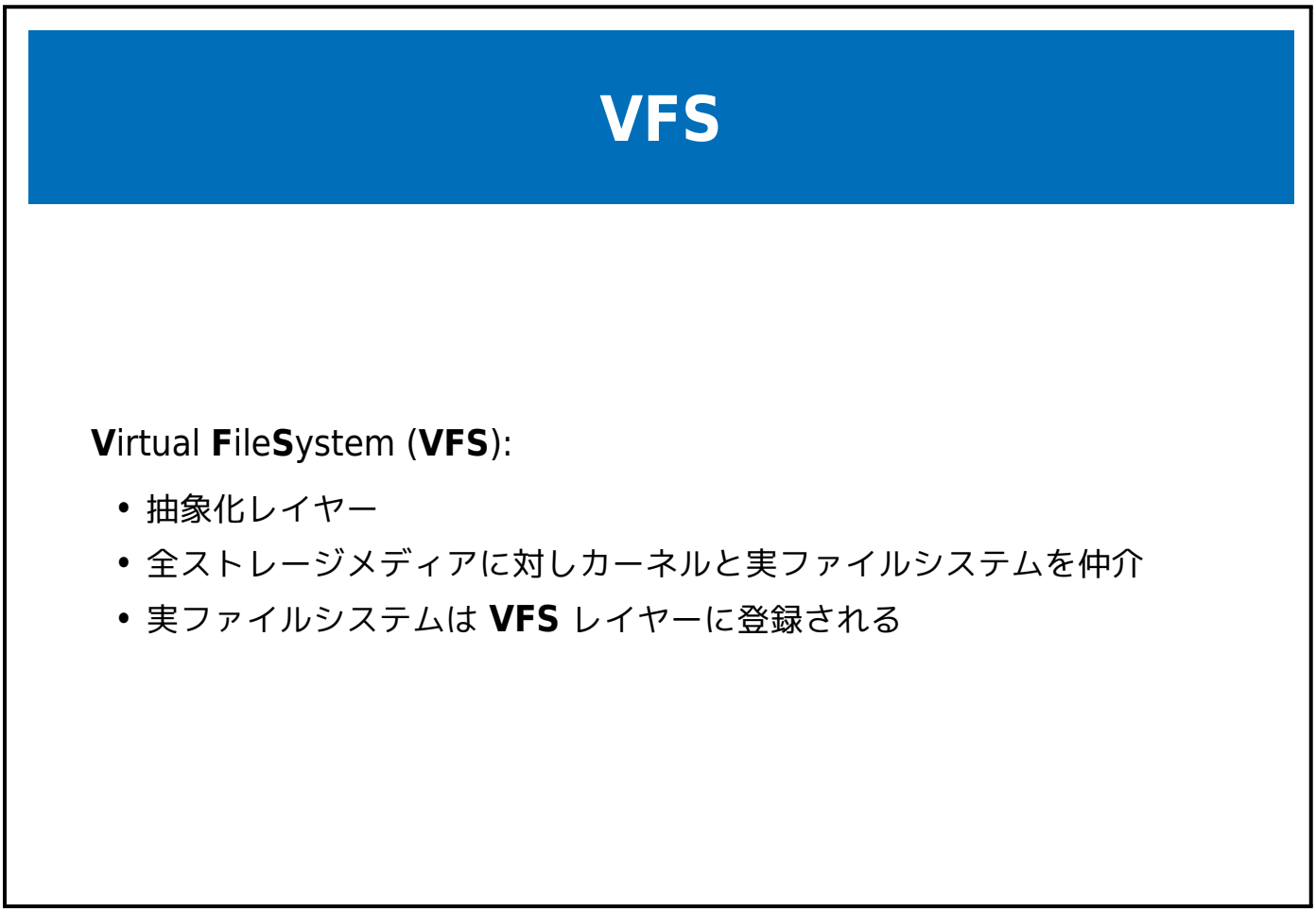
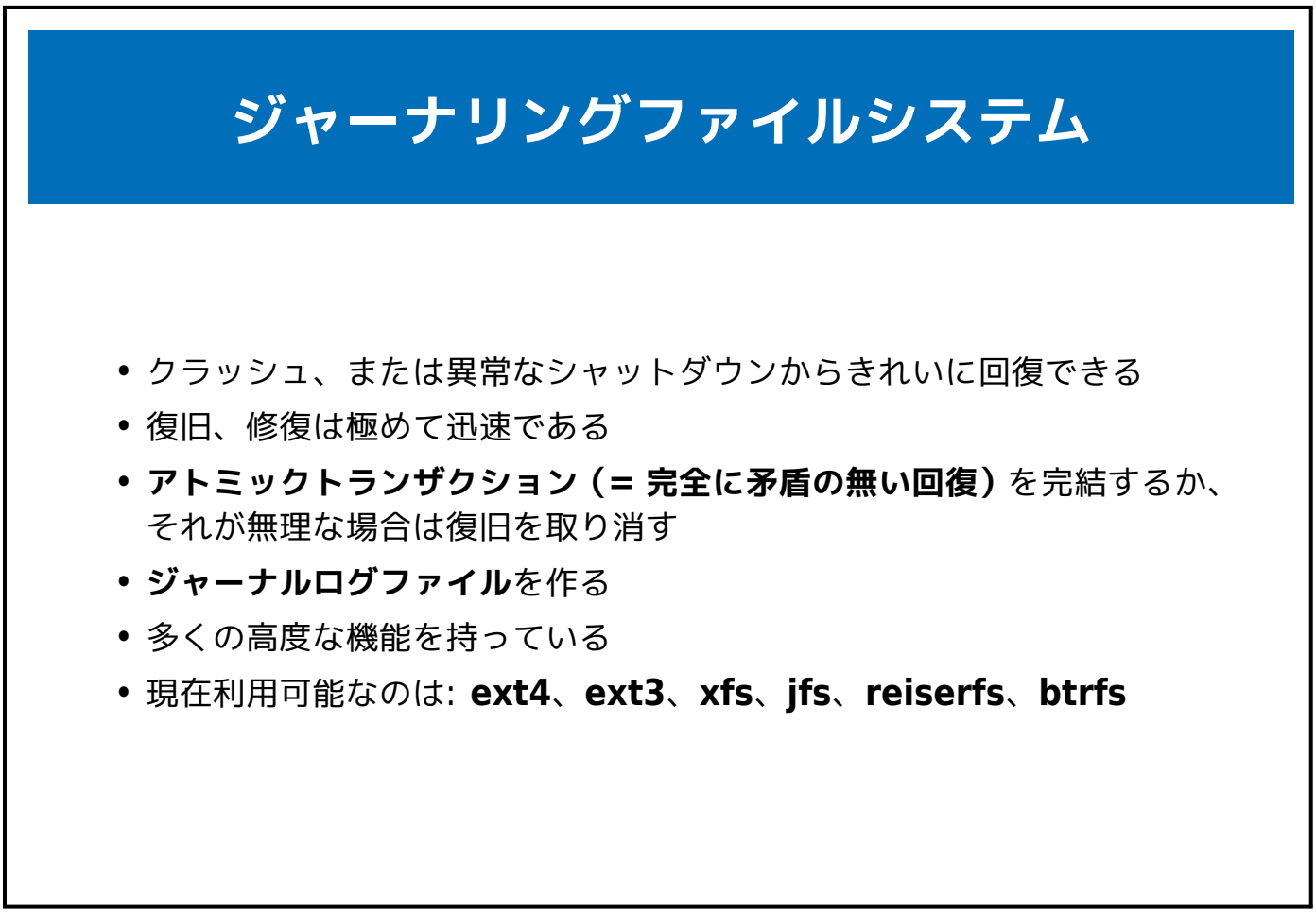
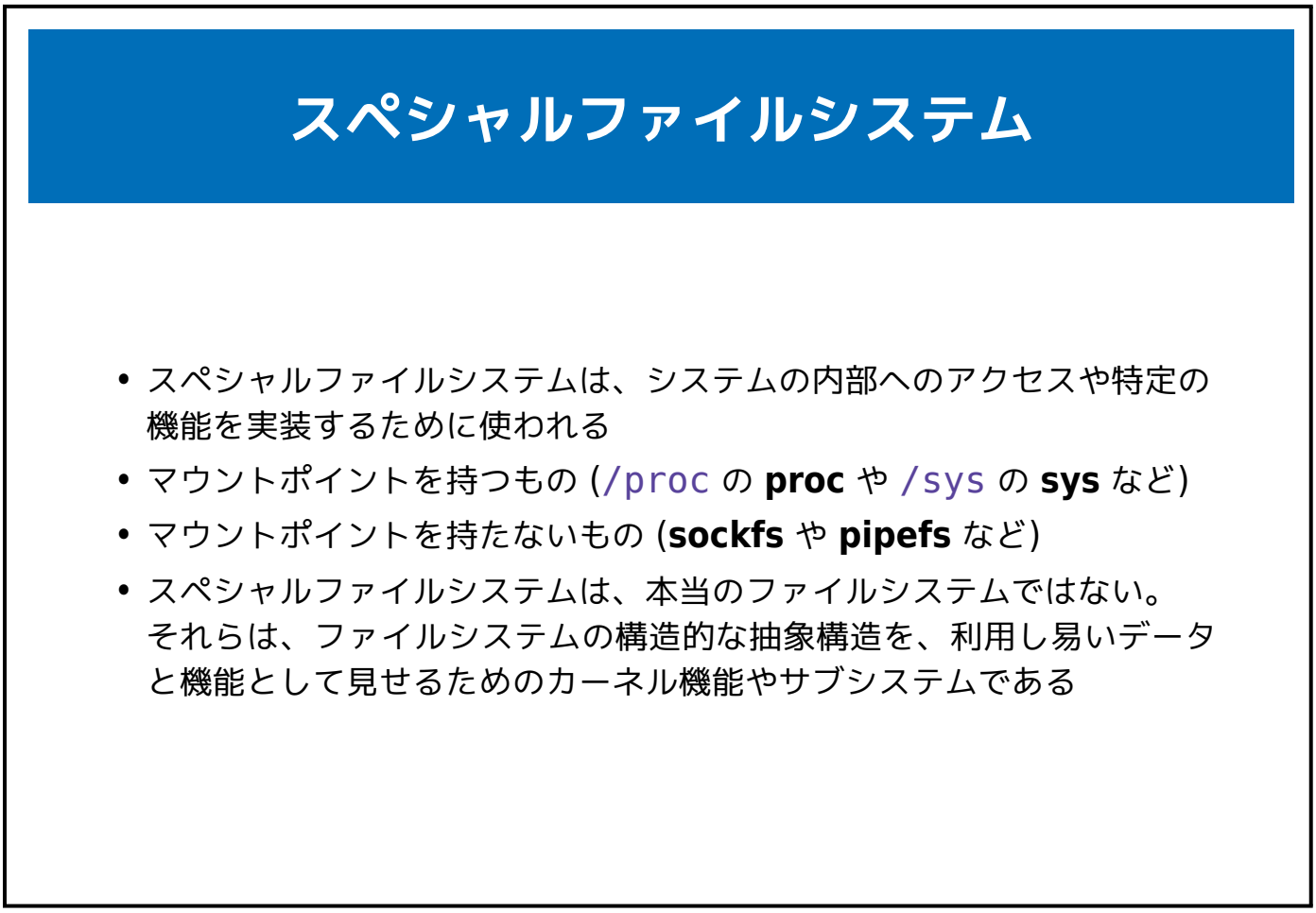
#################################################################
# simple test of parallel reads
do_read_test(){
    for n in $(seq 1 $NMAX) ; do
        cat file$n > /dev/null &
    done
# wait for previous jobs to finish
    wait
}

# simple test of parallel writes
do_write_test(){
    for n in $(seq 1 $NMAX) ; do
        [[ -f fileout$n ]] && rm -f fileout$n
        (cp file1 fileout$n && sync) &
    done
# wait for previous jobs to finish
    wait
}

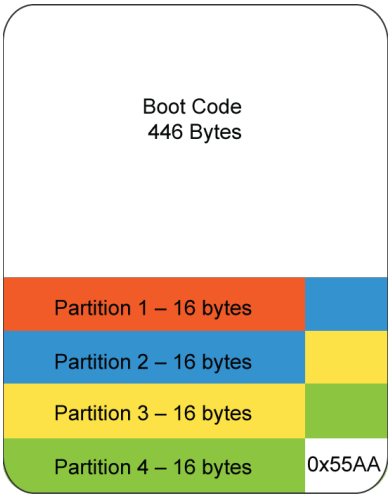
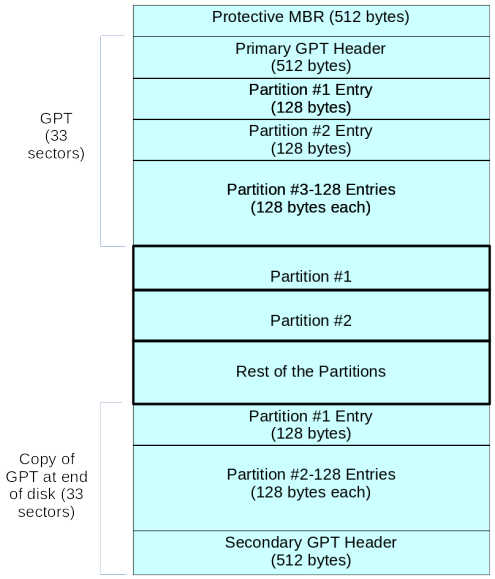
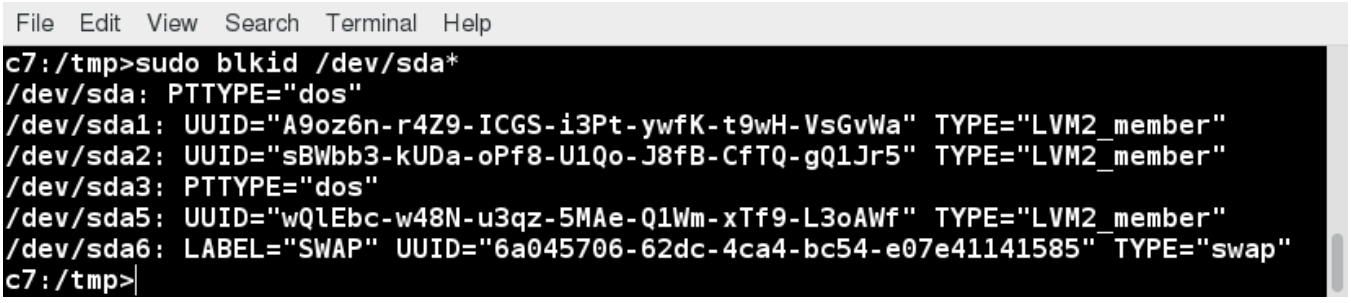
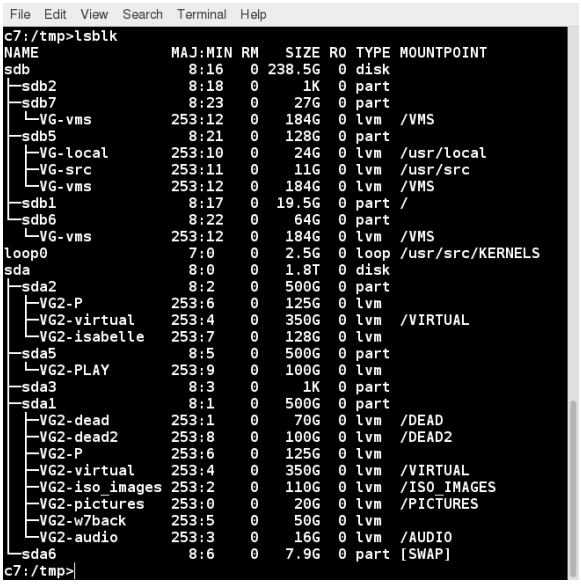
# create some files for reading, ok if they are the same
create_input_files(){
    [[ -f file1 ]] || dd if=/dev/urandom of=file1 bs=1M count=$NMEGS
    for n in $(seq 1 $NMAX) ; do
        [[ -f file$n ]] || cp file1 file$n
    done
}

echo -e "\ncreating as needed random input files"
create_input_files

#################################################################
# begin the actual work

# do parallel read test
echo -e "\ndoing timings of parallel reads\n"
echo -e " REAL    USER    SYS\n"
#for iosched in noop deadline cfq ; do
for iosched in \
    $(cat /sys/block/sda/queue/scheduler | sed -e s/'\['//g -e s/'\]'//g ) ; do
    echo testing IOSCHED = $iosched
```

SH

```
    echo $iosched > /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
#    echo -e "\nclearing the memory caches\n"
    echo 3 > /proc/sys/vm/drop_caches
    time do_read_test
done
################################################################
# do parallel write test
echo -e "\ndoing timings of parallel writes\n"
echo -e " REAL    USER    SYS\n"
for iosched in \
    $(cat /sys/block/sda/queue/scheduler | sed -e s/'\['//g -e s/'\]'//g ) ; do
    echo testing IOSCHED = $iosched
    echo $iosched > /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    time  do_write_test
done
################################################################
```

本コースのオンライン、セルフペース版を受講しているときは、**Lab** スクリーンからダウンロードできます。

I/O スケジューラの変更は特権命令ですから、以下のとおりに実行してください:

```
$ sudo ./lab_iosched.sh [# reads/writes (NMAX)] [file size in MB (NMEGS)]
```

使い方

スクリプト:

- 利用可能な I/O スケジューラについて、ハードディスクに対して、指定したサイズのファイルを指定した回数だけ並列読み書きするサイクルを測定します。(利用可能なスケジューラは、カーネルの構成コンパイル条件によりマシンごとに異なります)。
- 読み、書き別々に実行します。
- 読み出すときは、(root ユーザー権限で) キャッシュをフラッシュすることで、メモリー上のキャッシュからではなくディスクから読み出します;

  ```
  echo 3 > /proc/sys/vm/drop_caches
  ```

  読み出し前にディスクへの書き込みが発生するのを避けるため、スクリプトは **cat** コマンドの出力先に `/dev/null` を指定しています。
- 計測情報を得るまえに、全読み出しが完了していることを確認します;　このために、shell 下で **wait** コマンドを実行します。
- 書き込みテストは、ファイルを複数回並列コピーで行います (最初の読み出しで、メモリーにキャッシュされています)。計測情報を得るまえに、全書き込みが完了していることを確認します;　このために **sync** コールを実行します。

スクリプトは 2 つの引数を取ります。1 番目は、並列実行する読み書き回数です。2 番目は、各ファイルの大きさ (MB 指定) です。

`/proc` と `/sys` ディレクトリ階層の下に書き込むため、root 権限で実行します。

異なる I/O スケジューラの結果を比較してください。

追加課題

調整可能なパラメータを変更して、結果がどう変わるか見てください。

章 16

Linux ファイルシステムと VFS

16.1 ファイルシステムの基本

ファイルシステムの基本

- **Linux** プログラムは、ファイルに対してリード／ライトを行う（ディスクのセクターに対してではない）
- **ファイル** とは物理的な I/O レイヤーをカモフラージュし抽象化したもの
- **ファイルシステム** は物理パーティションに利用可能なフォーマットを作る
- **UNIX** 系のファイルシステムは、ツリー型の階層構造を持つ
  - ディレクトリにはファイルやディレクトリーを持つ
  - 全てのパスやノードはルート (/) ディレクトリーの配下に置かれる
- 複数のファイルシステムを単一のツリー構造に統合することができる（通常は統合される）
- **Linux** は、仮想ファイルシステムレイヤー (**VFS**) を介し、ファイルシステムと連携する

アプリケーションプログラムは、**ファイル** が保存されている実際のハードウェア上の物理的な場所にアクセスするのではなく、ファイルを読み書きします。

ファイルとその名前は、物理 I/O レイヤーをカモフラージュして抽象的に見せたものです。（**ファイルシステムレイヤー** を無視して）コマンドラインからディスクに直接書き込むのは非常に危険です。ディスクへの書き込みは、ユーザーアプリケーションではなく、低レベルのオペレーティングシステムソフトウェアによってのみ行われます。例外は、エンタープライズデータベースなど非常にハイエンドなソフトウェアで、これらはファイルシステムに起因するレイテンシを回避するために直接 **cmd** アクセスをします。

ローカルファイルシステムは、通常ディスク上の物理パーティション上に置かれます。または、**論理ボリュームマネージャ（LVM）** によって制御される論理パーティション上に置かれることもあります。ファイルシステムは、ネットワークの先に配置することもでき、この場合にはファイルシステムの実体はネットワークを介してローカルシステムからは完全に隠蔽されます。

16.2 ファイルシステムのコンセプト

Inodes

- ファイルの **名前** は、より本質的なファイルの **inode** の中のプロパティの１つにすぎない
- Inodes は、ファイルシステム内に保存され、同時にメモリー上のデータ構造である
- Inodes にはファイルに関する以下の情報が記載され保存されている:
  - パーミション
  - ユーザー、グループのオーナー
  - サイズ
  - タイムスタンプ (ナノ秒)
    * 最終アクセス時間
    * 最終変更時間
    * inode の変更時間
- ファイル名は inode に **保存されない。ディレクトリ** に保存される

inode は、場所を含むファイル属性を記述し保存する、ディスク上のデータ構造体です。

Linux のすべてのファイルは、個別の inode に関連付けられています。ファイルに関するデータは、全てそのファイルの inode に含まれます。オペレーティングシステムは、最新のファイル名、場所、アトリビュート（パーミション、オーナーシップなど）、アクセス時間 などを inode から取得します。このためファイルに関わる全ての I/O アクティビティには、通常そのファイルの inode が含まれます。

ハードリンクとソフトリンク

- **ハード** リンクは inode を指す
  - 2 つ以上のファイルが同じ inode を指す (ハードリンク)。
  - ハードリンクは、同一ファイルシステム内のファイルに限定
- **ソフト** (**シンボリック**) リンクは、ファイル名と Inode を結びつける
  - ソフトリンクは、別ファイルシステム上のファイルとのリンク可能
  - ターゲットは存在していなくても、さらにマウントさえされていなくても、リンクの設定が可能、**宙ぶらりん（未解決）**で良い
- ハードリンクされたファイルに注意（たぶんソフトリンクの間違い?)
  - 一カ所で行ったコンテンツ変更は、別の場所には反映されない



図 16.1: **ハードリンクとソフトリンク**

ディレクトリファイルは、ファイル名と inode を関連付けるために使用される特殊な種類のファイルです。ファイル名を inode に関連付ける（または **リンク**する）には 2 つの方法があります:

- **ハード** は単一の inode を指します。オプション無しの **ln** コマンドでハードリンクを作ることができます。
- **ソフト** (または **シンボリック**) リンクは inode に関連付いたファイル名を指す。**ln** に -s オプションを付加して作成します。

ディレクトリファイルの内容と inode の関連付けはリンクと呼ばれます。リンクを追加するには **ln** を使用します。

2 つ以上のディレクトリエントリが、同じ inode（ハードリンク）を指す可能性があります（多く使われます)。このようなファイルは、別の名前からアクセス可能でそれぞれ独自のディレクトリ構造内に独自の場所を持てます。ただし、どの名前から呼ばれても inode は 1 つだけです。

プロセスがパス名を参照すると、カーネルはディレクトリを検索して対応する inode 番号を見つけます。名前が inode 番号に変換された後、inode はメモリーにロードされ、後続のリクエストで使用されます。

16.3 仮想ファイルシステム (VFS)



Linux は、最新のすべてのオペレーティングシステムと同様に **仮想ファイルシステム**（VFS）を実装しています。アプリケーションがファイルにアクセスする必要がある場合 **VFS** 抽象化レイヤーとやり取りし、すべての I/O システムコール（読み取り、書き込みなど）を特定の実ファイルシステム用の固有のコードに変換します。

したがってアプリケーションは、特定の実ファイルシステムやそれが存在する物理媒体、およびハードウェアを考慮する必要はありません。さらに、ネットワークファイルシステム（**NFS** など）も透過的に処理できます。（＝ローカルファイルと全く同じに扱える）

これにより **Linux** では、他のどのオペレーティングシステムよりも多くの種類のファイルシステムが動作します。あらゆるファイルシステムが平等にサポートされたことは、**Linux** 成功の大きな要因となりました。

ほとんどのファイルシステムには完全な読み書きのアクセス権がありますが、一部のファイルシステムには読み取りのアクセス権と実験的な書き込みアクセス権しかありません。一部の種類のファイルシステム、特に非 **UNIX** ベースのものは、**VFS** として見せるためにより多くの操作が必要になる場合があります。

vfat などには owner/group/world フィールドに対して個別の 読み取り／書き込み／実行 のパーミッションがありません。**VFS** ではこの３種類のユーザーに対するパーミッションをそれぞれ想定する必要があります。この想定がマウント操作の影響を受ける可能性があります。また、**ntfs-3g** (https://sourceforge.net/projects/ntfs-3g/) のような 読み取り／書き込み をサポートするカーネル外のファイルシステム実装もあります。これは信頼性はありますが、カーネル内ファイルシステムよりもパフォーマンスは低くなります。

16.4 利用可能なファイルシステム

利用可能なファイルシステム

- **Linux** では、他のオペレーティングシステムより多くの種類のファイルシステムを利用することが可能である
- ファイルシステム選択の自由度（公平性）が Linux 成功の大きな要因
- 大部分のファイルシステムは完全な読み書きができるが、一部読み込みだけのものもある
- **ext4, xfs, btrfs, squashfs, nfs** と **vfat** が幅広く利用されている
- `/proc/filesystems` に現在サポートされているファイルシステムのリストがある

ファイルシステムの種類

Linux では多くの種類のファイルシステムを利用可能、大部分のファイルシステムは完全な読み書きをサポート、一例を示すと:

- **ext4**: **Linux** のネイティブファイルシステム (**ext2** と **ext3** も同様)
- **XFS**: 元々 **SGI** が開発した高性能ファイルシステム
- **JFS**: 元々 **IBM** が開発した高性能ファイルシステム
- **Windows** ネイティブ: **FAT12, FAT16, FAT32, VFAT, NTFS**
- メモリーに常駐する仮装ファイルシステムには **proc, sysfs, devfs, debugfs** などがある
- **NFS**,**coda**, **afs** などネットワークファイルシステムもある
- その他

以下のコマンドで、現在実行中の **Linux** カーネルに登録され認識されているファイルシステムのリストを見ることができます。

```
$ cat /proc/filesystems
```

```
	iso9660
	squashfs
	ext3
	ext2
	ext4
	fuseblk
nodev	sysfs
nodev	proc
nodev	tmpfs
nodev	debugfs
nodev	sockfs
nodev	hugetlbfs
nodev	fuse
nodev	nfsd
....
```

(nodev がついたものはストレージ上には存在しない **スペシャルファイルシステム**) です。システムがそのパーティションにアクセスしようとした時だけ、これ以外のファイルシステムがモジュールとしてロードされることがあります。

16.5 ジャーナリングファイルシステム



ジャーナリング ファイルシステムを使うと、システムクラッシュや異常なシャットダウンから、ほとんどまたはまったくファイルを破損することなく非常に迅速に回復させることができます。これを利用するには、さらに多くの操作を行うコストが伴いますが、追加の機能強化にはコストを払う価値があります。

ジャーナリングファイルシステムでは、操作は **トランザクション** にグループ化されます。トランザクションは **アトミック** に実行され、エラーなしで終了する必要があります。それが出来ない時にはファイルシステムは変更されません。トランザクションのログファイルは保持されます。エラーが発生した場合、通常は最後のトランザクションのみを調べます。

Linux では、以下のジャーナリングファイルシステムを自由に利用できます。

- **ext3** は、以前の非ジャーナリング **ext2** ファイルシステムの拡張です。
- **ext4** は、**ext3** を大幅に強化したものです。機能にはエクステントや 48 ビットのブロック番号を含み、対応できる最大サイズは 16TB です。ほとんどの **Linux** ディストリビューションがかなり長い間、**ext4** をデフォルトのファイルシステムとして使用しています。
- **reiserfs** は **Linux** で使用された最初のジャーナリング実装でしたが、リーダーシップを失い開発は中止されました。
- **JFS** はもともと **IBM** IBM の製品であり、**IBM** の **AIX** オペレーティングシステムから移植されました。
- **XFS** はもともと **SGI** の製品であり、**SGI** の **IRIX** ペレーティングシステムから移植されました。**RHEL 7** はデフォルトのファイルシステムとして **XFS** を採用しました。
- **btrfs** は最新のジャーナリングファイルシステムであり、今も急速に開発が進んでいます。**SUSE** と **openSUSE** システムのデフォルトのファイルシステムです。

16.6 スペシャルファイルシステム



Linux は、特定のタスクのために **スペシャルファイルシステム** を幅広く採用しています。これらは、さまざまなカーネルデータ構造へのアクセス、カーネルの動作調整、特定の機能の実装に非常に役に立ちます。一例としては:

Table 16.1: **スペシャルファイルシステム**

ファイルシステム	マウントポイント	目　的
rootfs	なし	カーネルのロード中に空のルートディレクトリを提供
hugetlbfs	任意	拡張ページのアクセスを提供（**x86** では 2 または 4MB）
bdev	なし	ブロックデバイスに使用
proc	/proc	多くのカーネル構造とサブシステムへの疑似ファイルシステムアクセス
sockfs	なし	**BSD ソケット**で利用
tmpfs	任意	スワッピングとサイズ変更を伴う RAM ディスク
shm	なし	System V IPC の共有メモリーとして使用
pipefs	なし	パイプに使用
binfmt_misc	任意	さまざまな実行可能形式で使用
devpts	/dev/pts	**Unix98** 擬似端末で使用
usbfs	/proc/bus/usb	動的デバイス用の USB サブシステムで使用
sysfs	/sys	デバイスツリーとして使用
debugfs	/sys/kernel/debug	簡単なデバッグファイルアクセスに使用

これらのスペシャルファイルシステムの中には、マウントポイントがないものがあることに注意してください。つまりユーザーアプリケーションは、それらのファイルシステムとはやり取りしません。しかしカーネルは、**VFS** レイヤーとコードを利用してそれらを使用します。

16.7 演習

デモ教材ビデオ

using_available_filesystems_demo.mp4

課題 16.1: tmpfs スペシャルファイルシステム

tmpfs は **Linux** で使われているスペシャルファイルシステムの 1 つです。これらのいくつかは、実際にはファイルシステムとして使われるのではなく、ファイルシステムとして抽象化することでその利点を利用できるようにしています。しかし、**tmpfs** は実際にアプリケーションが I/O を行うファイルシステムです。

基本的に **tmpfs** は **ramdisk** として機能します; メモリー中に存在しているのです。しかし、従来の RAM ディスクが持っていなかった素晴らしい機能を有しています:

1. 大きさは動的に調整されます（使用するメモリー量を調整します）; 0 から開始して、マウント時に指定された最大値まで必要に応じて広がります。
2. RAM が枯渇したら、**tmpfs** はスワップ領域を利用します。(しかし、許される最大値を越える量をファイルシステムに書き込むことはできません)。
3. **tmpfs** は **ext3** や **vfat** と違い、ファイルシステムを構築する必要がありません; 独自の方法でメモリー内の領域を考慮しながらファイルと I/O を処理します（それはブロックデバイスではありません)。そしてスピードを最適化しています。

   つまり、`mkfs` コマンドでファイルシステムをフォーマットする必要がありません; マウントして使うだけです。

新しく **tmpfs** を適当なディレクトリにマウントします:

```
$ sudo mkdir /mnt/tmpfs
$ sudo mount -t tmpfs none /mnt/tmpfs
```

ファイルシステムに割り当てられたスペースと、どのくらい使用されているかを見ます:

```
$ df -h /mnt/tmpfs
```

デフォルトで RAM の半分のサイズが割り当てられています; しかし全く利用されていません、`/mnt/tmpfs` にファイルを格納すると利用されます。

マウント時のオプションで割り当てサイズを変更できます:

```
$ sudo mount -t tmpfs -o size=1G none /mnt/tmpfs
```

最大容量まで格納して、何が起こるか見てください。終了したらアンマウントするのを忘れないでください。

```
$ sudo umount /mnt/tmpfs
```

最近の **Linux** ディストリビューションは、`/dev/shm`に **tmpfs** をマウントします:

```
$ df -h /dev/shm
```

```
Filesystem      Type   Size  Used Avail Use% Mounted on
tmpfs           tmpfs  3.9G   24M  3.9G   1% /dev/shm
```

たくさんのアプリケーションが、**POSIX** の共有メモリーを使うときに、これをプロセス間の通信手段として利用します。誰でも `/dev/shm` にファイルを作成、読み出し、書き込み できます。メモリー中に一時ファイルを作成するときにちょうど良いのです。

いくつかのファイルを `/dev/shm` 中に作成し、**df** でファイルシステムの使用状況を調べてください。

さらに、たくさんのディストリビューションが複数の **tmpfs** をマウントしています; **RHEL** システムを例にあげます:

```
$ df -h | grep ' tmpfs'
```

```
tmpfs           tmpfs     7.8G   38M  7.8G   1% /dev/shm
tmpfs           tmpfs     7.8G   18M  7.8G   1% /run
tmpfs           tmpfs     7.8G     0  7.8G   0% /sys/fs/cgroup
tmpfs           tmpfs     1.6G  1.2M  1.6G   1% /run/user/42
tmpfs           tmpfs     1.6G   56K  1.6G   1% /run/user/1000
```

このシステムは 16GB の RAM を搭載しています。そのため全ての **tmpfs** ファイルシステムが、デフォルトで割り当てられた ~8 GB を使うことはできません！

注目

いくつかのディストリビューション（**Fedora** など) は、（標準で） **tmpfs** を `/tmp` にマウントします; そのときは `/tmp` に巨大なファイルを格納することを避けて、メモリーが枯渇しないようにします。または `/tmp` について、以前に説明した方法でこれを停止させます。

章 17

ディスクのパーティション分割

17.1 一般的なディスクの種類

一般的なディスクの種類

- **SATA** (Serial AT Attachment)
  - **IDE** よりデータ転送が高速で、ケーブルが小さい
  - **SCSI** デバイスと見なされる
- **SCSI** (Small Computer Systems Interface)
  - 全般に高速である
  - Fast、Wide、Ultra、Ultrawide など多くのバージョンがある
- **SAS** (Serial Attached SCSI)
  - 新しいポイントツーポイント（point-to-point）プロトコルである
  - **SATA** ディスクよりも優れたパフォーマンスが出せる
- **USB**
  - フラッシュドライブとフロッピーが含まれ、SCSI デバイスと見なされる
- **SSD**
  - 可動部品がなく、汎用コントローラに接続できる
- **IDE** and **EIDE** (Integrated Drive Electronics, Enhanced IDE)
  - 使われなくなった

さまざまな種類のハードディスクがあり、それぞれが接続されている **データバス**の種類、速度、容量、複数のドライブが同時に動作する頻度や、その他の要因によって特徴付けられます。

SATA ディスクは、古い **IDE** ドライブを置き換えるために設計されました。小さなケーブルサイズ（7 ピン）、ネイティブホットスワップ、そして、より高速で効率的なデータ転送を提供します。

SAS はサーバー用途に向いています。以下に、違いについての明確でシンプルな説明があるので参照して下さい。
https://store.hp.com/us/en/tech-takes/sas-vs-sata

SCSI ディスクの範囲は narrow（8 ビットバス）から wide（16 ビットバス）で、転送速度は毎秒 5MB（narrow、標準 **SCSI**）から毎秒 160MB（**Ultra-Wide SCSI-3** ）です。

最新の **SSD** ドライブ (Solid State Drive) は価格が下がってきています。また、可動部品がなく回転式のメディアを備えたドライブよりも、消費電力が少なく転送速度が高速です。内蔵 **SSD** は、従来のドライブと同じ外形寸法で同じ筐体に取り付けられます。

SSD の価格はまだ少し高いですが、価格は下がってきています。同じマシンに **SSD** と回転式ドライブの両方を搭載するのが一般的です。**SSD** は、頻繁にアクセスされパフォーマンスがクリティカルなデータ転送に使われます。

17.2 ディスクジオメトリ

ディスクジオメトリ

- **回転式**のディスクは、１つ以上のプラッタで構成され、それぞれが１つ以上のヘッドで読み取られる
- ディスクが回転すると、ヘッドはプラッタ上の円形トラックを読み取る
- これらの円形のトラックは、セクタと呼ばれるデータブロックに分割されている
- セクターサイズは、当初は 512 バイトだったが、現在は 4096 バイトである
- シリンダは、すべてのプラッタ上の同じトラックで構成されるグループ
- **SSD** には、このようなジオメトリーの概念はない

ディスクコントローラが、その殆どを隠蔽するようになっているので、物理的なディスク上のどこにデータが配置されているかは以前ほど重要ではなくなっています。さらに **SSD** には何も可動パーツは無く、ジオメトリの概念自体がなくなりました。

fdisk でジオメトリを表示することができます:

```
File  Edit  View  Search  Terminal  Help
c7:/tmp>sudo fdisk -l /dev/sda

Disk /dev/sda: 2000.4 GB, 2000398934016 bytes, 3907029168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disk label type: dos
Disk identifier: 0x000852df

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1            2048  1048578047   524288000   8e  Linux LVM
/dev/sda2      1048578048  2097154047   524288000   8e  Linux LVM
/dev/sda3      2097154048  3907028991   904937472    5  Extended
/dev/sda5      2097156096  3145732095   524288000   8e  Linux LVM
/dev/sda6      3890448384  3907028991     8290304   82  Linux swap / Solaris
c7:/tmp>
```

図 17.1: **Using fdisk**

`-l` オプションを使うと対話モードに入らずに、単にパーティションテーブルをリストします。

歴史的には、ディスクは１セクター 512 バイトで製造されていましたが、現在では 4KB が主流です。セクターサイズが大きくなれば転送速度を稼ぐことができるのです。**Linux** は未だに **ロジカルセクターサイズ** に 512 バイトを使っていますが、これはソフトウエアでそう見せかけているだけです。

17.3 パーティション

パーティション構成

- ディスクは **パーティション** に分割される
- パーティションは、ディスク上の物理的に連続する領域である
- 一般的なアーキテクチャには 2 つのパーティションスキームがある:
  - **MBR** (Master Boot Record)
  - **GPT** (GUID Partition Table)
- **MBR** の原点は、**MSDOS** の初期にまでさかのぼる
  最大 4 つまでプライマリパーティションを持つことができ、そのうち 1 つは **拡張パーティション**に指定可能である **拡張パーティション** に最大 15 個に分割可能な **論理パーティション** を持たせることができる
- **UEFI** (Unified Extensible Firmware Interface) に基づく **GPT** は、すべての最新システムに搭載され、デフォルトで最大 128 個の **プライマリーパーティション** を持つことができる

- **MBR** スキームを使用する場合:

  たとえば `/dev/sda` などの **SCSI** の場合、`/dev/sda1` が最初のプライマリパーティションで、`/dev/sda2` が 2 番目のプライマリパーティションとなります。拡張パーティション `/dev/sda3` を作成した場合、それを論理パーティションに分割できます。4 つ目以上のパーティションは、すべて論理パーティションです（拡張パーティション内に含まれることを意味します）。拡張パーティションは 1 つしか設定できませんが、多数の論理パーティションに分割できます。

  注意: **Linux** ではパーティションはシリンダ境界で開始または終了している必要はありませんが、他のオペレーティングシステムではそうでないかもしれません。このため広く導入されている **Linux** のパーティション分割ユーティリティも、シリンダ境界で終了させようとします。もちろんパーティションは重複してはいけません。

- **GPT** スキームを使用する場合:

  拡張パーティションを使わずに（デフォルトで）128 個のプライマリパーティションを持つことができます。

  パーティションのサイズは、最大 2^{33} TB です。（MBR ではわずか 2TB です。）

パーティションを使う理由

- ユーザやアプリのデータを OS のファイルと **分離** する
- OS とマシン（ハードウエア）間で情報を **共有** する
- システムの領域毎に異なるサイズ制限や権限を与えて、**セキュリティ** を拡張する
- 一時変数や揮発性データを通常データと分割し **サイズ** を有効利用する
- 利用頻度の高いデータを高速ストレージへ配置し **性能** を向上させる
- データ退避用の **スワップ** 領域、ハイバネーションストレージにもなる

何を分割し、パーティションをどのように分離するか、よく考えて決定しなければなりません。個別のパーティションを持つ理由は、セキュリティ、クォータ設定、サイズ制限の細分化があります。データ保護のために、パーティションを分けることもできます。

一般的なパーティションレイアウトには、`/boot` パーティション、ルートファイルシステム `/` のパーティション、スワップパーティション、および `/home` ディレクトリツリーのためのパーティションが含まれます。

インストール時にパーティション作成した後で、サイズを変更するのは難しいことを念頭に置いて計画しましょう。

17.4 パーティションテーブル

MBR パーティションテーブル

- パーティションテーブルは、ディスクの最初のセクターにある 512 バイトの **MBR** (**M**aster **B**oot **R**ecord) の後半の 64 バイトの領域に置かれる
- パーティションテーブルのエントリは 16 バイト長、4 つのプライマリパーティションのうちの 1 つを記述:
  - アクティブビット
  - パーティションの開始セクタのアドレス。シリンダ／ヘッド／セクタ (**CHS**) 形式
  - パーティションの種類のコード: **Linux**、**Linux LVM**、**ntfs**、**swap**
  - **CHS** 形式のパーティションの最後のセクタのアドレス
  - 開始セクタのアドレス、0 からの通し番号
  - パーティション内のセクタの数

Linux は、リニアブロックアドレス (**LBA 方式**) を使用、最後の 2 つのフィールドのみでアドレス指定



図 17.2: **MBR ディスクパーティションテーブル**

ディスクパーティションテーブルは、ディスクのマスターブートレコード（**MBR**）に含まれており、446 バイトのブートレコードに続く 64 バイトがそれにあたります。

ディスク上の 1 つのパーティションが、アクティブとしてマークされることがあります。システム起動時にマスターブートローダーは、アクティブとしてマークされたパーティションからロードするアイテムを見つけます。

拡張パーティションを 1 つ定義することができ、そこに複数の論理パーティションを含めることができる事を思い出してください。

MBR の構造は、オペレーティングシステム非依存に記述されます。最初の 446 バイトはプログラムコード用に予約されています。通常、ここにブートローダープログラムの一部を保持します。次の 64 バイトは、最大 4 エントリのパーティションテーブルです。オペレーティングシステムは、このテーブルを使ってハードディスクを処理します。

Linux システムでは **CHS** の開始アドレスと終了アドレスは無視されます。

(参考：**MBR** の末尾 2 バイトにはマジックナンバー、シグネチャワード、または、セクタマーカーという名前で呼ばれる領域があり、必ず `0x55AA` の値を持っています。)

GPT パーティションテーブル

- 最新のハードウェアは **GPT** をサポート、**MBR** サポートは徐々に減っている
- GPT の Protective MBR には、UEFI システムに対する後方互換性があり、以前の方法でシステムを起動できる
- ディスクの先頭と末尾に **GPT** ヘッダーを保存、以下のメタデータを記述:
  - 使用可能なブロックのリスト
  - パーティションの数
  - パーティションエントリのサイズ
- 各パーティションエントリの最小サイズは 128 バイトである



図 17.3: **GPT パーティションテーブル**

blkid ユーティリティ（後で説明します）は、パーティションに関する情報を表示します。

- 最新の UEFI/GPT システムの場合:

```
$ sudo blkid /dev/sda8
```

```
/dev/sda8: LABEL="RHEL8" UUID="53ea9807-fd58-4433-9460-d03ec36f73a3" BLOCK_SIZE="4096"
↪  TYPE="ext4" PARTUUID="0c79e35b-e58b-4ce3-bd34-45651b01cf43"
```

- レガシー MBR システムの場合:

```
$ sudo blkid /dev/sdb2
```

```
/dev/sdb2: LABEL="RHEL8" UUID="6921b738-1e36-429a-89be-8b97cf2f0556" BLOCK_SIZE="4096"
↪  TYPE="ext4" PARTUUID="00022650-02"
```

どちらの例も、パーティション自体ではなく、パーティション上のファイルシステムを表す一意の UUID を提供しています。ファイルシステムが再フォーマットされると、この UUID も変更されます。

GPT パーティションには、パーティションを識別する PARTUUID も含まれます。この値は、ファイルシステムが再フォーマットされても変わりません。

ハードウェアがサポートしていれば、**MBR** システムを **GPT** に移行可能ですが、その際マシンが操作不能なることもありえます。それを考えると、リスクを犯してやるメリットはないでしょう。

17.5 ディスクデバイスの命名

ディスクデバイスとデバイスノードの命名

- デバイスファイルは /dev に置かれている
- ファイル名のフォーマット: xxy[z]
  - xx はデバイスタイプを示す (通常 sd)
  - y はデバイス番号を示す (a,b,c、など)
  - z はパーティション番号
- 例:
  - /dev/sda は最初に見つかったドライブである
  - /dev/sdb2 は 2 番目のドライブの 2 番目のパーティションである

Linux カーネルは、/dev ディレクトリにある **デバイスノード** を介して、ディスクとの低レベルでアクセスを行います。通常デバイスノードは、カーネルの **仮想ファイルシステム** のインフラを介してのみアクセスされます。直接デバイスノードを操作する生 (raw) アクセスを行うと、ファイルシステムは簡単に破壊されてしまいます。

低水準アクセスを使ったパーティションをフォーマットのコマンドの例:

```
$ sudo mkfs.ext4 /dev/sda9
```

SCSI と **SATA** ディスクのデバイスノードは、単純な命名規則を採用しています:

- 最初のハードディスクは /dev/sda となります。
- 2 番目のハードディスクは /dev/sdb となります。

パーティションも、次のように簡単に番号付けされます:

- /dev/sdb1 は、2 番目のディスクの最初のパーティション
- /dev/sdc4 は、3 番目のディスク上の 4 番目のパーティション

上記で sd は **SCSI** または **SATA** ディスクを意味します。**IDE** ディスクがあった時代には /dev/hda3、/dev/hdb などと命名されていました。

`ls -l /dev` を実行すると、現在利用可能なディスクデバイスノードが表示されます。

17.6 blkid と lsblk

blkid

- ブロックデバイスを見つけて、その属性をレポートするユーティリティ
- ブロックデバイスの一覧を表示する
- ブロックデバイスのアトリビュート（属性）を表示する
- コマンドの例:

```
$ sudo blkid
$ sudo blkid /dev/sda*
$ sudo blkid -L root
```

blkid は、ブロックデバイスを見つけてその属性をレポートするユーティリティ **libblkid** ライブラリで動作します。特定デバイス、またはデバイスのリストを引数に指定します。

blkid は、ブロックデバイスの種類（ファイルシステム、スワップなど）やコンテンツメタデータ（LABEL または UUID フィールドなど）から、デバイスの属性（トークン、NAME=value ペア）を特定します。

blkid は、フィンガープリント可能なデータを含むデバイスでのみ動作します。たとえば、空のパーティションではブロックを識別する **UUID** は生成されません。

blkid には、２種類のコマンド形式があります。特定の NAME=value を持つデバイスを検索するか NAME=value ペアを持つ複数のデバイスを表示します。

引数がなければ、すべてのデバイスについてレポートします。表示デバイスの指定とレポート属性の指定には、かなりの数のオプションがあります。



図 17.4: **blkid の使用例**

lsblk

- ツリー形式でブロックデバイスを表示



図 17.5: **lsblk の使用例**

```
$ lsblk -h
```

```
Usage:
 lsblk [options] [<device> ...]

 List information about block devices.

Options:
 -a, --all            print all devices
 -b, --bytes          print SIZE in bytes rather than in human readable format
 -d, --nodeps         don't print slaves or holders
 -D, --discard        print discard capabilities
 -z, --zoned          print zone model
 -e, --exclude <list> exclude devices by major number (default: RAM disks)
 -f, --fs             output info about filesystems
 -i, --ascii          use ascii characters only
 -I, --include <list> show only devices with specified major numbers
 -J, --json           use JSON output format
 -l, --list           use list format output
 -T, --tree           use tree format output
 -m, --perms          output info about permissions
 -n, --noheadings     don't print headings
 -o, --output <list>  output columns
 -O, --output-all     output all columns
 -p, --paths          print complete device path
 -P, --pairs          use key="value" output format
 -r, --raw            use raw output format
 -s, --inverse        inverse dependencies
 -S, --scsi           output info about SCSI devices
 -t, --topology       output info about topology
 -x, --sort <column>  sort output by <column>

 -h, --help           display this help
 -V, --version        display version

 Available output columns:
 ......
 For more details see lsblk(8).
```

17.7 パーティションのサイズ変更

パーティションのサイズ変更

- 多くの **Linux** システムは、パーティションを少なくとも２つ使用する:
  root (/)
  - ファイルシステムが利用する
  - ほとんどの **Linux** インストールは、複数のファイルシステムを持つ
  - サイズ変更はかなり困難である（**LVM** が利用できれば可能である）

  Swap
  - 物理メモリーの拡張として使用される
  - 物理メモリー（**RAM**）のサイズと同じにするのが一般的である
  - 複数のスワップパーティションやスワップファイルもありえる
- **Ubuntu** などはパーティションではなくファイルを swap に利用する、ただし:
  - フレキシブルだが
  - もしエラーやバグに当たった場合には危険度は高い

ほとんどのインストールでは、複数のパーティションに複数のファイルシステムがあり、**マウントポイント**で結ばれています。

ほとんどのファイルシステムでは、ルートパーティションサイズの変更は困難です。後述する **LVM** では、これが簡単にできます。

ルートパーティションだけで **Linux** を実行することも可能なのですが、ほとんどのシステムはより多くのパーティションを使用してバックアップの容易化、ディスクドライブのより効率的な使用、セキュリティの強化を図っています。

スワップサイズの推奨値は物理メモリーのサイズと同じですが、場合によっては２倍にすることが推奨されることもあります。正しい選択は、システムの利用シナリオ と ハードウェアの性能 などに依存します。

スワップ領域をどれだけ追加しても、どこかで限界はあるので必ずしも万能ではありません。メモリーの追加やシステムセットアップの再検討が必要な場合もあるでしょう。

17.8 パーティションテーブルのバックアップと復元

パーティションテーブルのバックアップと復元

- **MBR**
  - **dd** を利用する:
  - **MBR** のバックアップ（パーティションテーブルと一緒に）
    ```
    $ dd if=/dev/sda of=mbrbackup  bs=512 count=1
    ```
  - **MBR** を復元する
    ```
    $ sudo dd if=mbrbackup of=/dev/sda bs=512 count=1
    ```
- **GPT**
  - **MBR** システムでは **sgdisk** の利用も可能である
    ```
    $ sudo sgdisk --backup-file=gptbackup /dev/sda
    $ sudo sgdisk --load-backup-file=gptbackup /dev/sda
    ```

ディスクパーティションテーブルの変更によって、ファイルシステムの全データが削除されてしまう可能性があることを常に想定してください。（削除されるべきではありませんが、注意は必要です !）

したがって、変更などの作業を行う前に（まだバックアップされていない）すべてのデータをバックアップするのが賢明です。**MBR** システムでは **dd** を使って変換やコピーができます。**dd** では、単純な入力ミスやオプションの誤用によってディスク全体を破壊する可能性があるので注意してください。**dd** はプライマリーパーティションテーブルのみをコピーします。他のパーティション（例えば拡張パーティションなど）に含まれるパーティションテーブルは対象にしません。

GPT システムの場合次のように **sgdisk** ツールを使用するのが最適です:

```
x8:/tmp>sudo sgdisk -p /dev/sda
```

```
Disk /dev/sda: 1000215216 sectors, 476.9 GiB
Model: SAMSUNG MZNLN512
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): DBDBE747-8392-4B62-97AA-6214490073DC
Partition table holds up to 128 entries
....
Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048          534527   260.0 MiB   EF00  EFI System Partition
   2          534528          567295   16.0 MiB    0C01  Microsoft reserved ...
....
```

純粋な **MBR** システムで実行する場合は以下のようになります:

```
c8:/tmp>sudo sgdisk -p /dev/sda
```

```
****************************************************************
Found invalid GPT and valid MBR; converting MBR to GPT format
in memory.
****************************************************************
Disk /dev/sda: 500118192 sectors, 238.5 GiB
Model: Crucial_CT256MX1
Sector size (logical/physical): 512/4096 bytes
....
```

17.9 パーティションテーブルエディタ

fdisk は、メニュー方式のパーティションテーブルエディタです。これは、最も一般的で最も柔軟なパーティションテーブルエディタの１つです。他のパーティションテーブルエディタと同様、変更を行う前に現在のパーティションテーブル設定を書き留めるか、現在の設定のコピーを作成してください。

sfdisk は、非対話型の **Linux** ベースのパーティションエディタプログラムで、スクリプトで役立ちます。**sfdisk** ツールは注意して使用してください！

GPT システムでは、**gdisk** と **sgdisk** が同様の役割を果たします。どちらも **MBR** システムも操作できます。

parted は、GNU のパーティション操作プログラムです。パーティション（特定のファイルシステムを含む）の作成、削除、サイズ変更、移動 ができます。**parted** コマンドの GUI インターフェイス版が **gparted** です。

多くの **Linux** ディストリビューションには、CDROM または USB スティックから実行できる **ライブ／インストールバージョン** があります。通常これらのメディアには **gparted** が含まれているので、パーティションプログラムを実行していないディスク（＝システムのライブ起動に使っていない元々の起動ドライブなど）をグラフィカルパーティションツールを使って簡単に操作できます。

17.10 fdisk

fdisk の使用例

- どの Linux インストールにも常に含まれ、簡単に利用できる
- メニュー操作方式、プロンプトに答えていけばよい:
  - m: メニューを表示
  - p: パーティションテーブルの一覧を表示
  - n: 新しいパーティションを作成
  - d: パーティションを削除
  - t: パーティションの種類を変更
  - w: 新しいパーティションテーブル情報を書き込んで終了
  - q: 変更せずに終了
- パーティションテーブルの編集:
  ```
  $ sudo fdisk /dev/sda
  ```
- パーティションテーブルの表示だけで変更は何もしない:
  ```
  $ sudo fdisk -l /dev/sda
  ```

fdisk の実行には、root 権限が必要です。処理は少し複雑になる可能性があり、注意が必要です。**fdisk** はメニュー駆動式のインターフェイスを持っています。まず、特定のディスクで起動します。

```
$ sudo fdisk /dev/sdb
```

上に列挙した一文字のコマンドを使って操作します。

幸いなことに、w を入力してパーティションテーブルをディスクに書き込むまで、実際の変更は行われません。したがって、w を使用してディスクに書き込む前に（p を使用して）パーティションテーブルが正しいことを確認することが重要です。何かが間違っていた場合 q を使用すれば安全に終了できます。

システムを再起動するまで新しいパーティションテーブルは使用されません。ただし、次のコマンドで再読み込みできます:

```
$ sudo partprobe -s
```

を実行して、修正されたパーティションテーブルを試してください。ただし、これは常に確実に機能するとは限りません。パーティションの混在は壊滅的なものになる可能性があるため、新しいパーティションのフォーマットなどの前には再起動をお勧めします。

オペレーティングシステムが、現在どのパーティションを認識しているかを確認する次のコマンドは、いつでも利用可能です:

```
$ cat /proc/partitions
```

17.11 演習

デモ教材ビデオ

using_fdisk_demo.mp4

課題 17.0: ループバックデバイスを使って、パーティションのエミュレーションをする

本セクション中の演習では **mkfs** を使いファイルシステムのフォーマットと、**mount** を使いルートファイルシステムの階層構造中にマウントを行います。これらのコマンドの詳細は、次のセクションで説明します。

最初の３つの演習では **parted** プログラムを使う場合と、使わない場合の両方で、**ループデバイス** 機構を利用します。

重要

本コースの後半の演習では、パーティション分けされていないディスク領域が必要になります。1〜2GB 程度とそれほど大きな領域は必要ありません。

自分のマシンを使っている場合、その領域があるか確認してください。もし無ければ、**gparted** を使うか、説明した/する方法で、パーティションとファイルシステム (こちらを先に！) を縮小して、空き領域を用意してください。

注目

もし、物理的に空いているディスク領域がある場合は、以下の手順を踏む **必要はありません**。しかし、実習しておくことは有益です。

課題 17.1: ファイルをディスクパーティションイメージとして使う

最初の演習では、ハードディスクの完全なパーティションイメージを持つファイルを作成し、実際のハードパーティションと同様に扱えるようにします。その後の演習で、複数のパーティションを格納したり、完全なディスクのように振る舞えるようにします。

1. 大きさが 1GB の、ゼロで満たされたファイルを作成します:

   ```
   $ dd if=/dev/zero of=imagefile bs=1M count=1024
   ```

   もしパーティションに余裕が無ければ、小さいファイルでも構いません。

2. ファイルシステムを作成します:

   ```
   $ mkfs.ext4 imagefile
   ```

   ```
   mke2fs 1.42.9 (28-Dec-2013)
   imagefile is not a block special device.
   Proceed anyway? (y,n) y
   Discarding device blocks: done
   .....
   ```

   mkfs.ext3、**mkfs.vfat**、**mkfs.xfs** などを使って、別のファイルシステムにフォーマットしても構いません。

3. 適当な場所にマウントします:

   ```
   $ mkdir mntpoint
   $ sudo mount -o loop imagefile mntpoint
   ```

これで、ファイルを格納したりするなど普通に利用できるようになりました。

4. 終了したらアンマウントします:

```
$ sudo umount mntpoint
```

loopオプションを用いる別の方法でマウントします:

```
$ sudo losetup /dev/loop2 imagefile
$ sudo mount /dev/loop2 mntpoint
 ....
$ sudo umount mntpoint
$ sudo losetup -d /dev/loop2
```

この後の演習で **losetup** について説明します。/dev/loop[0-7] を使えるようになりますが、このあと説明するように、これらはすでに使用できなくなっています。

実パーティションの代わりにループデバイスファイルを用いる方法は便利ですが、性能測定やベンチマーキングにはふさわしくありません。なぜなら、ファイルシステム層をもうひとつの層の上に置いたことになるからです。そのため、性能面に悪い影響を与え、ファイルを作成したファイルシステムの動作に影響を受けるからです。

課題 17.2: ディスクイメージファイルをパーティション分割する

次のステップではイメージファイルを複数のパーティションに分割し、各々のパーティションをファイルシステムやスワップとして利用できるようにします。

先の演習で作成したイメージファイルを使うか、新規に作成してもかまいません。

1. イメージファイルに対して **fdisk** を実行します:

```
$ sudo fdisk -C 130 imagefile
```

```
Device does not contain a recognized partition table
Building a new DOS disk label with disk identifier 0x6280ced3.
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help):
```

-C 130により、ドライブ中の疑似シリンダ数を 130 に設定します。これは **RHEL 6** に含まれるような古いバージョンの **fdisk** にのみ必要です。しかし、(この設定をしても) 他のディストリビューションには悪影響を与えません。

2. m を入力してコマンド一覧を表示します:

```
m
```

```
Command (m for help): m
Command action
a   toggle a bootable flag
b   edit bsd disklabel
c   toggle the dos compatibility flag
d   delete a partition
g   create a new empty GPT partition table
G   create an IRIX (SGI) partition table
l   list known partition types
m   print this menu
n   add a new partition
o   create a new empty DOS partition table
p   print the partition table
q   quit without saving changes
```

```
s   create a new empty Sun disklabel
t   change a partition's system id
u   change display/entry units
v   verify the partition table
w   write table to disk and exit
x   extra functionality (experts only)

Command (m for help):
```

3. 大きさ 256MB（他でも構いません）のプライマリ パーティションを作成します:

```
Command (m for help): n
```

```
Partition type:
p   primary (0 primary, 0 extended, 4 free)
e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-2097151, default 2048):
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-2097151, default 2097151): +256M
Partition 1 of type Linux and of size 256 MiB is set
```

4. もうひとつ 256MB のプライマリ パーティションを追加します:

```
Command (m for help): n
```

```
Partition type:
p   primary (1 primary, 0 extended, 3 free)
e   extended
Select (default p): p
Partition number (2-4, default 2): 2
First sector (526336-2097151, default 526336):
Using default value 526336
Last sector, +sectors or +size{K,M,G} (526336-2097151, default 2097151): +256M
Partition 2 of type Linux and of size 256 MiB is set

Command (m for help): p

Disk imagefile: 1073 MB, 1073741824 bytes, 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x6280ced3

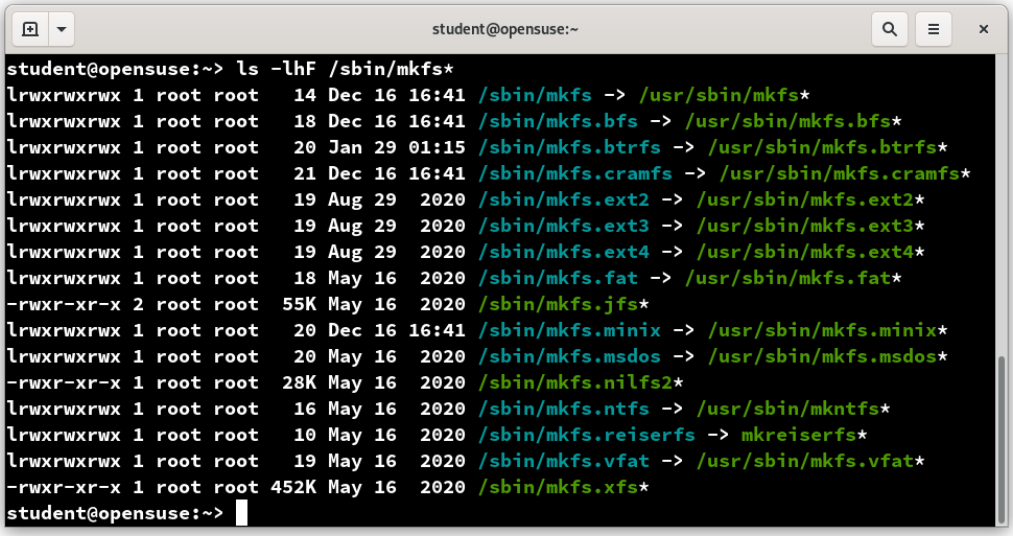
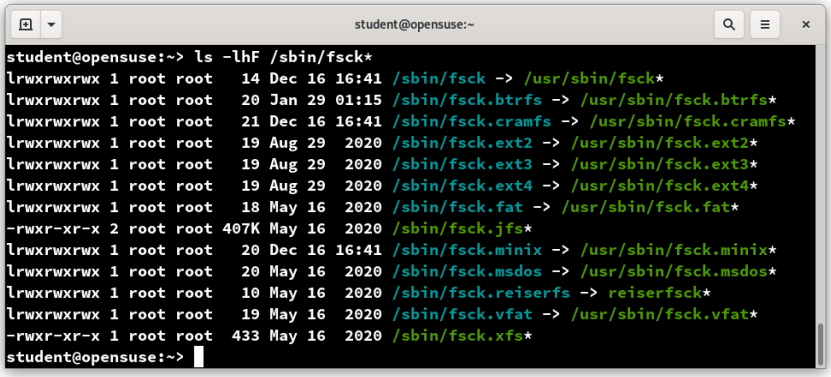
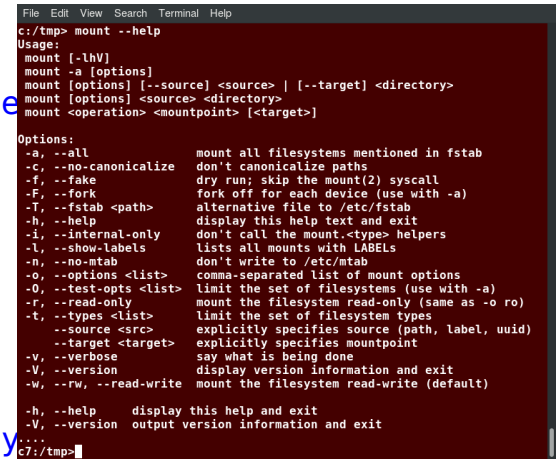
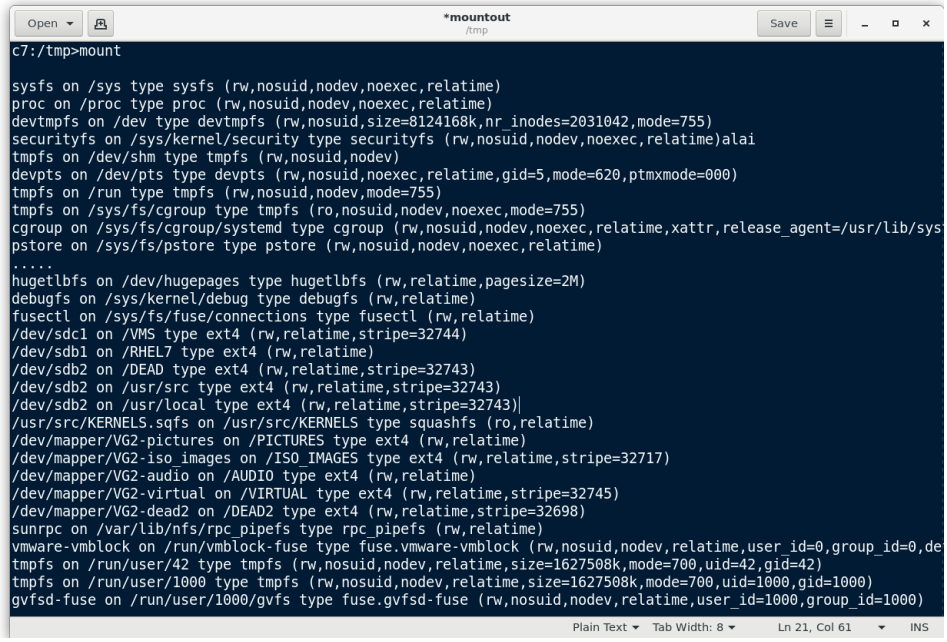
Device Boot      StartEnd      Blocks   Id  System
imagefile1          2048      526335      262144   83  Linux
imagefile2          526336     1050623      262144   83  Linux
```

5. ディスクにパーティションテーブルを書き込み、終了します:

```
Command (m for help): w
```

```
The partition table has been altered!

Syncing disks.
```

この演習はうまく行きましたが、ここで作成した 2 つのパーティションを利用する方法がまだわかりません。これを次の演習で説明をしていきます。

課題 17.3: losetup と parted を使う

ここでは、さらに次のことを実験します:

- ループ デバイスと **losetup**
- 非インタラクティブ（バッチ式）でパーティション分割するために **parted** を使う。

以降の手順を実行する前に **losetup** と **parted** の **man ページ**を読んでおいてください。

以前作成したイメージ ファイルそのままか、ゼロクリアしたもの、または新規に作成しても構いません。

1. **ループ**デバイスに割り当てます:

```
$ sudo losetup -f
/dev/loop1

$ sudo losetup /dev/loop1 imagefile
```

最初のコマンドは、**空いている** 最初のループデバイスを探します。すでにいくつかのループデバイスを利用している可能性があるからです。たとえば、上記のコマンドを実行する前にシステムが使っているかもしれません:

```
$ losetup -a
/dev/loop0: []: (/usr/src/KERNELS.sqfs)
```

圧縮された **squashfs** が /dev/loop0 を使って、読み出し専用でマウントされています。(コマンドの出力は、ディストリビューションによって異なります)。この結果を無視して **losetup** を使って/dev/loop0 を操作すると、ファイルは破壊されてしまいます。

2. ループデバイス（イメージファイル）に、ディスクパーティションラベルを作成します:

```
$ sudo parted -s /dev/loop1 mklabel msdos
```

3. ループデバイスに３つのプライマリパーティションを作成します:

```
$ sudo parted -s /dev/loop1 unit MB mkpart primary ext4 0 256
$ sudo parted -s /dev/loop1 unit MB mkpart primary ext4 256 512
$ sudo parted -s /dev/loop1 unit MB mkpart primary ext4 512 1024
```

4. パーティションテーブルを確認します:

```
$ fdisk -l /dev/loop1
Disk /dev/loop1: 1073 MB, 1073741824 bytes, 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x00050c11

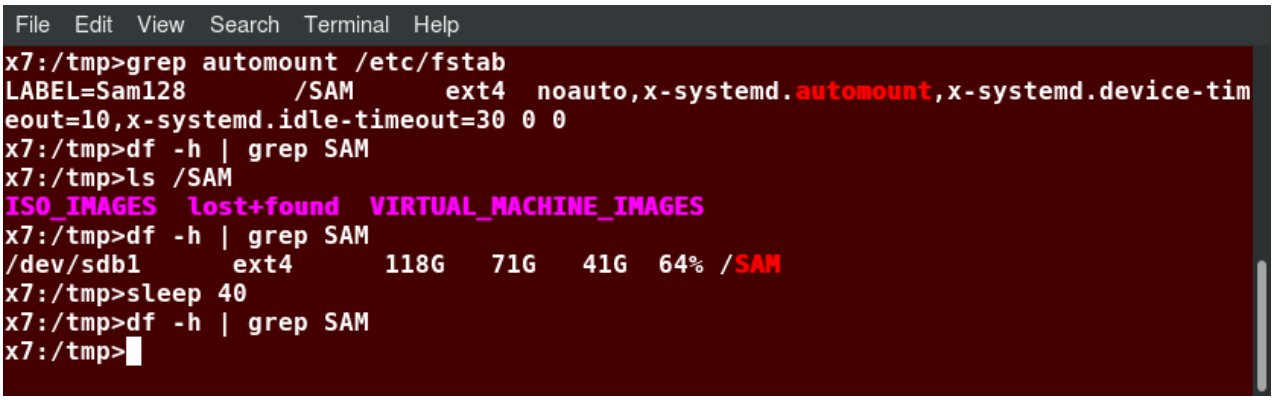
      Device Boot      Start         End      Blocks   Id  System
/dev/loop1p1               1      500000      250000   83  Linux
/dev/loop1p2          500001     1000000      250000   83  Linux
/dev/loop1p3         1000001     2000000      500000   83  Linux
```

5. 使用しているディストリビューションによって多少異なります。たとえば **RHEL** と **Ubuntu** では新しくデバイスノードが作成されます:

```
$ ls -l /dev/loop1*
brw-rw---- 1 root disk   7, 1 Oct  7 14:54 /dev/loop1
brw-rw---- 1 root disk 259, 0 Oct  7 14:54 /dev/loop1p1
brw-rw---- 1 root disk 259, 3 Oct  7 14:54 /dev/loop1p2
brw-rw---- 1 root disk 259, 4 Oct  7 14:54 /dev/loop1p3
```

次のように使います。

6. パーティションにファイルシステムを格納します:

```
$ sudo mkfs.ext3 /dev/loop1p1
$ sudo mkfs.ext4 /dev/loop1p2
$ sudo mkfs.vfat /dev/loop1p3
```

7. ３つのファイルシステムを全てマウントし、使えることを確認します:

```
$ mkdir mnt1 mnt2 mnt3

$ sudo mount /dev/loop1p1 mnt1
$ sudo mount /dev/loop1p2 mnt2
$ sudo mount /dev/loop1p3 mnt3

$ df -Th
```

```
Filesystem                Type      Size  Used Avail Use% Mounted on
/dev/sda1                 ext4       29G  8.5G   19G  32% /
....
/dev/loop1p1              ext3      233M  2.1M  219M   1% mnt1
/dev/loop1p2              ext4      233M  2.1M  215M   1% mnt2
/dev/loop1p3              vfat      489M     0  489M   0% mnt3
```

8. ファイルシステムの利用が終了したら、もとに戻します:

```
$ sudo umount mnt1 mnt2 mnt3
$ rmdir mnt1 mnt2 mnt3
$ sudo losetup -d /dev/loop1
```

課題 17.4: 実ハードディスクのパーティションを分割する

パーティションを作成していない、空き領域を持つハードディスクがあれば **fdisk** を使って、プライマリパーティション、拡張パーティションを作成してロジカルパーティションを新規に作成してください。作成したパーティションテーブルをディスクに書き込み、フォーマット、マウントしてください。

章 18

ファイルシステムの機能：属性、作成、検査、マウント

18.1 拡張属性

lsattr と chattr

- ファイルに拡張属性を設定することができる
  - i: 変更不可
  - a: 追加のみ許可
  - d: ダンプなし
  - A: atime の更新なし
- 属性フラグの値は **lsattr** で表示する
- 属性フラグの値は **chattr** で設定する
- 属性フラグの値は、ファイルの inode フラグに記録される
- ユーザーだけが変更および設定できる

拡張属性は、ファイルシステムが直接解釈しないメタデータをファイルに結びつけるものです。この拡張属性の管理には、4 種類の **namespaces (名前空間)** を使っています。それは、user, trusted, security, system です。system 名前空間は **アクセス制御リスト（ACL）**に使用され、security 名前空間は **SELinux** によって使用されます。

属性フラグの値は、ファイルの inode に保存され root ユーザーのみが変更および設定できます。それらは **lsattr** で表示され **chattr** で設定されます。属性フラグはファイルに設定できます:

- `i` (**変更不可 (immutable)**): immutable 属性を持つファイルは変更できません（root ユーザでも変更できません）。削除や名前の変更もできません。ハードリンクも作成できず、ファイルにデータを書き込むこともできません。この属性を設定または削除できるのはスーパーユーザーのみです。
- a (**追加のみ許可 (append-only)**): Append-only 属性が設定されたファイルは、書き込みする場合は追加モードでのみ開くことができます。この属性を設定または削除できるのはスーパーユーザーのみです。
- d (**ダンプなし (no-dump)**): No-dump 属性が設定されたファイルはダンプの対象外となります。バックアップが無意味なスワップファイルとキャッシュファイルに指定すると、無駄なダンプ処理を回避できます。
- A (**atime の更新なし (No atime update)**): No-atime-update 属性が設定されたファイルは、ファイルがアクセスされても変更されなかったときは **atime**（アクセス時間）レコードを変更しません。これにより、システム上のディスク I/O の量が減少するため、一部のシステムのパフォーマンスが向上します。

設定できるフラグは他にもあります。`man chattr` と入力するとリスト全体が表示されます。
chattr の書式は次のとおり:

```
$ chattr [+|-|=mode] filename
```

lsattr ファイルの属性を表示します:

```
$ lsattr filename
```

18.2 ファイルシステムの作成とフォーマット

mkfs

- **mkfs** の一般的なコマンド形式:

```
mkfs [-t fstype] [options] [device-file]
```

- `[device-file]` は `/dev/sda3` や `/dev/vg/lvm1` 等デバイス名
- 各ファイルシステム毎に固有なフォーマットオプションがある
- 各ファイルシステムは独自の **mkfs** プログラムを持っている
- 等価なコマンドの例:

```
$ sudo mkfs -t ext4 /dev/sda10
$ sudo mkfs.ext4 /dev/sda10
```

すべてのファイルシステムは、パーティション上のファイルシステムを **フォーマット**（作成）するためのユーティリティを持っています。そのようなユーティリティの一般的な名前は、**mkfs** ですが **mkfs** は各ファイルシステムに固有なフォーマットプログラムのフロントエンドです。それぞれのファイルシステムに固有なフォーマットプログラムには、特有なオプションがある場合があります。



図 18.1: **mkfs**

それぞれの **mkfs.*** プログラムの詳細は **man** ページを参照してください。

18.3 ファイルシステムのチェックと修復

fsck

- 設定した累積利用時間、またはマウント回数に到達すると自動的に実行する
- クリーンにアンマウント出来なかった時は、次回ブート時に実行する
- マウントされているオンラインのファイルシステムはチェックできない
- **fsck** の一般的なコマンド形式:

```
fsck [-t fstype] [options] [device-file]
```

[device-file] は通常 `/dev/sda3` や `/dev/vg/lvm1` 等のデバイス名

- パーティションのスーパーブロックを検証するので、通常はファイルシステムの指定は必要ない
- 等価なコマンドの例:

```
$ sudo fsck -t ext4 /dev/sda10
$ sudo fsck.ext4 /dev/sda10
```

すべてのファイルシステムは、エラーチェック（と可能なら検出されたエラーを修復する）ユーティリティを持っています。
これらのユーティリティの一般名は **fsck** ですが、これはファイルシステムに固有のチェックプログラムのフロントエンドです。



図 18.2: **fsck**

次回ブート時に、マウントされた全ファイルシステムを強制チェックするには:

```
$ sudo touch /forcefsck
$ sudo reboot
```

チェックでエラーがでなければ、作成されたファイル `/forcefsck` は自動的に削除されます。

`-r` オプションを指定して、見つかったエラーを１つずつ手動で修正するか、`-a` オプションを使用して可能な限り最善の方法で自動修正を試みるかを選択できます。さらに各ファイルシステムには、独自のチェックを実行する **fsck** のオプションがある場合があります。

18.4 ファイルシステムのマウント

ファイルシステムのマウント

- ファイルシステムを、指定したディレクトリーにマウントする
  `$ mount -t ext4 /dev/sdb4 /home`
  - **ext4** ファイルシステムをマウントする
  - 通常 `-t` でファイルシステムのタイプを指定する必要は無い
  - ファイルシステムはハードディスク (`/dev/sdb4`) 上の決められたパーティション上に配置される
  - ファイルシステムは、現在のディレクトリーツリーの `/home` という場所にマウントされる
  - `/home` ディレクトリーに最初からファイルがあった場合、それらのファイルはパーティションがアンマウントされるまで見えなくなる

Linux でアクセス可能なすべてのファイルは、先頭をルートディレクトリ (`/`) とする１つの大きな階層ツリー構造になっています。２つ以上のパーティションが（各パーティションに別のファイルシステムを指定することも可能です）、同じファイルシステムツリーに統合されることも珍しくありません。これらのパーティションは、別の物理デバイス（＝ドライブ）上や、更にはネットワーク上にあっても良いのです。

mount プログラムを使用すると、ツリー構造の任意の場所にファイルシステムをマウントできます。**umount** は、それらのマウントを解除（アンマウント）します。

マウントポイント とは、ファイルシステムがアッタチされるディレクトリです。**mount** を実行する前にそのディレクトリが存在している必要があります。**mkdir** を使用して空のディレクトリを作成できます。既存のディレクトリをマウントポイントとして使用しようとした時に、もしそのディレクトリに先に何かファイルがあった場合には、そのファイルはマウント後に非表示となります。それらのファイルは削除されるのではなく、ファイルシステムがアンマウントされると再び表示されます。

スーパーユーザーだけが、ファイルシステムを マウント／アンマウント できます。

mount

- **mount** の一般的なコマンド形式:

```
mount [options] <source> <dire
```

- **デバイスノード**、**ラベル**、**UUID** を使ったマウントが可能です

- ファイルシステム独自のオプションがある。
  共通オプション例:

```
$ sudo mount -o remount,ro /my
```

  リードオンリーでリマウント



図 18.3: **mount**

以下は、すべて同じマウント結果になります:

```
$ sudo mount /dev/sda2   /home
$ sudo mount LABEL=home  /home
$ sudo mount    -L home  /home
$ sudo mount UUID=26d58ee2-9d20-4dc7-b6ab-aa87c3cfb69a /home
$ sudo mount   -U 26d58ee2-9d20-4dc7-b6ab-aa87c3cfb69a /home
```

ラベルは、**e2label** などのファイルシステムの種類に固有のユーティリティによって割り当てられます。**UUID** は、パーティションがファイルシステムのコンテナとして作成され、**mkfs** でフォーマットされる時に割り当てられます。

これら３つのデバイス指定方法はいずれも使用できますが、最新のシステムではデバイスノード形式の使用は推奨していません。その理由は、システムの起動方法によってどのハードドライブが最初に見つかるかは一定でないため、デバイスの名前が変わってしまう可能性があるためです。

ラベルは改善されていますが、まれに２つのパーティションが同じラベルで作成される場合があります。一方で **UUID** はパーティションの作成時に作成され、常にユニークな値をとります。

現在マウントされているファイルシステム



図 18.4: **現在マウントされているファイルシステム**

umount

- ファイルシステムのアンマウントに使用する
- **umount** の一般的なコマンド形式:

```
umount [device-file | mount-point]
```

- 例:
  - /home ファイルシステムのアンマウント:

```
$ umount /home
```

  - /dev/sda3 デバイスのアンマウント:

```
$ umount /dev/sda3
```

ファイルシステムをアンマウントするコマンドは **umount** です。(**unmount** ではありません !)

mount と同様に **umount** にも多くのオプションがあり、その多くはファイルシステムの種類に固有のものです。繰り返しになりますが、**man** ページは特定のオプションに関する最適な情報源です。

ファイルシステムをアンマウントするときに発生する最も一般的なエラーは、現在使用中のファイルシステムに対してアンマウントしようとすることです。つまり、アプリケーションがファイルシステム内のファイル、または他のエントリを使用している場合です。

このエラーは、マウントされたファイルシステムのディレクトリでターミナルウィンドウを開くのと同じくらい、簡単に解決できます。そのウィンドウで **cd** と打つか、アプリケーションを強制終了するだけで、デバイスがビジー状態であるというエラーが解消されアンマウントが可能になります。

ただし、このエラーを引き起こすプロセスが他にある場合、ファイルシステムをアンマウントする前にそれらを強制終了する必要があります。**fuser** でどのユーザーがファイルシステムを使用しているか確認して、そのユーザーを強制終了できます (この操作には注意が必要です、強制終了する前にユーザーに警告すべきかもしれません)。**lsof** ("list open files") を使用して、どのファイルが使用中でアンマウントをブロックしているか確認することができます。

18.5 NFS

ネットワークの共有 (NFS)

- 最も一般的なネットワーク共有は、**NFS** である
- それ以外には、**AFS**、**SMB** (**CIFS**) がある
- **mount** でネットワーク共有を指定する方法:

```
$ sudo mount -t nfs myserver.com:/shdir /mnt/shdir
```

- `/etc/fstab` に書くと起動時にマウントされる、それ以降は `mount -a` を使う:

```
myserver.com:/shdir /mnt/shdir nfs rsize=8192,wsize=8192,timeo=14,intr 0
```

- ネットワーク接続が有効になる前にシステムは **NFS** をマウントするかもしれない。この調整に `netdev` と `noauto` オプションが利用可能

- システムは、ネットワークが起動する前に NFS ファイルシステムをマウントしようとする場合がある。これを回避するために `netdev` と `noauto` オプションが使用可能。(詳細は `man nfs` を見て **mount** のオプションを調査する)
- **autofs** または **automount** を使って解決することも可能
- **mount** には極めて多彩なオプションがあり、いくつかは **nfs** 専用である
  nfs と **mount** それぞれの詳細は **man** ページを参照

ネットワーク共有を介してリモートファイルシステムをマウントするのは一般的で、（マウントされたファイルシステムは）ローカルマシン上にあるかのように見えます。

おそらく、今までに使用されている最も一般的な方法は **NFS** (**N**etwork **F**ile **S**ystem) です。

NFS は 1989 年に **Sun Microsystems** によって最初に開発され、継続的に更新されてきました。最新のシステムでは 2000 年以降継続的に更新されている **NFSv4** を使用しています。

他のネットワークファイルシステムには **AFS** (**A**ndrew **F**ile **S**ystem) と **SMB** (**S**erver **M**essage **B**lock) があります。
SMB は **CIFS** とも呼ばれます。

ネットワークファイルシステムは、ネットワーク共有上にそれが存在しない、もしくはネットワーク自体が利用できないなどの要因で利用できない可能性があるため、システムはその可能性を考慮しておく必要があります。

このような状況になっても、システムが指定された時間より長く待っている間にハングアップ、またはブロックしないようにする必要があります。これらの設定は、mount コマンドか `/etc/fstab` で指定できます。

18.6 ブート時のマウントと/etc/fstab

ブート時のマウント

- システム初期化中に次のコマンドを実行すると、/etc/fstab 内にリストされている全てのファイルシステムがマウントされる:

```
$ mount -a
```

- 起動時にはローカルと、ネットワークマウントされるリモートファイルシステムのどちらもマウントが可能である

/etc/fstab には、ブート時にマウント可能なファイルシステムを定義します。どのファイルシステムがマウントできるか、ローカルのどの位置にマウントするか、誰がどのような権限でマウントするか、などが記載されています。/etc/fstab にファイルやディレクトリが書かれている場合、起動時にマウントされることもあります。

ここにはローカルと、NFS や samba のようなリモートにあるネットワーク共有ファイルの、両方を含めることができます。

/etc/fstab

- ファイルシステム、標準的なマウントポイント、マウント時のオプションの情報が含まれる
- 各レコードには、マウントされるファイルシステムに関するレコードが、スペースで区切られて記載される
  - デバイスファイル、ラベルまたは UUID
  - マウントポイント
  - ファイルシステムタイプ
  - カンマで区切られたオプションのリスト
  - ダンプ周期（または 0）
  - **fsck** パス番号（または 0）
- **mount** と **umount** コマンドは `/etc/fstab` 内の情報を参照する

このファイルの各レコードには、起動時にマウントされるファイルシステムに関する情報が含まれます。レコードの最初のアイテムは、(`/dev/sda1` などの) デバイスファイル名、ラベルまたは UUID のどれかになります。次にファイルシステムのマウントポイント（ツリー構造のどこに組み込まれるか）が続きます。次にカンマで区切られたオプションをもった、ファイルシステムタイプが記述されます。続いてダンプ周期（dump -w コマンドで利用）かゼロ（**dump** の対象外とするというの意味）、その次に **fsck** パス回数かゼロ（このパーティションを **fsck** の対象外とするという意味）が記述されます。

```
x8:/tmp> cat /etc/fstab
#
# /etc/fstab
# Created by anaconda on Tue Aug 27 15:29:19 2019
#
# Accessible filesystems, by reference, are maintained under '/dev/disk/'.
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info.
#
# After editing this file, run 'systemctl daemon-reload' to update systemd
# units generated from this file.
#
UUID=53ea9807-fd58-1234-9460-d03ec36f73a3 /     ext4    defaults 1 1
UUID=29C4-7777          /boot/efi               vfat    umask=0077,shortname=winnt 0 2
LABEL=ALL               /ALL                    ext4    defaults 1 2
LABEL=UBUNTU            /UBUNTU                 ext4    defaults 1 2

/ALL/usr/local          /usr/local              none    bind 0 0
/ALL/usr/src            /usr/src                none    bind 0 0
/ALL/VMS                /VMS                    none    bind 0 0
/ALL/RESOURCES          /RESOURCES              none    bind 0 0

LABEL=Sam128            /SAM                    ext4    noauto 0 0
/usr/src/KERNELS.sqfs   /usr/src/KERNELS        squashfs loop  0 0

/dev/sda1               /Cbootefi       vfat    umask=0077,shortname=winnt 0 2
x8:/tmp>
```

図 18.5: **/etc/fstab の例**

18.7 自動マウント

ファイルシステムの自動マウント

- ファイルシステムの使用時に利用可能 (**mount**) となり、アイドル状態になると切り離される (**umount**)
- **autofs**:
  - 何年も前から利用可能な技術である
  - **autofs** パッケージのインストレーションが必要
  - 柔軟性があり、設定ファイルは `/etc` に書く
- **automount**
  - **systemd** の機能
  - `/etc/fstab` を修正し再起動するか、**systemd** を再スタート:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart local-fs.target
```

必要なときにのみファイルシステムをマウントできる機能は、**Linux** システムには長い間存在しています。

このユーティリティを使うには、適切なパッケージマネージャを使用してインストールされた **autofs** パッケージと `/etc` 内の構成ファイルが必要になります。

autofs は非常に柔軟でよく知られていますが、一方で **systemd** ベースのシステム（全てのエンタープライズ向け **Linux** ディストリビューションが導入しています）には、**systemd** のフレームワークに組み込まれた **自動マウント (automount)** 機能が備わっています。これを利用するための設定は、適切なデバイス、マウントポイント、マウントオプションを指定する行を **/etc/fstab** に追加するだけです。具体的には以下の指定をして:

```
LABEL=Sam128 /SAM ext4 noauto,x-systemd.automount,x-systemd.device-timeout=10,x-systemd.idle-timeout=30 0 0
```

再起動するか、以下のコマンドを使います:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart local-fs.target
```

次ページで例を示し、オプションについて説明します。

自動マウントの例



図 18.6: **automount の例**

上の例では、使用時にのみシステムに接続される **USB** ペンドライブをマウントします。
`/etc/fstab` のオプションは以下のとおりです:

- `noauto` 起動時にマウントしません。`auto` は **automount** の意味ではありません。
- `x-systemd.automount` **systemd** の **自動マウント (automount)** 機能を使用します。
- `x-systemd.automount.device-timeout=10` このデバイスが使用できない場合、すなわち **NFS** を介してアクセス可能なネットワークデバイスの場合は、ハングアップする代わりに 10 秒後にタイムアウトします。
- `x-systemd.automount.idle-timeout=30` デバイスが 30 秒間使用されない場合は、アンマウントします。

デバイスはブート中にマウントされる **かも** しれませんが、指定タイムアウト時間にはアンマウントされることに注意してください。
スクリーンショットは、デバイスを使いたい時にだけ利用する（= マウントする）方法を示しています。

18.8 演習

課題 18.1: ファイル属性について

1. 自分のユーザーアカウントで、空の /tmp/appendit ファイルを **touch** で作成してください。
2. **cat** を利用して、/etc/hosts の内容を /tmp/appendit に追加してください。
3. /tmp/appendit を /etc/hosts と比較してください。違いは無いはずです。
4. **chattr** コマンドで/tmp/appendit に追加のみ（append-only）属性を追加してください。エラーが出るはずです。なぜでしょうか。
5. root ユーザーで追加のみ（append-only）属性を追加してください。今度は成功するはずです。**lsattr** コマンドで、ファイルの拡張属性を見てください。
6. 一般のユーザーで **cat** コマンドを使って/etc/passwd の内容を /tmp/appendit にコピーしてください。エラーがでるはずです。なぜでしょうか。
7. 同じことを root ユーザーで実行してください。これもエラーになります。なぜでしょうか。
8. 一般のユーザーで追加のリダイレクション（>>）を使って、/etc/passwd ファイルを /tmp/appendit に追加してください。成功します。結果を調べてください。
9. root ユーザーで/tmp/appendit の変更不可（immutable）属性をセットし、拡張属性を確認してください。
10. 一般のユーザーと root ユーザーの両方で/tmp/appendit への追加、ファイル名称の変更、ファイルへのハードリンク、ファイルの削除を実行してください。
11. 拡張属性を削除することでファイルを削除できるようになります。実行してください。

解 18.1

1. ```
   $ cd /tmp
   $ touch appendit
   $ ls -l appendit
   ```

   ```
   -rw-rw-r-- 1 coop coop 0 Oct 23 19:04 appendit
   ```

2. ```
   $ cat /etc/hosts > appendit
   ```

3. ```
   $ diff /etc/hosts appendit
   ```

4. ```
   $ chattr +a appendit
   ```

   ```
   chattr: Operation not permitted while setting flags on appendit
   ```

5. ```
   $ sudo chattr +a appendit
   $ lsattr appendit
   ```

   ```
   -----a-------e-- appendit
   ```

6. ```
   $ cat /etc/passwd > appendit
   ```

   ```
   bash: appendit: Operation not permitted
   ```

7. ```
   $ sudo su
   $ cat /etc/passwd > appendit
   ```
   ```

```
bash: appendit: Operation not permitted
```

```
$ exit
```

8.
```
$ cat /etc/passwd >> /tmp/appendit
$ cat appendit
```

9.
```
$ sudo chattr +i appendit
$ lsattr appendit
```

```
----ia-------e- appendit
```

10.
```
$ echo hello >> appendit
```

```
-bash: appendit: Permission denied
```

```
$ mv appendit appendit.rename
```

```
mv: cannot move `appendit' to `appendit.rename': Operation not permitted
```

```
$ ln appendit appendit.hardlink
```

```
ln: creating hard link `appendit.hardlink' => `appendit': Operation not permitted
```

```
$ rm -f appendit
```

```
rm: cannot remove `appendit': Operation not permitted
```

```
$ sudo su
$ echo hello >> appendit
```

```
-bash: appendit: Permission denied
```

```
$ mv appendit appendit.rename
```

```
mv: cannot move `appendit' to `appendit.rename': Operation not permitted
```

```
$ ln appendit appendit.hardlink
```

```
ln: creating hard link `appendit.hardlink' => `appendit': Operation not permitted
```

```
$ rm -f appendit
```

```
rm: cannot remove `appendit': Operation not permitted
```

```
$ exit
```

11.
```
$ sudo su
$ lsattr appendit
```

```
----ia-------e- appendit
```

```
$ chattr -ia appendit
$ rm appendit
```

```
rm: remove regular file `appendit'? y
```

```
$ ls appendit
```

```
ls: cannot access appendit: No such file or directory
```

課題 18.2: マウントのオプション

新しいパーティション、またはループバックファイル

本演習では、新しいパーティションを作成するか、ループバックファイルを利用するか、のどちらかをします。大きな違いはありませんが、両方のやり方を説明します。

1. **fdisk** を使って 250MB の新パーティションを作成します。通常 /dev/sda に作成します。または、ループバックファイルを利用して、新しいパーティションをシミュレートするために、0（zero）で満たされたファイルを作成します。
2. **mkfs** を使って、新しいパーティションまたはループバックファイル上のファイルシステムをフォーマットします。これを、ブロックサイズを変えながら 3 回行います。スーパーブロックの位置、ブロックグループの数、その他の必要な情報をメモしておきます。
3. 新たにディレクトリ (/mnt/tempdir）を作成し、新しいファイルシステムをマウントします。結果を確認します。
4. マウントしたファイルシステムをアンマウントし、読み取り専用でマウントします。
5. マウントしたディレクトリにファイルを作成します。エラーになります。なぜでしょうか。
6. ファイルシステムをアンマウントします。
7. ブート時にマウントされるように、/etc/fstab ファイルに行を追加します。
8. ファイルシステムをマウントします。
9. ファイルシステム上のバイナリ ファイルを実行不可にするようにファイルシステムを属性を変更します。(/mnt/tempdir エントリの noexec をデフォルトに変更します)。ファイルシステムを再マウントし、実行ファイル（例えば /bin/ls ）を /mnt/tempdir にコピーします。実行してみます。エラーになります。なぜでしょうか。

演習修了後クリーンアップするためには、/etc/fstab から当該エントリを削除してください。

解 18.2

物理的なパーティションによる方法

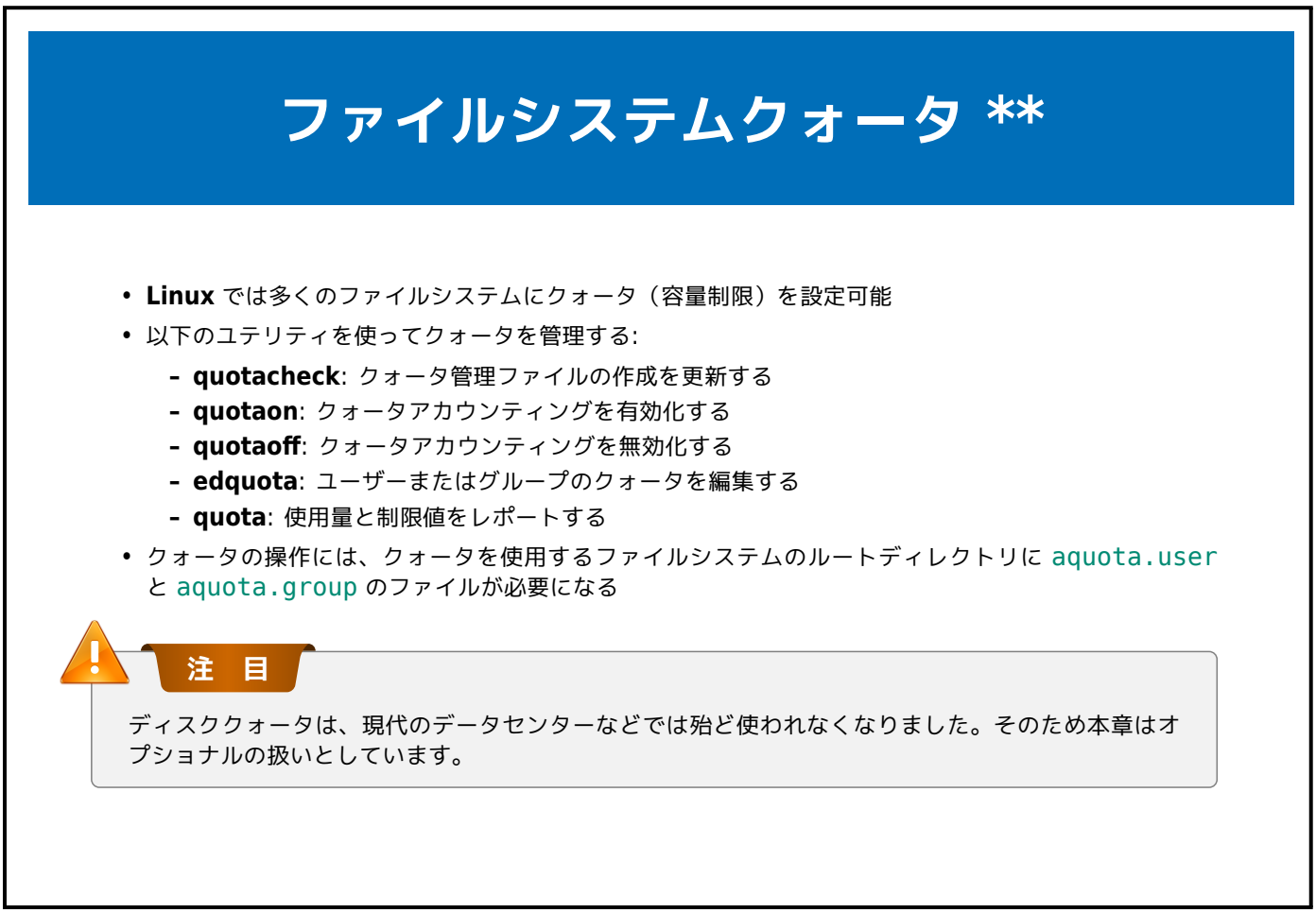
1. 以前に説明したので **fdisk** の詳細は省略します。パーティションを /dev/sda11 で作成したものとします。

   ```
   $ sudo fdisk /dev/sda
   ```

   ```
   .....
   w
   $ partprobe -s
   ```

   partprobe がうまく実行できない場合があります。システムに間違いなく新パーティションを認識させるため、リブートしたほうが良いでしょう。

2. ```
   $ sudo mkfs -t ext4         -v /dev/sda11
   $ sudo mkfs -t ext4 -b 2048 -v /dev/sda11
   $ sudo mkfs -t ext4 -b 4096 -v /dev/sda11
   ```
   ```

-v フラグ（verbose）で情報を表示します。今回のような小さいパーティションでは、ブロックサイズの標準は 1024 バイトです。

3.
```
$ sudo mkdir /mnt/tempdir
$ sudo mount /dev/sda11 /mnt/tempdir
$ mount | grep tempdir
```

4.
```
$ sudo umount /mnt/tempdir
$ sudo mount -o ro /dev/sda11 /mnt/tempdir
```

アンマウント時にエラーが発生したときは、当該ディレクトリにいないことを確認してください。

5.
```
$ sudo touch /mnt/tempdir/afile
```

6.
```
$ sudo umount /mnt/tempdir
```

7. /etc/fstab にこの行を追記してください：

```
/dev/sda11 /mnt/tempdir ext4 defaults 1 2
```

8.
```
$ sudo mount /mnt/tempdir
$ sudo mount | grep tempdir
```

9. /etc/fstab の行を次のように変更してください：

```
/dev/sda11 /mnt/tempdir ext4 noexec 1 2
```

そして次を実行します：

```
$ sudo mount -o remount /mnt/tempdir
$ sudo cp /bin/ls /mnt/tempdir
$ /mnt/tempdir/ls
```

エラーになります。なぜでしょうか。

ループバックファイルによる方法

1.
```
$ sudo dd if=/dev/zero of=/imagefile bs=1M count=250
```

2.
```
$ sudo mkfs -t ext4         -v
$ sudo mkfs -t ext4 -b 2048 -v /imagefile
$ sudo mkfs -t ext4 -b 4096 -v /imagefile
```

パーティションではなくファイルであるという警告がでますが、そのまま進んでください。

-v フラグ（verbose）で情報を表示します。今回のような小さいパーティションでは、ブロックサイズの標準は 1024 バイトです。

3.
```
$ sudo mkdir /mnt/tempdir
$ sudo mount -o loop /imagefile /mnt/tempdir
$ mount | grep tempdir
```

4.
```
$ sudo umount /mnt/tempdir
$ sudo mount -o ro,loop /imagefile /mnt/tempdir
```

アンマウント時にエラーが発生したときは、当該ディレクトリにいないことを確認してください。

5.
```
$ sudo touch /mnt/tempdir/afile
```

6.
```
$ sudo umount /mnt/tempdir
```

7. /etc/fstab にこの行を追記してください:

/etc/fstab の追加内容

```
/imagefile /mnt/tempdir ext4 loop 1 2
```

8.
```
$ sudo mount /mnt/tempdir
$ sudo mount | grep tempdir
```

9. /etc/fstab の行を次のように変更してください:

/etc/fstab の変更内容

```
/imagefile /mnt/tempdir ext4 loop,noexec 1 2
```

次を実行してください:

```
$ sudo mount -o remount /mnt/tempdir
$ sudo cp /bin/ls /mnt/tempdir
$ /mnt/tempdir/ls
```

エラーになります。なぜでしょうか。

章 19

ファイルシステム機能：スワップ、クォータ、使用量

注目

** これらのセクションの一部または全体をオプション扱いとする場合があります。これらには補足資料、専門トピック、または高度な話題が含まれインストラクターが教室の状況や時間制約に応じてこれらの内容を紹介するかを判断します。

19.1 ファイルシステムの使用量

df: ファイルシステムの使用量

- **df** (**d**isk **f**ree) は、ファイルシステムの使用量の確認に利用する
- ファイルシステムの使用量を表示する（デフォルトではキロバイト単位）
  `$ df`
  ファイルシステムの使用量を **人が読みやすい単位** で表示する:
  `$ df -h`
- ファイルシステムのタイプを表示する:
  `$ df -T`
- inode 情報を表示する:
  `$ df -i`

```
x8:/tmp>df -hT
Filesystem     Type      Size  Used Avail Use% Mounted on
devtmpfs       devtmpfs  7.8G     0  7.8G   0% /dev
tmpfs          tmpfs     7.8G     0  7.8G   0% /dev/shm
tmpfs          tmpfs     7.8G  9.6M  7.8G   1% /run
tmpfs          tmpfs     7.8G     0  7.8G   0% /sys/fs/cgroup
/dev/sda8      ext4       26G   10G   15G  42% /
/dev/sda7      vfat      200M   14M  187M   7% /boot/efi
/dev/sda5      ext4       15G  9.6G  4.1G  71% /UBUNTU
/dev/sda6      ext4      389G  252G  118G  69% /ALL
/dev/sda1      vfat      256M   43M  214M  17% /Cbootefi
tmpfs          tmpfs     1.6G  1.2M  1.6G   1% /run/user/42
tmpfs          tmpfs     1.6G  4.0K  1.6G   1% /run/user/1000
/dev/loop0     squashfs  2.5G  2.5G     0 100% /usr/src/KERNELS
x8:/tmp>
```

図 19.1: **df の使用例**

19.2 ディスクの使用量

du: ディスクの使用量

- du (**d**isk **u**sage) は、ディスクの使用量と用途を表示する
- カレントディレクトリの使用量を表示する:
  ```
  $ du
  ```
- ディレクトリーだけではなく全てのファイルをリストする:
  ```
  $ du -a
  ```
- 人が読みやすい単位で表示する:
  ```
  $ du -h
  ```
- ディレクトリを指定してディスク使用量を表示する:
  ```
  $ du -h somedir
  ```

以下のコマンドを試してください:

```
$ find . -maxdepth 1 -type d -exec du -shx {} \; | sort -hr
```

```
c8:/home/coop/.cache>find . -maxdepth 1 -type d -exec du -shx {} \; | sort -hr
338M    .
229M    ./google-chrome
86M     ./spotify
7.7M    ./tracker
6.8M    ./thunderbird
4.9M    ./gnome-software
2.5M    ./mesa_shader_cache
956K    ./gstreamer-1.0
436K    ./samba
52K     ./evolution
12K     ./fontconfig
8.0K    ./vmware
8.0K    ./Slack
8.0K    ./skypeforlinux
4.0K    ./libgweather
4.0K    ./googleearth_CACHE
4.0K    ./gnome-shell
4.0K    ./gnome-screenshot
c8:/home/coop/.cache>
```

図 19.2: **du の使用例**

19.3 スワップ

スワップの使用

- ディスク上に置かれた、仮想的なメモリー領域である
- **fdisk** でスワップパーティションを作成する
- **mkswap**: スワップパーティション、またはファイルをフォーマット
- **swapon**: スワップパーティション、またはファイルを有効化する
- **swapoff**: スワップパーティション、またはファイルを無効化する
- (`/etc/fstab` を使って) 起動時にマウント可能である
- １つ以上のスワップパーティションを設定できる

```
$ cat /proc/swaps

Filename                Type            Size    Used    Priority
/dev/sda6               partition       8290300 0       -1
/tmp/swpfile            file            102396  0       -2
```

Linux は **仮想メモリー**システムを採用しており、オペレーティングシステムは、実際よりも多くのメモリーがあるかのように機能します。この種の **メモリーのオーバーコミッション（＝投機的な割り当て）** は２つの方法で実現されています:

- 多くのプログラムは、使用すると宣言したメモリーを実際に全て使用するわけではありません。それは、子プロセスが **COW** (**C**opy **O**n **W**rite) 手法を使用して、親のメモリー領域のコピーを継承するためです。子プロセスは、変更があったメモリー領域のみ（ページごとに）固有のコピーを作成します。
- メモリーの負荷が重くなってくると、アクティブでないメモリー領域はディスクに **スワップアウト** され、再び必要になったときにのみ、読み戻されます。

このようなスワッピングは、通常１つ以上の専用パーティション又はファイルに対して行われます。**Linux** では複数の **スワップ領域** が認められているため、スワップの要求を動的に調整できます。各スワップ領域には優先順位があり、優先順位の高い領域がいっぱいになるまで優先順位の低い領域は使用されません。

多くの場合、推奨されるスワップサイズはシステムの **RAM** サイズです。`cat /proc/swaps` で現在システムが何をスワップに使用しているかを確認することが可能で、`free` でシステムが現在使用しているスワップの量を確認できます。

いつでも大部分のメモリーは、必要以上のディスクへアクセスや、最善とは言えない順序やタイミングでのディスクアクセスを防ぐための、ファイルコンテンツのキャッシュ用途に使用されています。そのようなメモリーページのデータは、一時的なキャッシュデータが書かれているだけなので（＝バックアップ記憶なので)、スワップに書き出す意味は無く、決してスワップアウトされません。代わりに **ダーティページ** (メモリー上のデータが元々のディスク上の値から更新されている) は、ディスクに書き戻されます。

Linux では、アプリケーションメモリーとは対照的に、カーネル自体が使用するメモリーは **決してスワップアウトされない** ことにも触れておくべきでしょう。この挙動は他のオペレーティングシステムとの違いです。

19.4 ファイルシステムクォータ **



ディスククォータの機能により、管理者は特定のユーザー（またはグループ）に許可する最大ストレージサイズを制御できます。かなり柔軟性があり、クォータはファイルシステムごとに割り当てることもできます。共有リソースを使い果たす一部のユーザーに対するプロテクションが働きます。

クォータは、ファイルシステムごとに有効または無効にできます。さらに **Linux** はユーザー ID とグループ ID に基づくクォータの使用をサポートしています。

ファイルシステムによっては、**xfs_quota** などの追加のクォータ関連ユーティリティがある場合があります。

クォータの設定

- ユーザー、グループまたはその両方のクォータオプションでファイルシステムをマウントする:
  - usrquota、grpquota またはその両方のオプションを /etc/fstab のファイルシステムエントリに追加する
  - ファイルシステムを再マウント（新しい場合は、マウント）する
- ファイルシステムで **quotacheck** を実行して、クォータを設定する
- ファイルシステムでクォータを有効化する
- **edquota** プログラムでクォータを設定する

ファイルシステムのクォータ作成には、まずユーザー、グループまたはその両方のクォータマウントオプションでファイルシステムをマウントする必要があります。これらができていなければ、何も機能しません。

例えば、クォータを利用するには、まず /etc/fstab に適切なオプションを設定します:

```
/dev/sda5 /home ext4 defaults,usrquota 1 2
```

ここでは、/home は専用パーティション上にあると想定しています。

そして、以下のコマンドをテストします:

```
$ sudo mount -o remount /home
$ sudo quotacheck -vu /home
$ sudo quotaon -vu /home
$ sudo edquota someusername
```

edquota を使った **grace periods** の設定も、必要かもしれません。

quotacheck

- **quotacheck** は、ファイルシステムのアカウンティングファイル (aquota.user と aquota.group) を作成し更新する
- /etc/fstab に定義された全てのファイルシステムのユーザーファイルの更新は、ユーザークォータオプションで:

```
$ sudo quotacheck -ua
```

- /etc/fstab に定義された全ファイルシステムのグループファイルの更新は、グループクォータオプションで:

```
$ sudo quotacheck -ga
```

- 特定のファイルシステムのユーザーファイルを更新するには:

```
$ sudo quotacheck -u [somefilesystem]
```

- 特定のファイルシステムのグループファイルを更新するには:

```
$ sudo quotacheck -g [somefilesystem]
```

  -v オプションを使うと、更に詳細な出力が得られます

quotacheck は、通常クォータが最初にオン（有効）になった（または更新する必要がある）場合にのみ実行されます。
このプログラムは、システムの起動時に **fsck** がファイルシステムのエラーを報告したときにも実行されることがあります。

クォータのオン（有効化）とオフ（無効化）

- ファイルシステムクォータのオン／オフは **quotaon** と **quotaoff**

```
$ sudo qoutaon --help
quotaon: Usage:
        quotaon [-guvp] [-F quotaformat] [-x state] -a
        quotaon [-guvp] [-F quotaformat] [-x state] filesys ...
-a, --all                   すべてのファイルシステムのクォータをオンにします
-f, --off                   クォータをオフにします
-u, --user                  ユーザー クォータを操作します
-g, --group                 グループ クォータを操作します
-p, --print-state           クォータがオンかオフかを表示します
-x, --xfs-command=cmd       XFS クォータ コマンドを実行します
-F, --format=formatname     特定のクォータ フォーマットで動作します
-v, --verbose               より詳細なメッセージを出力します
-h, --help                  ヘルプテキストを表示して終了します
-V, --version               バージョン情報を表示して終了します
```

- **quotaon** と **quotaoff** は、実際にはまったく同じプログラムで、呼び出される名前によって動作が変わることに注意してください。

例えば:

```
$ sudo quotaon -av
/dev/sda6 [/]: group quotas turned on
/dev/sda5 [/home]: user quotas turned on
```

```
$ sudo quotaoff -av
/dev/sda6 [/]: group quotas turned off
/dev/sda5 [/home]: user quotas turned off
```

```
$ sudo quotaon -avu
dev/sda5 [/home]: user quotas turned on
```

```
$ sudo quotaoff -avg
 /dev/sda6 [/]: group quotas turned off
```

ファイル `aquota.user` または `aquota.group` が存在しない場合、クォータ操作は失敗します。

クォータの調査

- **quota** ユーティリティはクォータに関する情報を表示する:
  - quota (または quota -u) 現在のユーザークォータの情報を表示
  - quota -g 現在のグループクォータの情報を表示する
  - スーパーユーザーはユーザー名またはグループ名を指定して、任意のユーザーまたはグループの quota を調べることができる

```
$ sudo quota george
```

```
Disk quotas for user george (uid 1000):
 Filesystem blocks quota limit grace files quota limit grace
 /dev/sda5  837572   500  1000        5804   0       0
```

```
$ sudo quota gracie
```

```
Disk quotas for user gracie (uid 1001):
 Filesystem blocks quota limit grace files quota limit grace
  /dev/sda5  83757  5000 10000        5804     0     0
```

クォータの設定

- username のユーザーの制限を編集する:

```
$ sudo edquota -u [username]
```

- groupname のグループの制限を編集する:

```
$ sudo edquota -g [groupname]
```

- userproto のユーザークォータ値を、指定された username のクォータ値にコピーする:

```
$ sudo edquota -u -p [userproto] [username]
```

- groupproto のグループクォータ値を、指定された groupname のクォータ値にコピーする:

```
$ sudo edquota -g -p [groupproto] [groupname]
```

  - 最後の 2 つのコマンドは、新しいアカウント作成時にそのアカウントのクォータを設定するスクリプトに利用すると便利である

- 猶予期間の設定:

```
$ sudo edquota -t
```

edquota と入力すると、クォータエディタが表示されます。指定されたユーザーまたはグループに対して、そのユーザーまたはグループの現在のディスククォータの一時ファイルがテキスト形式で作成されます。

次に、エディタがそのファイルを開き、その中でクォータを変更します。エディタを終了すると、一時ファイルが読み取られバイナリクォータファイルに変更が反映されます。

quota で編集できるフィールドは、ソフトとハードの制限のみです。他のフィールドは情報提供のみです。

ユーザーとグループのクォータは、ディスクブロック、inode またはその両方に設定できます。さらに、猶予期間と同様にソフト制限とハード制限を設定できます。猶予期間中にソフト制限を超えることがあります。しかし、ハード制限を超えることはできません。

猶予期間はファイルシステムごとに設定されます。

```
$ sudo edquota gracie
$ sudo edquota -t
```

19.5 演習

デモ教材ビデオ

using_swap_and_oom_demo.mp4

課題 19.1: スワップ領域の管理

現在のスワップ領域を確認します:

```
$ cat /proc/swaps
```

```
Filename                                Type            Size    Used    Priority
/dev/sda11                              partition       4193776 0       -1
```

新パーティションかファイルをスワップ領域として追加します。ファイルの場合:

```
$ dd if=/dev/zero of=swpfile bs=1M count=1024
```

```
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 1.30576 s, 822 MB/s
```

```
$ mkswap swpfile
```

```
Setting up swapspace version 1, size = 1048572 KiB
no label, UUID=85bb62e5-84b0-4fdd-848b-4f8a289f0c4c
```

(実パーティションの場合 **mkswap** のオプションとしてパーティション名を与えます。そのパーティションの内容は消去されます !)

新スワップ領域をアクティブにします:

```
$ sudo swapon swpfile
```

```
swapon: /tmp/swpfile: insecure permissions 0664, 0600 suggested.
swapon: /tmp/swpfile: insecure file owner 500, 0 (root) suggested.
```

RedHat、Centos、Fedora では

RHEL はセキュアでないと警告を出します。次のように対応します:

```
$ sudo chown root:root swpfile
$ sudo chmod 600 swpfile
```

`swpfile` が使用されていることを確認します:

```
$ cat /proc/swaps
```

```
Filename                                Type            Size    Used    Priority
/dev/sda11                              partition       4193776 0       -1
/tmp/swpfile                            file            1048572 0       -2
```

`Priority` に注目してください; 高プライオリティのスワップ領域が満杯になるまで、低プライオリティのスワップパーティションもしくはファイルは使用されません。

領域を確保するために、スワップファイルを無効にして削除します:

```
$ sudo swapoff swpfile
$ sudo rm swpfile
```

課題 19.2: ファイルシステム　クォータ

注目

本サブセクションはオプションです。したがって、本内容をカバーしなくても演習には問題ありません。

1. /etc/fstab 中の新パーティション用のエントリをクォータを利用するように変更します（/mnt/tempdir 用のエントリの noexec を usrquota に変更します）。ファイルシステムを再マウントします。
2. 新ファイルシステムのクォータを初期化します。クォータチェックをオンにします。
3. 一般のユーザーのクォータリミットを設定します:　ソフトリミットを 500 ブロック、ハード リミットを 1000 ブロックとします。
4. 一般のユーザーとして **dd** を使って、クォータリミットを越えるファイルを作成します。bigfile1（200 blocks）と bigfile2（400 blocks）を作成します。

   警告が出ます。なぜでしょうか。
5. bigfile3（600 blocks）を作成します。

   エラーが出ます。なぜでしょうか。ファイルサイズを注意深く見てください。
6. /etc/fstab に追加したマウント行を削除してください。

解 19.2

1. 実パーティションかループバックファイルに応じて、以下の 2 行のいずれかを /etc/fstab に追加してください:

/etc/fstab の追加内容

```
/dev/sda11     /mnt/tempdir ext4 usrquota      1 2
/imagefile     /mnt/tempdir ext4 loop,usrquota 1 2
```

   そして再マウントします:

   ```
   $ sudo mount -o remount /mnt/tempdir
   ```

2. ```
   $ sudo quotacheck -u /mnt/tempdir
   $ sudo quotaon -u /mnt/tempdir
   $ sudo chown student.student /mnt/tempdir
   ```

   （通常は、上記を実行する必要はありません。しかし、次の実行を楽にするために実行します）。
3. student ユーザーアカウントに自分の名前を入れてください。
4. ```
   $ sudo edquota -u student
   ```
5. ```
   $ cd /mnt/tempdir
   $ dd if=/dev/zero of=bigfile1 bs=1024 count=200
   ```

```
200+0 records in
200+0 records out
204800 bytes (205 kB) copied, 0.000349604 s, 586 MB/s
```

   ```
   $ quota
   ```
   ```

```
Disk quotas for user student (uid 500):
Filesystem blocks quota lim grace files qu lim gr
/dev/sda11    200   500  1000   1   0   0
```

```
$ dd if=/dev/zero of=bigfile2 bs=1024 count=400
```

```
sda11: warning, user block quota exceeded.
400+0 records in
400+0 records out
4096600 bytes (410 kB) copied, 0.000654847 s, 625 MB/s
```

（600 ブロックの）bigfile3 を作成します。

6. $ quota

```
Disk quotas for user student (uid 500):
Filesystem blocks quota limit grace files qu lim gr
/dev/sda11    600*   500  1000 6days   2  0  0
```

```
$ dd if=/dev/zero of=bigfile3 bs=1024 count=600
```

```
sda11: write failed, user block limit reached.
dd: writing `bigfile3': Disk quota exceeded
401+0 records in
400+0 records out
409600 bytes (410 kB) copied, 0.00177744 s, 230 MB/s
```

```
$ quota
```

```
Disk quotas for user student (uid 500):
Filesystem blocks  quota limit grace files quota limit grace
/dev/sda11   1000*   500  1000 6days     3     0     0
```

```
$ ls -l
```

```
total 1068
-rw------- 1 root    root      7168 Dec 10 18:56 aquota.user
-rw-rw-r-- 1 student student 204800 Dec 10 18:58 bigfile1
-rw-rw-r-- 1 student student 409600 Dec 10 18:58 bigfile2
-rw-rw-r-- 1 student student 409600 Dec 10 19:01 bigfile3
drwx------ 2 root    root     16384 Dec 10 18:47 lost+found
-rwxr-xr-x 1 root    root     41216 Dec 10 18:52 more
```

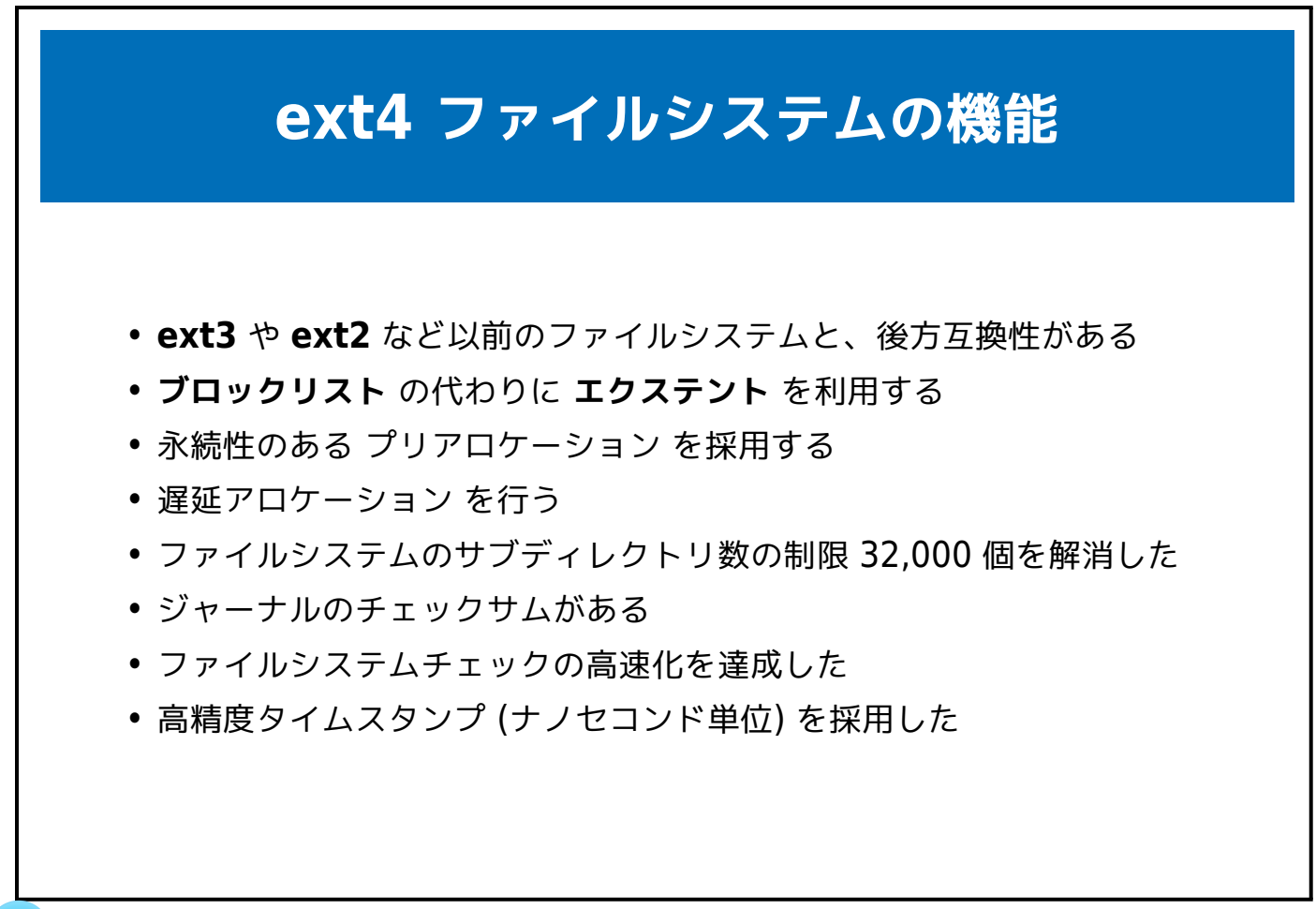
ファイルサイズに注目してください。

7. /etc/fstab の当該行を削除してください。

章 20

Ext4 ファイルシステム

20.1 ext4 の機能



エクステント (Extents)

エクステントは、連続したブロックのグループです。エクステントの利用により、大きなファイルを扱う時の性能が向上しフラグメンテーションが抑制されます

The **ext4** は 1EB までのボリュームと 16TB までのファイルをサポートします。これまでのブロックマッピングをエクステントに置き換えています。

ext4 は **ext3** と **ext2** との後方互換性を持っています。ファイルに対して領域を事前に確保することができます。確保された領域は、通常は連続領域となることが保証されます。また、**allocate-on-flush** と呼ばれる性能向上のテクニック (実際のディスクに書くタイミングまで、ブロックアロケーションを保留する）も使われています。**ext4** では **ext3** にあったサブディレクトリ数の上限 32,000 個の制約も解消されました。

ext4 はジャーナルデータにチェックサムを使っているので、信頼性も向上しています。これにより、ジャーナル期間中のディスク I/O 待ちが安全に回避できるので、性能も若干改善しています。

この他、タイムスタンプも強化されています。**ext4** はナノセコンド単位のタイムスタンプを記録します。

20.2 ext4 のレイアウト、スーパーブロック、ブロックグループ

ext4 のスーパーブロックとブロックグループ

- 先頭の **スーパーブロック** にファイルシステムの全ての情報が含まれる
- 以降に、**ブロックグループ** とよばれる連続ブロックのセットがある
  - 管理情報が書かれている
  - 複数のブロックグループには、冗長性の為のスーパーブロックのコピーが記録される
  - その他のブロックには、ファイルのデータが書かれる
- ブロックサイズは、ファイルシステム作成時に決まる
  - 512, 1K, 2K, 4K, 8K バイトなど、但しメモリーの**ページサイズ** (**x86** では 4k バイト) より大きくは出来ない

ext4 ファイルシステムは、ブロックグループのセットに分割されます。ブロックアロケータは、シーク時間を減らすためにひとつのファイルのブロックを同じブロックグループ内に配置しようとします。デフォルトのブロックサイズ 4 KB の場合、ブロックグループのサイズは 128MB になります。

ext4 ではジャーナルを除くすべてのフィールドは **リトルエンディアン** でディスクに書き込まれます。

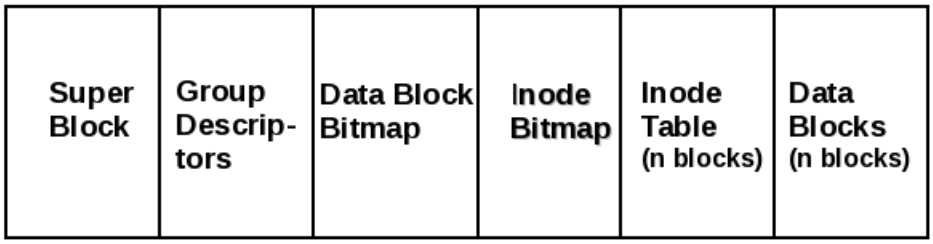


図 20.1: **ext[3,4] ファイルシステムレイアウト**

標準的なブロックグループのレイアウトは単純です。先頭の 1024 バイト（ブロック 0）は使いません。（ここはブートセクターなどの用途のために明けておく）スーパーブロックは、ブロックグループ 0 を除く先頭（つまりブロック 1）から始まります。その後ろには、グループディスクリプタといくつかの **GDT** ブロックが続きます。その後ろにデータブロックビットマップ、inode ビットマップ、inode テーブルそしてデータブロックが格納されます。

ブロックグループ

- スーパーブロック と グループディスクリプタコピー は、多重化されていてファイルシステムのリカバリー時にはコピーが利用されます

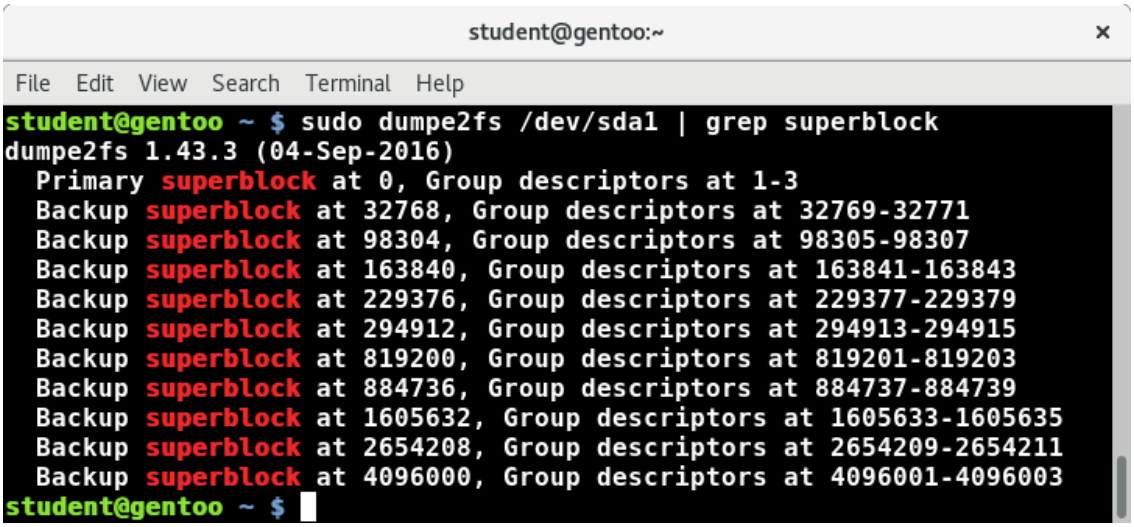


図 20.2: **ブロックグループ**

全ブロック共通で、一番目と二番目のブロックには **スーパーブロック** と **グループディスクリプタ** が書かれています。

カーネルは、通常は最初のブロックグループを参照します。多重化されたコピーは、ファイルシステムチェック時だけ参照されます。何も問題が見つからなければ、カーネルは単純に最初のブロックグループの内容を多重化コピーに上書きします。

マスターコピーに問題が見つかった場合には、健全なデータをつかってファイルシステム構造がリビルドできるまで、順に次のデータを参照していきます。この冗長性により、ファイルシステムチェックが定期的に実行されている限り、ext2/ext3/ext4 ファイルシステムが完全に壊されることはありません。

ext ファイルシステムファミリーの初期には、全てのブロックグループにすべてのブロックグループのグループディスクリプタとスーパーブロックのコピーが含まれていました。しかし現在では、最適化のためすべてのブロックグループにスーパーブロックとグループディスクリプタのコピーがあるわけではありません。自分の環境がどうなっているかを確認するには、添付のスクリーンショットのコマンドを（適切なデバイスノードを指定して）利用して、正確な配置を確認することができます。スーパーブロックの複製の配置は、ファイルシステム作成時に `sparse_super` オプションで設定されます。

スーパーブロックの詳細

- スーパーブロックには、ファイルシステムのグローバル情報が書かれる
  - マウント数と最大マウント数
  - このファイルシステムのブロックサイズ
  - グループ毎のブロック数
  - 未使用のブロック数
  - 未使用の inode 数
  - オペレーティングシステム ID
- スーパーブロックは、冗長性のため複数のブロックグループに保存

マウント数はディスクが正常にマウントされるたびに増えます。最大マウント回数、もしくは 180 日のどちらか先に発生したタイミングで、ファイルシステムが検証されます。

ブロックサイズは **mkfs** コマンドによって設定されます。

ファイルシステムのスーパーブロックは、ディスクのブロック 0 に保存されています。スーパーブロックには、ファイルシステム自体に関する情報が格納されています。

20.3 dumpe2fs

ext4 のブロックと inode 確認: dumpe2fs

- ブロックサイズには以下の各項目の最大値が設定されてきた:
  - ブロック
  - inode
  - スーパーブロック
- **dumpe2fs** でファイルシステム情報を見る:
  - 制限
  - 機能とフラグ
  - その他の属性

dumpe2fs を使うと、指定したパーティションの情報を取得できます。

```
$ sudo dumpe2fs /dev/sdb1

dumpe2fs 1.45.6 (20-Mar-2020)
Filesystem volume name:   VMS
Last mounted on:          /VMS
Filesystem UUID:          fce521c7-e2ce-414a-8a7e-e2311640802f
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal ext_attr resize_inode dir_index filetype needs_recovery extent 64bit flex_bg \
sparse_super large_file huge_file uninit_bg dir_nlink extra_isize
Filesystem flags:         signed_directory_hash
Default mount options:    user_xattr acl
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linuxxf .
Inode count:              14352384
Block count:              57388288
Reserved block count:     2869413
Free blocks:              22270800
Free inodes:              14352217
First block:              0
Block size:               4096

  Block bitmap at 1056 (bg #0 + 1056)
  Inode bitmap at 1072 (bg #0 + 1072)
  Inode table at 1599-2110 (bg #0 + 1599)
  415 free blocks, 8192 free inodes, 0 directories, 8192 unused inodes
  Free blocks: 33822-33985, 34550-34691, 38803-38911
  Free inodes: 8193-16384
Group 2: (Blocks 65536-98303) csum 0xdde9 [INODE_UNINIT, ITABLE_ZEROED]
  Block bitmap at 1057 (bg #0 + 1057)
  Inode bitmap at 1073 (bg #0 + 1073)
  Inode table at 2111-2622 (bg #0 + 2111)
  0 free blocks, 8192 free inodes, 0 directories, 8192 unused inodes
  Free blocks:
  Free inodes: 16385-24576
....
```

20.4 tune2fs

tune2fs: ファイルシステムのパラメータ変更

tune2fs を使うと、ファイルシステムのパラメータの変更が可能

- ファイルシステムのチェック間隔を決める最大マウント回数を変更する (`max-mount-count`):

  ```
  $ sudo tune2fs -c 25 /dev/sda1
  ```

- チェックインターバルを変更 (`interval-between-checks`):

  ```
  $ sudo tune2fs -i 10 /dev/sda1
  ```

- 変更可能な値を含む、現在のスーパーブロックのコンテンツのリスト:

  ```
  $ sudo tune2fs -l /dev/sda1
  ```

  基本的には **dumpe2fs** からのグローバル情報を表示する

```
$ sudo tune2fs -l /dev/sdb1

tune2fs 1.45.6 (20-Mar-2020)
Filesystem volume name:   VMS
Last mounted on:          /VMS
Filesystem UUID:          fce521c7-e2ce-414a-8a7e-e2311640802f
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal ext_attr resize_inode dir_index filetype needs_recovery extent 64bit flex_bg sparse_super large_file huge_file uninit_bg dir_nlink extra_isize
Filesystem flags:         signed_directory_hash
Default mount options:    user_xattr acl
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              14352384
Block count:              57388288
Reserved block count:     2869413
Free blocks:              22270800
Free inodes:              14352217
First block:              0
Block size:               4096
Fragment size:            4096
.....
Filesystem created:       Mon Sep 25 14:14:57 2017
Last mount time:          Mon Mar  8 06:05:03 2021
Last write time:          Mon Mar  8 06:05:03 2021
Mount count:              2003
Maximum mount count:      -1
Last checked:             Wed Feb 28 14:24:15 2018
Check interval:           0 (<none>)
Lifetime writes:          14 TB
....
```

20.5 演習

課題 20.1: デフラグ（Defragmentation）

Linux の初心者の中には、**Windows** で良く使われているファイルシステムの **デフラグツール** についての説明が欠けていることに驚く人がいます。

しかし、**Linux** を含む **UNIX** タイプのオペレーティングシステムでは、ファイルシステムのフラグメンテーションが原因で重大な問題が発生することはありません。

アクセススピードが速いディスクの内周にファイルを詰め込むことはしていないのです。そのかわり、ディスク全体を使用しているのです。そのため、ファイルを作成するときファイルを格納するために必要な領域を簡単に見つけることができます。

最新のハードウェアでは、ディスクの内周かどうかはハードウェアが隠蔽してしまいます。

これは、止めてください

デフラグは 読み/消去/書き込み を繰り返し実行するので、**SSD** の寿命を短くします。最新のオペレーティングシステムでは、**SSD** については、標準でデフラグが使われないようになっています。

さらに、新しい **journalling** ファイルシステム（**ext4** を含む）は、設計上 **extents** （連続した大きな領域）と呼ばれる領域を持っています。

しかし（ここで説明したような理由から）、**ext4** ファイルシステムをデフラグするツールはありません:

```
$ sudo e4defrag
```

```
Usage   : e4defrag [-v] file...| directory...| device...
        : e4defrag  -c  file...| directory...| device...
```

e2fsprogs パッケージに含まれる **e4defrag** は、最新の **Linux** ディストリビューションには必ず含まれています。

2 つのオプションがあります:

- -v: 詳細な出力を出力します。
- -c: 解析してレポートを出力しますが、他は何もしません。

引数:

- ファイル
- ディレクトリ
- デバイス

例:

```
$ sudo e4defrag -c /var/log
```

```
<Fragmented files>                          now/best       size/ext
1. /var/log/lastlog                           5/1             9 KB
2. /var/log/sa/sa24                           3/1            80 KB
3. /var/log/rhsm/rhsm.log                     2/1           142 KB
4. /var/log/messages                          2/1          4590 KB
5. /var/log/Xorg.1.log.old                    1/1            36 KB

 Total/best extents                          120/112
 Average size per extent                     220 KB
```

```
 Fragmentation score                          1
 [0-30 no problem: 31-55 a little bit fragmented: 56- needs defrag]
 This directory (/var/log) does not need defragmentation.
 Done.
```

```
$ sudo e4defrag  /var/log
```

```
ext4 defragmentation for directory(/var/log)
[2/152]/var/log/Xorg.2.log:     100%    [ OK ]
[3/152]/var/log/Xorg.0.log.old: 100%    [ OK ]
[4/152]/var/log/messages-20141019.gz:   100%    [ OK ]
[5/152]/var/log/boot.log:       100%    [ OK ]
[7/152]/var/log/cups/page_log-20140924.gz:      100%    [ OK ]
[8/152]/var/log/cups/access_log-20141019.gz:    100%    [ OK ]
[9/152]/var/log/cups/access_log:        100%    [ OK ]
[10/152]/var/log/cups/error_log-20141018.gz:    100%    [ OK ]
[11/152]/var/log/cups/error_log-20141019.gz:    100%    [ OK ]
[12/152]/var/log/cups/access_log-20141018.gz:   100%    [ OK ]
[14/152]/var/log/cups/page_log-20141018.gz:     100%    [ OK ]
...
[152/152]/var/log/Xorg.1.log.old:       100%    [ OK ]

        Success:                        [ 112/152 ]
        Failure:                        [ 40/152 ]
```

まず、いろいろなファイル、ディレクトリ、デバイスに対して `-c` を指定して **e4defrag** を実行してみてください。

Linux ファイルシステムで、デフラグが必要になるのは、使用率が 90%以上になったときや、**boot** パーティションのように小さいファイルシステムに比較的大きなファイルを格納しようとしたときだけです。

課題 20.2: tune2fs コマンドで、ファイルシステムのパラメータを変更する

フォーマット済の **ext4** ファイルシステムのプロパティを扱います。これは、ファイルシステムのアンマウントは不要です。

以下のコマンドで作成したイメージファイルを利用します。または、`imagefile` の代わりに `/dev/sdaX`（変更したいファイルシステムがマウントされている）を利用します:

```
$ dd if=/dev/zero of=imagefile bs=1M count=1024
$ mkfs.ext4 imagefile
```

1. **dumpe2fs** を利用して、変更したいファイルシステムのプロパティを得ます。
   表示:
   - 最大マウント回数（回数を超えたら、ファイルシステムチェックを強制実行する）。
   - チェック間隔 （指定間隔を超えたら、ファイルシステムチェック実行）
   - 予約ブロック数とブロック数の合計
2. 変更:
   - 30 を上限として最大マウント回数。
   - チェック間隔 最大 3 週間。
   - 予約ブロックの割合。最大 10%。
3. **dumpe2fs** を使ってファイルシステムの情報を得たら、オリジナルの出力と比較します。

解 20.2

1. `$ dumpe2fs imagefile > dumpe2fs-output-initial`

```
dumpe2fs 1.42.9 (28-Dec-2013)
```

```
$ grep -i  -e "Mount count" -e "Check interval" -e "Block Count" dumpe2fs-output-initial
```

```
Block count:              262144
Reserved block count:     13107
Mount count:              0
Maximum mount count:      -1
Check interval:           0 (<none>)
```

2. `$ tune2fs -c 30 imagefile`

```
tune2fs 1.42.9 (28-Dec-2013)
Setting maximal mount count to 30
```

```
$ tune2fs -i 3w imagefile
```

```
tune2fs 1.42.9 (28-Dec-2013)
Setting interval between checks to 1814400 seconds
```

```
$ tune2fs -m 10 imagefile
```

```
tune2fs 1.42.9 (28-Dec-2013)
Setting reserved blocks percentage to 10% (26214 blocks)
```

3. `$ dumpe2fs imagefile > dumpe2fs-output-final`

```
dumpe2fs 1.42.9 (28-Dec-2013)
```

```
$ grep -i  -e "Mount count" -e "Check interval" -e "Block Count" dumpe2fs-output-final
```

```
Block count:              262144
Reserved block count:     26214
Mount count:              0
Maximum mount count:      30
Check interval:           1814400 (3 weeks)
```

```
$ diff dumpe2fs-output-initial dumpe2fs-output-final
```

```
14c14
< Reserved block count:     13107
---
> Reserved block count:     26214
29c29
< Last write time:          Wed Oct 26 14:26:19 2016
---
> Last write time:          Wed Oct 26 14:26:20 2016
31c31
< Maximum mount count:      -1
---
> Maximum mount count:      30
33c33,34
< Check interval:           0 (<none>)
---
> Check interval:           1814400 (3 weeks)
> Next check after:         Wed Nov 16 13:26:16 2016
```

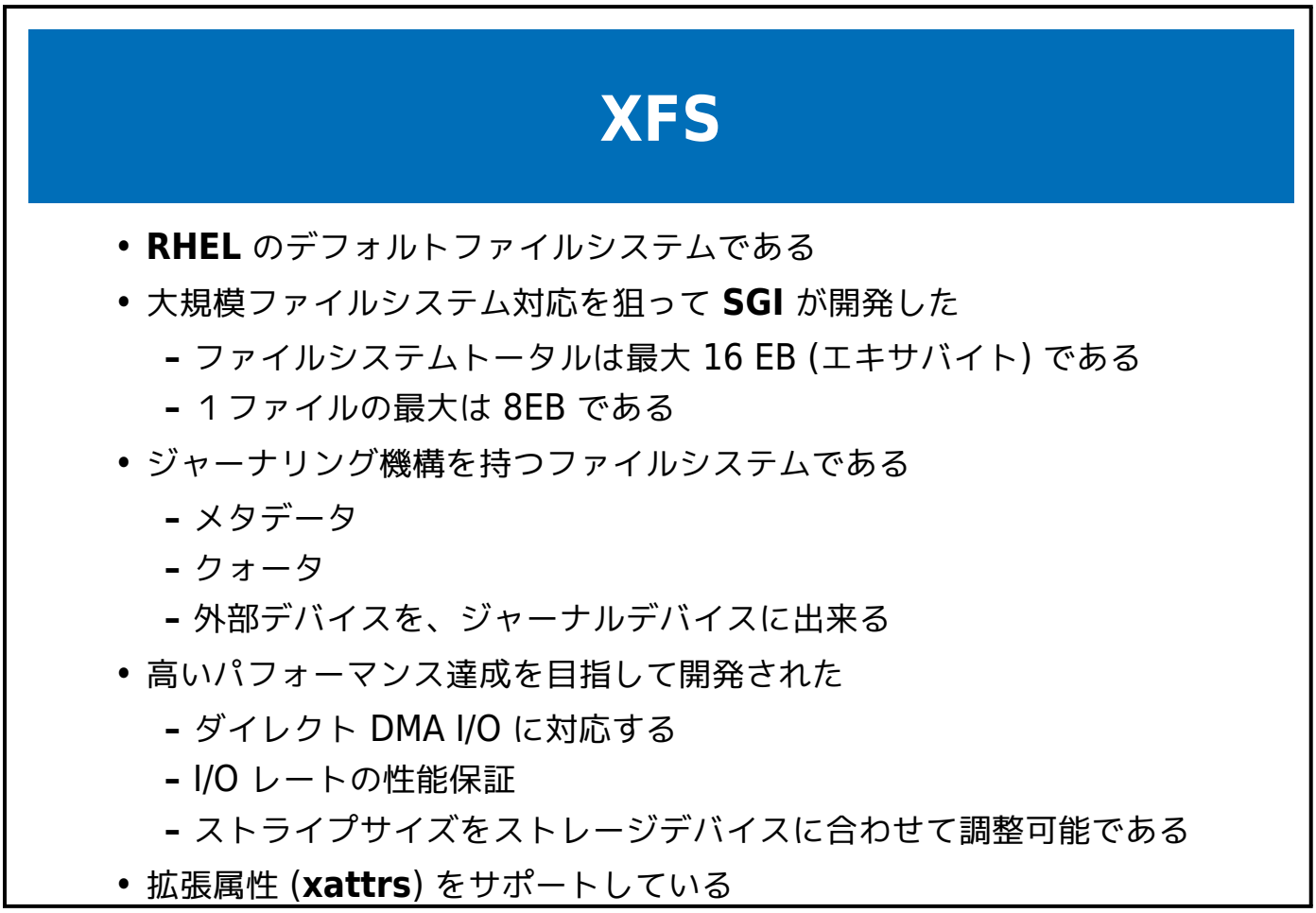
章 21

XFS と btrfs ファイルシステム **

注目

** これらのセクションの一部または全体をオプション扱いとする場合があります。これらには補足資料、専門トピック、または高度な話題が含まれインストラクターが教室の状況や時間制約に応じてこれらの内容を紹介するかを判断します。

21.1 XFS



XFS ファイルシステムは **Silicon Graphics Inc（SGI）**によって設計・開発されました。**SGI** システムの大規模なデータセットを処理し並列 I/O タスクを非常に効率的に処理するように設計されています。

他の **Linux** ファイルシステムとは異なり、**XFS** はクォータ情報をジャーナルに保管することもできます。これにより、クォータが有効なファイルシステムが正しくアンマウントされなかった時のリカバリ時間が短縮されています。

性能の追求は **XFS** 設計時の極めて重要なテーマの 1 つでした。**XFS** は、I/O レートを保証するための手段として **ダイレクト DMA I/O** と、外部ジャーナルデバイスをサポートしています。更にストレージデバイスが RAID や **LVM** 構成の場合、デバイスに合わせてストライプサイズ（= ストライピングされたディスクを読み書きする時のブロックサイズ）を調整する機能があります。

他の **Linux** のファイルシステムと同様に **XFS** は拡張属性をサポートしています。

XFS の詳細

- オンライン状態でも利用可能なメンテナンスツールがある
  - デフラグメント
  - サイズの拡大
  - ダンプ／レストア
- クォータをサポートする
  - 標準のクォータツールの利用が可能
  - デバイス毎のクォータもサポートする
    - **XFS** 専用のクォータツールを使って、クォータを変更する
- ファイルシステム自体には、スナップショット機能は無い
  - **xfs_freeze** コマンド
    - **LVM** ツールを利用している場合（自動的に呼ばれるので）不要
- **xfsdump** と **xfsrestore** を使ってバックアップ／レストアが出来る

XFS ファイルシステムのメンテナンスは、ほとんどのメンテナンス作業がマウント時かオンラインで（つまりファイルシステムがマウントされている状態で）実行できるので簡単です。メンテナンスメニューには、デフラグメント、リサイズ（拡大のみ）、ダンプ／レストア が含まれます。

xfsdump と **xfsrestore** コマンドは、中断、及び後に再開させることができます。またこれらの処理は、マルチスレッド化されているので **XFS** のダンプ／レストアは非常に高速です。

従来のクォータコマンドを使用して **XFS** ボリューム上のファイルシステムごとのクォータの操作ができます。一方で、**xfs_quota** コマンドを使用すれば、**XFS** がサポートするディレクトリごとのクォータを使用できます。

XFS ファイルシステムは、ハードウェアまたはソフトウェアによるスナップショットを直接サポートしていません。しかしながら、**xfs_freeze** ユーティリティを使用してファイルシステムを静止させ、ストレージデバイスを対象にスナップショットツールを動作させることができます。**Linux LVM** ツールは **xfs_freeze** を自動的に使用して（= 呼び出して）スナップショットが取れるようにファイルシステムを停止します。

21.2 btrfs

btrfs

- **B**-**TR**ee **F**ile **S**ystem
- コピーオンライト (**COW**) 手法を多用している
- ファイルシステム全体、またはサブボリュームのスナップショットを取得
- LVM（Logical Volume Management）に組み込まれている
- 最大ファイルサイズ: 16 EB
- ファイル数の最大値: 2^{64}
- **SLES**（SUSE Linux Enterprise Server）と **openSUSE** のデフォルトファイルシステムである

btrfs は RHEL から除かれました

btrfs は **RHEL/CentOS 8** から外れました。アップストリームソースからのインストールは可能ですが、使用するディストリビューションのサポート外なら重要なファイルシステムに使うのは推奨できません。

Linux Linux 開発者と、高性能と大容量もしくは特殊なニーズを持つ **Linux** ユーザーの両方が、Chris Mason が作成した **btrfs** filesytem の開発と段階的なデプロイに注目しています。**btrfs** という名前は **B-tr**ree **f**ile **s**ystem の略です。完全なドキュメントは https://btrfs.wiki.kernel.org/index.php/Main_Page にあります。

btrfs は **ext4** など他の **Linux** ファイルシステムが抱えていた プーリング、スナップショット、チェックサム、および統合マルチデバイススパニング不足といった課題に対処することを目指しています。これらの機能は **Linux** を大規模なエンタープライズストレージ構成で利用するためには重要です。

btrfs はカーネル 2.6.29 でメインラインにマージされましたが、しばらくは実験的なものと見られていました。それでも一部のベンダーが **btrfs** を新製品に使用し、現在ではいくつかのディストリビューションのデフォルトのルートファイルシステムです。

btrfs の主要機能の１つは、ファイルシステム全体またはファイルシステム全体に対して設定した仮想的なサブボリューム （サブボリュームの実態は、ファイルとディレクトリを保持している名前の付いた B-tree です。）の **スナップショット**を、頻繁に取得できることです。**btrfs** は **COW**（Copy on Write）手法を広範囲に使用するため、スナップショット取得時にはメタデータの更新以外はデータブロックの初期領域以外の利用や IO アクティビティは発生しません。

以前のスナップショットに戻すのは簡単で、カーネルに以前のルートファイルシステムのスナップショットを組み合わせて再起動させることもできます。

btrfs は **LVM**（Logical Volume Management）と同様に、既存のファイルシステムに新しいパーティションや物理メディアを追加または削除するための独自の内部フレームワークを持っています。

btrfs を重要なファイルシステムとして日常使いするには、幾つかのタスク（＝機能追加のための開発のこと）を終了させる必要があります。**btrfs** の開発経緯と今後期待されている進化については https://lwn.net/Articles/575841 を参照してください。

21.3 演習

課題 21.1: xfs について、もっと知る

注目

xfs に関する詳細な演習は行いません。多くのシステムにはカーネルモジュールや関連するユーザー用ユーティリティがインストールされていません。しかし **Linux** カーネルとディストリビューションがサポートしていれば、ユーザは `mkfs -t xfs` を使って簡単にファイルシステムを作成できます。

以下のようにして **xfs** に関するユーティリティを見つけることができます:

```
$ man -k xfs
```

```
attr (1)             - extended attributes on XFS filesystem objects
bcc-xfsdist (8)      - Summarize XFS operation latency. Uses Linux eBPF/bcc.
bcc-xfsslower (8)    - Trace slow xfs file operations, with per-event details.
filesystems (5)      - Linux filesystem types: ext, ext2, ext3, ext4, hpfs, i...
fs (5)               - Linux filesystem types: ext, ext2, ext3, ext4, hpfs, i...
fsck.xfs (8)         - do nothing, successfully
fsfreeze (8)         - suspend access to a filesystem (Ext3/4, ReiserFS, JFS,...
mkfs.xfs (8)         - construct an XFS filesystem
xfs (5)              - layout, mount options, and supported file attributes f...
xfs_admin (8)        - change parameters of an XFS filesystem
xfs_bmap (8)         - print block mapping for an XFS file
xfs_copy (8)         - copy the contents of an XFS filesystem
xfs_db (8)           - debug an XFS filesystem
xfs_estimate (8)     - estimate the space that an XFS filesystem will take
xfs_freeze (8)       - suspend access to an XFS filesystem
xfs_fsr (8)          - filesystem reorganizer for XFS
xfs_growfs (8)       - expand an XFS filesystem
xfs_info (8)         - display XFS filesystem geometry information
xfs_io (8)           - debug the I/O path of an XFS filesystem
xfs_logprint (8)     - print the log of an XFS filesystem
xfs_mdrestore (8)    - restores an XFS metadump image to a filesystem image
xfs_metadump (8)     - copy XFS filesystem metadata to a file
xfs_mkfile (8)       - create an XFS file
xfs_ncheck (8)       - generate pathnames from i-numbers for XFS
xfs_quota (8)        - manage use of quota on XFS filesystems
xfs_repair (8)       - repair an XFS filesystem
xfs_rtcp (8)         - XFS realtime copy command
xfs_spaceman (8)     - show free space information about an XFS filesystem
xfsdist (8)          - Summarize XFS operation latency. Uses bpftrace/eBPF.
xfsdump (8)          - XFS filesystem incremental dump utility
xfsinvutil (8)       - xfsdump inventory database checking and pruning utility
xfsrestore (8)       - XFS filesystem incremental restore utility
xqmstats (8)         - Display XFS quota manager statistics from /proc
```

これらのユーティリティの詳細を読んで、作成したファイルシステムに使ってみてください。

課題 21.2: btrfs について、もっと知る

RHEL/CentOS 8 では btrfs は使えません

RHEL 8 など **btrfs** をサポートしていないシステムでは、この演習を行うことができません。

注目

btrfs に関する詳細な演習は行いません。多くのシステムでは、カーネルモジュールや関連するユーザー用ユーティリティがインストールされていません。しかし **Linux** カーネルとディストリビューションがサポートしていれば、ユーザは `mkfs -t btrfs` を使って簡単にファイルシステムを作成できます。

`btrfs` を入力することで、**btrfs** に関するユーティリティを見つけることができます:

```
$ btrfs
```

```
usage: btrfs [--help] [--version] <group> [<group>...] <command> [<args>]

Command groups:
  subvolume          manage subvolumes: create, delete, list, etc
  filesystem         overall filesystem tasks and information
  balance            balance data across devices, or change block groups using filters
  device             manage and query devices in the filesystem
  scrub              verify checksums of data and metadata
  rescue             toolbox for specific rescue operations
  inspect-internal   query various internal information
  property           modify properties of filesystem objects
  quota              manage filesystem quota settings
  qgroup             manage quota groups
  replace            replace a device in the filesystem

Commands:
  check              Check structural integrity of a filesystem (unmounted).
  restore            Try to restore files from a damaged filesystem (unmounted)
  send               Send the subvolume(s) to stdout.
  receive            Receive subvolumes from a stream
  help               Display help information
  version            Display btrfs-progs version
```

以下をタイプすることで、もう少し詳細情報を得られます：

```
$ btrfs --help
```

これらのユーティリティの詳細を読んで、作成したファイルシステムに使ってみてください。

以下のコマンドを入力することで、たくさんの情報が得られます:

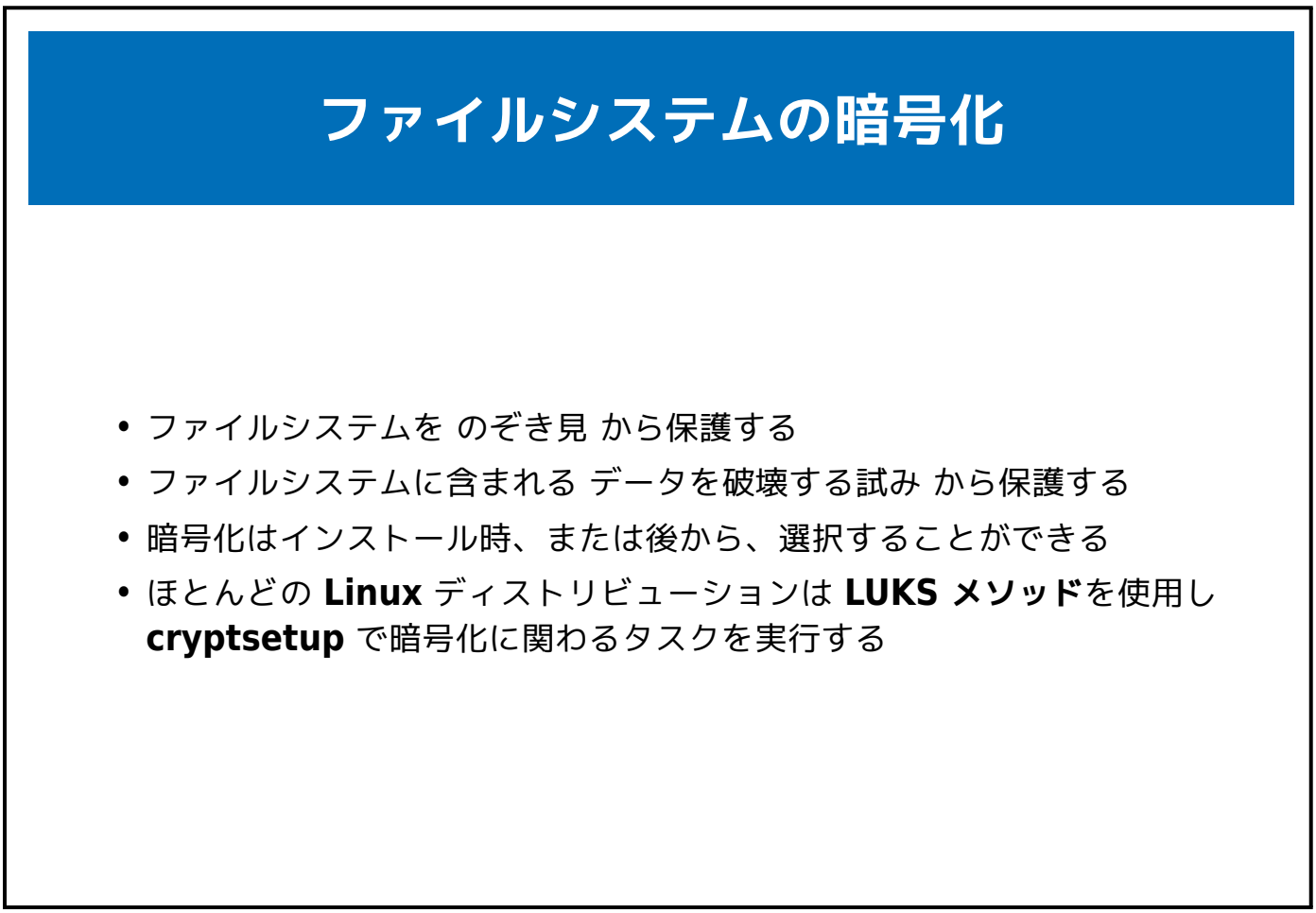
```
$ man -k btrfs
```

```
btrfs-image (8)      - create/restore an image of the filesystem
btrfs-show (8)       - scan the /dev directory for btrfs partitions and print re...
btrfsck (8)          - check a btrfs filesystem
btrfsctl (8)         - control a btrfs filesystem
mkfs.btrfs (8)       - create an btrfs filesystem
btrfs (5)            - topics about the BTRFS filesystem (mount options, support...
btrfs (8)            - a toolbox to manage btrfs filesystems
btrfs-balance (8)    - balance block groups on a btrfs filesystem
....
```

章 22

ディスクの暗号化

22.1 ファイルシステムの暗号化



この章を終了すると次のことができるようになります:

- 暗号化を使用する正当な理由と、暗号化がいつ要求されるかを示すことができます。
- **cryptsetup** を使用して、**LUKS** がどのように動作するかを理解できます。
- 暗号化されたファイルシステムとパーティションの設定と、使用ができるようになります。
- 起動時に暗号化されたパーティションをマウントするように、システムを構成できるようになります。

なぜ暗号化を使用するのか？

- ストレージ上ないしネットワーク経由の転送時に機密データを保護する
- ブロックデバイスレベル暗号化は、データ喪失や侵害に対する最強の保護となる
- **Linux** ディストリビューションのインストール時に、ディスク全体ないし、一部分の暗号化を選択できる
- インストール後に暗号化することも可能だが、既存のパーティションの暗号化にはデータのコピーが求められる

機密データの保存や送信を行う時には暗号化が必要です。ブロックデバイスレベルの暗号化を構成して使用すれば、ハードドライブやその他のメディアに含まれるデータの損失や侵害によって引き起こされる損害に対する最強の保護となります。

最新の **Linux** ディストリビューションでは、インストール中にディスクパーティション全体、または一部の暗号化を選択できます。暗号化パーティションを後で作成してフォーマットすることも簡単にできますが、既存のパーティションの暗号化にはデータのコピーが必要になります。

22.2 LUKS

LUKS

- ブロックデバイスレベルの暗号化では、主に **LUKS** (**L**inux **U**nified **K**ey **S**etup) が使われる
- **LUKS** は **cryptsetup** 上にインストールされる
- **cryptsetup** では以下も利用可能:
  - **plain dm-crypt ボリューム**
  - **loop-AES**
  - **TrueCrypt 互換フォーマット**
- しかし、**LUKS** が **Linux** で使われる最も一般的なメソッドである
- 他のディスクやシステムにパーティションを移行するのは容易である
- **swap** パーティションの透過的な暗号化にも利用される

最新の **Linux** ディストリビューションでは、主に **LUKS** (**L**inux **U**nified **K**ey **S**etup) を使用してブロックデバイスレベルの暗号化を行います。ラップトップ、タブレット、スマートフォン等のポータブルシステムではブロックデバイスの暗号化を強く推奨します。

LUKS は **cryptsetup** 上にインストールされます。**LUKS** は強力なユーティリティです。**cryptsetup** 上では **plain dm-crypt ボリューム**、**loop-AES** 、**TrueCrypt 互換フォーマット** などの他のメソッドも利用可能ですが、ここではこれらの代替ツールは紹介しません。**LUKS**（は元々 **Linux** 向けに開発されたもので、他の OS にも移植されています）が、**Linux** で最もよく使用される標準的な方法だからです。

dm-crypt カーネルモジュールは、次に紹介する **LVM** も頻繁に使用する **デバイスマッパー** というカーネルの仕組みを使用します。

LUKS はパーティションヘッダー内に必要な全情報を保存するので、他のディスクやシステムへのパーティション移行は簡単です。

LUKS はスワップパーティションを透過的に暗号化するのにも使われます。

22.3 cryptsetup

cryptsetup

- ほぼ全ての操作は、多機能の **cryptsetup** ユーティリティで行う
- セットアップとフォーマットが完了した暗号化ボリュームは、通常のディスクユーティリティでマウント／アンマウントできる
- 使い方:
  `cryptsetup [OPTION...] <action> <action-specific>`
- 全機能を参照するには:
  `$ cryptsetup --help`

基本的には、Swiss army knife プログラム（= 多機能ツールの意味）の **cryptsetup** を使います。
暗号化ボリュームは、セットアップが終わっていれば通常のディスクユーティリティでマウントおよびアンマウントが可能です。
コマンドの一般的な書式は次の通りです:

```
cryptsetup [OPTION...] <action> <action-specific>
```

以下のコマンドで全機能をリストも可能です:

```
$ cryptsetup --help
```

```
cryptsetup 2.3.3
Usage: cryptsetup [OPTION...] <action> <action-specific>
  -v, --verbose                     Shows more detailed error messages
      --debug                       Show debug messages
      --debug-json                  Show debug messages including JSON
                                    metadata
  -c, --cipher=STRING               The cipher used to encrypt the disk
                                    (see /proc/crypto)
  -h, --hash=STRING                 The hash used to create the encryption
                                    key from the passphrase
  -y, --verify-passphrase           Verifies the passphrase by asking for
                                    it twice
  -d, --key-file=STRING             Read the key from a file
      --master-key-file=STRING      Read the volume (master) key from file.
      --dump-master-key             Dump volume (master) key instead of
                                    keyslots info
  -s, --key-size=BITS               The size of the encryption key
.....
```

22.4 暗号化されたパーティションの使用

暗号化されたパーティションの使用

- 既に作られているパーティション (/dev/sdc12) を **LUKS** に渡す:

```
$ sudo cryptsetup luksFormat /dev/sdc12
```

- もし使っているカーネルが **cryptsetup** が利用するデフォルトの暗号化方式をサポートしていない場合には /proc/crypto を調べてサポートされている方式を使う:

```
$ sudo cryptsetup luksFormat --cipher aes /dev/sdc12
```

- 暗号化ボリュームとして利用可能にする

```
$ sudo cryptsetup --verbose luksOpen /dev/sdc12 SECRET
```

- パーティションのフォーマット、マウント、利用、アンマウント:

```
$ sudo mkfs.ext4  /dev/mapper/SECRET
$ sudo mount /dev/mapper/SECRET /mnt
....
$ sudo umount /mnt
```

- 次のコマンドで、現在の関連付けを削除:

```
$ sudo cryptsetup --verbose luksClose SECRET
```

パーティション /dev/sdc12 が既に存在する場合は、次のコマンドで、暗号化の設定、**LUKS** で使用可能にして、フォーマットして、マウントして、使用し、そしてアンマウントします。まず **LUKS** でパーティションを初期化します:

```
$ sudo cryptsetup luksFormat /dev/sdc12
```

暗号化されたボリュームを後で使用する時に必要となる パスフレーズ の入力を求められます。暗号化の設定時に、この手順を 1 回だけ行う必要があります。あなたが使っているカーネルが **cryptsetup** が使用するデフォルトの暗号化方式をサポートしていない場合があります。その場合は /proc/crypto を調べてシステムがサポートしている方式を確認します。その方法は:

```
$ sudo cryptsetup luksFormat --cipher aes /dev/sdc12
```

ボリュームをいつでも使用可能にする場合:

```
$ sudo cryptsetup --verbose luksOpen /dev/sdc12 SECRET
```

ここでパスフレーズを入力するように求められます。パーティションをフォーマットしてマウントするには:

```
$ sudo mkfs.ext4  /dev/mapper/SECRET
$ sudo mount /dev/mapper/SECRET /mnt
```

これで暗号化されていないパーティションと全く同じように利用できます。完了したら次のコマンドでアンマウントします。

```
$ sudo umount /mnt
$ sudo cryptsetup --verbose luksClose SECRET
```

22.5 ブート時のマウント

ブート時のマウント

- 以下のファイルに適切なエントリーがあれば、起動時に暗号化されたパーティションをマウントできる:
  - `/etc/fstab`
  - `/etc/crypttab`
- `/etc/crypttab` のエントリー、シンプルである:
  `SECRET /dev/sdc12`
- このエントリーにパスワードを設定したり（**バッドアイディア** ですが)、他の設定を追加することもできる
- `/etc/fstab` へのエントリー、これもシンプルである:
  `/dev/mapper/SECRET /mnt ext4 defaults 0 0`

ブート時に暗号化されたパーティションをマウントするには、２つの条件を満たす必要があります:

1. `/etc/fstab` に適切なエントリを作成する。ここでは暗号化については一切触れる必要はありません。
2. `/etc/crypttab` にエントリを追加する。これも次のように簡単です:

   ```
   SECRET /dev/sdc12
   ```

推奨しません

起動時にプロンプトを表示したくないのでこのファイルパスワードを指定する、などさらに多くの設定が可能です。（これはセキュリティ観点では、逆効果になりますが）何が設定できるかは `man crypttab` を参照してください。

22.6 演習

デモ教材ビデオ

using_encrypted_loopback_demo.mp4

課題 22.1: ディスクの暗号化

本演習では、ハードディスクやラップトップの盗難に備えてパーティションを暗号化する方法について学びます。最初に **cryptsetup** に関するドキュメントをレビューします（man cryptsetupとcryptsetup --help）。

1. **fdisk** を用いて暗号化するブロックデバイスに新しくパーティションを作成します。カーネルが新しいパーティションを認識していることを確認します。リブートしても良いですが、他の方法もあります。
2. **cryptsetup** を使ってパーティションをフォーマットします。暗号化レイヤに **LUKS** を使用します。
3. 暗号化されたブロックデバイス（つまり、secret-disk）を開いて、デバイス用の暗号化されていない pass を作成します。
4. /etc/crypttabに項目を追加し、リブート時にシステムがパスフレーズの入力を待つようにします。
5. **ext4** ファイルシステムを作成します。
6. 新しいファイルシステム用のマウントポイントを作成します。例えば /secret とします。
7. ブート時にマウントできるように /etc/fstab に項目を追加します。
8. 暗号化ファイルシステムをマウントします。
9. リブートして全ての構成が問題無いことを確認します。

解 22.1

1. `$ sudo fdisk /dev/sda`

   新しいパーティションを作成します。ここでは /dev/sda4 とします。その他の場合は適宜読み替えてください:

   `$ sudo partprobe -s`

   本コマンドでシステムに変更されたパーティションテーブルを読み出します。より確実な方法はリブートすることです。

   注意: 実パーティションを利用することができない場合、前の章で説明した loop デバイスを用いるかイメージファイルを使うことができます。

2. `$ sudo cryptsetup luksFormat /dev/sda4`

3. `$ sudo cryptsetup luksOpen /dev/sda4 secret-disk`

4. 以下を/etc/crypttabに追加します:

   /etc/crypttab の追加内容

   ```
   secret-disk    /dev/sda4
   ```

5. `$ sudo mkfs -t ext4 /dev/mapper/secret-disk`

6. `$ sudo mkdir -p /secret`

7. 以下を /etc/fstab に追加します:

/etc/fstab の追加内容

```
/dev/mapper/secret-disk    /secret    ext4    defaults    1 2
```

8. 当該ファイルシステムをマウントします:

   ```
   $ sudo mount /secret
   ```

   または、/etc/fstab に記載されている全てのファイルシステムをマウントします:

   ```
   $ sudo mount -a
   ```

9. リブートします。

課題 22.2: Swap 領域の暗号化

本演習では **swap パーティション** を暗号化します。swap デバイスに書き込まれるデータは非常に重要です。swap は実パーティションに書き込まれるため、swap パーティションを暗号化しないことはセキュリティ上の問題となります。

暗号化の方法は前の課題と同様です。異なる点は、暗号化されたブロックデバイス上にファイルシステムを作成しないことです。

まず、既存の swap デバイスを無効化し、暗号化用にフォーマットします。これは下記に示す、新規にパーティションを作成するよりも安全な方法です。または、前の課題で作成した暗号化パーティションを再利用することもできます。最後に、問題が発生したときの対処方法を説明します。

全てを終えてもとの暗号化されていないパーティションに戻したいときは、そのパーティションが使用中でないときに **mkswap** を使ってもとに戻せます。同時に、/etc/crypttab と /etc/fstab をもとに戻します。

1. 使用中の swap パーティションを見つけて未使用にします。

   ```
   $ cat /proc/swaps
         \end{cmd:
         \begin{response}
   Filename                                Type            Size    Used    Priority
   /dev/sda11                              partition       4193776 0       -1
         \end{response}
         \begin{cmd}
   $ sudo swapoff /dev/sda11
   ```

2. 前章と同様の暗号化手順を実行します:

   ```
   $ sudo cryptsetup luksFormat /dev/sda11  # may use --cipher aes option
   $ sudo cryptsetup luksOpen   /dev/sda11  swapcrypt
   ```

3. swap として使用するために、暗号化デバイスをフォーマットします:

   ```
   $ sudo mkswap /dev/mapper/swapcrypt
   ```

4. 問題ないか、テストする：

   ```
   $ sudo swapon /dev/mapper/swapcrypt
   $ cat /proc/swaps
   ```

5. ブート時に暗号化 swap パーティションを使用可能にするために、以下の 2 つの手順を実行します:

   (a) リブート時にパスフレーズの入力をするために /etc/crypttab に追加する：

   /etc/crypttab への追加内容

   ```
   swapcrypt  /dev/sda11   /dev/urandom  swap,cipher=aes-cbc-essiv:sha256,size=256
   ```

（注意: crypttab の **man** で説明されているように **エントロピー不足** の可能性があるため /dev/random より /dev/urandom を使う方が望ましいのです。）詳細なオプションは必須ではありませんが、他にできることの例示として記載します。

(b) ブート時に swap デバイスを使用するために /etc/fstab に追加します。

/etc/fstab の追加内容

```
/dev/mapper/swapcrypt  none    swap    defaults    0 0
```

6. リブートして、設定が正しいことを確認してください。

もとの状態に戻すためには:

```
$ sudo swapoff /dev/mapper/swapcrypt
$ sudo cryptsetup luksClose swapcrypt
$ sudo mkswap /dev/sda11
$ sudo swapon -a
```

swapon コマンドが失敗する場合は /etc/fstab 内のスワップパーティションの記述が正しくない可能性が高いです。実デバイスノード（/dev/sda11）が書かれている場合には問題ありません。以下のように直してください。

/etc/fstab の修正内容

```
/dev/sda11  swap   swap  defaults 0 0
```

または、フォーマット時にラベルを定義します:

```
$ sudo mkswap -L SWAP /dev/sda11
```

以下を追加してください:

/etc/fstab への追加内容

```
LABEL=SWAP  swap   swap  defaults 0 0
```

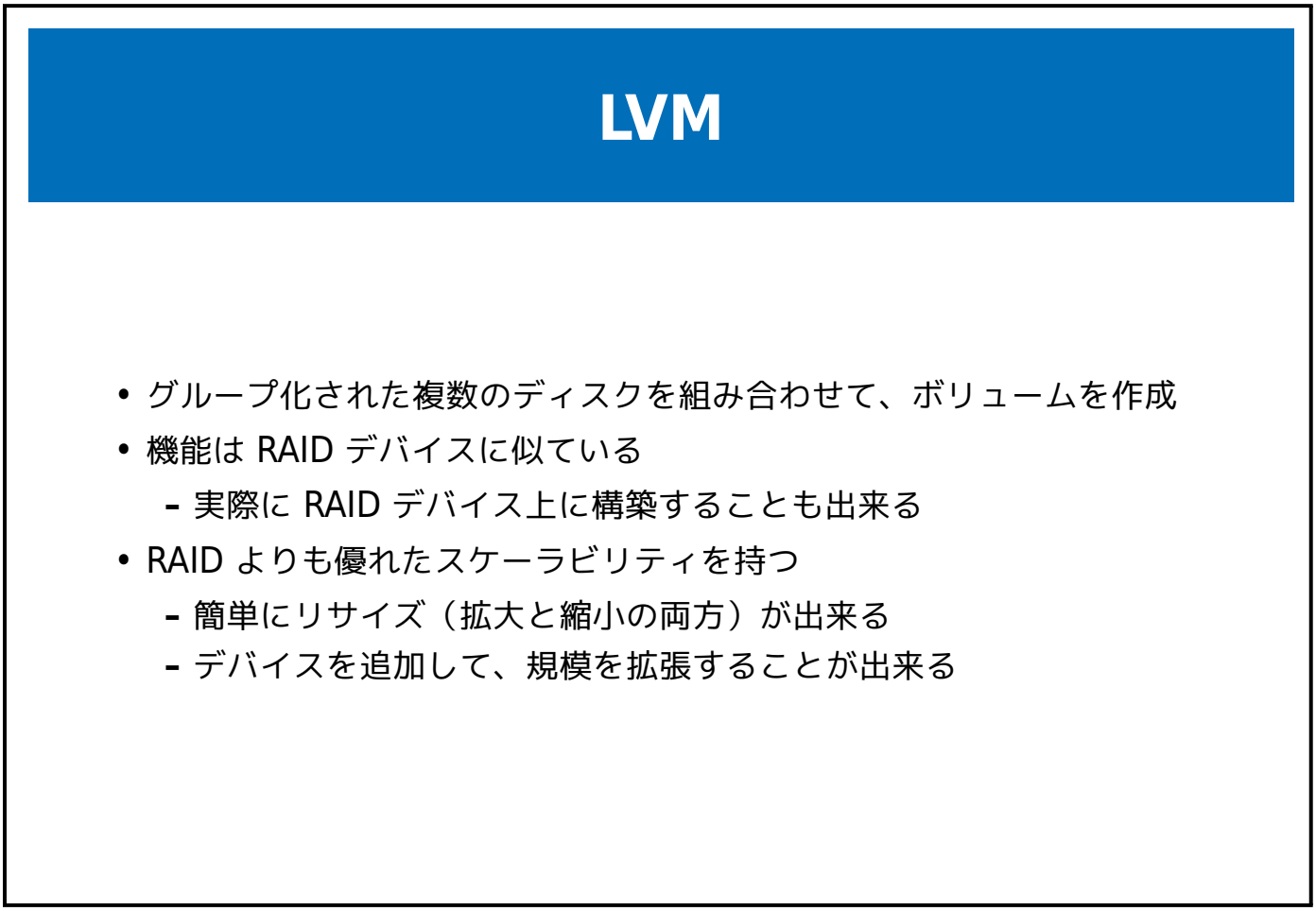
章 23

論理ボリューム管理 (LVM)

注目

** これらのセクションの一部または全体をオプション扱いとする場合があります。これらには補足資料、専門トピック、または高度な話題が含まれインストラクターが教室の状況や時間制約に応じてこれらの内容を紹介するかを判断します。

23.1 論理ボリューム管理 (LVM)



LVM (**L**ogical **V**olume **M**anagement) は、１つの仮想パーティションを複数のチャンクに分割します。各チャンクは、異なるパーティションやディスクに配置できます。

LVM には多くの利点があります。特に、論理パーティションとファイルシステムのサイズ変更や、ストレージ領域の追加やデータの再配置などは非常に簡単にできます。

１つ以上の **物理ボリューム**（ディスクパーティション）を、**ボリュームグループ** にグループ化します。次に、ボリュームグループを **論理ボリューム** に再分割します。論理ボリュームはシステムの物理ディスクパーティションを模倣したもので、通常のパーティションとしてフォーマットしてファイルシステムにマウントできます。

物理ボリュームと論理ボリュームの 作成、削除、サイズ変更 などを行う、さまざまなコマンドラインユーティリティがあります。大半の **Linux** ディストリビューションは **system-config-lvm** と呼ばれるグラフィカルツールを使用します。しかしながら **RHEL** はこのツールのサポートを止めたので、最新のファイルシステムタイプで利用可能な信頼性が高いグラフィカルツールがありません。幸いなのは、コマンドラインユーティリティは使いにくいものではなく、むしろ柔軟です。

LVM はパフォーマンスに悪影響します。**LVM** レイヤーのオーバーヘッドに起因する明確な追加コストがあります。ただし、**RAID** システムで無くても **ストライピング**（複数のディスクへのデータ分割）を使用すれば、並列化による性能向上は期待できます。

論理ボリュームには **RAID** （については次章で説明します）に似た機能があります。実際に **RAID** デバイス上に構築することが出来ます。この構成では **LVM** のスケーラビリティを持った論理ボリュームに **RAID** デバイスの冗長性を付加することが出来ます。

LVM は **RAID** よりも優れたスケーラビリティを備えています。論理ボリュームは簡単にサイズ変更できます。つまり、必要に応じて拡大または縮小できます。さらに領域が必要な場合は、いつでも論理ボリュームにデバイスを追加できます。

23.2 ボリュームとボリュームグループ

ボリュームとボリュームグループ

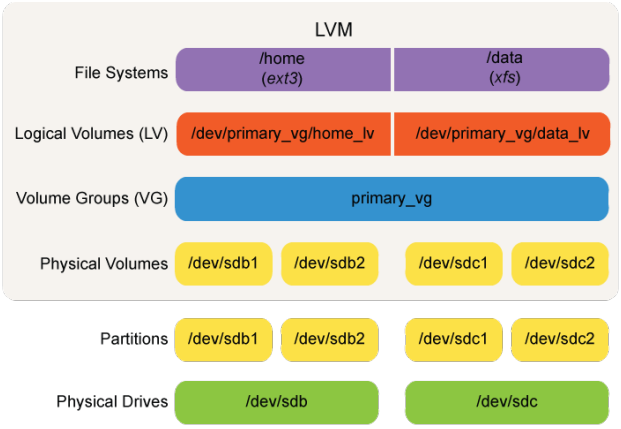


図 23.1: **LVM のコンポーネント**

- デバイスは物理ボリュームに変換される
- 複数の物理ボリュームがグループ化され、ボリュームグループになる
  - ボリュームグループは、エクステントに分割される
  - エクステントのサイズは設定できる (デフォルトは 4MB)
- ボリュームグループから、論理ボリュームが配置される
  - エクステントのサイズと数は、変更することができる
  - 論理ボリューム上にファイルシステムを構築される
  - 任意の名前を付けることが出来る

パーティションは物理ボリュームに変換され、複数の物理ボリュームはボリュームグループにグループ化されます。システム上に複数のボリュームグループが存在する場合があります。

ボリュームグループ内の領域は **エクステント** に分割されます。**エクステント** のサイズはデフォルトで 4MB ですが、割り当て時に簡単に変更できます。

ボリュームグループの生成と操作には、以下のように必ず **vg** で始まるコマンドラインユーティリティが用意されています:

- **vgcreate**: ボリュームグループの生成します。
- **vgextend**: ボリュームグループに物理ボリュームを追加します。
- **vgreduce**: ボリュームグループを縮小します。

ボリュームグループに参加または離脱させる物理パーティションを操作するユーティリティは **pv** で始まり、次のものがあります:

- **pvcreate**: 指定パーティションを物理ボリュームとして管理できるようにします。
- **pvdisplay**: 使用されている物理ボリュームの詳細な情報を表示します。
- **pvmove**: ボリュームグループ内の１つの物理ボリュームから、他の物理ボリュームにデータを移動します。これは何らかの理由でディスクまたはパーティションが削除された場合に、必要になることがあります。これは次のコマンドに続きます:
- **pvremove**: 指定したパーティションにある物理ボリュームを削除します。

`man lvm` と入力すると **LVM** ユーティリティの説明が表示されます。

23.3 論理ボリュームの利用

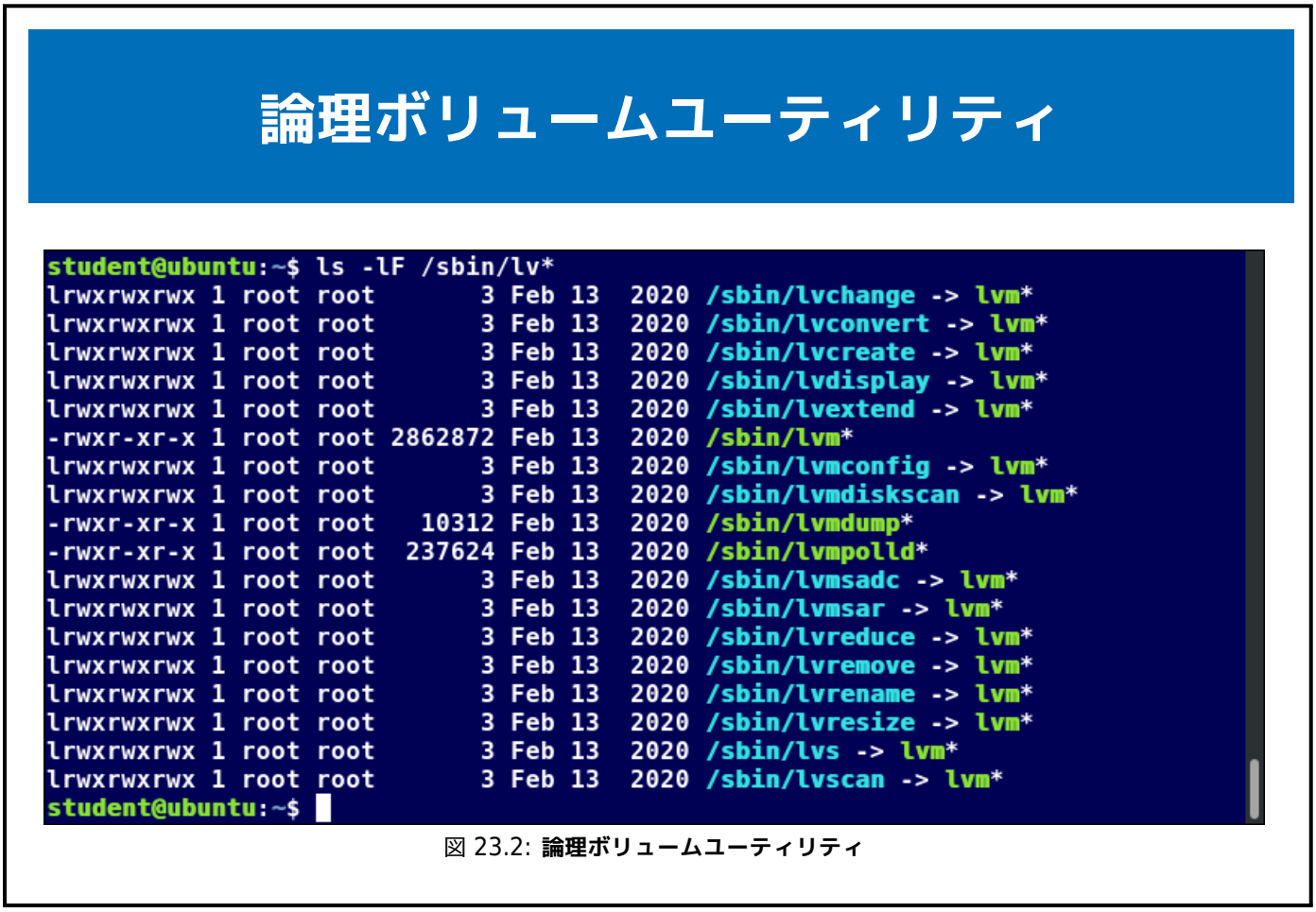


図 23.2: **論理ボリュームユーティリティ**

論理ボリュームの操作には、多くのユーティリティがあります。当然のことながらそれらはすべて **lv** で始まります。
ここでは最も一般的に使用されるものについて説明しますが、次のコマンドで短いコマンドリストを取得できます:

```
$ ls -lF /sbin/lv*
```

これらのユーティリティは `/usr/sbin` ではなく `/sbin` にあります。これらユーティリティがブート、または修復とリカバリのいずれかに必要な場合があるためです。

そのほとんどは、すべての作業を行う Swiss army knife（＝万能）プログラムである **lvm** にシンボリックリンクされており、呼び出される名前から何のためのユーティリティなのか想像できます。この命名ルールはほとんどの **pv*** と **vg*** ユーティリティにも当てはまり、とてもわかりやすくなっています。

```
student@ubuntu:~$ ls -lF /sbin/pv*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/pvchange -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/pvck -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/pvcreate -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/pvdisplay -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/pvmove -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/pvremove -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/pvresize -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/pvs -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/pvscan -> lvm*
student@ubuntu:~$
```

図 23.3: **物理ボリュームユーティリティ**

論理ボリュームの作成

1. ディスクドライブに、パーティションを作成する
   (**fdisk** で指定するディスクタイプ番号は 8e)
2. パーティションに、物理ボリュームを作成する
3. ボリュームグループを作成する
4. ボリュームグループに、論理ボリュームを割り当てる
5. 論理ボリュームをフォーマットする
6. 論理ボリュームをマウントする（必要に応じ `/etc/fstab` も更新）

lvcreate はボリュームグループから論理ボリュームを割り当てます。サイズは、バイト単位、またはエクステント数で指定できます。(デフォルトは 4MB です)。任意の名前を使用できます。

lvdisplay は使用可能な論理ボリュームの情報を表示します。

ファイルシステムを論理ボリュームに割り当て、いつもと同じように **mkfs** でフォーマットします。

たとえば、既に 2 つのパーティション `/dev/sdb1` と `/dev/sdc1` があり、これらのパーティションのファイルタイプが 8e に設定済みでなら、以下のコマンドで論理ボリュームを構成できます:

```
$ sudo pvcreate /dev/sdb1
$ sudo pvcreate /dev/sdc1
$ sudo vgcreate -s 16M vg /dev/sdb1
$ sudo vgextend vg /dev/sdc1
$ sudo lvcreate -L 50G -n mylvm vg
$ sudo mkfs -t ext4 /dev/vg/mylvm
$ mkdir /mylvm
$ sudo mount /dev/vg/mylvm /mylvm
```

マウント設定を永続化（= 再起動してもマウントされる）するには、`/etc/fstab` に以下の行を追加してください。

```
/dev/vg/mylvm /mylvm ext4 defaults 1 2
```

論理ボリュームの表示

- **pvdisplay** - 物理ボリュームを表示する

```
$ pvdisplay
$ pvdisplay /dev/sda5
```

- **vgdisplay** - ボリュームグループを表示する

```
$ vgdisplay
$ vgdisplay /dev/vg0
```

- **lvdisplay** - 論理ボリュームを表示する

```
$ lvdisplay
$ lvdisplay /dev/vg0/lvm1
```

pvdisplay は物理ボリュームの情報を表示します。物理ボリューム名を省略すると全ての物理ボリュームがリストされます。
vgdisplay はボリュームグループを表示します。ボリュームグループ名を省略すると全てのボリュームグループがリストされます。
lvdisplay は論理ボリュームの情報を表示します。論理ボリューム名を省略すると全ての論理ボリュームがリストされます。

```
student@ubuntu:~$ ls -lF /sbin/vg*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgcfgbackup -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgcfgrestore -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgchange -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgck -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgconvert -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgcreate -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgdisplay -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgexport -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgextend -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgimport -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgimportclone -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgmerge -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgmknodes -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgreduce -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgremove -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgrename -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgs -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgscan -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13  2020 /sbin/vgsplit -> lvm*
student@ubuntu:~$
```

図 23.4: **ボリュームグループユーティリティ**

23.4 論理ボリュームのサイズ変更

論理ボリュームのサイズ変更

- 物理パーティションのサイズ変更より、はるかに簡単である
- どのボリュームグループのエクステントを使うこともできる
- ファイルシステムがオンライン状態のまま、縮小ないし拡大ができる
- サイズは、エクステント数でもバイト数 (M,G, 等) で指定しても良い
- 一番確実なのは、以下のように **lvresize** を使う方法である:
  ```
  $ sudo lvresize  -r -L 20 GB  /dev/VG/mylvm
  ```
  -r オプションを使うと、ボリュームサイズ変更とファイルシステムサイズ変更が一緒に行われる
- **lvextend**、**lvreduce** を **resize2fs** と組み合わせて設定もできる
- サイズ変更ユーティリティは、ファイルシステムに依存する

論理ボリューム上に構築された **ext4** ファイルシステムを拡大するには:

```
$ sudo lvresize -r -L +100M /dev/vg/mylvm
```

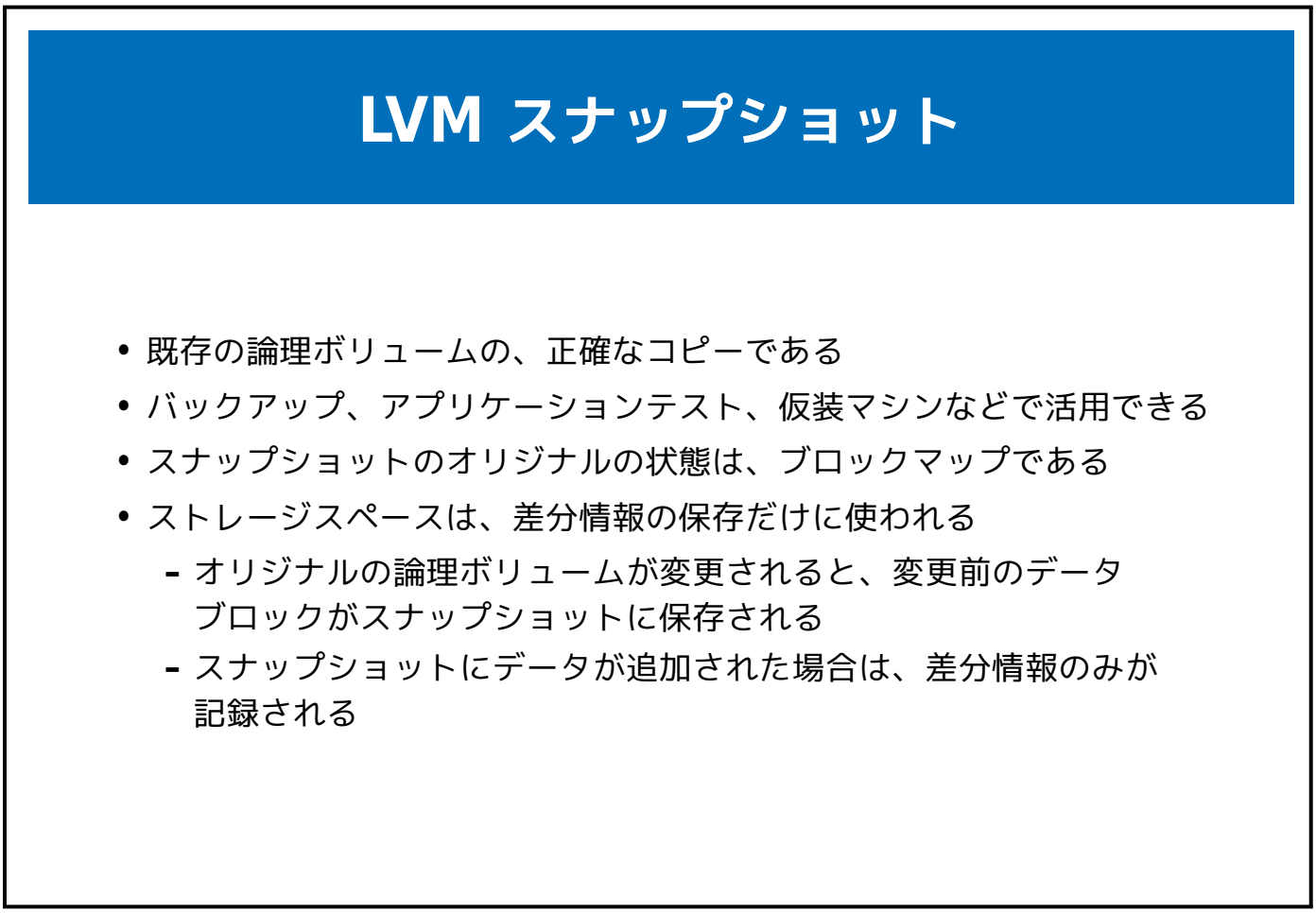
プラス記号はサイズを増やすという意味です。サイズを拡大する時には、アンマウントさせる必要はありません。
ファイルシステムを縮小するには:

```
$ sudo lvresize -r -L 200M /dev/vg/mylvm
```

ボリュームグループのサイズ縮小をしたい場合は:

```
$ sudo pvmove /dev/sdc1
$ sudo vgreduce vg /dev/sdc1
```

23.5 LVM スナップショット **



LVM Snapshots は、既存の論理ボリュームの正確なコピーを作成する。

スナップショットは、バックアップ、アプリケーションのテスト、**VM** (仮装マシン) のデプロイ時などに有効活用できます。スナップショットのオリジナルの状態はブロックマップとして保存されます。

スナップショットは差分情報の記録スペースだけを消費します:

- オリジナルの論理ボリュームが変更されると、変更前のデータブロックがスナップショットに保存されます。
- スナップショットにデータが追加された場合は、差分情報のみが記録されます。

既存の論理ボリュームにスナップショットを作るには:

```
$ sudo lvcreate -l 128 -s -n mysnap /dev/vg/mylvm
```

マウントポイントを作成し、スナップショットをマウントするには:

```
$ mkdir /mysnap
$ mount -o ro /dev/vg/mysnap   /mysnap
```

スナップショットの利用とスナップショットの削除方法:

```
$ sudo umount /mysnap
$ sudo lvremove /dev/vg/mysnap
```

スナップショットを使い終わったら、必ずスナップショットを削除してください。スナップショットを削除せず、変更の情報でいっぱいになってしまうと、スナップショット自体が自動的に無効になります。オリジナルのサイズのスナップショットは、オーバーフローすることはありません。

23.6 演習

デモ教材ビデオ

using_lvm_demo.mp4

課題 23.1: 論理ボリューム

2 つの 250MB パーティションを使って論理ボリュームを作成します。ディスク上に実パーティションを確保できると想定します。

1. 2 つの 250MB の論理ボリューム（8e）のパーティションを確保します。
2. パーティションを物理ボリュームに変更します。
3. myvg という名称のボリュームグループを作成し、2 つの物理ボリュームを標準のエクステントサイズで追加します。
4. ボリュームグループ myvg から、mylvm という名称で 300MB の論理ボリュームを割り当てます。
5. フォーマット後、論理ボリューム mylvm を /mylvm にマウントします。
6. **lvdisplay** を使って、論理ボリュームに関する情報を見ます。
7. 論理ボリュームと対応するファイルシステムを 350MB まで大きくします。

解 23.1

1. 以下を実行します:

   ```
   $ sudo fdisk /dev/sda
   ```

   どのようなハードディスクでも構いませんが、パーティションを 2 つ作成します。**fdisk** の t コマンドでパーティションタイプを 8e とします。タイプを設定しなくても問題ありませんが、設定した方が混乱が少ないです。w コマンドで、パーティションテーブルに書き込み終了します。次に

   ```
   $ sudo partprobe -s
   ```

   を実行するか、新パーティションが確実に有効になるようにリブートします。

2. ここでは、新パーティションを /dev/sdaX と /dev/sdaY とします:

   ```
   $ sudo pvcreate /dev/sdaX
   $ sudo pvcreate /dev/sdaY
   $ sudo pvdisplay
   ```

3. ```
   $ sudo vgcreate myvg /dev/sdaX /dev/sdaY
   $ sudo vgdisplay
   ```

4. ```
   $ sudo lvcreate -L 300M -n mylvm myvg
   $ sudo lvdisplay
   ```

5. ```
   $ sudo mkfs.ext4 /dev/myvg/mylvm
   $ sudo mkdir /mylvm
   $ sudo mount /dev/myvg/mylvm /mylvm
   ```

   継続的にマウントしたいときは、/etc/fstab に以下のように記入してください:

/etc/fstab の記述内容

```
/dev/myvg/mylvm /mylvm ext4 defaults 0 0
```
```

6. `$ sudo lvdisplay`

7. ```
   $ df -h
   $ sudo lvresize -r -L 350M /dev/myvg/mylvm
   $ df -h
   ```

   または

   ```
   $ sudo lvresize -r -L +50M /dev/myvg/mylvm
   ```

   または、古い方法ですが：

   ```
   $ df -h
   $ sudo lvextend -L 350M /dev/myvg/mylvm
   $ sudo resize2fs /dev/myvg/mylvm
   $ df -h
   ```
   ```

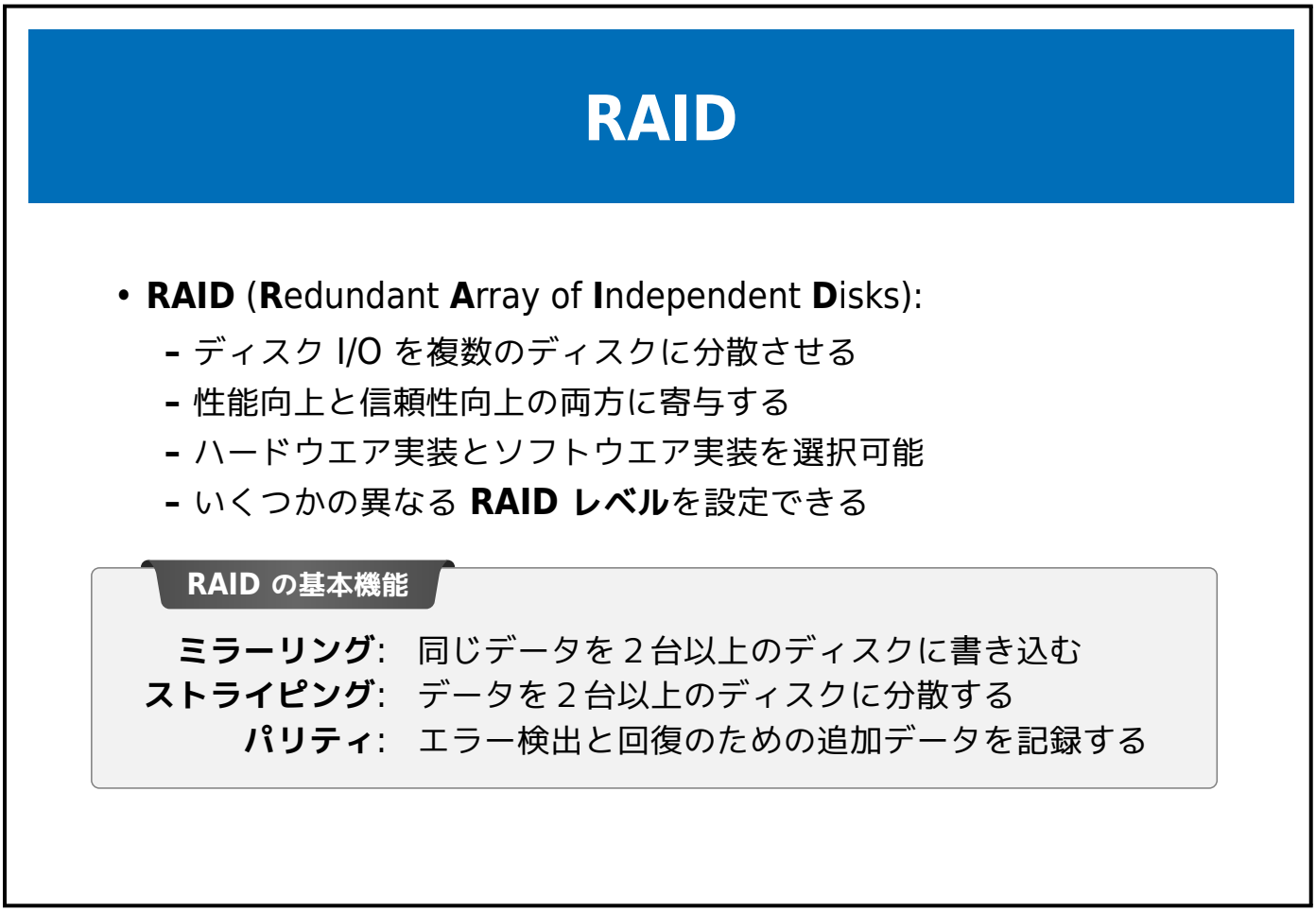
章 24

RAID **

注目

** これらのセクションの一部または全体をオプション扱いとする場合があります。これらには補足資料、専門トピック、または高度な話題が含まれインストラクターが教室の状況や時間制約に応じてこれらの内容を紹介するかを判断します。

24.1 RAID



RAID (**R**edundant **A**rray of **I**ndependent **D**isks) は、複数のディスクに I/O を分散します。これにより、並列動作による効率改善に対応した **SCSI** などの最新のディスクコントローラーインターフェースの性能を大きく向上させることができます。

RAID は、ソフトウェア（**Linux** カーネルのソフトウエア **RAID** 実装は枯れています）でも ハードウェア でも実装できます。もし使用するハードウェア **RAID** の実装が十分に信頼に足るものであれば、ハードウエア **RAID** の方がソフトウェア **RAID** より高性能です。ハードウェア実装では、オペレーティングシステムは **RAID** が使われていることを直接認識しません。たとえば、RAID-5 で構成された３台の 512GB ハードドライブ（データ用に２つ、パリティ用に１つ）はオペレーティングシステムには、単純に一台の 1TB ディスクに見えます。

ただし、ハードウェア **RAID** にも欠点はあり、万一使っているディスクコントローラに障害が発生した場合には互換性のあるコントローラに交換する必要があるのですが、それを入手するのは簡単ではありません。ソフトウエア **RAID** なら、同じディスクドライブに交換するだけでコントローラには手を付けずに復旧できます。このため、中小規模のハードウエア（＝サーバー）ではソフトウエア **RAID** が選択されることも多いのです。

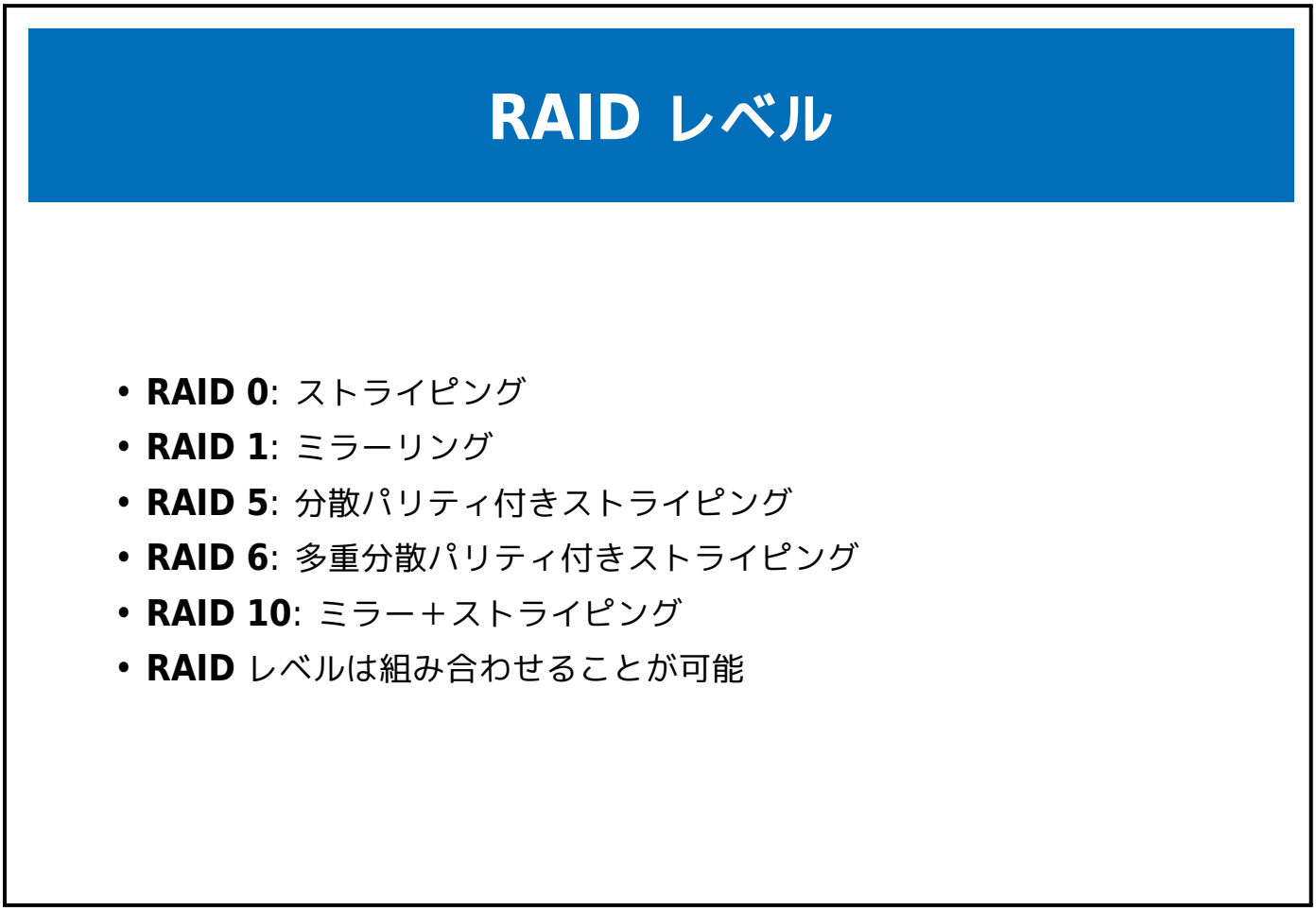
RAID 構成を採用する目的の一つは、ファイルシステムを複数ディスクに分散させることです。これにより、１つのドライブよりも大きなファイルシステムを作成できます。通常は **RAID** は複数のディスクのパーティションを組み合わせて構築します。

RAID デバイスのもう１つの利点は、パフォーマンスか冗長性、またはその両方を改善できることです。ストライピングは情報を複数のデバイスに分散することでパフォーマンスを向上させ、並列書き込みが可能になる場合もあります。ミラーリングは同じ情報を複数のドライブに書き込むので冗長性が向上します。

（ソフトウエア）**RAID** デバイスの作成と管理には **mdadm** が使用されます。

生成されたアレイに付けられた名前 `/dev/mdX` は `/dev/sda1` などの他のデバイス名と同じように使用できます。

24.2 RAID レベル



RAID には、その複雑さと用途に応じて様々なレベルが存在します。最も一般的に使用されているのはレベル 0、1、5 です。

- **RAID 0** はストライピングのみを使用します。データは複数のディスクに分散されます。ただし **RAID** と言う名称にもかかわらず冗長性はなく安定性向上や修復の機能はありません。実際ディスクに障害が発生するとデータが失われます。ただし、I/O タスクの並列化によりパフォーマンスを大幅に向上させることができます。
- **RAID 1** はミラーリングのみを使用します。各ディスクは複製を持っています。これは障害からの修復に適しています。少なくとも 2 台のディスクが必要です。
- **RAID 5** は rotating parity stripe (= 輪番性パリティ分散、全てのディスクにパリティ情報を分割して記録する方式) を利用します。一つのドライブに障害が発生しただけならデータは失われずパフォーマンスが低下するだけです。 3 台以上のディスクが必要です。
- **RAID 6** にはデュアルパリティ付きのストライプディスクがあります。 2 台のディスクの損失を処理できますが少なくとも 4 台のディスクが必要です。**RAID 5** はディスクに大きなストレスを与え修復中に障害を引き起こす可能性があるため **RAID 6** の重要性が増えています。
- **RAID 10** はミラーリングとストライピングが組み合わされたデータセットです。少なくとも 4 台のドライブが必要です。

一般論で言えば、ディスクを追加すればパフォーマンスは向上します。

24.3 ソフトウェア RAID 構成

ソフトウェア RAID 構成

ソフトウェア **RAID** デバイスを構成するための基本的な手順:

- 各ディスクにパーティションを作成する
  （**fdisk** で指定するパーティションタイプは fd ）
- **mdadm** で **RAID** デバイスを作成する
- **RAID** デバイスをフォーマットする
- /etc/fstab にデバイスを追加する
- **RAID** デバイスをマウントする
- **RAID** の詳細を取得（理解）して永続性を実現する

以下のコマンドで **RAID** を停止させる:

```
$ sudo mdadm -S
```

一例として、ディスク sdb と sdc (例えば /dev/sdbX と /dev/sdcX) のそれぞれに対して **fdisk** を実行し、パーティションタイプを fd に設定します。

```
$ sudo fdisk /dev/sdb
$ sudo fdisk /dev/sdc
```

次にアレイを設定し、フォーマットし、構成ファイルに追加し、マウントします:

```
$ sudo mdadm --create /dev/md0 --level=1 --raid-disks=2  /dev/sdbX /dev/sdcX
$ sudo mkfs.ext4 /dev/md0
$ sudo bash -c "mdadm --detail --scan >> /etc/mdadm.conf"
$ sudo mkdir /myraid
$ sudo mount /dev/md0 /myraid
```

/etc/fstab にマウントポイント設定行を追加してください。

```
/dev/md0    /myraid        ext4    defaults 0 2
```

/proc/mdstat を使って **RAID** のステータスを確認できます:

```
$ cat /proc/mdstat
 Personalities : [raid1]
 md0 : active raid1 sda8[1] sda7[0]
 ---------- 521984 blocks [2/2]
 unused devices: <none>
```

mdadm -S /dev/md0 で RAID デバイスを停止できます。

24.4 RAID の監視

以下に示すように、複数の方法で RAID デバイスを監視できます:

```
$ sudo mdadm --detail /dev/md0
$ cat /proc/mdstat
```

`/etc/mdadm.conf` を設定しておけば **mdmonitor** を使うこともできます。
以下のコマンドで、システム上の全ての **RAID** デバイスのステータスを表示できます:

```
$ sudo mdadm --detail /dev/mdX
```

この設定により **RAID** デバイス `/dev/mdX` のステータスが表示されます。別の設定方法としては `/proc` ファイルシステムを利用することもできます:

```
$ cat /proc/mdstat
```

`/etc/mdadm.conf` を以下の行を追加しておけば **mdmonitor** を使うこともできます:

```
MAILADDR eddie@haskell.com
```

ここでは **RAID** にアレイの起動が失敗したとかアレイがデグレーデッド（= 冗長性が保てない状態で運用中）状態になった等の問題が発生した時に `eddie@haskell.com` 宛てに告知メールが送信されるように設定しています。これを有効にするには:

```
$ sudo systemctl start mdmonitor
```

ブート時にこれを有効にするには:

```
$ sudo systemctl enable mdmonitor
```

24.5 RAID ホットスペア

RAID が提供する重要な機能の１つは冗長性です。障害から可能な限り迅速に修復するために **ホットスペア** の使用が可能です。
RAID アレイの作成時にホットスペアを作成するには:

```
$ sudo mdadm --create /dev/md0 -l 5 -n3 -x 1 /dev/sda8 /dev/sda9 /dev/sda10 /dev/sdb2
```

`-x 1` スイッチは１台のスペアデバイスを使用するという意味です。ホットスペアは、後で追加することもできます。
以下のコマンドは、冗長性とアレイのホットスペアのテストを行います:

```
$ sudo mdadm --fail /dev/md0 /dev/sdb2
```

テストに使われたドライブの修復、もしくは障害状態において新しいドライブを使う修復では、まず「障害のある」メンバを削除してから次に「新しい」メンバを追加します:

```
$ sudo mdadm --remove /dev/md0 /dev/sdb2
$ sudo mdadm --add /dev/md0 /dev/sde2
```

24.6 演習

課題 24.1: RAID デバイスの作成

通常 **RAID** デバイスを作成するときは、異なったディスク上のパーティションを使います。しかし本演習ではハードウェアを用意できないと思われます。そのため、同じディスク上にある 2 つのパーティションを利用します。

パーティションが 1 つのディスクにあるか、複数のディスクにあるかを問わず、手順は変わりません（しかしながら、1 つのディスク上に **RAID** を作成するには、少々理由があります）。

1. **fdisk** か **LVM** を使って raid タイプ（fd）のパーティションをハードディスク上に 2 つ作成します。
2. 2 つのパーティションを使い `/dev/md0` という名称の **RAID 1** デバイスを作成します。
3. **RAID** デバイスを **ext4** ファイルシステムにフォーマットします。続いて `/myraid` にマウントし、恒久的にマウントされるようにします。
4. **mdadm** を使って `/dev/md0` に関する情報を `/etc/mdadm.conf` に書き込みます。（ディストリビューションによっては、このファイルがすでに用意されていることがあります）。
5. `/proc/mdstat` を調べて **RAID** デバイスのステータスを見ます。

解 24.1

1. 新しいパーティションを作成するときは:

   ```
   $ sudo fdisk /dev/sda
   ```

   を実行して、以前のようにパーティションを作成してください。これ以降パーティションを `/dev/sdaX` と `/dev/sdaY` とします。新規パーティションをシステムに確実に認識させるため、**partprobe** か **kpartx** を実行するか、リブートしてください。

2. ```
   $ sudo mdadm -C /dev/md0 --level=1 --raid-disks=2 /dev/sdaX /dev/sdaY
   ```

3. ```
   $ sudo mkfs.ext4 /dev/md0
   $ sudo mkdir /myraid
   $ sudo mount /dev/md0 /myraid
   ```

   そして `/etc/fstab` に追加します。

   in /etc/fstab への追加内容

   ```
   /dev/md0  /myraid    ext4    defaults  0 0
   ```

4. ```
   $ sudo bash -c "mdadm --detail --scan >> /etc/mdadm.conf"
   ```

5. ```
   $ cat /proc/mdstat
   ```

```
Personalities : [raid1]
md0 : active raid1 dm-14[1] dm-13[0]
      204736 blocks [2/2] [UU]

unused devices: <none>
```

注目

リブートしても **RAID** ボリュームが自動的にマウントされていることを確認します。全てが終わったら `/etc/fstab` から該当行を削除し、パーティションを削除してください。

章 25

カーネルサービスと構成

25.1 カーネルの概要

カーネルの概要

- カーネルは、オペレーティングシステムのコアである
- カーネルは以下の役割を担う:
  - システム（＝ハードウエア）の初期化と起動を行う
  - プロセスのスケジューリングを制御する
  - メモリーを管理する
  - ハードウエアアクセスを制御（仲介）する
  - アプリケーションとストレージデバイス間の I/O (Input/Output)
  - ローカルとネットワーク越しのファイルシステムをインプリする
  - セキュリティ管理、（ファイルシステムの権限管理などの）ローカルとネットワーク接続の両方をカバーする
  - ネットワークを制御する
- オペレーティングシステムには、カーネル以外にも多くのコンポーネントが含まれている

狭義の **Linux** は、オペレーティングシステムの **カーネル** のみを指します。**オペレーティングシステム** にはカーネルとやりとりするライブラリやアプリケーションなど、他の多くのコンポーネントが含まれています。

カーネルはハードウェアとソフトウェアとを接続し、競合するアプリケーションやサービス間でメモリーや CPU 時間の割り当てなどのシステムリソースを管理（調停）する重要な中枢コンポーネントです。接続されたすべてのデバイスを **デバイスドライバ** を使って処理し、デバイスをオペレーティングシステムで使用できるようにします。

カーネル **のみ** を実行するシステムの場合、機能はかなり制限されます。特定用途向けに特化した **組み込みデバイス** でのみ、このようなケースが見受けられます。

25.2 カーネルコンフィグレーション

カーネルコマンドライン

- パラメータはシステム起動時に **カーネルコマンドライン** で渡される
- **GRUB** 構成ファイル内の `linux` ないし `linux16` に置かれる: `grub.cfg` は `/boot` ディレクトリ内にある
- 起動時に対話的な変更（= パラメータの上書き）ができる
- 実際の設定方法は **Linux** ディストリビューションとバージョンに依存する:

```
linux /boot/vmlinuz-4.15.0-58-generic \
       root=UUID=5cec328b-bcd6-46d2-bcfa-8430257cd7a5 \
       ro find_preseed=/preseed.cfg auto noprompt \
       priority=critical locale=en_US quiet crashkernel=512M-:192M
```

または

```
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.15.0-58-generic \
     root=UUID=5cec328b-bcd6-46d2-bcfa-8430257cd7a5\
     ro find_preseed=/preseed.cfg auto noprompt \
     priority=critical locale=en_US quiet crashkernel=512M-:192M
```

カーネルコマンドラインのサンプルはディストリビューションによりますが、以下のようなものです:

```
linux /boot/vmlinuz-4.19.0 root=UUID=7ef4e747-afae-48e3-90b4-9be8be8d0258 ro quiet crashkernel=384M-:128M
```

```
linuxefi /boot/vmlinuz-5.2.9 root=UUID=77461ee7-c34a-4c5f-b0bc-29f4feecc743 ro crashkernel=auto rhgb quiet
```

そしてこの行は `/boot/grub` などの `boot` 下のサブディレクトリの `grub.cfg` または `/boot/efi/EFI/centos/grub.cfg` などのファイルの中にあります。

`vmlinuz` ファイル以降で指定されているものはすべてオプションです。カーネルが理解できないオプションは、すべてシステムで最初に実行されるユーザープロセスの **init** (`pid=1`) に渡されます。システムの起動に使用したコマンドラインを確認するには:

```
$ cat /proc/cmdline
```

```
BOOT_IMAGE=(hd0,msdos1)/boot/vmlinuz-5.11.0 root=UUID=7a8244d5-f289-4067-8ad6-9090080b7e35 ro
↪  resume=UUID=d602c4e1-ef8a-4945-8e3b-e98fcc8bfba2 rhgb quiet
```

BLSFG で状況が変わっていく

現在 **Fedora** と **RHEL/CentOS 8** はカーネルコマンドラインを置き換える **BLSFG** (Boot Loader Specification Configuration) を使用します。情報は `/boot/grub2/grubenv` で見ることができます。これに関する詳細は **grub** の章にページを設けて説明します。

25.3 カーネル ブート パラメータ

カーネル ブート パラメータ

- grub.cfg 内のカーネル行で指定する
- 起動時に対話的に指定することもできる
- パラメータは以下のいずれかの方法で指定する:
  - 単純な引数として指定
  - param=value の形式で value には文字列、整数、整数の配列などを指定できる

```
vmlinuz root=/dev/sda6 .....  noapic .... crashkernel=256M
```

- パラメータを意図的に秘密にしたり、隠されたりはできない
- カーネルソース内の kernel-parameters.txt に一覧形式で説明

カーネルオプションは、カーネルを指定する行の最後に、項目をスペースで区切りで記述します。カーネルブートパラメータの例（途中に改行を入れてはいけません）:

```
linux16 /boot/vmlinuz-3.19.1.0 root=UUID=178d0092-4154-4688-af24-cda272265e08 ro
        vconsole.keymap=us crashkernel=auto  vconsole.font=latarcyrheb-sun16 rhgb quiet LANG=en_US.UTF-8
```

- root: ルートファイルシステムの指定
- ro: ブート時に読み取り専用でルートデバイスをマウントする設定
- vconsole.keymap: コンソールで使用するキーボードのキー配列の指定
- crashkernel: **カーネルクラッシュダンプ** 用に確保するメモリー量の指定
- vconsole.font: コンソールで使用するフォントの指定
- rhgb: ブート時にグラフィカル表示をする設定
- quiet: ほとんどのログメッセージを出さない設定
- LANG: システム言語の指定

man bootparam とカーネルソース内のドキュメント kernel-parameters.txt を参照してください。カーネルドキュメントは **kernel-doc** や **linux-doc** などのパッケージをインストールするか、オンライン http://kernel.org/doc/Documentation/ で見ることができます。

25.4 sysctl

sysctl

- /proc/sys 内の設定が **sysctl** で実行される
- 変数名は /proc/sys のパスに対応している
- カーネルパラメータの表示、設定、変更は:

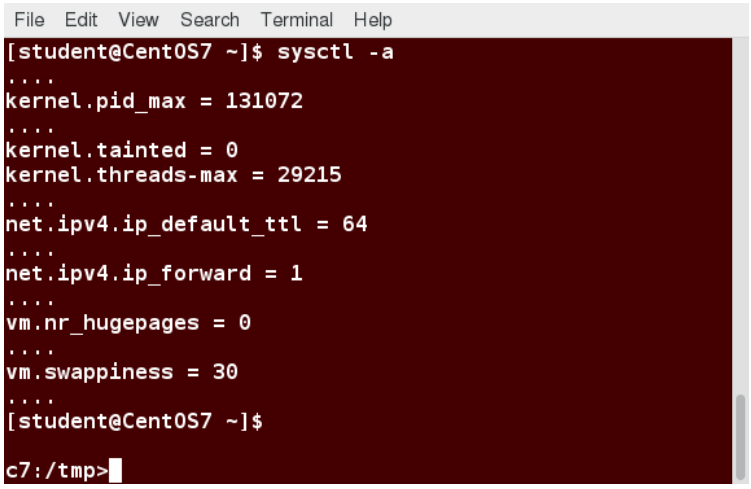


図 25.1: **sysctl**

sysctl インターフェイスは、実行中のカーネルパラメータを読み取って調整するために使用します。以下を実行することで現在の値を表示できます:

```
$ sysctl -a
```

以下の 2 つのコマンドは同じことをします:

```
$ sudo sh -c 'echo 1 > /proc/sys/net/ipv4/ip_forward'
$ sudo sysctl net.ipv4.ip_forward=1
```

(コマンドの = 記号の前後にスペースを入れないでください。) 最初のコマンド文では **echo** 付きの単純な **sudo** を使用できないことに注意してください。コマンドは、ここに示した複雑な方法で実行するか root として実行する必要があります。

設定が /etc/sysctl.conf に書かれている場合、ブート時に設定を修正できます。以下のように入力します:

```
$ sudo sysctl -p
```

ファイルを即座に読み込み、すべてのパラメータを指定されたとおりに設定します。これもブートプロセスの一部です。

/usr/lib/sysctl.d と /usr/lib/sysctl.d

systemd の登場により状況はもう少し複雑になりました。ベンダーは /usr/lib/sysctl.d/ /ディレクトリ内に設定ファイルを置きます。これらのファイルは /etc/sysctl.d にあるファイルに追加または置換できます。ただし以前からあるファイル (/etc/sysctl.conf) もファイルが自己文書化されているので (= 見れば分かるよう具体的に書かれている)、引き続きサポートされます。

25.5 演習

デモ教材ビデオ

using_sysctl_demo.mp4

課題 25.1: sysctl によるシステムの調整

1. システムで **ping** が可能かを調べます。
2. net.ipv4.icmp_echo_ignore_all の値を表示します。これはシステムが **ping** に応答するかしないかの指定です。0 の場合 ping に応答します。
3. **sysctl** コマンドを使って net.ipv4.icmp_echo_ignore_all に 1 を設定します。ping に応答するか確認します。
4. net.ipv4.icmp_echo_ignore_all の値を 0 に戻して、もとの動作に戻るか確認します。
5. /etc/sysctl.conf を修正して値を変更することで、リブートしなくても本設定を有効にできます。
6. これらが正しく動作することを確認します。

全てを終えたら、システムがもとに戻るようにリセットします。

解 25.1

ping の宛先は localhost、127.0.0.1（ループバックアドレス）のどちらか、または実際の IP アドレスでもかまいません。

1.
```
$ ping localhost
```
2.
```
$ sysctl net.ipv4.icmp_echo_ignore_all
```
3.
```
$ sudo sysctl net.ipv4.icmp_echo_ignore_all=1
$ ping localhost
```
4.
```
$ sudo sysctl net.ipv4.icmp_echo_ignore_all=0
$ ping localhost
```
5. 以下の行を /etc/sysctl.conf に追加してください：

in /etc/sysctl.conf の追加内容

```
net.ipv4.icmp_echo_ignore_all=1
```

そして以下を実行してください:

```
$ sysctl -p
```

6.
```
$ sysctl net.ipv4.icmp_echo_ignore_all
$ ping localhost
```

/etc/sysctl.conf への変更は恒久的なので、前の状態に戻したいと思うでしょう。

課題 25.2: プロセス ID の最大値を変更する

通常 **Linux** システムでは、最初のユーザープロセスの **init** プロセスがプロセス番号（PID）PID=1 となりその後、プロセスが生成されるごとに 1 つずつ番号が大きくなっていきます（プロセスが死ぬときも同様です）。

しかし、PID が /proc/sys/kernel/pid_max（32768 : 32K）に達したとき、また下から開始します。つまり 32K 個以上のプロセスは実行できないということです。

1. PID の最大値を取得します。
2. 現在発行済の PID を取得します。
3. pid_maxを現在の発行値より小さくします。
4. 新しいプロセスを生成して、PID の値を確認します。

解 25.2

重要

以下では 2 つの方法を演習します。一つは **sysctl** を実行する方法、もう一つは、直接 /proc/sys/kernel/pid_max に echo する方法です。**echo** する方法は、root になる必要があります; **sudo** は使えません。なぜかがわからないときは、自分で調べてください。

1. ```
   $ sysctl kernel.pid_max
   $ cat /proc/sys/kernel/pid_max
   ```

2. 入力します：

   ```
   $ cat &
   ```

   ```
   [1] 29222
   ```

   ```
   $ kill -9 29222
   ```

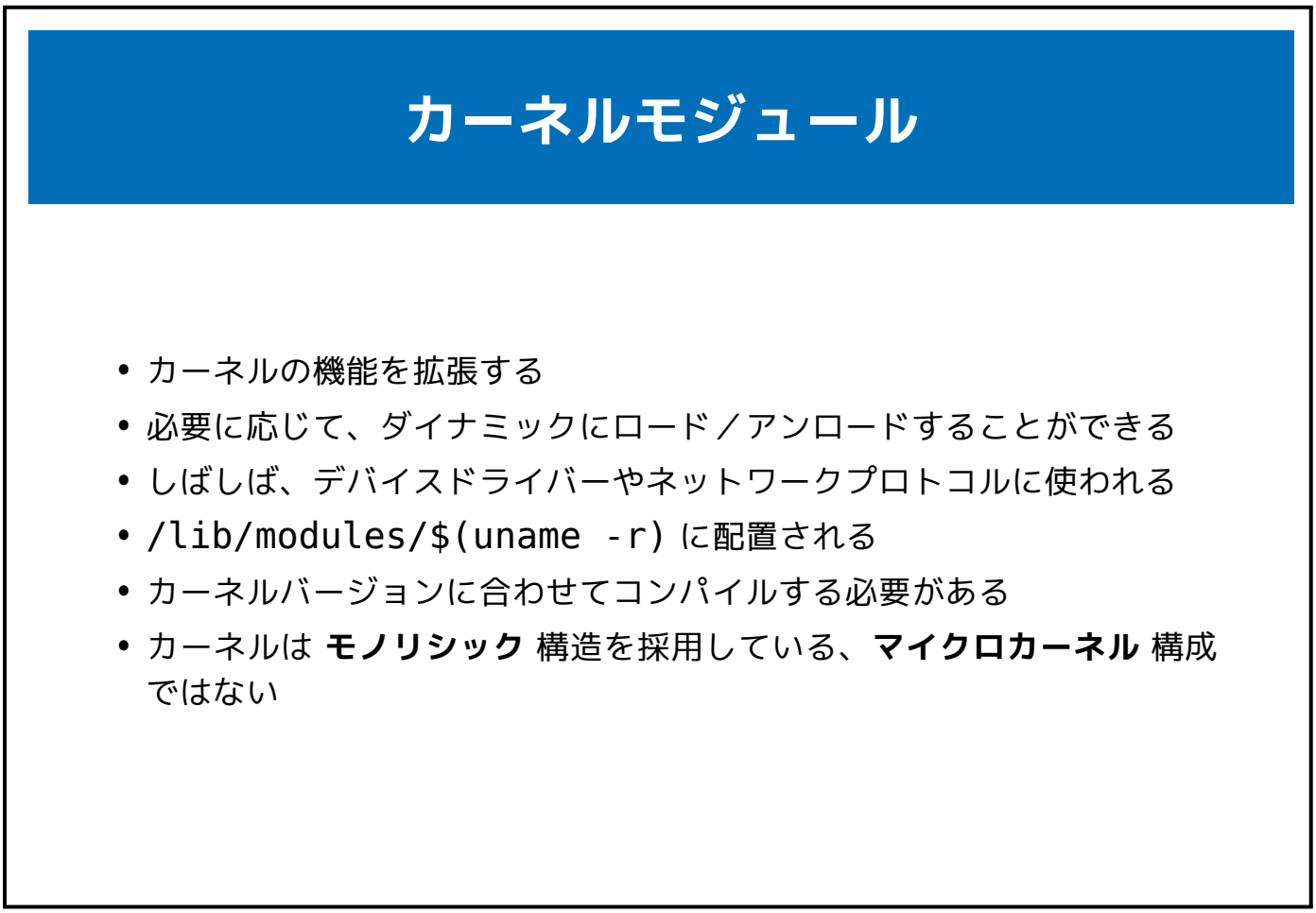
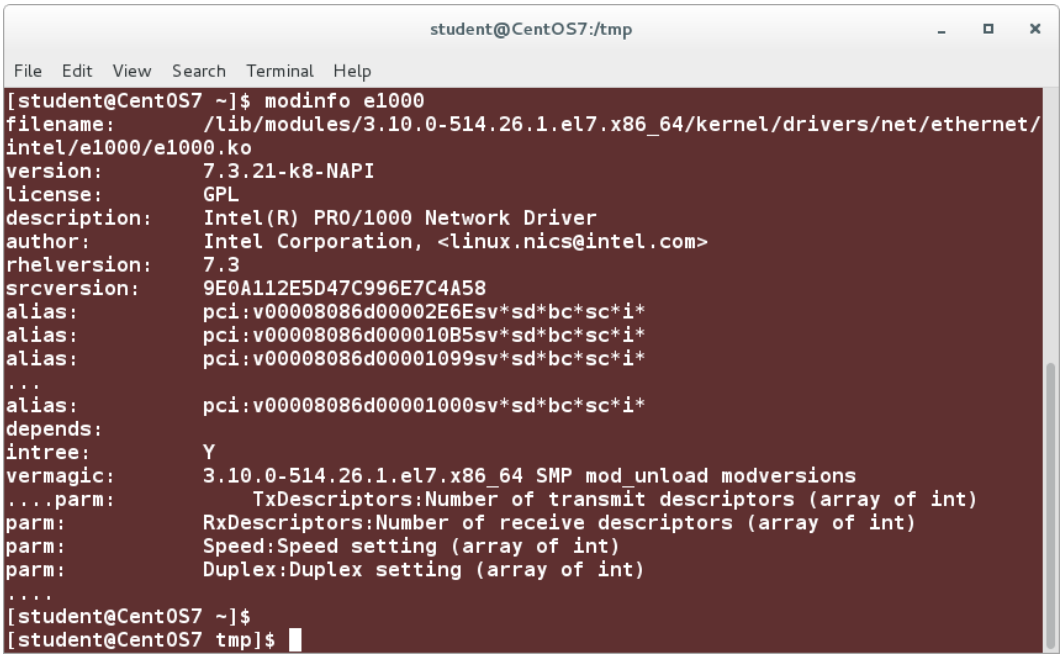
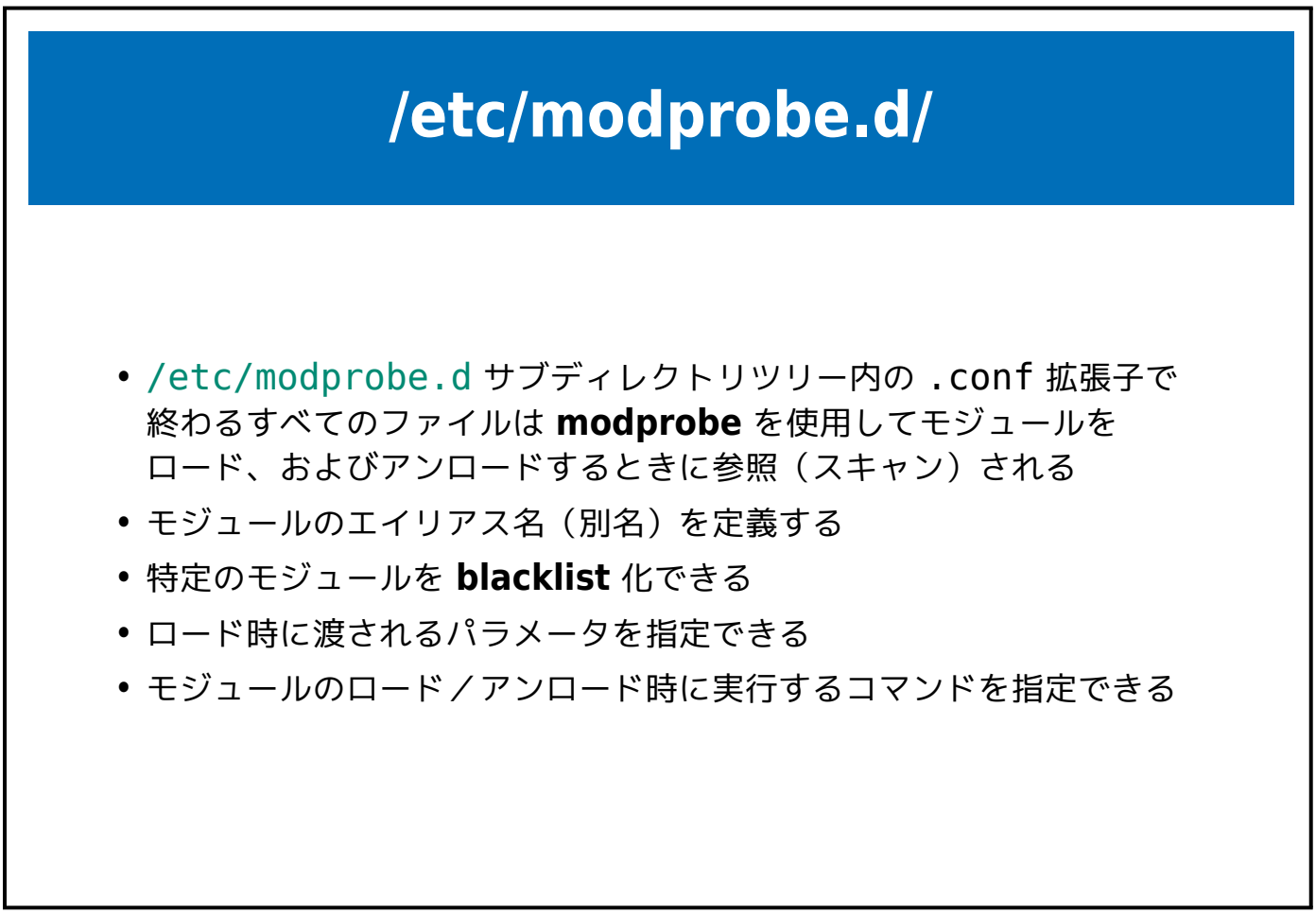
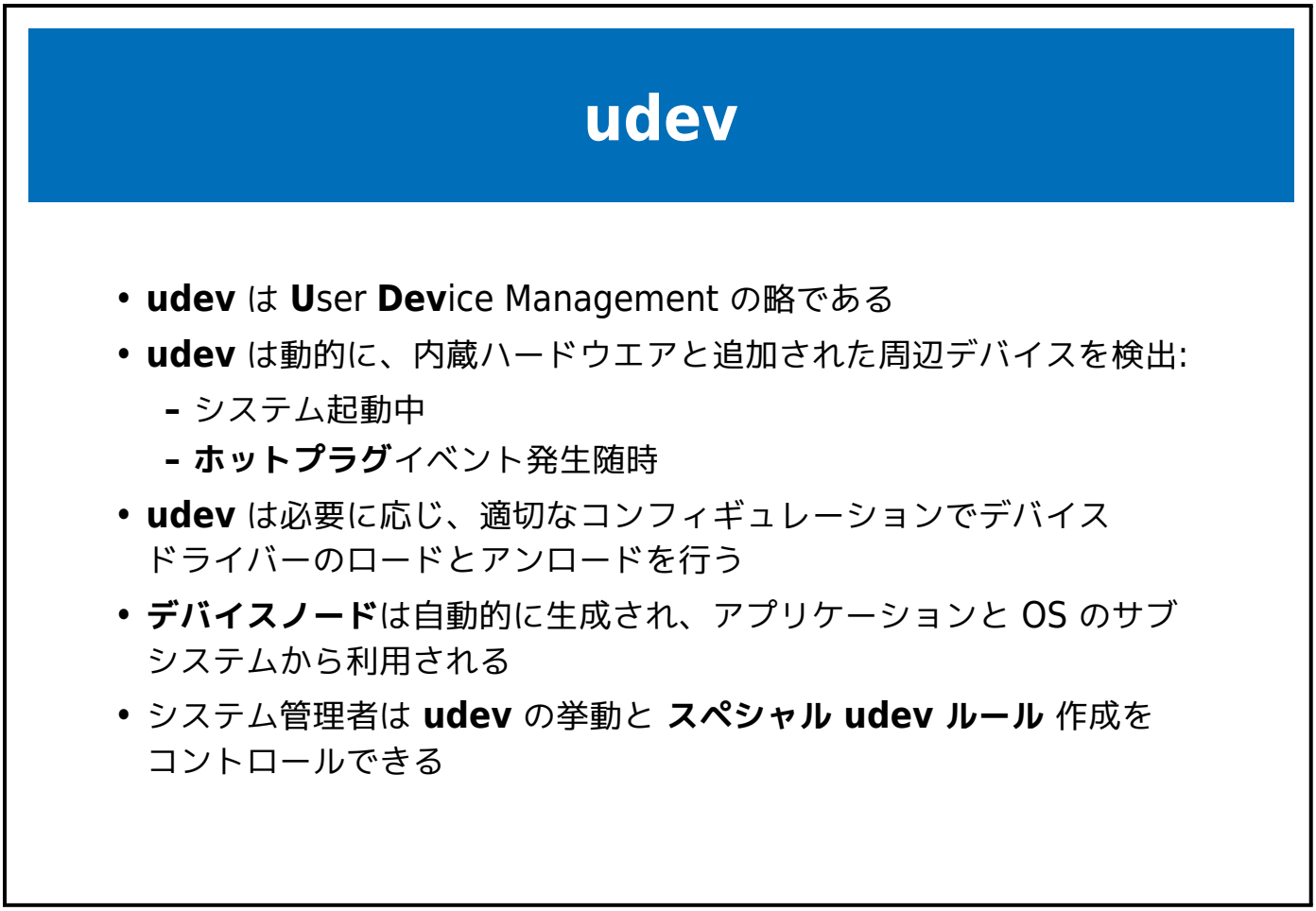
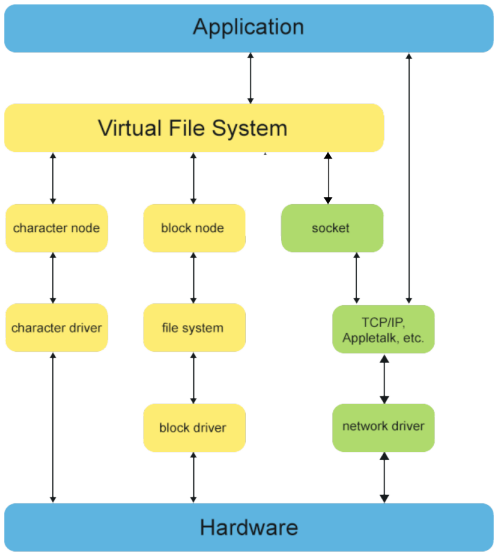
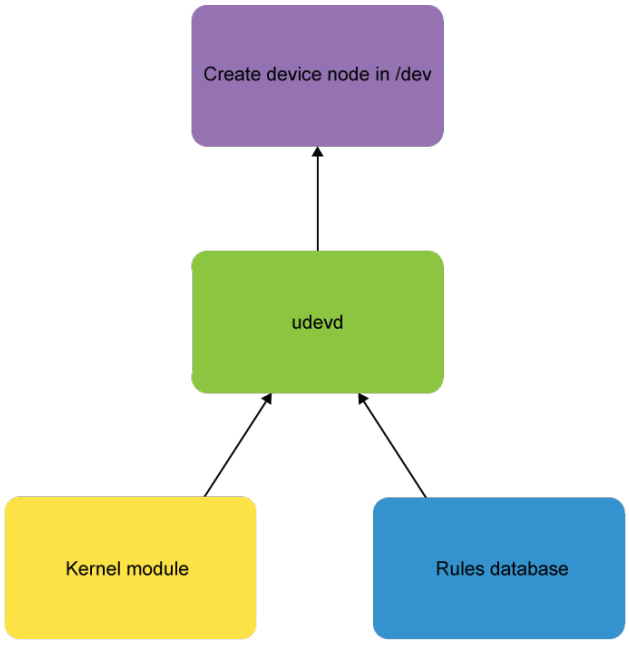
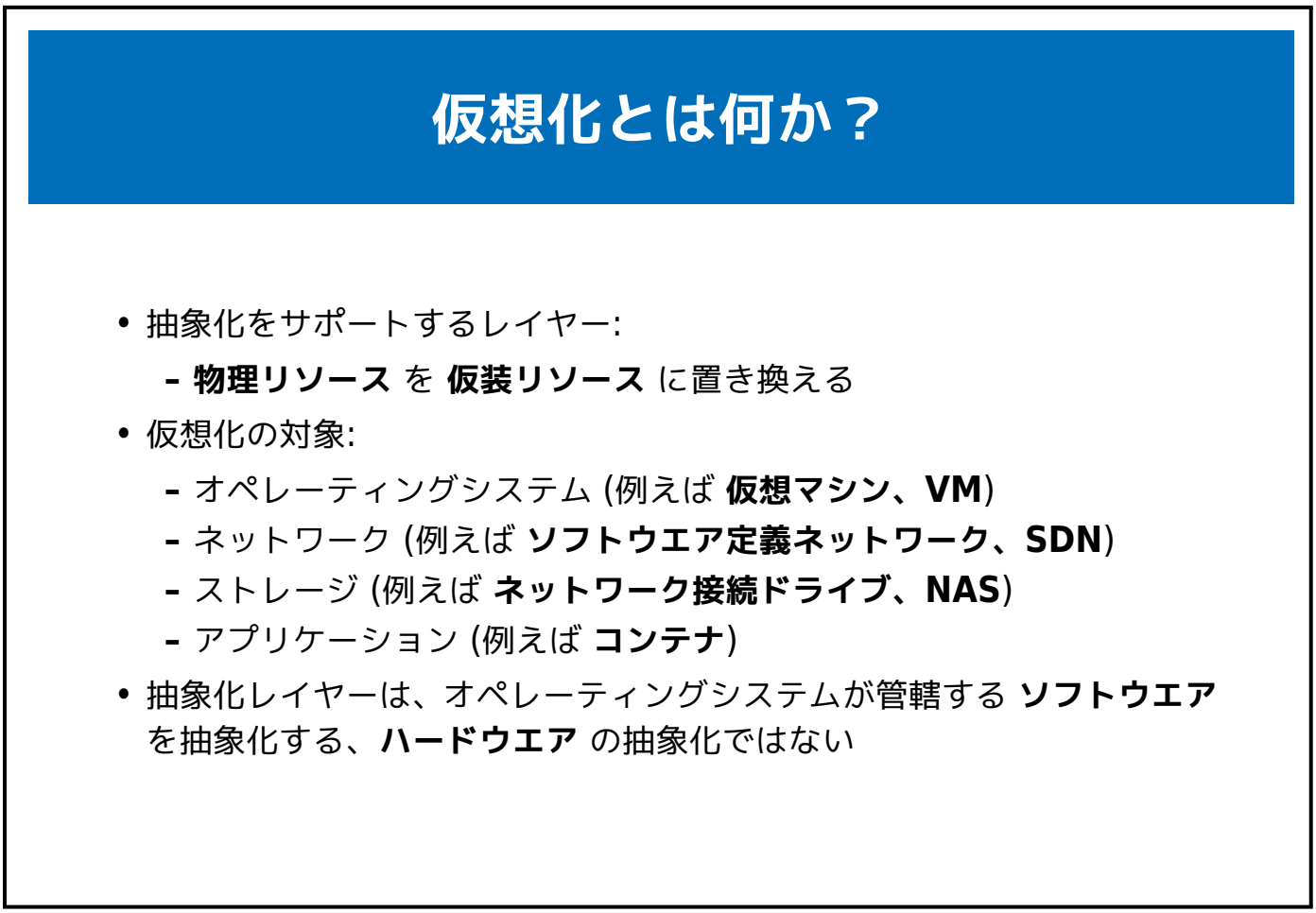
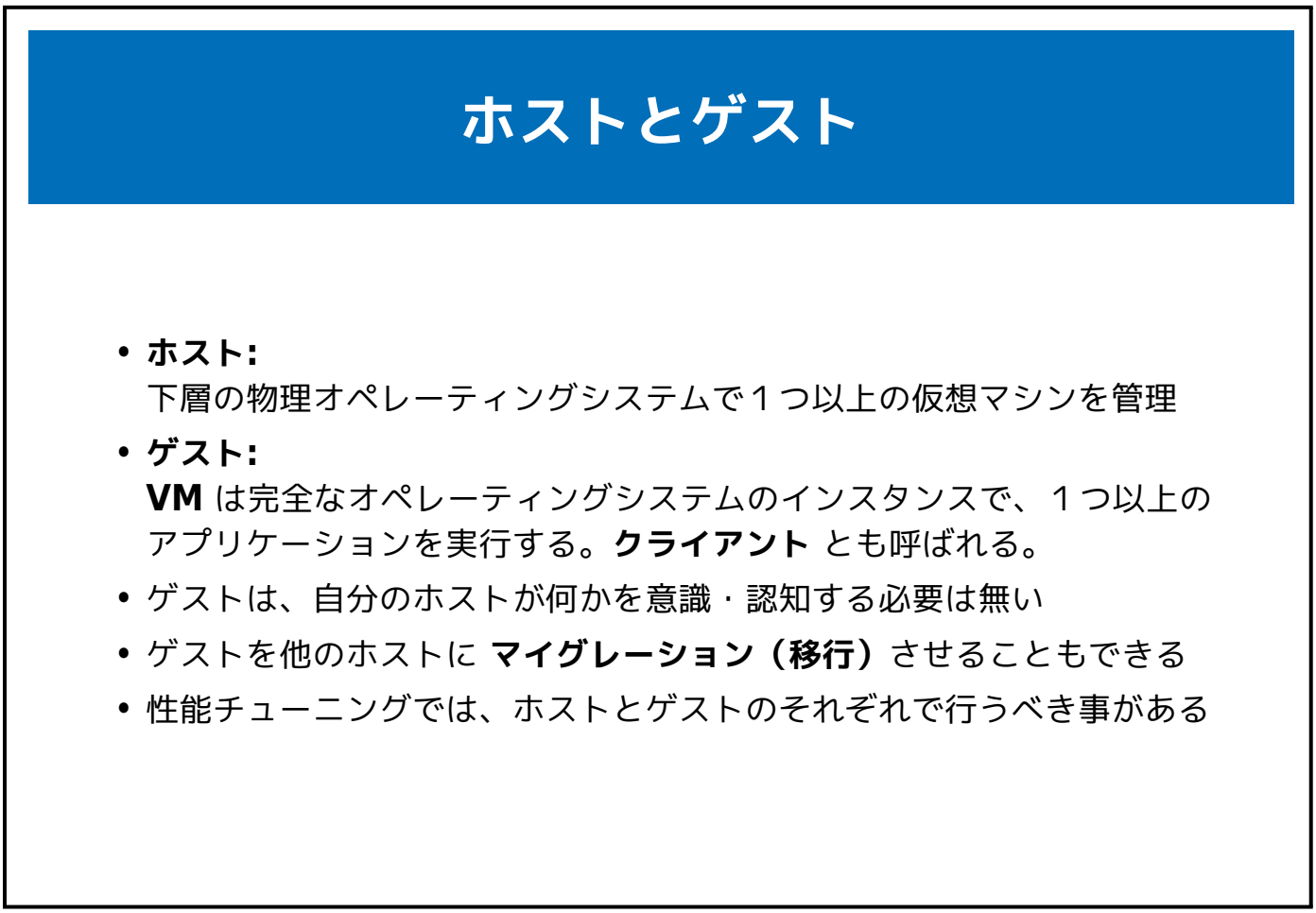
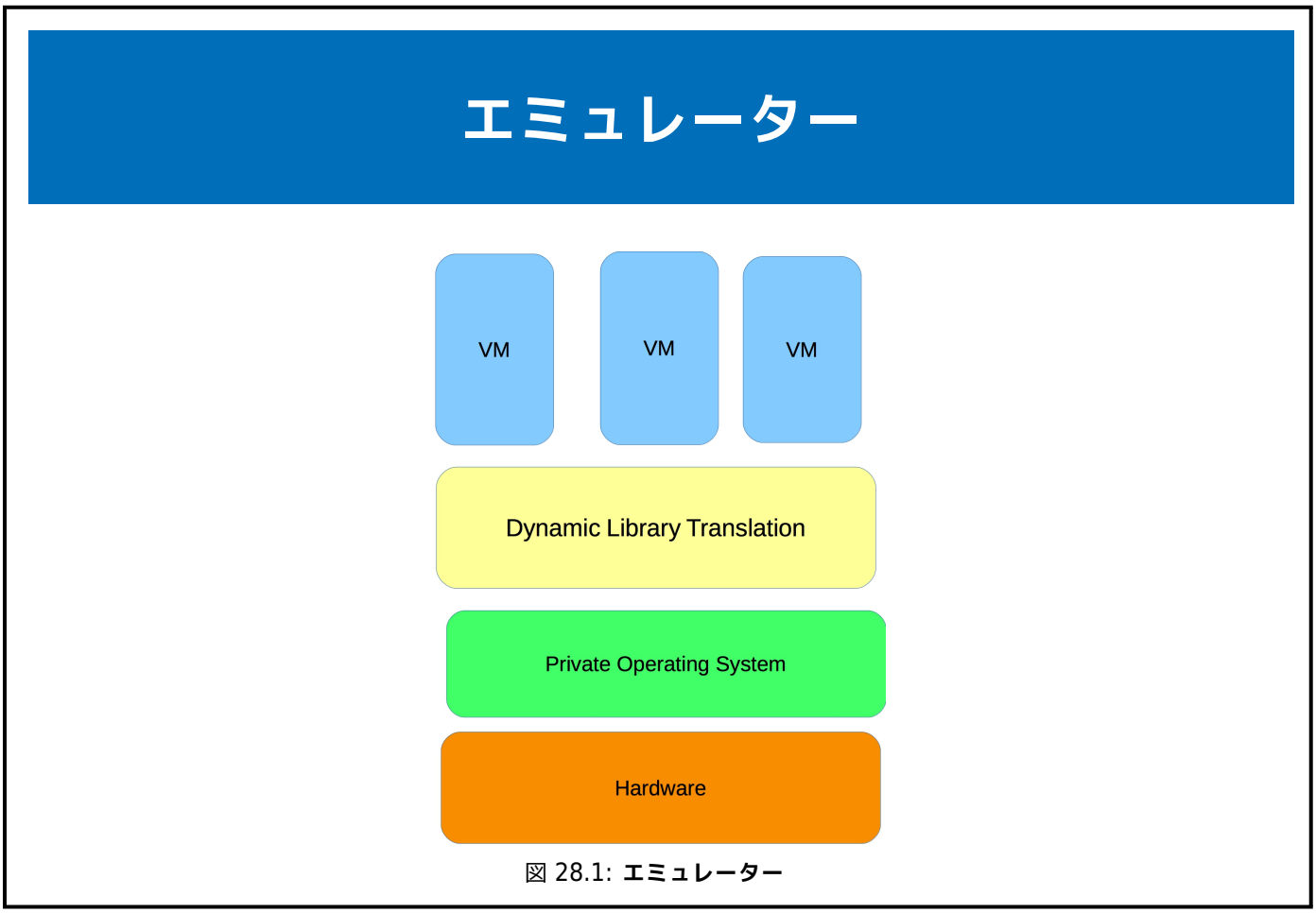
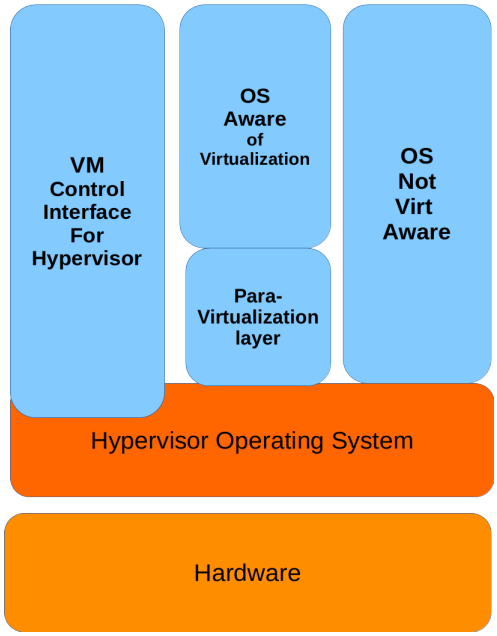
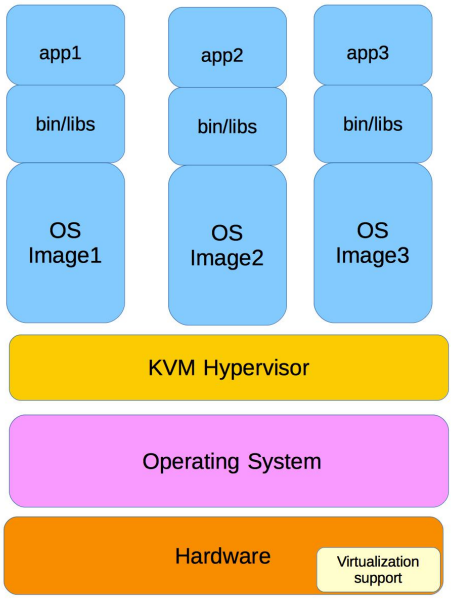
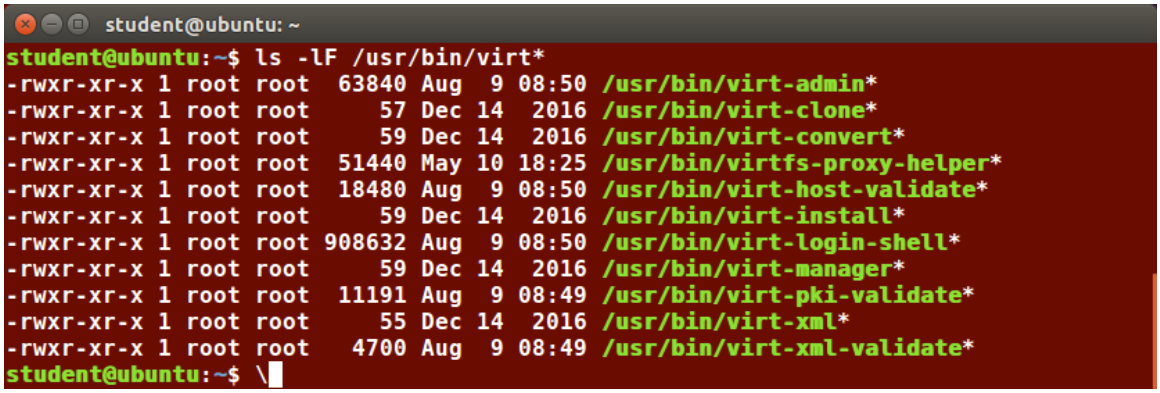
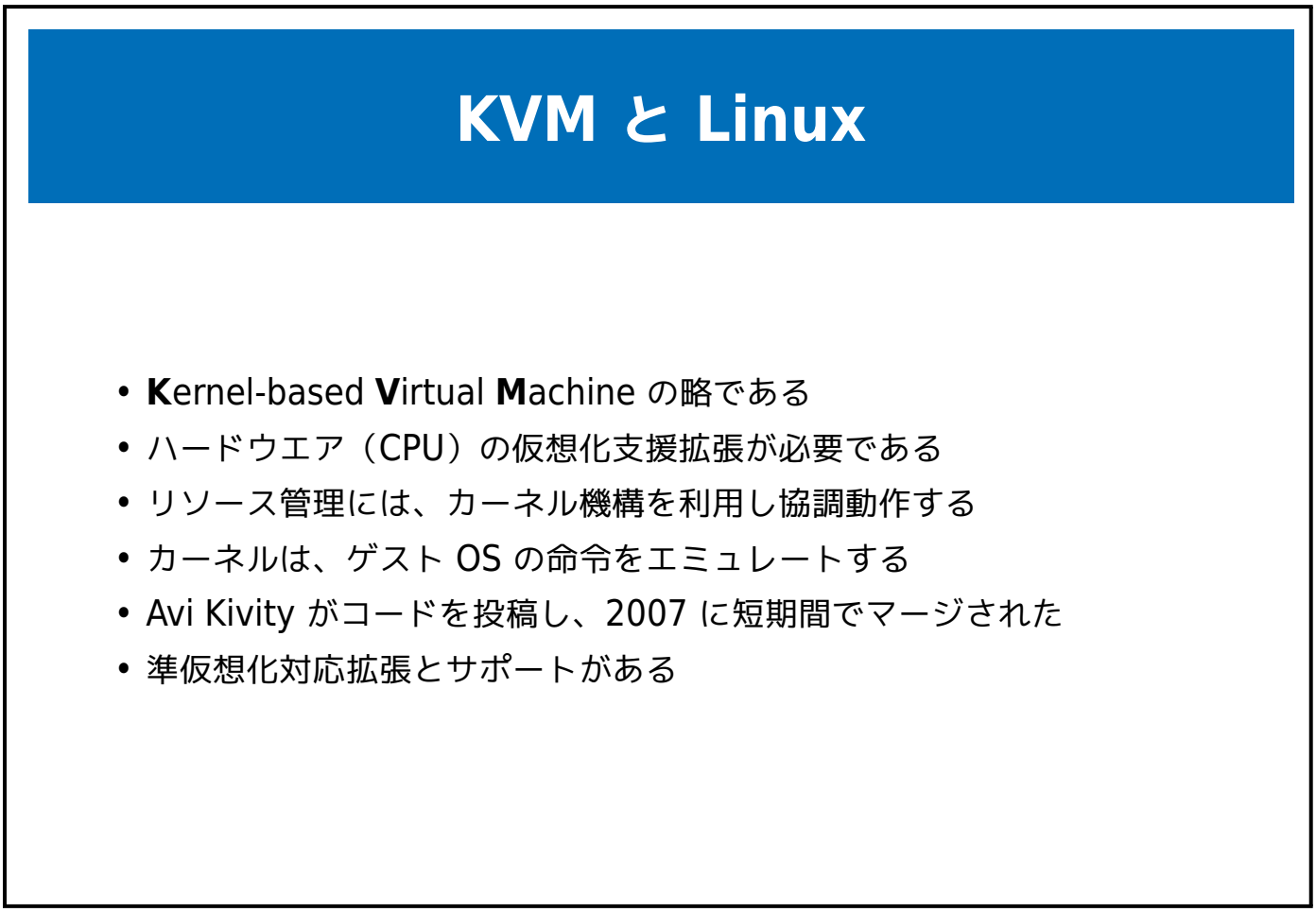
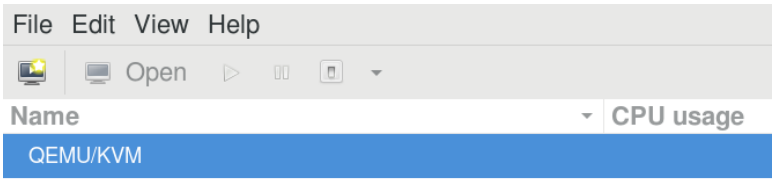
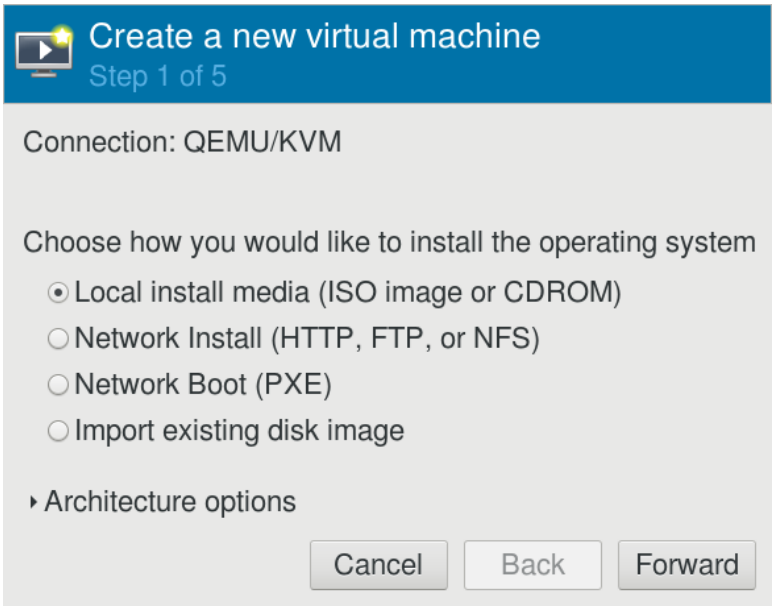
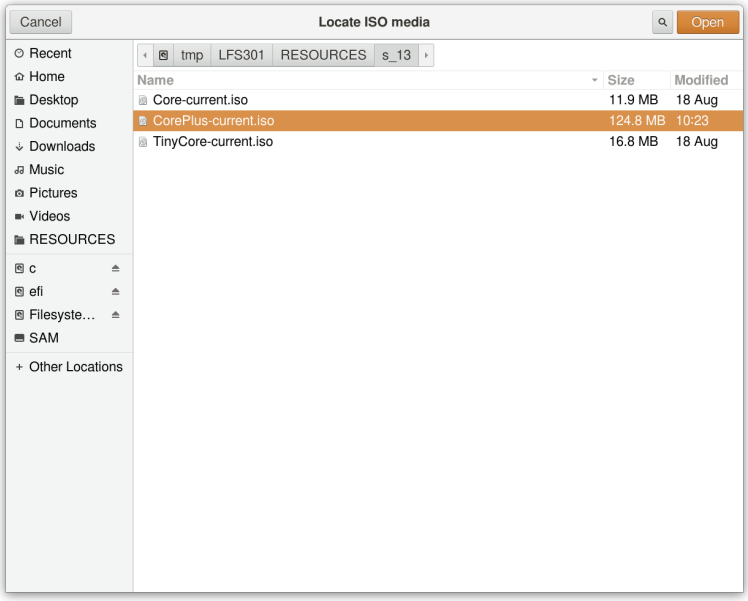
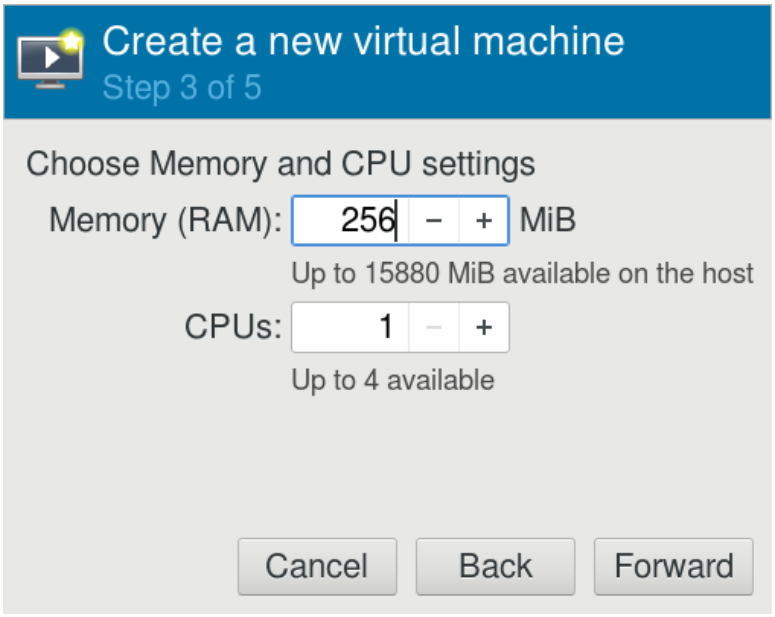
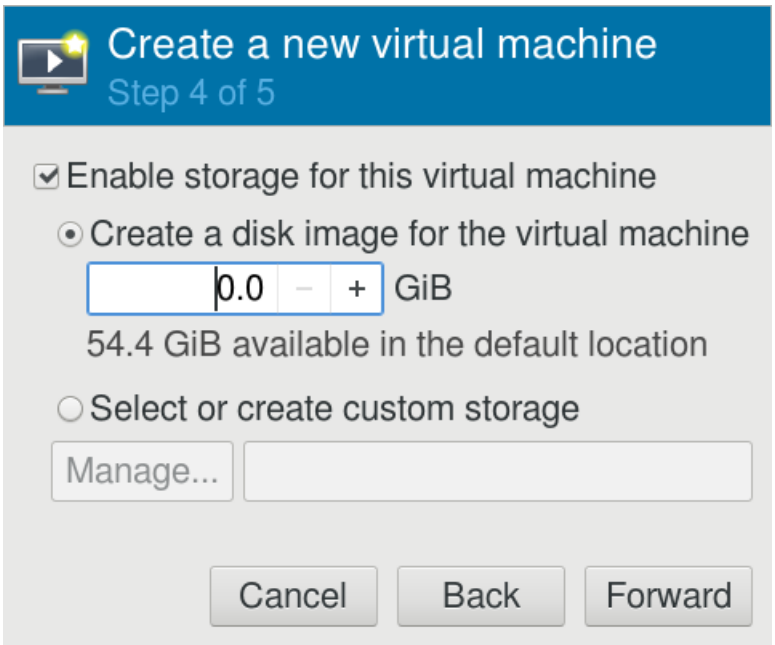
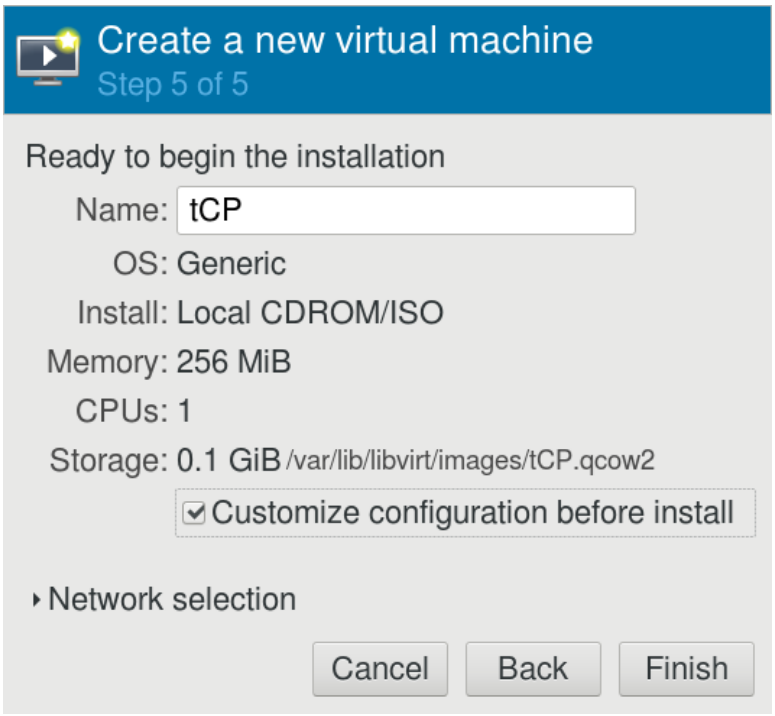
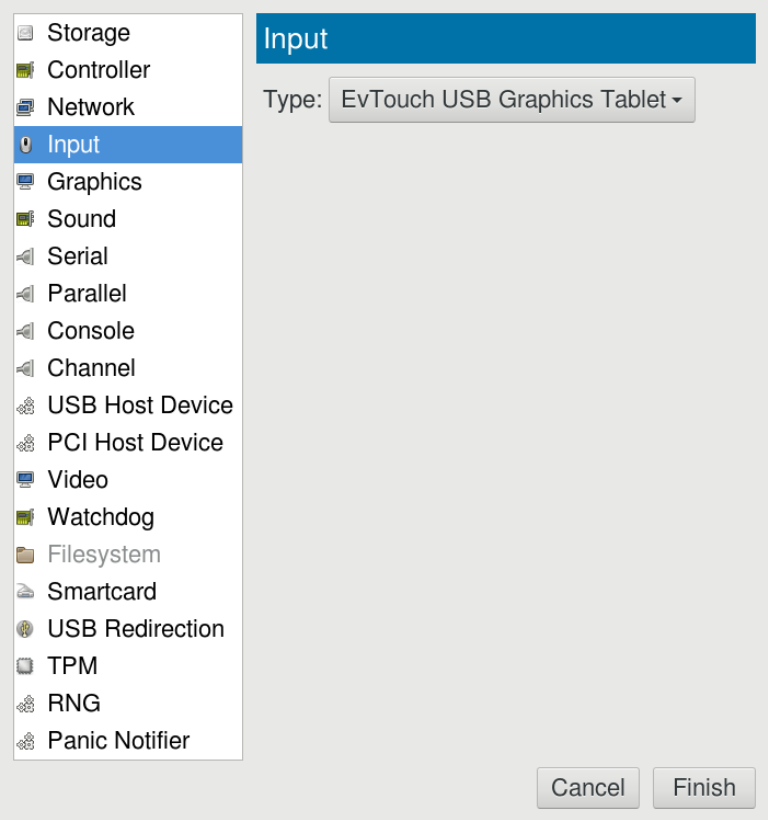
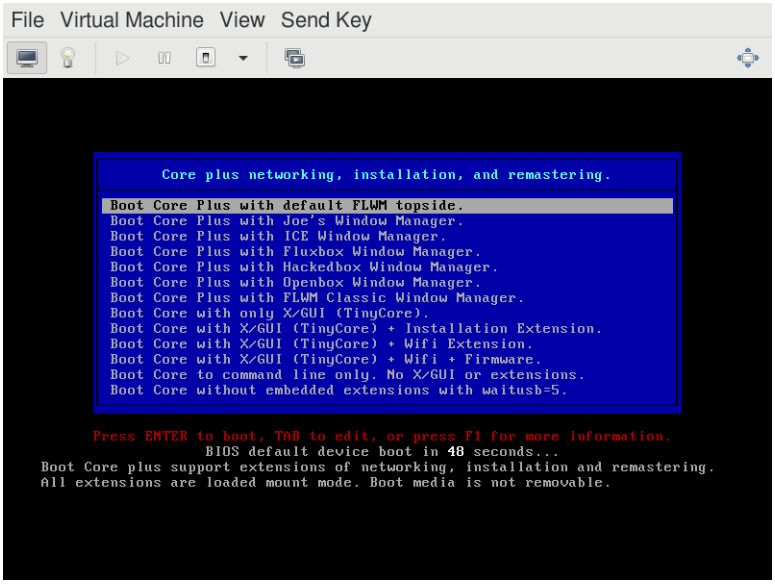
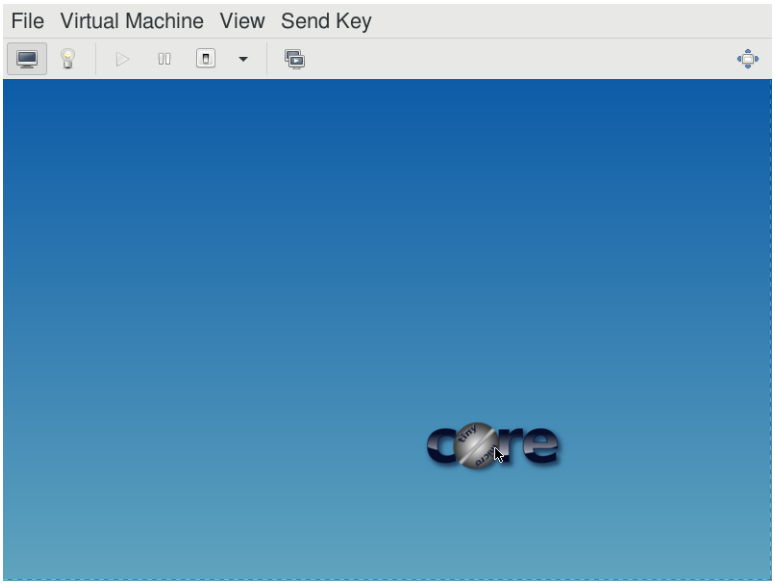
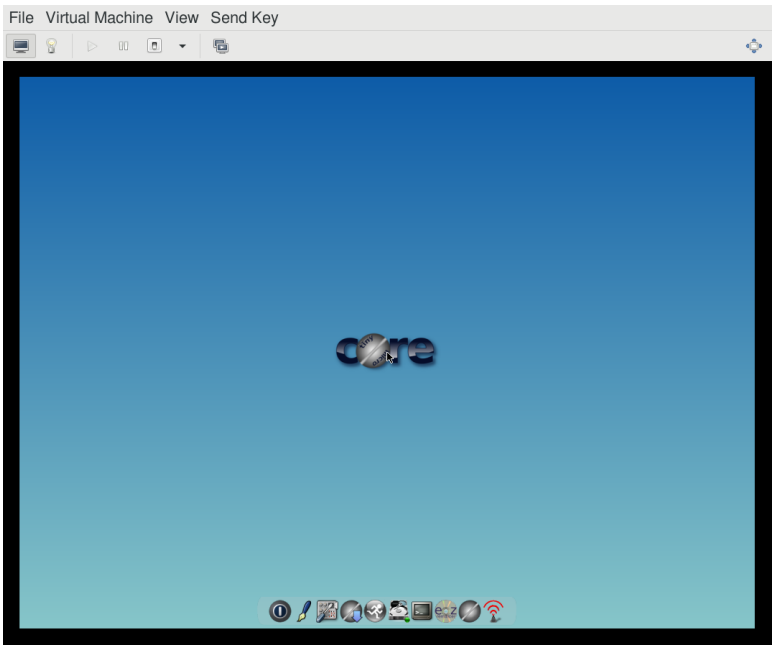
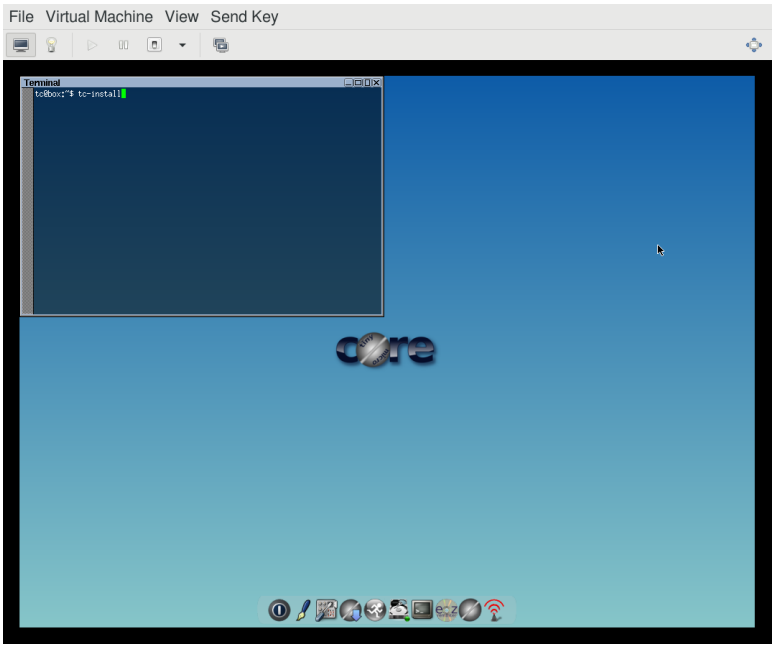
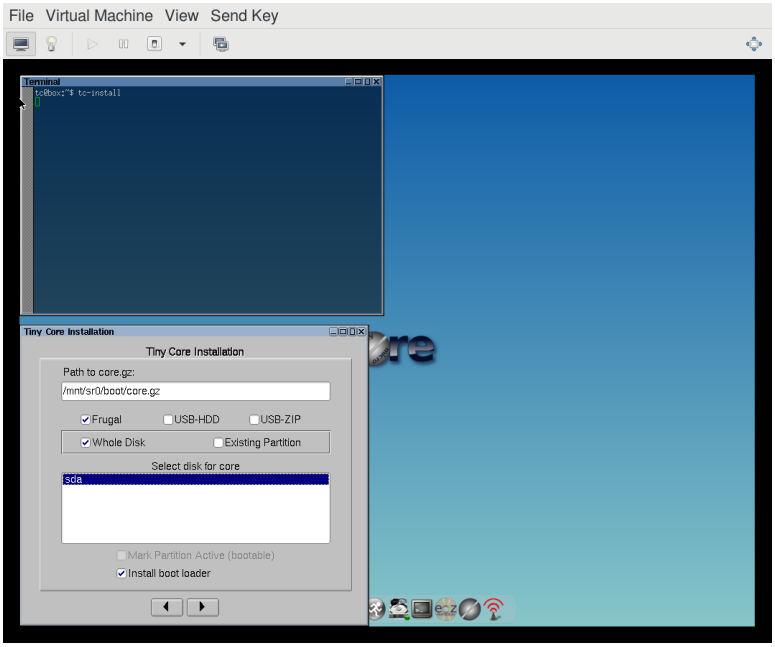
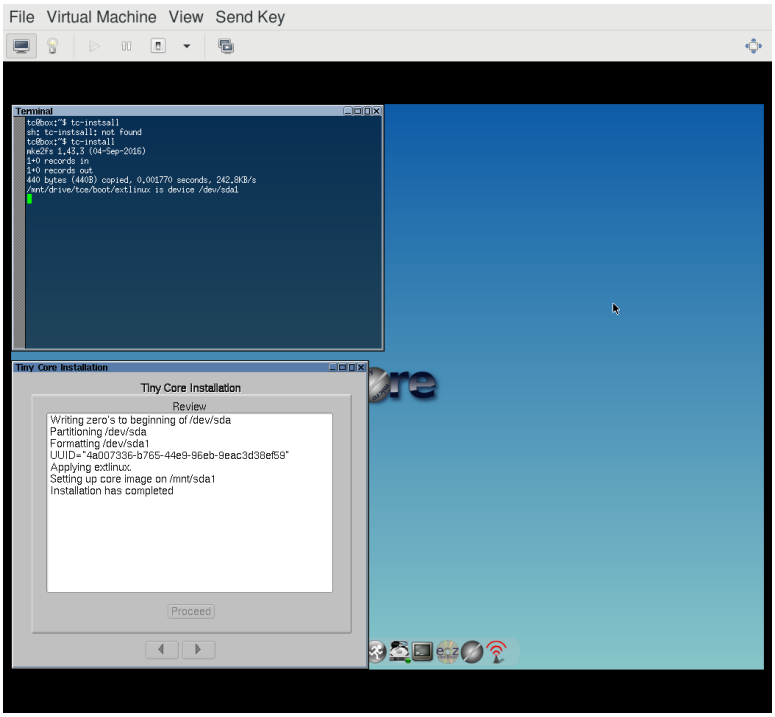
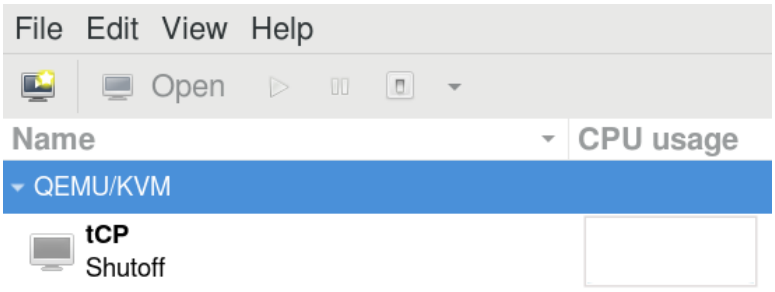
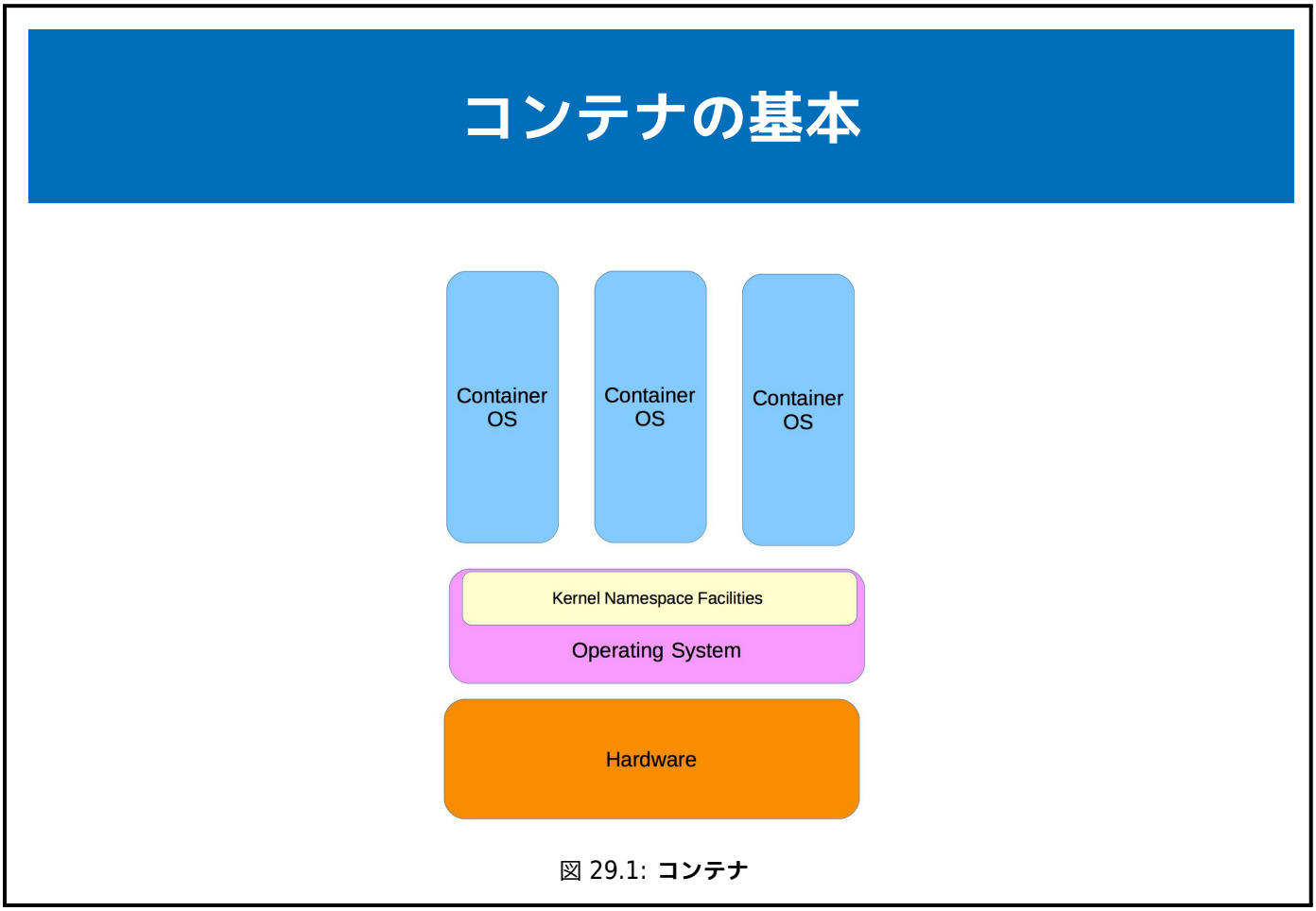
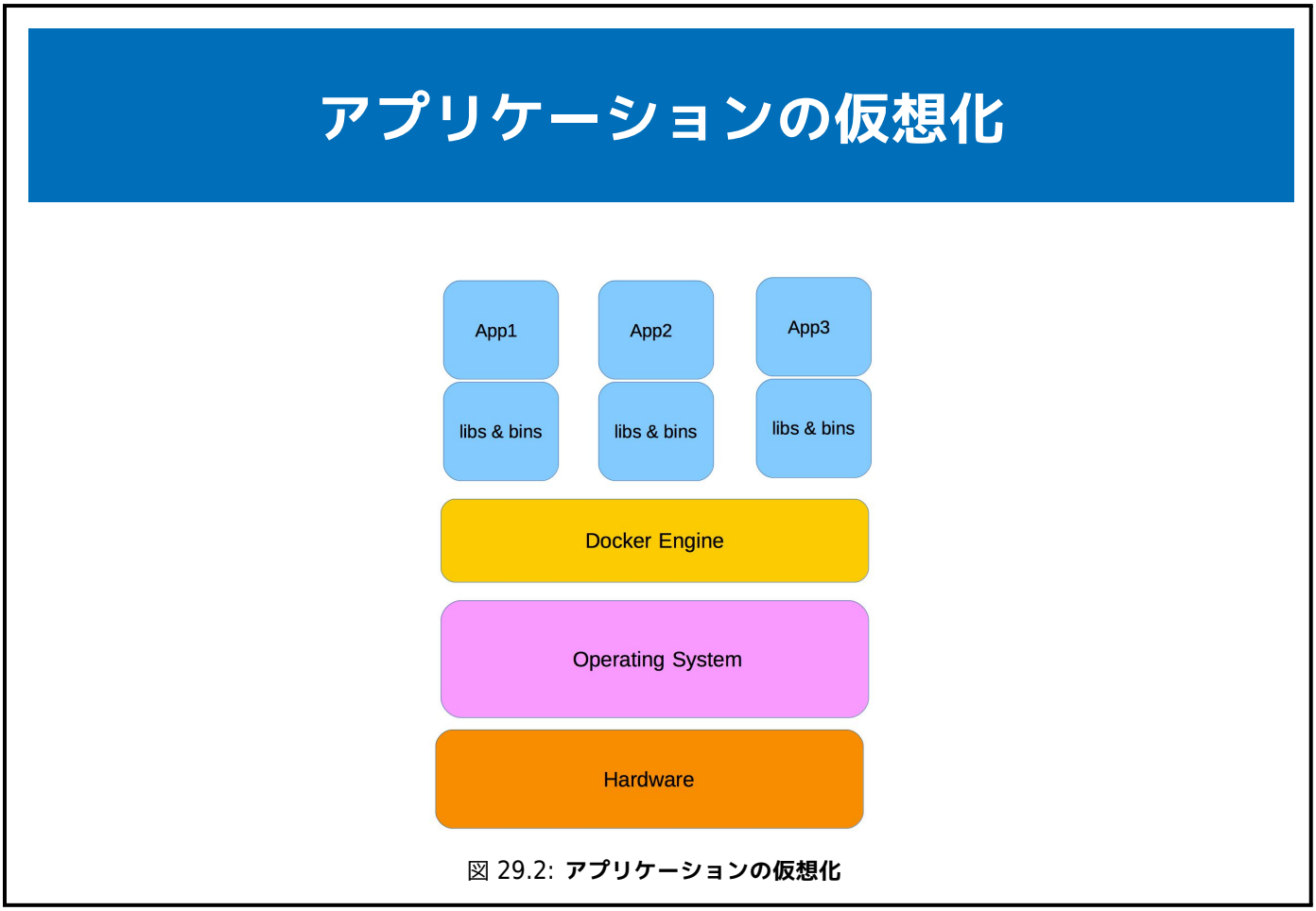
3. ```
   $ sudo sysctl kernel.pid_max=24000
   $ echo 24000 >  /proc/sys/kernel/pid_max # This must be done as root
   $ cat /proc/sys/kernel/pid_max
   ```

4. ```
   $ cat &
   ```

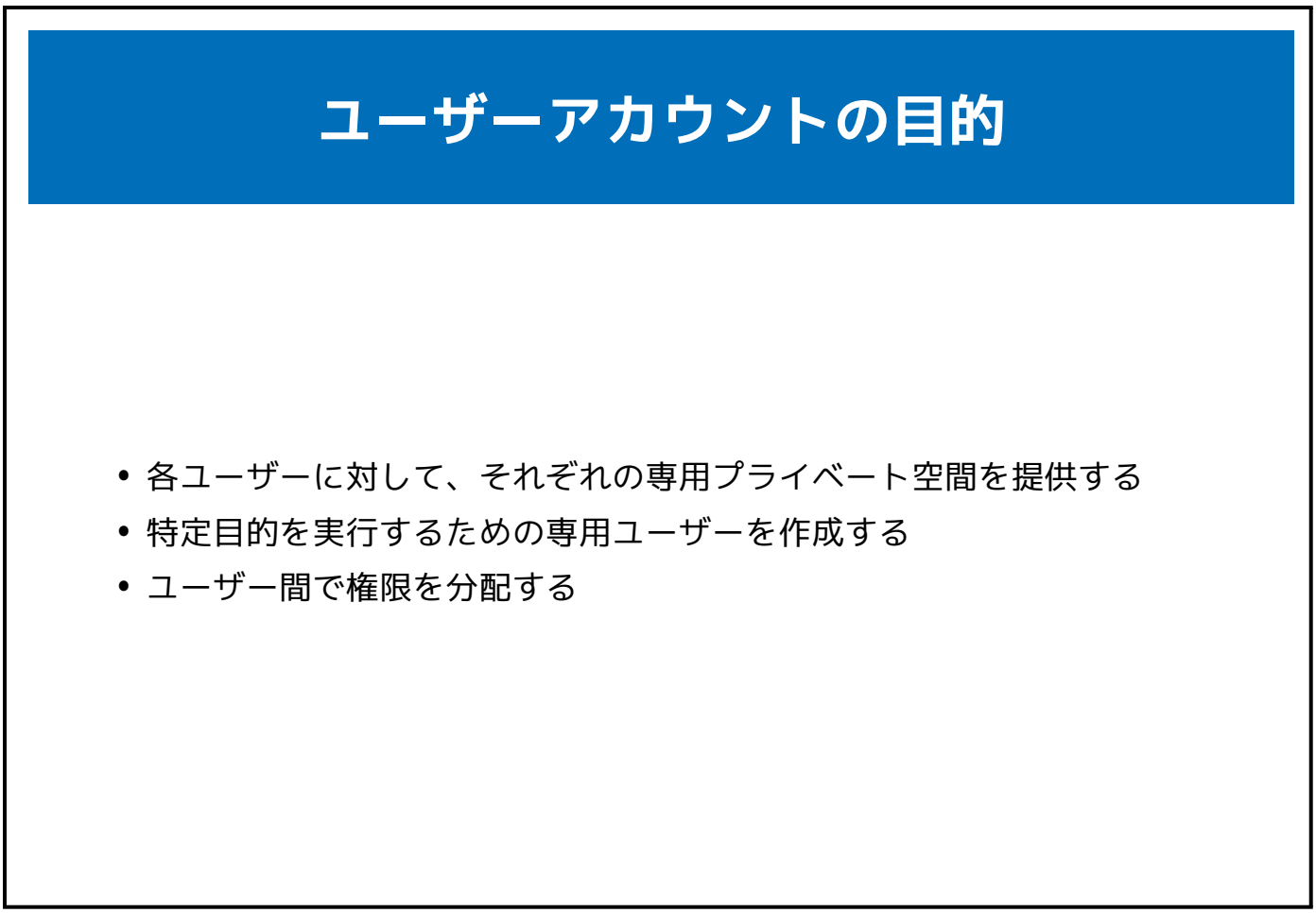
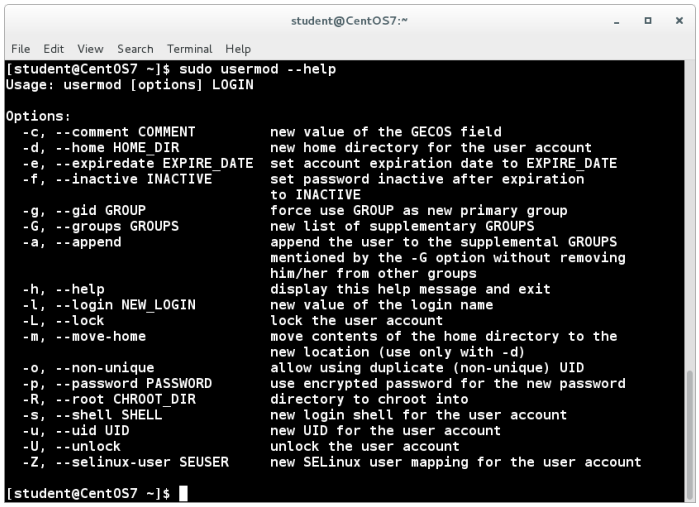
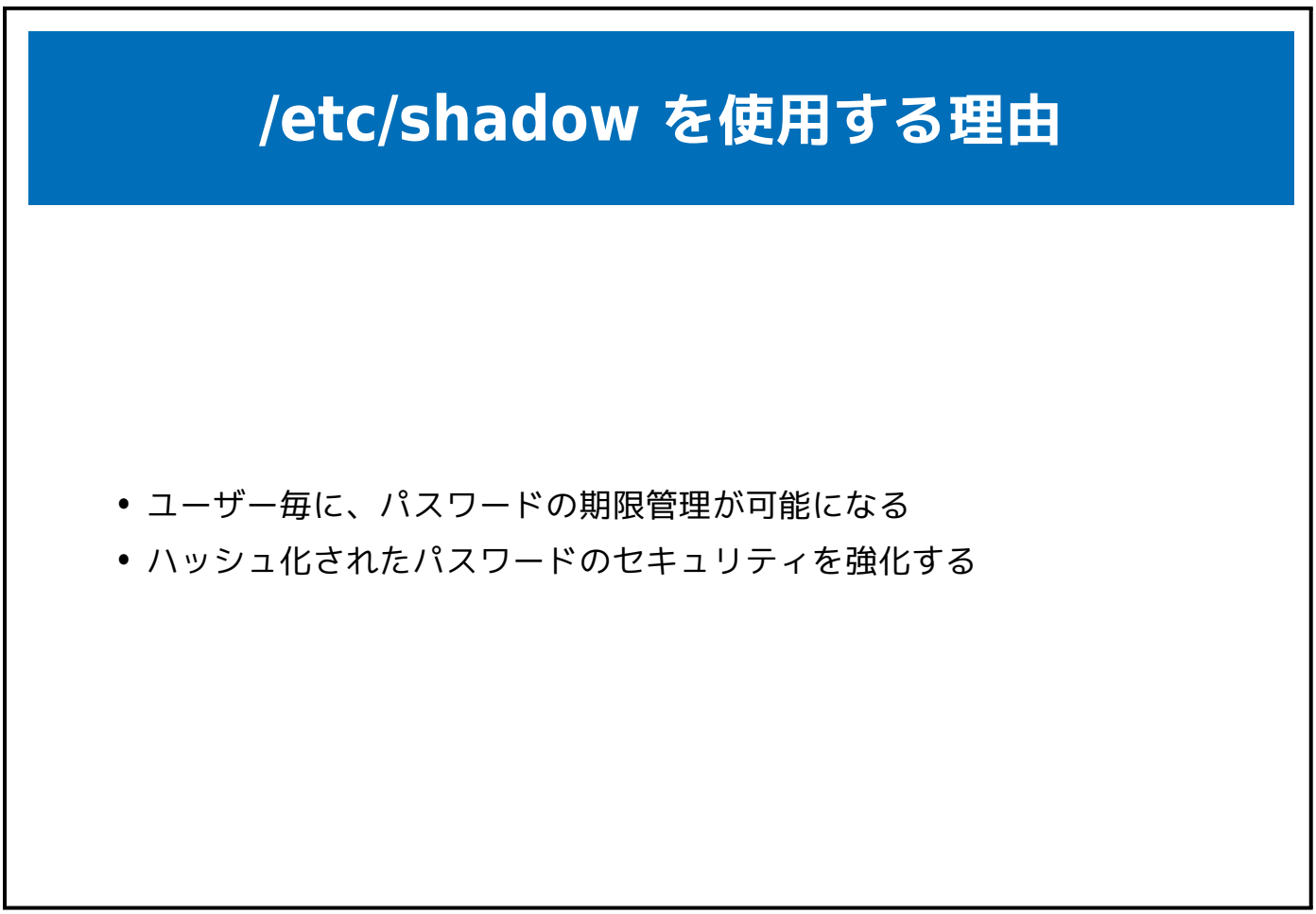
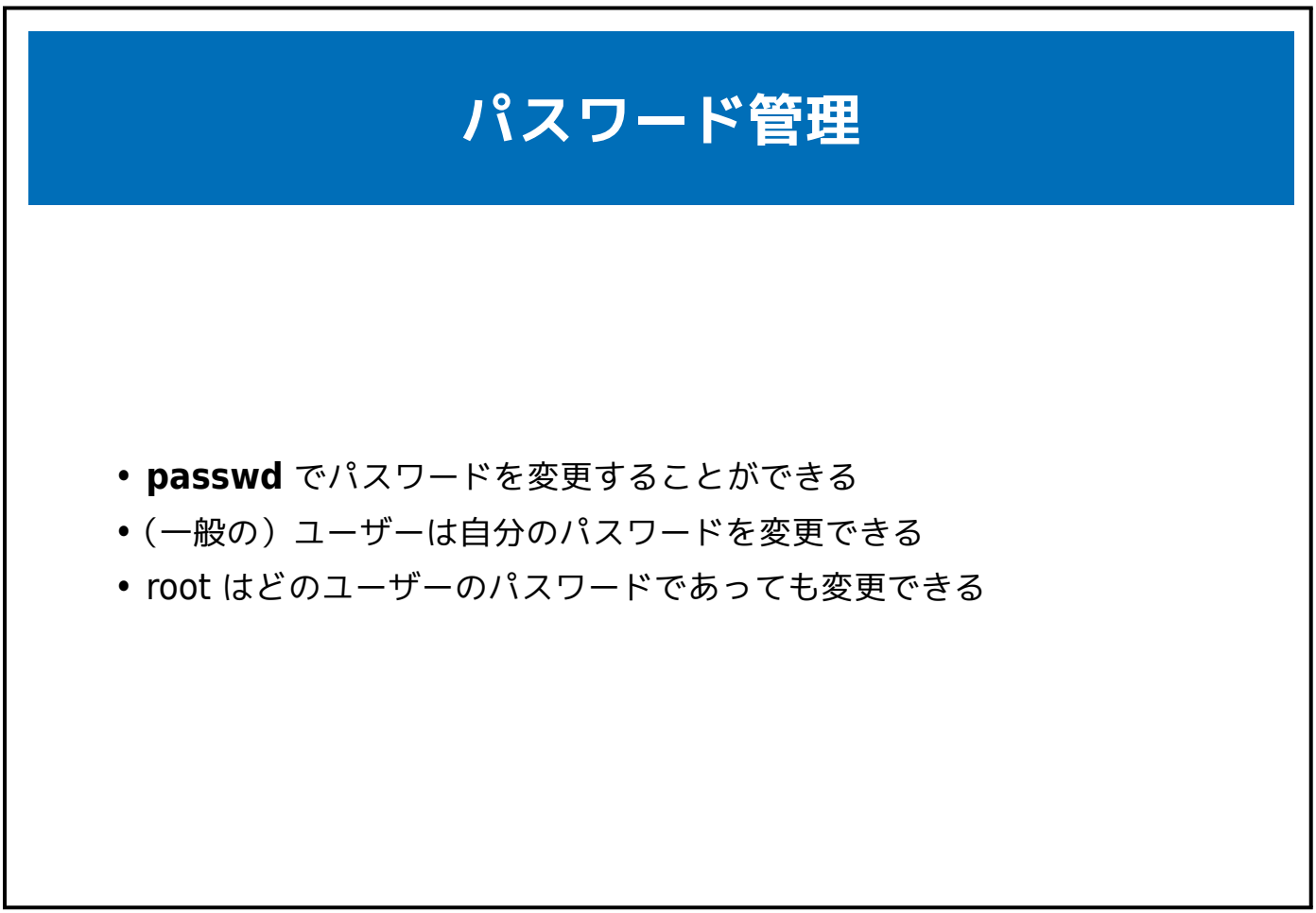
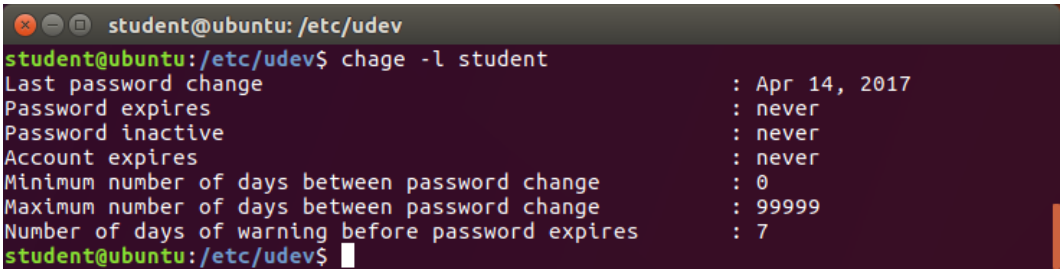
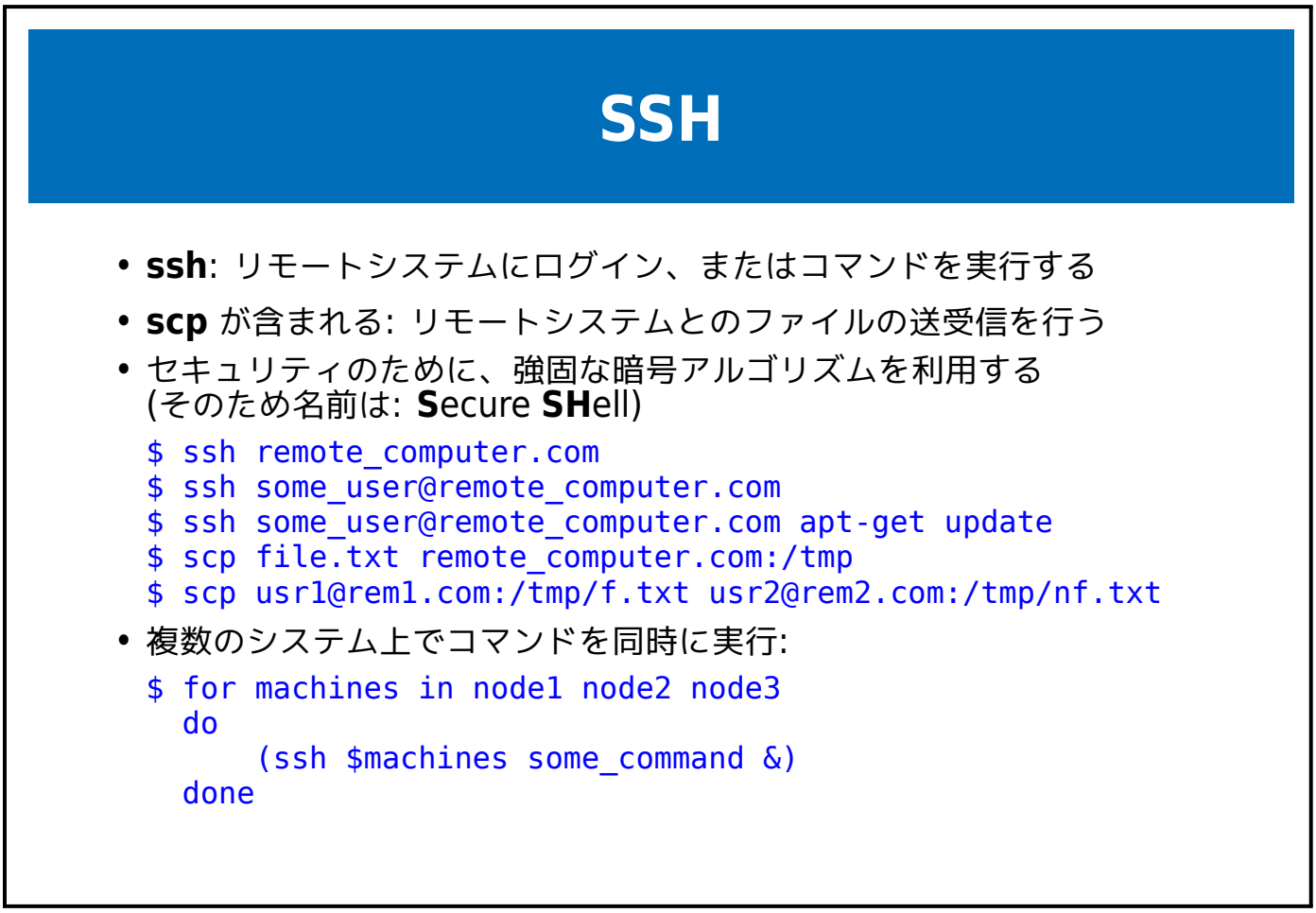
   ```
   [2] 311
   ```

   ```
   $ kill -9 311
   ```

注目

システム開始時に PID が最低値ではなく 300 から始まっていることに注意してください。実際、新しいプロセスへの PID 割り当ては些細なことではありません。すでに PID がターンオーバーしているかもしれませんし、カーネルは PID がすでに使用されていないことを確認して新プロセスに PID を割り当てています。**Linux** カーネルは、システム上のプロセス数にとらわれない効率的な方法でこれらを実行しています。
   ```

章 26

カーネルモジュール

26.1 カーネルモジュール



Linux カーネルは広範囲に **モジュール** を使用します。**モジュール** には重要なソフトウェアが含まれ、システム起動後であっても、必要に応じてロードおよびアンロードができます。

システム内や周辺機器として接続された、ハードウエアを制御するための **デバイスドライバ** の多くは、モジュールとして組み込まれています。ネットワークプロトコルの制御、さまざまなファイルシステムサポート、その他さまざまな目的のためにもモジュールが利用されます。

モジュールをロードする時には、モジュールの挙動を制御するパラメータを指定することが可能です。パラメータを変更することによって、条件や要件の変化に対して極めてフレキシブルかつ迅速に適合できます。

lsmod でモジュール一覧を表示

```
c8:/tmp>lsmod
Module                  Size  Used by
vmnet                  57344  13
vmmon                 126976  0
rfcomm                 90112  4
nft_fib_inet           16384  1
bnep                   28672  2
vmw_vsock_vmci_transport    32768  0
vmw_vmci               81920  1 vmw_vsock_vmci_transport
kvm_intel             270336  0
kvm                   958464  1 kvm_intel
irqbypass              16384  1 kvm
ext4                  905216  5
mbcache                16384  1 ext4
jbd2                  147456  1 ext4
e1000e                290816  0
c8:/tmp>
```

図 26.1: **lsmod の利用**

ほとんどのカーネル機能は、常に使用される可能性がある機能も含め、モジュールとして構成できます。この柔軟性によって、開発やデバッグ中のテストでシステムを再起動する必要がが殆どなくなるので、新機能の開発が効率化されます。

他のオペレーティングシステムにもモジュールのような方法はありますが、**Linux** は他のオペレーティングシステムよりもはるかに積極的にモジュールを活用しています。

26.2 モジュール ユーティリティ

モジュール ユーティリティ

- **lsmod** ロードされたモジュールを一覧表示する
- **insmod** モジュールを直接ロードする
- **rmmod** モジュールを直接削除する
- **modprobe** 依存関係と場所の情報を持つ、事前に構築されたモジュールデータベースを使用して、モジュールをロードまたはアンロードする
- **depmod** モジュールの依存関係データベースを更新する
- **modinfo** モジュールに関する情報を表示する

modprobe を使用してモジュールをロードします:

```
$ modprobe e1000e
```

`modprobe -r` を使用してモジュールをアンロードまたは削除します:

```
$ modprobe -r e1000e
```

modprobe は、モジュール依存関係のデータベースの更新を要求します。
depmod を使用してファイル、`/lib/modules/$(uname -r)/modules.dep` を生成または更新します。
insmod を使用してモジュールを直接ロードします。(この時は、モジュールの完全修飾名を指定する必要があります):

```
$ insmod /lib/modules/$(uname -r)/kernel/drivers/net/ethernet/intel/e1000e.ko
```

rmmod を使用してモジュールを直接削除します:

```
$ rmmod e1000e
```

lsmod を使用して、ロードされたモジュールをリストします:

```
$ lsmod
```

modinfo を使用して、モジュールに関する情報(パラメータを含む)を表示します:

```
$ modinfo e1000e
```

モジュールに関する留意事項

- 以下の状況では、モジュールがアンロードできない:
  - アンロードしようとしたものが、他のモジュールに使われている
  - 実行中のプロセスが使っている
- 以下の状況では、モジュールがロードできない:
  - 別のカーネルバージョン向けにコンパイルされている、または コンパイルオプションが異なる
  - 現在実行中のモジュールと競合する
- 通常モジュールは **modprobe** でロードする、**insmod** を使わない
- 許容できないオープンソースライセンスをもったモジュールがロードされた時には、カーネルは **tainted** とマークする

モジュールをロードおよびアンロードする際に、留意すべき重要な点がいくつかあります:

- １つ以上の他の **モジュール** が参照しているものはアンロードできません。これらは **lsmod** で確認できます。
- １つ以上の **プロセス** で使用されているモジュールもアンロードできません。これも **lsmod** で確認できます。ただし、ネットワークデバイスドライバモジュールなど、参照カウントを記録しないモジュールがあります。このため、ネットワークスタック全体の大部分をシャットダウン再起動することなく、特定のモジュールだけを一時的に置き換えるのは非常に困難です。
- **modprobe** でモジュールをロードした場合、先にロードする必要がある他のモジュールが自動的にロードされます。
- **modprobe -r** でモジュールをアンロードする場合、そのモジュールが使用していた他のモジュールも、別のモジュールがそれを参照していなければシステムが自動的にアンロードします。

26.3 modinfo

modinfo の使用例



図 26.2: **modinfo**

上のスクリーンショットのように、**modinfo** はカーネルモジュールに関する情報を表示します。(現在ロードされているかどうかに関係なく) これには、バージョン、ファイル名、デバイスドライバモジュールが処理できるハードウェアデバイス、読み込み時に指定可能なパラメータに関する情報が含まれます。

さらに、多くのモジュールに関する多くの情報が `/sys` 疑似ファイルシステムのディレクトリツリーで確認できます。上の例では `/sys/module/e1000` を見ていて、いくつかのパラメータは `/sys/module/e1000/parameters` で読み書きできます。次に、それらを設定する方法を示します。

多くのモジュールは、以下に示すようにロード時にパラメータ値の指定が可能です:

```
$ sudo /sbin/insmod <pathto>/e1000e.ko debug=2 copybreak=256
```

または、モジュールがファイルシステムが想定する場所に置かれている場合は、より簡単に:

```
$ sudo /sbin/modprobe e1000e debug=2 copybreak=256
```

26.4 モジュールのコンフィギュレーション



/etc/modprobe.d 内のファイルは、**modprobe** を使用してロードするときに使ういくつかのパラメータを直接設定します。これらのパラメータには、モジュール名の**エイリアス（別名）**と自動的に提供されるオプションが含まれます。特定のモジュールを **blacklist** ファイルに登録して、それらがロードされないように指定できます。

設定はモジュールのロードまたはアンロード時に適用され、必要に応じて構成を変更できます。

/etc/modprobe.d 内のファイル形式はシンプルです。１行に１つのコマンドを書き、空行と # で始まる行は無視されます。（コメントの追加に便利です）行の終わりにバックスラッシュがあると次の行に継続するため（＝途中で改行できるので）、ファイルが少し見やすくなります。

26.5 演習

デモ教材ビデオ

using_kernel_modules_demo.mp4

課題 26.1: カーネルモジュール

1. 現在ロードされているモジュール一覧を表示します。
2. モジュールをロード、アンロードします。

   ディストリビューション標準のカーネルを実行してる場合には、モジュールは /lib/modules/<kernel-version>/kernel/drivers/net ディレクトリに見つけることができます。(ディストリビューションのカーネルには、システムが必要とする可能性のある、全てのデバイス、ファイルシステム、ネットワークプロトコルなどのドライバが含まれています。) しかし、独自のカーネルを実行している場合は、コンパイル済のモジュールがロードされない状態で多数存在しているケースは希でしょう。

   一般的なギガビットイーサネットドライバの `e1000.ko` または、`e1000e.ko` を利用します。尚、この 2 つが同時にロードされることはありえません。
3. ロードされたカーネルモジュール一覧を再表示し、本当にロードされたか確認します。
4. ロードされたモジュールを削除します。
5. 一覧を再表示し、モジュールが削除されたか確認します。

解 26.1

1. `$ lsmod`
2. これから `e1000e` には、自分が利用するモジュール名を入れてください。これらの方法は問題なく実行できますが、もちろん 2 番目の方法が楽です。

   ```
   $ sudo insmod /lib/modules/$(uname -r)/kernel/drivers/net/ethernet/intel/e1000e.ko
   $ sudo /sbin/modprobe e1000e
   ```
3. `$ lsmod | grep e1000e`
4. あらためて、どちらの方法も問題なく動きます。

   ```
   $ sudo rmmod e1000e
   $ sudo modprobe -r e1000e
   ```
5. `$ lsmod | grep e1000e`

章 27

デバイスと udev

27.1 udev とデバイス管理



udev アクションには以下が含まれます:

- デバイス名の命名
- デバイスファイルとシンボリックリンクの生成
- ファイル属性の設定
- 必要なアクションの実行

システムにデバイスが追加ないし削除されると、**udev** はカーネルからメッセージを受け取ります。それをもとに `/etc/udev/rules.d` ディレクトリにあるルールファイル（.files）を構文解析して、それらのデバイスが追加ないし削除された時のルールがあるかを確認します。

これらのルールは完全にカスタマイズ可能で、デバイスファイル名、デバイスファイルの生成、シンボリックリンクの生成、デバイスファイルの属性設定（ユーザー、グループ所有者の設定を含む）、更に起動されるアクション（プログラムの実行を指定できる）などを設定できます。

27.2 デバイスノード

デバイスノードはハードウエアとソフトウエアを結びつける



図 27.1: デバイスノード

デバイスノード

- **デバイスノード**は /dev ディレクトリに配置される
- キャラクターデバイスとブロックデバイスはデバイスノードを持ち、ネットワークデバイスは持たない
- プログラムはデバイスノードを介して通常の I/O でデバイスと通信する
- 一つのデバイスが複数のデバイスノードを使うことがある
- デバイスノードの生成方法:

  ```
  $ sudo mknod [-m mode] /dev/name <type> <major> <minor>
  ```

  例えば

  ```
  $ sudo mknod -m 666 /dev/mycdrv c 254 1
  ```

```
$ ls -l /dev
```

```
total 0
crw------- 1 coop audio      14,   4 Jul  9 01:54 audio
crw------- 1 root root       10,  62 Jul  9 01:54 autofs
lrwxrwxrwx 1 root root            4 Jul  9 01:54 cdrom -> scd0
lrwxrwxrwx 1 root root            4 Jul  9 01:54 cdrw -> scd0
crw------- 1 coop root        5,   1 Jul  9 06:54 console
....
lrwxrwxrwx 1 root root            4 Jul  9 01:54 dvd -> scd0
lrwxrwxrwx 1 root root            4 Jul  9 01:54 dvdwriter -> scd0
....
brw-r----- 1 root disk        8,   0 Jul  9 01:53 sda
brw-r----- 1 root disk        8,   1 Jul  9 01:53 sda1
brw-r----- 1 root disk        8,   2 Jul  9 06:54 sda2
....
brw-r----- 1 root disk        8,  16 Jul  9 01:53 sdb
brw-r----- 1 root disk        8,  17 Jul  9 01:53 sdb1
brw-r----- 1 root disk        8,  18 Jul  9 01:53 sdb2
....
crw-rw-rw- 1 root tty         5,   0 Jul  9 01:54 tty
crw-rw---- 1 root root        4,   0 Jul  9 14:54 tty0
crw------- 1 root root        4,   1 Jul  9 06:54 tty1
cr--r--r-- 1 root root        1,   9 Jul  9 01:53 urandom
....
crw-rw-rw- 1 root root        1,   5 Jul  9 01:54 zero
```

udev コンポーネント

- **libudev** ライブラリ
  デバイスに関する情報への
  アクセスを許可
- **udevd** または **systemd-udevd** デーモン
  /dev ディレクトリを管理
- **udevadm** ユーティリティ
  制御および診断



図 27.2: **udev ルール**

udev は **デーモン**（**udevd** または **systemd-udevd**）として実行され **netlink** ソケットを監視します。新しいデバイスが初期化されたり削除されると、**uevent** カーネル機構はソケットを介してメッセージを送信します。**udev** はそれを受信しルールに従って正しい名前とプロパティのデバイスノードを作成したり削除したりします。

udev は、以下の３つのコンポーネントで構成されます:

1. デバイスに関する情報へのアクセスを許可する **libudev** ライブラリ
2. /dev ディレクトリを管理する **udevd** デーモン
3. 制御および診断を行う **udevadm** ユーティリティ

最もクリーンな **udev** の使いかたは、プレーンなシステムを使うことです。最初のカーネル起動時には、空の /dev ディレクトリに必要な分だけ（＝実際に認識されたデバイスに対してのみ）デバイスノードが生成されます。この方法を使うには、暫定的なデバイスノードセットと **udev** インフラストラクチャが含まれた **initramfs** イメージを使ってブートする必要があります。

27.3 Rules

udev ルールファイル

- ルールファイルの場所とファイルの形式:

```
/etc/udev/rules.d/<rulename>.rules
/usr/lib/udev/rules.d/<rulename>.rules
```

- 例:

```
30-usb.rules
90-mycustom.rules
70-mouse.rules
60-persistent-storage.rules
```

１行に２つの情報を定義します:

- 最初の部分は == で示される１つ以上の一致するペアで構成します。これらは、デバイスの属性や特性もしくはその両方がここに指定した値に一致するかをみます。
- ２番目の部分はファイル名、グループの割り当て、さらにはファイルのパーミッションなど名前に値を割り当てる１つ以上の割り当てキーと値のペア（=KVS、キーバリューストア）で構成します。

一致するルールが見つからない場合、デフォルトのデバイスノード名とその他の属性が使用されます。

```
$ cat /etc/udev/rules.d/99-fitbit.rules
SUBSYSTEM=="usb", ATTR{idVendor}=="2687", ATTR{idProduct}=="fb01", SYMLINK+="fitbit", MODE="0666"
```

```
$ cat /etc/udev/rules.d/60-vboxdrv.rules
KERNEL=="vboxdrv", NAME="vboxdrv", OWNER="root", GROUP="vboxusers", MODE="0660"
KERNEL=="vboxdrvu", NAME="vboxdrvu", OWNER="root", GROUP="root", MODE="0666"
KERNEL=="vboxnetctl", NAME="vboxnetctl", OWNER="root", GROUP="vboxusers", MODE="0660"
SUBSYSTEM=="usb_device", ACTION=="add", RUN+="/usr/lib/virtualbox/VBoxCreateUSBNode.sh \
            $major $minor $attr{bDeviceClass}"
SUBSYSTEM=="usb", ACTION=="add", ENV{DEVTYPE}=="usb_device", RUN+="/usr/lib/virtualbox/VBoxCreateUSBNode.sh \
            $major $minor $attr{bDeviceClass}"
SUBSYSTEM=="usb_device", ACTION=="remove", RUN+="/usr/lib/virtualbox/VBoxCreateUSBNode.sh --remove \
            $major $minor"
SUBSYSTEM=="usb", ACTION=="remove", ENV{DEVTYPE}=="usb_device", RUN+="/usr/lib/virtualbox/VBoxCreateUSBNode.sh\
            --remove $major $minor"
```

udev ルールの作成

- ルールの形式:

  <match><op>value [, ...] <assignment><op>value [, ...]

- サンプル

```
KERNEL=="sdb", NAME="my-spare-disk"
KERNEL=="sdb", DRIVER=="usb-disk", SYMLINK+="sparedisk"
KERNEL=="sdb", RUN+="/usr/bin/my-program"
KERNEL=="sdb", MODE="0660", GROUP="mygroup"
```

ルールファイルは３箇所に置くことができます。もし同じ名前が存在する場合の優先順位は:

1. /etc/udev/rules.d
2. /run/udev/rules.d
3. /usr/lib/udev/rules.d

udev ルールの書式は簡単です。１行に２つの情報を定義します。最初の部分には１つ以上の一致ペアで構成します。（二重等号 == で示されます）これらはデバイスの属性や特性、もしくはその両方が、ここに指定した値に一致するか確認します。２番目の部分にはファイル名、グループの割り当て、さらにはファイルのパーミッションなど名前に値を割り当てる１つ以上の割り当てキーと設定値のペアを含みます。

一致するルールが見つからない場合、デフォルトのデバイスノード名が使用されます。

SYMLINK や RUN などのキーの詳細は man udev を参照してください。

ルールファイルの例

- **Fitbit** デバイス:

```
$ cat /etc/udev/rules.d/99-fitbit.rules
SUBSYSTEM=="usb", ATTR{idVendor}=="2687", ATTR{idProduct}=="fb01", \
                  SYMLINK+="fitbit", MODE="0666"
```

- **kdump/kexec** によるクラッシュダンプと高速カーネルロード設定:

```
$ cat /usr/lib/udev/rules.d/98-kexec.rules
SUBSYSTEM=="cpu", ACTION=="add", PROGRAM="/bin/systemctl \
                        try-restart kdump.service"
SUBSYSTEM=="cpu", ACTION=="remove", PROGRAM="/bin/systemctl \
                        try-restart kdump.service"
SUBSYSTEM=="memory", ACTION=="online", PROGRAM="/bin/systemctl \
                        try-restart kdump.service"
SUBSYSTEM=="memory", ACTION=="offline", PROGRAM="/bin/systemctl \
                       try-restart kdump.service"
```

- **kvm** 仮想マシンハイパーバイザーの設定:

```
$ cat /usr/lib/udev/rules/.d/80-kvm.rules
KERNEL=="kvm", GROUP="kvm", MODE="0666"
```

27.4 演習

デモ教材ビデオ

using_udev_demo.mp4

課題 27.1: udev

1. **USB** デバイスが挿入されたときに、myusbというシンボリックリンクを生成するルールを作成、インプリメントします。
2. **USB** デバイスを挿入します。それは、ペン、マウス、ウェブカメラでも何でもかまいません。

   注意：ハイパーバイザーのもとで仮想マシンを実行中ならば、ゲスト OS が **USB** デバイスを見えるようにしておく必要があります。マウスのクリックで（= ハイパーバイザーの設定ユティリティで）ホスト OS から切り離すことができます。
3. /dev ディレクトリのリストを表示し、シンボリックリンクが生成されたことを確認します。
4. **USB** を取り外します。（ドライブであれば、安全のためにまず **umount** してください。）
5. /dev にシンボリックリンクがあるか、確認してください。

解 27.1

1. 内容が次の１行だけのファイル /etc/udev/rules.d/75-myusb.rules を作成します:

   ```
   $ cat /etc/udev/rules.d/75-myusb.rules
   ```

   ```
   1 SUBSYSTEM=="usb", SYMLINK+="myusb"
   ```

   すでに使われていない BUS を SUBSYSTEM の代わりに使用しないでください。**udev** の最近のバージョンではすでに無くなっています。

   注意: ファイル名は問題ではありません。ルール中に ACTION コンポーネントがあれば、システムはそれを実行します; 他のルールを参考にしてください。
2. デバイスを挿入します。
3. `$ ls -lF /dev | grep myusb`
4. デバイスがマウントされていれば:

   ```
   $ umount /media/whatever
   ```

   /media/whateverは、マウント・ポイントです。これで、安全にデバイスを抜くことができます。
5. `$ ls -lF /dev | grep myusb`

章 28

仮想化概要

28.1 仮想化の紹介



この章では、**仮想マシン（VM）**の作成、デプロイ、およびメンテナンスについて説明します。

仮想化インスタンスのスコープを、オペレーティングシステム全体とすることも可能で、こうすれば **サーバー** または **デスクトップ／ワークステーション** 全体を仮想化することができます。

VM を外部から見ると、ネットワーク上に存在する物理マシンに見えます。ほとんどの場合 VM 上で実行されているアプリケーションは、自分が仮想化環境上で動作していることを認識していません。

その他の仮想化（抽象化のスコープが異なるもの）には:

- **ネットワーク:**
  ハードウェア（インターフェース）、ルーターなどの物理ネットワークデバイスが抽象化され、それらを実行、設定するソフトウェアは物理デバイスの詳細を知る必要はありません。
- **ストレージ:**
  複数のネットワークストレージデバイスが１つの大きなストレージユニット（ディスク）に見えるように構成されます。
- **アプリケーション:**
  アプリケーションは **コンテナ** などアプリ毎に独立したスタンドアロン形式に分離されます。これについては後で説明します。

（仮想化によって機能の抽象化ができるとはいっても）実際には物理マシンと仮想マシンには、無視できない違いもあります。
たとえば、パフォーマンスチューニングを行う時には VM とその下にある物理マシンの両方で低レベル（＝ハードウエアを意識した）の最適化を（個別に）行う必要があります。

仮想化の歴史

- 数十年前にメインフレームに実装されたのが最初である:
  - ハードウエアを有効活用する道を開いた
  - オペレーティングシステムの進化はハードウエアの進化より早い
  - (ハードウエア内には) マイクロコードが含まれる
  - 特にユーザーフレンドリー指向に作られてはいない
- 仮想化技術は PC とワークステーションに波及した:
  - 当初は **エミュレーション** (= 機能のソフトによる置き換え) で実現
  - CPU に仮想化支援機構が導入されたことで仮想化での性能、VM のインストール／移行の容易性やフレキシビリティが大幅に向上した

仮想化 には、長く輝かしい歴史があります。初期の **メインフレーム** から **ミニコン** の時代では、様々な性能制約の解消やデバッグ性と管理機能の拡張のために仮想化技術が使われました。

今日では、仮想化はさまざまな形で多くのシーンで活用されています。

目的に応じた仮想化の形式と実装があり、それぞれに特有の利点を生み出しています。

28.2 ホストとゲスト



多くの場合、ゲストは自分がどのホスト上で実行されているかを意識すべきでなく、場合によっては実行中のホスト間でも**マイグレーション（移行）**できます。

事実、コンテンツ全体を VM にコピーすることで、物理マシンを仮想マシンに変換することが可能です。専用のソフトウェアユーティリティを使用すると簡単です。

CPU 使用率、ネットワークスループット、メモリー使用率などの低レベルのパフォーマンスチューニングは、ホスト側で行った方が効果が得られます。ゲストでは（実ハードウエアを制御している訳では無いので）、意味の無いシミュレーションにしかなりません。

アプリケーションのチューニングは、主にゲスト側に対して実施されます。

28.3 エミュレーション



図 28.1: **エミュレーター**

PC アーキテクチャにおける最初の仮想化実装は **エミュレータ** を使用したものでした。現在の（ホスト）オペレーティングシステムで実行している（エミュレータ）アプリケーションは、ゲスト OS から見ると何らかのハードウェアに見えます。エミュレータは（ソフトウエアによる完全なハードウエアのエミュレーションなので）、動作するために特別なハードウェアを必要としません。

Qemu はそのようなエミュレーターの 1 つです。

エミュレーション vs. 仮想化

- **エミューレータ** は完全にソフトウエアとして実行される
- ハードウエア構成はソフトウエアによって置き換えられる
- 異なる CPU アーキテクチャのプログラムを実行させることができる、例えば **x86** ホスト上で **ARM** ゲストマシンの実行が可能である
- しばしば新しい CPU（まだ実ハードウエアが無い場合も含め）向けのオペレーティングシステム開発に利用される
- 性能は期待できない

28.4 ハイパーバイザー

仮想化ハイパーバイザーの種類

- 仮想マシンの下で動作するレイヤーで、仮想マシンを管理する
- **仮想マシンモニター** (**VMM**) とも呼ばれる
- 複数の VM に対して、ハードウエア資源を透過的に共有させる
- 基本となる 2 つの形:
  - **ハードウエア仮想化**
    ゲストシステムは、自分が VM として動作している事を意識しない、**完全仮想化** とも呼ばれる
  - **準仮想化 (= パラ バーチャライゼーション)**
    ゲスト OS は自分が VM として動作していることを認識していて、VM としてより良く実行できるよう変更される
- 最新のハードウエア（= CPU）の多くは、仮想化支援機構を持っている（BIOS で有効化する必要がある）

ホストシステムは、通常のオペレーティングシステム機能とは別に、ゲスト OS の初期化、停止、管理といった **ハイパーバイザー** の役割も担う。仮想化を実現する 2 つの基本的な方法:

- **ハードウエア仮想化:** ゲストシステムは、自分が仮想化されたゲスト OS として動作している事を認識していません。ゲスト OS 自体には手を入れる必要はありません。(＝このため、通常のインストーラを使ったゲスト OS のインストールが可能です)
- **準仮想化 (= パラ バーチャライゼーション):** ゲストシステムには、仮想化環境内で動作する前提でゲスト OS 自体に最適化のための変更が追加されます。(仮想化に対応した、専用 OS をインストールする必要があります)

広く使用されている **Intel** と **AMD** の CPU には **x86** アーキテクチャに仮想化拡張が組み込まれており、ハイパーバイザーが完全に仮想化された（つまり変更されていない）ゲストオペレーティングシステムを、わずかなパフォーマンスペナルティで実行できます。

Intel 拡張 (**I**ntel **V**irtualization **T**echnology) は、通常 **VT**、**IVT**、**VT-32** または **VT-64** と略され、**Vanderpool** という開発コード名でも知られています。2005 年の春から公開されています。

AMD 拡張は、通常 **AMD-V** と呼ばれ、最近でもまだ **Pacifica** という開発コード名で呼ばれることがあります。

これら 2 つの拡張の詳細な説明と比較については https://lwn.net/Articles/182080/ を参照してください。

`/proc/cpuinfo` を見ると、CPU がハードウェア仮想化拡張機能をサポートしているか確認できます。**IVT** 対応チップを使用している場合、フラグフィールドに `vmx` が表示されます。**AMD-V** 対応チップを使用している場合は、同じフィールドに `svm` が表示されます。また **CMOS** で（=BIOS のこと）仮想化機能を有効にする必要があるかもしれません。

準仮想化に対応するオペレーティングシステムの選択は（専用の拡張が必要となるので）制限される傾向がありますが、もともと完全仮想化ゲストよりも効率的に実行できる利点がありました。(CPU などハードウエア側の）仮想化技術の進歩により、そのような準仮想化の利点が大きな性能差とならなくなってきました。さらに、完全仮想化に対応したハードウェアの選択肢が広がったことにより、準仮想化の利点がより少なくなり人気がなくなりました。

外部とカーネル内のハイパーバイザー

- ホスト OS 以外に別のハイパーバイザーを持つことができる: **VMWare**
- OS カーネル内にハイパーバイザーを持たせることもできる: **KVM**
- この講座では、全てがオープンソースで別に 3rd パーティ製ハイパーバイザーを必要としない **KVM** を紹介する

専用ハイパーバイザー



図 28.2: **専用ハイパーバイザー**

エミュレーションから始まった仮想化技術開発が目指した次ステップは、専用設計の軽量カーネルにハイパーバイザーを統合することでした。**VMware ESX**（および関連機能）は、オペレーティングシステムに組み込まれたハイパーバイザーの一例です。

カーネル内のハイパーバイザー



図 28.3: **カーネル内のハイパーバイザー**

KVM プロジェクトは **Linux** カーネルにハイパーバイザー機能を追加しました。これにより、**Linux** カーネルの機能を活用したハイパーバイザーを実現しました。

前述のように、この種類の仮想化（＝最新の完全仮想化）では CPU チップに内蔵された仮想化支援機構と設定ユーティリティが必要です。

28.5 libvirt

libvirt

- 仮想化技術を利用するためのツールキットである
- 多くのハイパーバイザーに対する共通 API を提供する
  - **KVM/QEMU**
  - **LXC**
  - **VirtualBox**
  - ...その他多数
- 以下に対して管理機構を提供する
  - 仮想マシン
  - 仮想ネットワーク
  - ストレージ
- 全エンタープライズ向け **Linux** ディストリビューションで利用可能

多くのアプリケーションが **libvirt** と連携します。最も一般的ものとしては、**virt-manager**、**virt-viewer**、**virt-install**、**virsh** などがあります。現在サポートされているハイパーバイザーの完全なリストは https://www.libvirt.org を参照してください:

- **QEMU/KVM**
- **Xen**
- **Oracle VirtualBox**
- **VMware ESX**
- **VMware Workstation/Player**
- **Microsoft Hyper-V**
- **IBM PowerVM (phyp)**
- **OpenVZ**
- **UML** (ユーザーモード Linux)
- **LXC** (Linux コンテナ)
- **Virtuozzo**
- **Bhyve** (BSD のハイパーバイザー)
- **Test** (テスト用)

libvirt を使用するプログラム

- 多くのユーティリティが **libvirt** を利用する:



図 28.4: **libvirt ベースのユーティリティ**

- 正確な機能のリストは、利用する **Linux distribution** に依存する
- 完全なリストは https://www.libvirt.org/apps.html を参照

このコースでは、堅牢な **GUI** である **virt-manager** を使用します。本当は、コマンドラインユーティリティの方が柔軟性があり、GUI をサポートしないサーバ版 OS でも利用可能という良さがあるのですが。

28.6 QEMU

QEMU とは何か？

- **Q**uick **EMU**lator
- ３つのレベルがある:
  1. **libvirt**: 共通ライブラリーと API
  2. **QEMU**: エミューレータ
  3. **KVM**: アクセラレータ
- ハイパーバイザーが、ハードウエアエミューレーションを行う
- ホストとゲストの CPU アーキテクチャが異なっても良い
- ネイティブ速度で実行できるように、アクセラレータ（**KVM** 等）を結合する
- 新しいプロセッサーチップの開発に活用できる

QEMU は、**Q**uick **EMU**lator の略です。2002 年に Fabrice Bellard によって開発されました。(Bellard は 2.7 兆桁に達する π を計算して、当時の世界記録を達成するなどの偉業を成し遂げたことでも知られています。)
QEMU は **ハードウェアエミュレーション**、または **仮想化** を実行するハイパーバイザーです。ホストアーキテクチャとエミュレートされたアーキテクチャの間で、バイナリ命令を動的に変換して CPU をエミュレートします。
ホスト側とエミュレートされた側の CPU アーキテクチャは異なっていても共通でも良いのです。このため、ホストとゲストそれぞれのオペレーティングシステムには非常多くの選択肢があります。
QEMU はオペレーティングシステム全体ではなく、特定のアプリケーションのみをエミュレートするためにも使用できます。
QEMU 自体はホストマシンよりもはるかに低速です。ただし **KVM** (**K**ernel **V**irtual **M**achine) と組み合わせて使用することで、ネイティブホストに近い速度を達成できます。
ゲストオペレーティングシステムは **QEMU** で実行するために書き換える必要はありません。
QEMU はいつでも仮想マシンを保存、一時停止、復元できます。
QEMU は **GPL** ライセンスが付与されたフリーソフトウェアです。
QEMU 以下の例のように、多くのアーキテクチャをサポートします:

IA-32 (i386), **x86-64**, **MIPS**, **SPARC**, **ARM**, **SH4**, **PowerPC**, **CRIS**, **MicroBlaze**, **etc.**

QEMU のクロスコンパイル機能は、組み込みプロセッサ向けの開発では非常に役立ちます。
実際、**QEMU** はまだ実チップが存在しないか、市場にリリースされていないプロセッサ向けのソフトウエア先行開発用途で、しばしば使用されています。

サードパーティのハイパーバイザーとの統合

- **QEMU** 単独では低速だが、以下の **アクセラレータ** を組み合わせると殆どネイティブスピードが出せる:
  - **KVM**
  - **Xen**
  - **Oracle Virtual Box**
- **KVM** は **QEMU** と同じプロジェクト生まれで、極めて密接に統合されている

QEMU 単独では処理速度はかなり低速です。ただし、サードパーティのハイパーバイザーと統合すれば、ほぼネイティブの処理速度を出すことができます。これらのシステムのいくつかは **QEMU** とは非常に近い従兄弟のような関係であることに注目してください。他のものは、より遠い関係にあります。近しい関係のものをいくつか紹介します:

- **KVM** は **QEMU** と特に密接に統合されています。ホストとターゲットのアーキテクチャが同じ場合、完全なアクセラレーションと高速化が達成されます。**KVM** は **Linux** そのものです。これについて後で詳しく説明します。
- **Linux** がネイティブにサポートする（= PV ドライバーが予めカーネルに組み込まれているという意味）**Xen** も **x86** といくつかの **ARM** バリアント CPU のように、アーキテクチャに機能がある場合はハードウェア仮想化モードで実行できます。
- **Oracle Virtual Box** は **qcow2** 形式のイメージを使用でき、**qemu** と非常に親密な関係にあります。

このコースでは、仮想マシンの構成と実行に **virt-manager** を使用することをお勧めします（演習でも使用します）。また、**qemu** コマンドラインユーティリティの使い方についても説明します。

イメージフォーマット

- ２つの主要なフォーマット:
  1. **cmd**: 最もシンプルなフォーマットである
  2. **qcow2**: **C**opy **O**n **W**rite 2
- これ以外にも多くのフォーマットがサポートされている
- フォーマット間での変換が可能:

```
$ qemu-img convert -O vmdk myvm.qcow2 myvm.vmdk
```

QEMU はディスクイメージファイルのいろいろなフォーマットをサポートしています。ただし、主に使用されるのは２つだけで、残りは歴史的な理由と変換ユーティリティのために存在しています。２つの主要なフォーマットは:

- **cmd** (デフォルト)
  これが最も単純で、他の非 **QEMU** エミュレーターにエクスポートするのも一番簡単です。空のセクタは領域を必要としません。
- **qcow2**
  COW は **C**opy **O**n **W**rite の略です。多くのオプションがあります。詳細は **man qemu-img** を参照してください。

サポートされているフォーマットのリストを取得するには:

```
$ qemu-img --help | grep formats:
```

```
Supported formats: blkdebug blklogwrites blkreplay blkverify copy-on-read file ftp ftps
↪  gluster host_cdrom host_device http https iscsi iser luks nbd null-aio null-co nvme qcow2
↪  quorum raw rbd ssh throttle vhdx vmdk vpc
```

特に以下に注目してください:

- **vdi**: **Oracle Virtual Box** が採用する
- **vmdk**: **VMware** が採用する

qemu-img はフォーマット間の変換に使用できます。例えば:

```
$ qemu-img convert -O vmdk myvm.qcow2 myvm.vmdk
```

これはデフォルトのオプションを使用しています。全機能は **man** ページを参照してください。

28.7 KVM



KVM はメモリー管理、スケジューリング、同期などを含むコンピューティングリソースの管理に **Linux** カーネルそのものを使用しています。仮想マシン実行時には **KVM** は **Linux** カーネルと協調動作します。

KVM は 1 つ以上の CPU コア内で VMX（Intel）、または SVM （AMD）命令を使用して、仮想マシンモニタ（VMM）を実行します。この時 **Linux** カーネルは、他の CPU コアで実行されます。

仮想マシンモニタ（VMM）は、仮想マシンモニタ自身に制御を引き取る命令を実行するまでは、ゲストを最大限のハードウェアスピードで実行させます。

引き継いだ時点で、仮想マシンモニタは任意の **Linux** リソースを使用してゲストの指示をエミュレートしたり、最後の指示でゲストを再起動したり、他の操作を実行したりできます。

KVM モジュールをロードしてゲストを起動すると **Linux** がハイパーバイザーとして動作します。**Linux** の特徴は残しつつ、ハードウェア仮想マシンモニタとしても機能します。このため、**cgroups**、**nice**、**numactl** などの標準 **Linux** リソースとプロセス制御ツールを使用して、仮想マシンを制御できるのです。

KVM は、最初 Windows 仮想デスクトップ製品の一部として登場 (事前マージ) しました。2007 年にアップストリームにマージされた時点では、**KVM** の実行には最新の **x86_64** プロセッサが必須でした。**x86_64** プラットフォームでは **KVM** は主としてプロセッサの仮想化サブシステム用のドライバでした。

KVM は、2007 年に **Linux** カーネルモジュールとして登場しました。(同時に提供された）**QEMU** の修正バージョンと組み合わせることで、ほとんどのランタイムサービスに **Linux** カーネルを使用するハイパーバイザーが実現されました。

KVM の作者である Avi Kivity が **Linux** 開発コミュニティにソースコードを投稿すると、Linus はすぐに **KVM** を **Linux** ツリーにマージしました。これは多くの人にとって驚きでした。

KVM の管理

- コマンドラインと GUI の両方が利用可能である
- コマンドラインツールには:
  - **virt-***
  - **qemu-***
- GUI ツールには:
  - **virt-manager**
  - **kimchi**
  - **OpenStack**
  - **oVirt**
- この講座では **virt-manager** を紹介する:
  - 様々なディストリビューション、バージョンを通じて最も安定した GUI ツールである

コマンドラインツールには、仮想マシンのイメージを作成、変換、操作、デプロイ、保守するための多くの低レベルのコマンドが用意されています。

より多くの経験を積めばコマンドラインツールの方が効率的と感じられるでしょうが、基本的な操作だけなら **virt-manager** で十分なので、ここではそれを使用します。

28.8 演習

課題 28.1: KVM が正しくセットアップされていることを確認する

重要

- 以下の演習は **Linux** をネイティブに実行できる物理 PC 上で行うことが望ましいのです。
- 演習は **VMWare** や **Virtual Box**、更に **KVM** などのハイパーバイザー配下で動作する仮想マシンでも動かせるかもしれませんが、この場合は **多段仮想化（=nested virtualization）**が正常に動作することが前提となります。
- 動くか動かないかは、どのハイパーバイザーを使うかだけでなく下位のホスト OS（例えば、**Windows**、**Mac OS**、**Linux** など）や、さらに **Linux** や **Windows** のどのバージョンを使うか、カーネルは何かなどにも依存します。
- さらに、あなたが使用するハードウエアも影響します。例えば、色々な **x86_64** マシン上で **VMWare** では多段仮想化が動作することを確認していますが、**Oracle Virtual Box** 上では動作するものは限られます。
- もし動いたとしても、ネイティブハードウエア環境と比べれば性能は劣るでしょうが、ここで実行する簡単なデモ演習ではそれは重要ではないでしょう。
- 皆さんの経験値も色々でしょう。（仮想化環境上で）きちんと動かすところまでは、我々も責任を持てません。

1. まず（CPU コアに）ハードウェア仮想化支援機構が備わっていて、有効化されていることを確認します:

   ```
   $ grep -e vmx -e svm /proc/cpuinfo
   ```

   vmx は **INTEL** CPU を意味し、svm は **AMD** を意味します。どちらもない時には、以下のどちらかが考えられます:

   - 物理マシン上で作業しているときは、この問題を解決できます。マシンをリブートし BIOS 設定で仮想化支援機能を有効にします。IT 部門が、セキュリティ上の理由で仮想化支援機能を利用できないようにしている場合があります。そのときは、そのポリシーを変更してもらうよう頑張ってください。
   - ハイパーバイザー配下の仮想マシン上で作業している場合で、多段仮想化（=nested virtualization）が利用できない場合

2. どちらの理由だったにせよ、ハードウェア仮想化支援はありません。**virt-manager** を実行することは **できるでしょう**が、性能は低いままです。

3. 全ての関連するパッケージをインストールします。正確なリストを作っても良いのですが、パッケージの正確な名称や必要とするものは良く変わるものです。大半のエンタープライズ系のディストリビューションには、必要なソフトウェアの全部（ほとんど全部）が含まれています。

4. 提供したスクリプトを使ってもらうのが、一番簡単な方法です:

   ```
   $ ./ready-for.sh --install LFS301-JP
   ```

   これで、全ての作業が終わります。

   RPM システムでは、別の地道な方法もあります:

   ```
   $ sudo dnf|zypper install kvm* qemu* libvirt*
   ```

   このようにしても、ストレージをあまり消費しません。

   Ubuntu を含む **Debian** パッケージベースのシステムでは、同様のことを好みのパッケージインストーラで行います。

重要

- 他のハイパーバイザーを実行中に **libvirtd** を実行しないでください。恐ろしい結果を引き起こします。システムがクラッシュしたり、使用中の仮想マシンにダメージを与えたりします。
- 他のハイパーバイザーを **停止** したり **使用不可** にしたりすることを勧めます:

```
$ sudo systemctl stop vmware
$ sudo systemctl disable vmware
```

または

```
$ sudo systemctl stop vboxdrv
$ sudo systemctl disable vboxdrv
```

課題 28.2: virt-manager と KVM を使い、仮想マシンをインストールして実行する

本演習では、事前に **TinyCoreLinux** (http://www.tinycorelinux.net) で構築済の **iso** イメージを利用します。なぜなら、それは非常に良くできていて小さいからです。

もし他の **Linux** ディストリビューション、たとえば **Debian**、**CentOS**、**Ubuntu**、**Fedora**、**openSUSE** などがあれば、それらのインストール **iso** イメージを利用しても構いません。基本的な進め方は同じで、違うのは新規 VM 構築のためのインストール段階だけです; それは実機にインストールするのと大して差はありません。

スクリーンキャプチャを使いながら、ステップバイステップで説明します; 混乱したら **virt-manager** を立ち上げてください、**GUI** はわかりやすく作られているので、必要なステップを進めていくことができるはずです。

1. **libvirtd** が実行中であることを確認して、**virt-manager** を起動します:

```
$ sudo systemctl start libvirtd
$ sudo virt-manager
```



図 28.5: **virt-manager を開始する**

2. `File->Create New Machine`をクリックします:



図 28.6: **virt-manager で仮想マシンを作成する**

3. **TinyCoreLinux** から（入手した）3つの異なる **iso** インストールイメージを RESOURCES/s_28 ディレクトリに置きました:

   ```
   Core-current.iso
   CorePlus-current.iso
   TinyCore-current.iso
   ```

   (http://www.tinycorelinux.net に、より新しいものがあるかもしれません。しかし、ここに置いたものでも問題無く動作します)。

   CorePlus-current が大きく安定しているため、これを用います。完全なグラフィックをインストールしてくれます。しかし、他も手早く使うことができます。

   ファイルシステムを見て、望むイメージを使ってください:



図 28.7: **virt-manager で TinyCoreLinux の iso イメージを選択する**

4. つぎに必要なメモリー量、**CPU** もしくは **core** 数を指定します。これらのイメージは比較的小さいものです。256 MB で十分です: どのくらい小さいメモリーでできるか見ものです！



図 28.8: **virt-manager で、メモリーと CPU の構成を設定する**

5. つづいて VM の生成場所と大きさを指定します。**TinyCoreLinux** は本当に小さくても十分なのですが、GUI では 0.1 GB 以下は指定できません（約100 MB。）（コマンドラインからは、もっと小さい値を指定できます）。



図 28.9: **virt-manager で、ディスクストレージの構成を設定する**

`Select or Create custom storage` をクリックしないと、イメージは `/var/lib/libvirt/images` に置かれます。イメージはそこそこ大きいため、別の場所に置くことも必要です。シンボリックリンクを使って、`/var/lib/libvirt` 中のイメージディレクトリを変更することもできます:

```
$ cd /var/lib/libvirt
$ sudo mv images images_ORIGINAL
```

```
$ sudo mkdir /tmp/images
$ sudo ln -s /tmp/images images
```

（イメージファイル用のディレクトリを `/tmp` 以外の場所に設定したい場合があります。これでできますね。）

6. **TinyCoreLinux** インストールディスクから VM をインストールする準備ができました:



図 28.10: **virt-manager で、仮想マシンのインストールを開始する**

注目

インストール前に **Customize** コンフィグをクリックすることをお勧めします。他の変更をする前に、マウスが **PS2** となっているのを **USB tablet** インプットポインタとすべきです。

これは次のスクリーンで、`Add Hardware` をクリックすることでできます。



図 28.11: **virt-manager で、仮想マシンに入力装置を追加する**

7. さあ、インストールを開始します:



図 28.12: **virt-manager で、インストールメディアをブートする**

グラフィカルインターフェースを選択します。ここでは、一番目、デフォルトを選択し、リターンを押下します。

8. 時間がかかります。そして、次のような画面が表示されます:



図 28.13: **TinyCoreLinux の最初の画面**

ここで何をすべきかはっきりしませんが、画面下にアイコンがあり、スクリーンの大きさを変更して縦長にします:



図 28.14: **TinyCoreLinux の最初の画面の大きさを変更**

9. ターミナルアイコンをクリックします（または、バックグラウンドで右クリックしてターミナルを開きます）。残念ながら非常に小さいフォントです。次にターミナルから以下を入力します。

```
tc-install
```



図 28.15: **tc-install を実行する**

10. `Whole disk` と `sda` を選択し、次矢印をクリックします:



図 28.16: **仮想マシンのディスクを選択**

11. しばらく動き続けて、各ステップの出力が画面に表示されます。



図 28.17: **virt-manager で、インストールを終了する**

インストール完了後、File メニューから仮想マシンをシャットダウンします。

12. **virt-manager** をリブート（もし、kill していたら）すると、以下のように表示されます:



図 28.18: **virt-manager で、新しい仮想マシンを実行する**

VM とopenとrunを右クリックします。仮想マシンが立ち上がり、実行されるはずです！(もし問題があり、もとのインスト

ールイメージが実行されているようでしたら、新ディスクイメージに **tc-install** プログラムがないせいかもしれません。）

課題 28.3: 追記: コマンドラインから実行する

コマンドラインから:

```
$ sudo qemu-img create -f qcow2 /var/lib/libvirtd/myimg.qcow2 24M
$ sudo qemu-system-x86_64 -hda /var/lib/libvirtd/myimg.qcow2  \
       -cdrom /teaching/LFCW/RESOURCES/LFS301-JP/CorePlus-current.iso  -usbdevice tablet
```

章 29

コンテナ概要

29.1 コンテナ



図 29.1: **コンテナ**

ハイパーバイザーと **Linux** カーネルをさらに統合して、オペレーティングシステムレベルの仮想マシンや **コンテナ** を実現します。コンテナは **Linux** カーネルの多くの機能を共有し、**ネームスペース（名前空間）**や **cgroup** など、最近カーネルに追加されたいくつかの機能も利用します。コンテナは非常に軽量であり、仮想マシン全体に関連するオーバーヘッドを削減します。

コンテナの初期の形は **OS コンテナ** でした。この種類のコンテナは **init** プロセスを実行して、複数のアプリケーションを生成する機能を備えたオペレーティングシステムのイメージを実行します。

LXC (**Linux Containers**) は、その一例です。

29.2 アプリケーションの仮想化



図 29.2: **アプリケーションの仮想化**

仮想マシンに関連するオーバーヘッドをさらに削減するために、**アプリケーション仮想化** の人気が高まっています。
アプリケーション仮想化では、アプリケーション毎に独立したコンテナを実行します。通常１台のマシン上で、複数のアプリケーションコンテナが初期化されます。小さいコンポーネントの使用により柔軟性が向上し、仮想化に伴うオーバーヘッドを減少させています。

Docker は、そのようなプロジェクトの１つです。

29.3 コンテナ vs 仮想マシン

コンテナ vs 仮想マシン

- プロセスとサービスを実行する
  - VM はフル機能のオペレーティングシステムを実行するので、多くのサービスやアプリケーションを動かすことができる
  - コンテナは、通常は１つのこと（アプリ）に専念する
- VM はコンテナより多くのリソースを利用する
- 複数のコンテナが１つの OS カーネルを共有する。一方 VM ではそれぞれが専用のカーネルを持つ
- コンテナの方が移植性が高い
- VM 内でコンテナを動かすこともできる
- コンテナでセキュリティを確保するのは容易ではない
- コンテナの起動は、通常より高速である
- それでも、多くのケースで仮想マシン（=VM）が最適なソリューションとなる

仮想マシンとコンテナは、両方とも重要なニーズを満たすものです。登場してからしばらくの間は両方とも順調で、ほとんど全てのものに適用されると思われていました。

両方とも長い歴史があります:

- メインフレームコンピュータは何十年もの間、かなり特殊な方法でソフトウェア分離と仮想マシンを採用していました。
- オペレーティングシステムに長年実装されている **chroot** と **BSD Jail** の基本的な分離の考え方は、コンテナと同じです。

多くの異なるサービスとアプリケーションを緊密に統合させる必要がある場合、サービス単位で仮想マシンを動かす、が最適なソリューションとなります。

アプリケーションが、多くのサービスやライブラリなどを持ったフル機能のオペレーティングシステム環境を前提に記述されている場合も、仮想マシンが最適でしょう。

コンテナと仮想マシンでは、ワークロードのスケーリングの考え方が異なります。**Kubernetes** や **Mesos** などの **オーケストレーション** システムは、必要に応じてコンテナ数の変更、負荷分散、イメージの複製や削除を行うことができます。

29.4 Docker

Docker

Docker はアプリケーションレベルの仮想化です。対象アプリケーション実行に必要なサービスを構築するため、多くの個別イメージを使用します。これらのイメージがコンテナ内にパッケージ化されています

- イメージは、コンテナ内のコンポーネントである
- イメージに含まれる可能性のあるのは
  - アプリケーションプログラム
  - ランタイムライブラリー
  - システムツール
  - その他、アプリ実行に必要なもの全て
- イメージは **Docker Hub** または **レジストリーサーバー** に置かれる

Docker の広範なドキュメントが https://docs.docker.com にある

Docker の最も魅力的な機能の１つは、すべての依存コードとサービスをアプリケーションと共にパッケージ化し、最小限のオーバーヘッドで単一ユニットとしてデプロイできることです。このデプロイは、要求に応じて何度でも簡単に繰り返すことができます。これにより、アプリケーションをサポートするために何層ものサービスをサーバーに構築する必要が少なくなります。

Docker の手順

数ステップで **Docker** コンテナ化されたアプリケーションを起動することが可能である

- お気に入りのツールで **Docker** サービスパッケージをインストール
- **Docker** サービスを開始する
- **Docker Hub** 、またはプライベートリポジトリでイメージを検索する
- イメージを取得する
- イメージを実行する
- 最後に、アプリケーションをテストする

上記の手順は、**Docker** アプリケーションのテストの最小限の例です。

もちろん、イメージを作成し、システム変数または構成パラメータを設定し、結果を新しいイメージとして保存する機能を含めるなど、使用できるオプションが多数あります。場合によっては、書き込み不可ではなく書き込み可能なイメージが必要となる場合もあります。

ほとんどの **Docker** コマンドには、個別の **man** ページが用意されています。具体的には `docker(1)`、`docker-search(1)`、`docker-pull(1)`、`docker-create(1)`、`docker-run(1)` などがあります。

29.5 Docker コマンド

docker コマンド

- **docker** コマンドには 40 以上のサブコマンドがある
  - 50 以上のオプションを持つコマンドもある
- `docker <command> --help` で詳細を表示できる

• run	• network	• start
• create	• rm	• stop
• exec	• rmi	• top
• ps	• save	• update
• images	• search	• volume

など

多くの **docker** サブコマンドは、ある程度自己文書化されています。(=コマンド実行時に、ヘルプドキュメントを表示できます。) よく混同されるのは、`run`、`create`、`exec` です。`ps` コマンドは実行中のコンテナを表示し、**-all** オプションを指定するとすべてのコンテナをリストします。

`run` は新しいコンテナを開始し、その中でコマンドを実行します。一般的なオプションは tty に接続する `-t` とコンテナをバックグラウンドで実行させる `-d` です。

`create` コマンドはコンテナを作成します。コンテナの設定と接続をするための多くのオプションがあります。

コンテナがすでに実行されていてその中で、何かを実行したい場合は `exec` コマンドを使用できます。また `-t -d` オプションも使えます。

`images` コマンドは、さまざまなイメージ情報を表示します。`rmi` コマンドは、デフォルトではイメージを削除しタグ付けされていない親を削除します。

すべてのコンテナ上でシェル関数を活用した操作をすることもできます。たとえば、停止したすべてのコンテナを削除するには: `docker rm $(docker ps -a -q)`.

29.6 Podman

Podman

- **RHEL8/CentOS8** では純粋な **docker** が **Podman** に置き換えられた
- **Podman** は、コンテナの作成と管理に子／親の fork モデルを採用する
- **Docker** では、バックグラウンドで実行されるデーモンを備えたサーバー／クライアントモデルが使われていた
- 約束された利点:
  - セキュリティの向上
  - オーバーヘッドの削減

  ```
  $ sudo dnf install podman podman-docker
  ```
- エミュレーションレイヤーによって **docker** コマンドとの後方互換性がある

docker エミュレーションレイヤーを使用して **RHEL/CentOS 8** で演習を行う方法を示します。他のディストリビューションも **podman** をパッケージシステムに追加してきました。

Ubuntu 20.04 の公式リポジトリには既に **podman** が含まれているのですぐに利用できます。

```
$ sudo apt-get update
$ sudo apt-get install podman
```

Ubuntu 18.04 では最初に以下を行う必要があります:

```
$ sudo apt-get update
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository ppa:projectatomic/ppa
```

29.7 演習

課題 29.1: Docker アプリケーションとして、Apache（httpd）のインストール、テストをする

Overview

本演習では **docker** パッケージをインストール、実行そしてテストします。さらに、**httpd**、**Apache** web サーバコンテナの入手とデプロイを行います。

RedHat、Centos、Fedora では

RHEL/CentOS 8 と、最近の **Fedora** ディストリビューションでは **podman** を用います。これらのシステムでは、後方互換性レイヤーによって **docker** がサポートされています。

```
$ sudo dnf install podman podman-docker
```

1. **Docker** がインストールされている（または **podman** がエミュレートしている）ことを確認してください。ディストリビューション毎に適切なコマンドを実行してください。

```
$ sudo yum install docker                # RHEL/CentOS 7
$ sudo dnf install podman podman-docker  # RHEL/CentOS 8, Fedora
$ sudo apt-get install docker.io         # Ubuntu, Debian
$ sudo zypper install docker             # openSUSE
```

Docker を再インストール?

- 演習中に解析不可能なエラーが発生した場合には、**docker** を **再インストール** するのが良いでしょう。（**RHEL 7** などで）システム更新後に **docker** の構成が壊れる現象が見つかっています。

2. **docker** サービスの開始:

重要

podman ベースのシステムでは、開始すべき **docker** サービスが無いため、次のステップは省略できます。

```
$ sudo systemctl start docker
```

`systemctl status docker`を用いることで、実行状態を確認できます:

```
~
student@ubuntu:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; disabled; vendor preset: enabled)
     Active: active (running) since Thu 2021-03-25 11:48:49 PDT; 8min ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 3063 (dockerd)
      Tasks: 12
     Memory: 125.7M
     CGroup: /system.slice/docker.service
             └─3063 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Mar 25 11:48:48 ubuntu dockerd[3063]: time="2021-03-25T11:48:48.994073149-07:00" level=warning msg="Your kernel does not support cgroup rt runtime"
Mar 25 11:48:48 ubuntu dockerd[3063]: time="2021-03-25T11:48:48.994324640-07:00" level=warning msg="Your kernel does not support cgroup blkio weig>
Mar 25 11:48:48 ubuntu dockerd[3063]: time="2021-03-25T11:48:48.994565112-07:00" level=warning msg="Your kernel does not support cgroup blkio weig>
Mar 25 11:48:48 ubuntu dockerd[3063]: time="2021-03-25T11:48:48.995099733-07:00" level=info msg="Loading containers: start."
Mar 25 11:48:49 ubuntu dockerd[3063]: time="2021-03-25T11:48:49.337342379-07:00" level=info msg="Default bridge (docker0) is assigned with an IP a>
Mar 25 11:48:49 ubuntu dockerd[3063]: time="2021-03-25T11:48:49.633735493-07:00" level=info msg="Loading containers: done."
Mar 25 11:48:49 ubuntu dockerd[3063]: time="2021-03-25T11:48:49.667539540-07:00" level=info msg="Docker daemon" commit=afacb8b7f0 graphdriver(s)=d>
Mar 25 11:48:49 ubuntu dockerd[3063]: time="2021-03-25T11:48:49.667912940-07:00" level=info msg="Daemon has completed initialization"
Mar 25 11:48:49 ubuntu systemd[1]: Started Docker Application Container Engine.
Mar 25 11:48:49 ubuntu dockerd[3063]: time="2021-03-25T11:48:49.691798667-07:00" level=info msg="API listen on /run/docker.sock"
student@ubuntu:~$
```

図 29.3: **Docker の状況を調べる**

なにか問題が発生したら `/var/log/messages` を見るか、システム上の他のログを見てください。標準のカーネルを実行している場合は問題が無いと思いますが、手を加えた **Linux** カーネルを実行している場合は、特にネットワーク関係の構成を適切に設定しなければなりません。これは複雑なのでここで説明することができません。ですから、よほど挑戦的なことをしたいのでない限り、ディストリビューションが提供するカーネルを利用してください。

3. 次のコマンドで **httpd** コンテナを検索してください。

   ```
   $ sudo docker search apache
   ```

```
student@ubuntu:~$ sudo docker search apache
NAME                              DESCRIPTION                                     STARS               OFFICIAL            AUTOMATED
httpd                             The Apache HTTP Server Project                  3424                [OK]
tomcat                            Apache Tomcat is an open source implementati…   2979                [OK]
cassandra                         Apache Cassandra is an open-source distribut…   1251                [OK]
maven                             Apache Maven is a software project managemen…   1174                [OK]
zookeeper                         Apache ZooKeeper is an open-source server wh…   1041                [OK]
solr                              Solr is the popular, blazing-fast, open sour…   819                 [OK]
apache/airflow                    Apache Airflow                                  226
apache/nifi                       Unofficial convenience binaries and Docker i…   206                                     [OK]
eboraas/apache-php                PHP on Apache (with SSL/TLS support), built …   144                                     [OK]
apache/zeppelin                   Apache Zeppelin                                 144                                     [OK]
eboraas/apache                    Apache (with SSL/TLS support), built on Debi…   92                                      [OK]
apacheignite/ignite               Apache Ignite - Distributed Database            76                                      [OK]
nimmis/apache-php5                This is docker images of Ubuntu 14.04 LTS wi…   69                                      [OK]
bitnami/apache                    Bitnami Apache Docker Image                     67                                      [OK]
apachepulsar/pulsar               Apache Pulsar - Distributed pub/sub messagin…   34
linuxserver/apache                An Apache container, brought to you by Linux…   27
apache/nutch                      Apache Nutch                                    23                                      [OK]
antage/apache2-php5               Docker image for running Apache 2.x with PHP…   23                                      [OK]
webdevops/apache                  Apache container                                15                                      [OK]
newdeveloper/apache-php           apache-php7.2                                   8
newdeveloper/apache-php-composer  apache-php-composer                            7
lephare/apache                    Apache container                                6                                       [OK]
secoresearch/apache-varnish       Apache+PHP+Varnish5.0                           2                                       [OK]
apache/arrow-dev                  Apache Arrow convenience images for developm…   1
jelastic/apachephp                An image of the Apache PHP application serve…   0
student@ubuntu:~$
```

図 29.4: **Docker 検索を使う**

（**apache** の代わりに **httpd** を使っても同じ結果を得ることができます。）
なにか出力されればうまく行っているはずなので、ここからは詳細な実行結果は掲載しません。

4. コンテナの処理:

   ```
   $ sudo docker pull docker.io/httpd
   ```

   すべてのコンポーネントをダウンロードするのに時間がかかります。

5. インストール済のコンテナを表示します:

   ```
   $ sudo docker images
   ```

6. イメージに関連するコンポーネントを表示します。

```
$ sudo  docker images --all
```

7. **httpd docker** コンテナを開始します。**httpd** デーモンに接続するため、端末がハングしたように見えます。

```
c7:/tmp>sudo docker run httpd

AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
   using 172.17.0.2. Set the 'ServerName' directive globally to suppress this message
......
```

8. 出力中の IP アドレスを指定してブラウザを起動します。(上記のアドレスを使わないでください !)

   新しい端末を開いて、テキストベースのブラウザを実行することもできます。(グラフィカル環境を持たない場合など。)(**docker httpd** コンテナが実行されている端末を kill しないように注意してください)。次のコマンドを実行してください:

```
$ lynx   http://172.17.0.2
$ w3m    http://172.17.0.2
$ elinks http://172.17.0.2
```

   インストールされているテキストブラウザを利用してください。

9. コンテナと **docker** サービスを停止し、クリーンアップをします。

```
c7:/tmp>sudo docker ps
```

```
CONTAINER ID    IMAGE    COMMAND              CREATED           STATUS            PORTS       NAMES
b936b0afeb23    httpd    "httpd-foreground"   41 seconds ago    Up 40 seconds     80/tcp      boring_turing
```

```
c7:/tmp>sudo docker stop b936b0afeb23
```

```
b936b0afeb23
```

10. システムとディストリビューションにより /var/lib/docker または /var/lib/containers のどちらかの下に、イメージと関連するストレージを残します。不要な場合は、以下のコマンドで削除します :

```
c7:/tmp>sudo docker rmi -f docker.io/httpd
Untagged: docker.io/httpd:latest
Untagged: docker.io/httpd@sha256:cf774f082e92e582d02acdb76dc84e61dcf5394a90f99119d1ae39bcecbff075
Deleted: sha256:cf6b6d2e846326d2e49e12961ee0f63d8b5386980b5d3a11b8283151602fa756
```

   いくつかのシステムでは、他にすることがあります: コンテナを削除します:

```
c7:/tmp>sudo docker system prune -a
.....
```

   podman 以外のシステムでは、次のことができます:

```
c7:/tmp>sudo systemctl stop docker
```

章 30

ユーザーアカウントの管理

注目

** これらのセクションの一部または全体をオプション扱いとする場合があります。これらには補足資料、専門トピック、または高度な話題が含まれインストラクターが教室の状況や時間制約に応じてこれらの内容を紹介するかを判断します。

30.1 ユーザーアカウント



Linux システムは、複数のユーザーやプロセスに対して並列的に独立した作業領域を許容する **マルチユーザー環境** を提供します。

特別なユーザーアカウントとして、システムで **どんなことでも** 実行できる **root** ユーザーがあります。二度と取り返しがつかない大失敗を回避するため、またセキュリティ上の理由から、root アカウントは真に必要がある場合に限って使用するべきです。

一般ユーザーアカウントは、システム上で作業する人向けです。プロセスが root アカウントを使わずにプログラムを実行できるようにするための専用アカウント（**daemon** など）がいくつか存在します。

特定のユーザーの集団が、共通の目的のためにファイルや権限などを共有する **グループ管理** についても紹介します。

ユーザーアカウントの属性

/etc/passwd より:

```
....
beav:x:1000:1000:Theodore Cleaver:/home/beav:/bin/bash
warden:x:1001:1001:Ward Cleaver:/home/warden:/bin/bash
dobie:x:1002:1002:Dobie Gillis:/home/dobie:/bin/bash
....
```

- ユーザー名
- ユーザーパスワード
- ユーザー識別番号 (**UID**)
- グループ識別番号 (**GID**)
- コメント、または **GECOS** 情報
- ホームディレクトリ
- ログインシェル

/etc/passwd ファイルの各行には、システムの各ユーザーの基本アカウント属性が記載されています。(パスワードとこのファイルについては後で説明します)。 各行に含まれる 7 つの要素は:

1. ユーザー名
   各ユーザーに割り当てられた一意の名前です。
2. ユーザーパスワード
   各ユーザーに割り当てられたパスワードです。
3. ユーザー識別番号 (**UID**)
   ユーザーアカウントに割り当てられた一意の番号です。システムはユーザーの特権の決定やアクティビティ追跡など、さまざまな目的で **UID** を使用します。
4. グループ識別番号 (**GID**)
   ユーザーの**グループ**がプライマリグループ、プリンシパルグループ、またはデフォルトグループのいずれであるかを示します。
5. コメント、または **GECOS** 情報
   連絡先情報(フルネーム、電子メールアドレス、オフィス、連絡先番号)の記録ために使用される、コメントフィールドです。
   (GECOS の意味は意識する必要はありません。これは非常に古い用語です。)
6. ホームディレクトリ
   ほとんどのユーザーでは、そのユーザーのワークングエリアとなる専用のディレクトリです。通常はユーザーがそのディレクトリの所有者になります。**root** 以外のユーザーについては、システムの /home の下に作られます。
7. ログインシェル
   通常ここには **/bin/bash** や **/bin/csh** などのシェルプログラムを指定します。ただし、特別なケースではここに別のプログラムを指定することがあります。一般論で言えば、本フィールドには任意の実行可能ファイルを指定することができます。

30.2 ユーザーアカウントの管理

useradd によるユーザーアカウント作成

- **useradd** は、デフォルト（ひな形）に従ってユーザーを作成する
  `$ sudo useradd dexter`
  - dexter に対するアカウントを作成する
  - デフォルトの設定でユーザー／グループ ID、ホームディレクトリ、シェルを設定する
- **useradd** にオプションを付加して、デフォルトを上書きする
  `$ sudo useradd -s /bin/csh -m -k /etc/skel -c "Bullwinkle J Mo が ose" bm`
- オプションを使って、シェル、skel ディレクトリ、コメントを設定する

```
$ sudo useradd dexter
```

このコマンドは、以下のステップを実行します:

- dexter の UID のデフォルト値は (/etc/login.defs に指定されている) UID_MIN より一つ大きな値となります。
- dexter のグループ ID は GID=UID に設定され、同時に dexter のプライマリーグループに設定されます。
- ホームディレクトリ /home/dexter が作成され dexter を所有者に指定します。
- dexter のログインシェルを **/bin/bash** に指定します。
- /etc/skel のコンテンツを /home/dexter にコピーします。デフォルトでは /etc/skel には **bash** と **X Window** システムの初期化ファイルが含まれています。
- /etc/shadow ファイルの dexter 行のパスワード欄には !! が書き込まれます。これにより、このアカウントを利用可能な状態にするには、管理者がパスワードを付与することが求められます。

デフォルト設定は、以下のように **useradd** のオプションを使って簡単に上書きが可能となっています:

```
$ sudo useradd -s /bin/csh -m -k /etc/skel -c "Bullwinkle J Moose" bmoose
```

これは、ユーザー属性の一部にデフォルト以外の明示的な値を与える場合に使います。

ユーザーアカウントの変更と削除

- root ユーザーは **userdel** を使ってユーザーアカウントを削除できる
  ```
  $ sudo userdel morgan
  ```
  morgan のユーザーアカウントが削除されるが、ホームディレクトリは削除されない
- **usermod** を使ってユーザーアカウントを変更できる
  ```
  $ sudo usermod -L dexter
  ```
  dexter のユーザーアカウントをロックする、これでログインできなくなる

```
$ sudo userdel morgan
```

ユーザー morgan のすべての情報が /etc/passwd、/etc/shadow、/etc/group から消去されます。これにより、アカウントは削除されますが、後でアカウントが再設定される可能性に備えてホームディレクトリ（通常は /home/morgan）は削除されません。**userdel** 実行時に -r オプションを指定すると、ホームディレクトリも消去されます。ただし、削除されたユーザーが所有するシステム上の他のすべてのファイルは残ったままです。

usermod は、グループメンバシップ、ホームディレクトリ、ログイン名、パスワード、デフォルトシェル、ユーザー ID など、ユーザーアカウント属性を変更するために使用できます。
使い方はとても簡単です。**usermod** は必要に応じて /etc ディレクトリ内のファイルを変更することに注意してください。



図 30.1: **Using usermod**

30.3 ロックされたアカウント

ロックされたアカウント

- **Linux** はシステムアカウントを **ロック** した状態で提供される
  - bin、daemon、sys などのアカウント
  - プログラムを実行する可能性はあるが、ログインには使われない
- 別の目的でロックされたアカウントが作られることもある
  - データーベースなど主要なアプリケーションが利用するアカウント
- ロックされたアカウントには、有効なパスワードが存在しない
  - 通常 "!!" と表現される
  - パスワードは 100 文字以下または 100 文字以上（要確認、意味不明）
- ロックされたアカウントには、全て無効なシェル（通常は `/sbin/nologin`）が設定されることを確認する

Linux にはロックされたアカウントがあります。このアカウントはプログラムの実行は可能ですが、システムへログインはできず、有効なパスワードも付与されません。たとえば、/etc/passwd には次のようなエントリがあります:

```
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

ロックされたユーザーがシステムにログインを試みると、**nologin** シェルは次を返します:

```
This account is currently not available.
```

または /etc/nologin.txt に何かメッセージが設定されているときはそれを表示します。

システムサービスまたはアプリケーションは、特別な目的でこのようなロックされたアカウントを作成します。/etc/passwd の中でログインシェルに **nologin** が指定されたユーザーを探すと、どんなユーザーがいるか見ることができます。次のように、特定のユーザーアカウントをロックすることもできます:

```
$ sudo usermod -L dexter
```

これらのアカウントはシステムに残りますが、ログインはできなくなります。-U オプションを使うと、ロックを解除できます。
慣例として、ユーザーが組織を離れるとき、または長期の休職中は、ユーザーのアカウントをロックします。アカウントロックの別の方法として **chage** を使用してアカウントの有効期限を変更する方法もあります。

```
$ sudo chage -E 2014-09-11 morgan
```

アカウント有効期限を過去の日付に設定すると、アカウントが無効となります。

30.4 パスワード

ユーザー ID と /etc/passwd

```
beav:x:1000:1000:Theodore Cleaver:/home/beav:/bin/bash
rsquirrel:x:1001:1001:Rocket J Squirrel:/home/rsquirrel:/bin/bash
```

- /etc/passwd ファイルには、1 ユーザーあたり 1 レコード（1 行）の情報が含まれる。各レコードのフィールドはコロン (:) で仕切られる:
  - username: ユーザーのユニーク名
  - password: ハッシュ化データ (/etc/shadow 未使用時) またはプレースホルダー (/etc/shadow 使用時は "x" と表示される）
  - UID: ユーザー識別番号
  - GID: ユーザーのプライマリーグループ識別子
  - comment: コメント領域、通常はユーザーの実名
  - home: ユーザーのホームディレクトリのディレクトリパス名
  - shell: ログイン時に呼び出すシェルの絶対パス名
- UID は UID_MIN = 1000 からスタートする

ほとんどの **Linux** ディストリビューションでは、**UID** が 1000 未満の値のを持つアカウントはシステムに属する特別なユーザーとみなされます。通常ユーザーのアカウントは 1000 から始まります。実際には、(1000 という実値ではなく）/etc/login.defs 内に UID_MIN として定義された値が使われます。

歴史的にみると **Red Hat** から派生したディストリビューションは、1000 ではなく UID_MIN=500 を使用していましたが **RHEL 7** 以降からは、一般的な値である 1000 になりました。

useradd の使用時に **UID** が指定されていない場合、システムは UID_MIN からインクリメンタルに ID を割り当てます。

さらに、各ユーザーはデフォルトで **UID** と同じ番号である **プライマリグループ ID** を取得します。これらは **ユーザープライベートグループ**（UPG）と呼ばれることもあります。

/etc/passwd、/etc/group、/etc/shadow を直接編集することは避けてください。代わりに **usermod** などの適切なユーティリティを使用してください。

30.5 /etc/shadow



/etc/passwd のデフォルトのパーミッションは 644 (-rw-r--r--) なので、誰でもこのファイルを読むことができてしまいます。システムプログラムとユーザーアプリケーションがこのファイルの情報を読み取る必要があるため、残念ながらこのパーミッションにする必要があるのです。システムプログラムが root 権限で実行されていないからですが、しかしどんな状況でも、このファイルの変更を許されているのは root のみです。

懸念されるのは、ハッシュされたパスワードそのものです。それが /etc/passwd にあれば、誰でもハッシュ化されたパスワードのコピーを作成して **Crack** や **John the Ripper** などのユーティリティを使用してハッシュ化されたパスワードから元のテキストパスワードを推測できてしまうからです。これはセキュリティリスクです！

/etc/shadow のパーミッションの設定は 400 (-r--------) です。つまりこのファイルにアクセスできるのは root のみです。これにより、誰かがハッシュされたパスワードを収集するのが難しくなっています。

説得力のある正当な理由がない限り /etc/shadow ファイルを使用すべきです。

/etc/shadow II

- /etc/shadow ファイルには１ユーザーにつき１つ（１行）のレコード
- 各レコードのフィールドはコロン（：）で仕切られている
  - username: ユニークなユーザー名
  - password: パスワードのハッシュ値 (sha512)
  - lastchange: 最後変更日（エポック日起点の日数）
  - mindays: パスワードを変更できるまでの日数
  - maxdays: パスワードを変更しなくてよい日数
  - warn: 有効期限切れをユーザーに警告開始する日数
  - grace: 期限切れからアカウントが無効になるまでの日数
  - expire: アカウントが無効になる日（エポック日起点の日数）
  - reserved: 予約フィールド

/etc/shadow には次のようにユーザーごとに１つのレコード（１行）があります:

/etc/shadow

```
daemon:*:16141:0:99999:7:::
.....
beav:$6$iCZyCnBJH9rmq7P.$RYNm10Jg3wrhAtUnahBZ/mTMg.RzQE6iBXyqaXHvxxbK\
   TYqj.d9wpoQFuRp7fPEE3hMK3W2gcIYhiXa9MIA9w1:16316:0:99999:7:::
```

各レコードのユーザー名は /etc/passwd のユーザー名との完全一致が必要で、さらに同じ順序で表示される必要があります。
すべての日付は 1970 年 1 月 1 日（**エポック日**）からの日数として保存されます。
“6” の後に８文字のソルト値、その後に $ と 88 文字（sha512）のパスワードハッシュが続きます。

30.6 パスワード管理



デフォルトでは選択されたパスワードが pam_cracklib.so でチェックされ、より安全なパスワードを推薦されることがあります。

一般ユーザーが自分のパスワードを変更する場合は:

```
$ passwd
Changing password for clyde
(current) UNIX password: <clyde's password>
New UNIX password: <clyde's-new-password>
Retype new UNIX password: <clyde's-new-password>
passwd: all authentication tokens updated successfully
```

root が一般ユーザーのパスワードを変更する場合、root は現在のパスワードの入力を求められないことに留意してください。

```
$ sudo passwd kevin
New UNIX password:  <kevin's-new-password>
Retype new UNIX password: <kevin's-password>
passwd: all authentication tokens updated successfully
```

一般ユーザーは、短すぎるものや辞書にある単語など、不適切なパスワードを設定することが許されていません。ただし root には許可されています。

30.7 パスワードの有効期限の設定・確認

パスワードの有効期限の設定・確認（chage）

- chage 使って管理する

```
$ chage [-m mindays] [-M maxdays] [-d lastday] \
      [-I inactive] [-E expiredate] [-W warndays] user
```

- 例:

```
$ sudo chage -l stephane
$ sudo chage -m 14 -M 30 kevlin
$ sudo chage -E 2012-4-1 isabelle
$ sudo chage -d 0 clyde
```

一般に、パスワードを定期的に変更することは重要と考えられています。こうすることで、もしパスワードがクラックされても、それを利用して侵入できてしまう期間を短くできます。また、使われなくなったアカウントのロックにもなります。しかし欠点もあります。このポリシーを煩わしく思ったユーザーが、頻繁に変更されるパスワードを忘れないように目につくところに書き留めてしまい、結果的にパスワードが盗まれやすくなることです。

chage ユーティリティを使ってパスワード更新を管理します:

```
chage [-m mindays] [-M maxdays] [-d lastday] [-I inactive] [-E expiredate] [-W warndays] user
```



図 30.2: **chage の利用**

chage を利用できるのは root ユーザーだけです。ただし 1 つの例外があります。`-l` オプションを指定して実行すれば、誰でも自身のアカウントやパスワードの有効期限を確認できます。

ユーザーに次回ログイン時にパスワード変更をを強制的するには:

```
$ sudo chage -d 0 USERNAME
```

30.8 制限付きシェルとアカウント **

制限付きシェル

Linux には **制限付きシェル**というものがあり、それを使うには:

```
$ bash -r
```

- **cd** を使用したディレクトリ変更を禁止する
- SHELL、ENV、PATH など環境変数の設定を禁止する
- / で始まる絶対パスまたはコマンド名の指定を禁止する
- 入力や出力のリダイレクトを制限する
- /bin/rbash という名前で /bin/bash へのシンボリックリンクを作成しそれを/etc/passwd 内でログインシェルに指定することで **制限付きアカウント** を有効化できる、制限付きアカウントは次に説明する

rbash は安全ではありません！

実際には非常に簡単に **rbash** による制限は回避できてしまうので **SELinux** などの最新技術を利用した方がはるかに堅牢です。ここで示した方法に遭遇した時に備えるためだけに説明しました。

制限付きシェルを無効化にする方法を紹介したハッカーガイドが https://www.metahackers.pro/breakout-of-restricted-shell/ に公開されています。また別のものが https://www.exploit-db.com/docs/english/44592-linux-restricted-shell-bypass-guide.pdf にあります。

制限付きシェルを使用すれば安全、という誤った安心感を与えてしまうかもしれません。

制限付きアカウント

ユーザーにアクセス権限を付与する必要がある時でも、権限許諾範囲を限定したいケースがある。そのような時に、制限付きユーザーアカウントの設定が解決策になる可能性がある。

- 制限付きシェルを利用する
- 利用可能なシステムプログラムや、ユーザーアプリケーションを制限
- システムリソースを制限する
- アクセス時間を制限する
- アクセス場所を制限する

制限付きシェル起動時には、制限なしに $HOME/.bash_profile が実行されます。(制限をつけた?) ユーザーが、ホームディレクトリへの書込みないし実行権限を持ってはいけないのはこのためです。

このようなアカウントを設定する場合、誤って環境変数 PATH にシステムディレクトリを追加しないでください。追加してしまうと、制限ユーザーが制限されていないシェルなどの他のシステムプログラムを実行できるようになってしまいます。

制限付きシェルは `/bin/bash -r` を使用して起動できます。残念ながら、この時に `/etc/passwd` ファイルにフラグが指定されていない（= ログインシェルに rbash が指定されていない）可能性があります。もし制限付きシェルを使用するなら、シェルインタープリタにシンボリックリンク、またはハードリンクファイルを設定する必要があります。

コマンドラインまたはスクリプトから `/bin/bash -r` を使用して制限付きシェルを呼び出すことができます。ただし `/etc/passwd` ファイルのシェルフィールドには /bin/bash -r と記述できていない場合があります。この問題を解決する簡単な方法は、次のいずれかを実行することです。

```
$ cd /bin ; sudo ln -s bash rbash
$ cd /bin ; sudo ln    bash rbash
$ cd /bin ; sudo cp    bash rbash
```

そして `/bin/rbash` を `/etc/passwd` でログインシェルに指定します。

このようなアカウントを設定するときは、システムディレクトリを誤って環境変数 PATH に追加しないでください。追加すると、制限ユーザーが制限のないシェルなど他のシステムプログラムを実行できてしまいます。

制限付き（Restricted）アカウントは **limited アカウント** とも呼ばれます。

30.9 root アカウント

root アカウント

root の利用は、極めて限定的かつ例外的とすべきである

- 真に必要な場面に限定して root でログインする
  - 間違いを防ぐため
  - セキュリティ脆弱性を回避するため
- 権限を取得するには **su** を利用させる必要がある
  - 直接 root ではログインさせない
  - 誰が root になったかの履歴を監査できるよう設定する
- セキュリティ担保のために **sudo** を利用する

root アカウントの利用は管理目的だけに使用するもので、真に必須な場合のみに限定とし、決して通常アカウントとして使用してはいけません。root アカウントで何らかのミスをすると、整合性と安定性の両方にとって、そしてシステムセキュリティ面でも、致命的な犠牲を払うことになるリスクがあります。

セキュリティ上の理由から、デフォルトではネットワークを介した root ログインは禁止されています。**ssh** を使った **Secure Shell** ログインは許可されます。**ssh** は `/etc/ssh/sshd_config` と **PAM** (Pluggable Authentication Modules) で設定されます。**PAM** は pam_securetty.so モジュールと関連する `/etc/securetty` ファイルを使います。root ログインは `/etc/securetty` にリストされているデバイスからのみ許されます。

通常すべての root アクセスは **su** または **sudo** を使って行うことをお勧めします。（**sudo** を使うとすべての root アクセスを追跡調査できます。）一部（**Ubuntu** など）のディストリビューションは、デフォルトでは root アカウントで直接ログインすることを禁止しています。

PAM を使用して **su** で root になることができるユーザーを制限することもできます。また root として実行されたすべてのコマンドのログを記録するように **auditd** を設定することも重要です。

30.10 SSH



同一ユーザー名、または別ユーザー名でネットワーク経由でリモートシステムにログインする必要はよくあります。または、リモートマシンとの間でファイルを転送する必要があります。いずれの場合でも、傍受されない安全な方法でこれを行う必要があります。

SSH (**S**ecure **SH**ell) はこの目的を実現するためのものです。**SSH** は、強力なアルゴリズムに基づく暗号化を使用します。適切な **ssh** パッケージがシステムにインストールされていれば **ssh** を使うためにこれ以上セットアップする必要はありません。
リモートシステムにサインインするには：

```
$ ssh farflung.com
```

```
1 student@farflung.com's password: (type here)
```

farflung.com に `student` アカウントがあると仮定しています。別のユーザーとしてログインする場合：

```
$ ssh root@farflung.com
$ ssh -l root farflung.com
```

あるシステムから別のシステムにファイルをコピーするには:

```
$ scp file.txt farflung.com:/tmp
$ scp file.tex student@farflung.com/home/student
$ scp -r some_dir farflung.com:/tmp/some_dir
```

(スペース節約のためパスワード要求のやり取りは省略しました。)

SSH 設定ファイル

- ホームディレクトリには:
  - `id_rsa`: ユーザーの **秘密鍵**
  - `id_rsa.pub`: ユーザーの **公開鍵**
  - `authorized_keys`: ログインを許可した公開鍵
  - `known_hosts`: これまでにログインを認めたホスト
  - `config`: 色々なオプションを指定するファイル
- **ssh-keygen** で公開鍵と秘密鍵を生成する

SSH をいろいろ設定すると、さらに高度な使いかたができます。特に、パスワードなしのログインが可能になります。ユーザー固有の設定ファイルは、各ユーザーのホームディレクトリにある非表示の `.ssh` ディレクトリに作成されます。

ユーザーは最初に **ssh-keygen** を使用して、秘密鍵および公開鍵を生成する必要があります:

```
$ ssh-keygen
Generating public/private rsa key pair.
...
```

秘密鍵は **決して** 誰とも共有してはいけません！ 一方で公開鍵は、パスワード無しアクセスをしたいどんなマシンにも渡すことができます。
そして、あなたの `authorized_keys` ファイルには、あなたがパスワード無しアクセスを求められて認めた全ての公開鍵が追加されていきます。（色々なリモートシステムに対する）**ssh** アクセスを繰り返していくと `known_hosts` の内容も徐々に積み上がっていきます。システムが **ssh** でログインしようとするリモートユーザーの変化を検出すると、あなたにワーニングを出してアクセスを拒否するチャンスを与えてくれます。

`authorized_keys` には、（リモートアクセスに関わった）ユーザーとマシンに関する情報が含まれます:

```
$ cat authorized_keys
ssh-rsa AAAAB3Nza........student@debian
ssh-rsa AAAAB3Nza........student@fedora
```

一方で `known_hosts` にはコンピュータノードに関する情報だけが記録されます:

```
$ cat known_hosts
x7 ecdsa-sha2-nistp256 AAAAE2VjZHNhLXN..... M=
192.168.1.200 ecdsa-sha2-nistp256 AAAA..... bcs=
```

`man ssh_config` にアクセスして **ssh** のコンフィギュレーションファイルに設定するオプションを確認してください。

リモートからのグラフィカルログイン

- フルグラフィカルデスクトップで、リモートマシンにログインする
- しばしば **VNC** (**V**irtual **N**etwork **C**omputing) が使われる
- よく使われる実装は **tigervnc** である
- 使い方は簡単です:
  - リモートマシン側で (例えば some_machine):
    ```
    $ vncserver
    ```
  - ローカルマシン側で:
    ```
    $ vncviewer -via server student@some_machine localhost:2
    ```

これをテストするには、まず **vnc** パッケージがインストールされていることを確認してください:

```
$ which vncserver vncviewer
```

```
1 /usr/bin/vncserver
2 /usr/bin/vncviewer
```

これらのプログラムが見つからない場合は、次のようにしてインストールする必要があります。

```
$ sudo [dnf|zypper|apt-get] install tigervnc*
```

適切なパッケージ管理システムを利用してください。(具体的なパッケージ名は **Linux** ディストリビューションによってまちまちです。そのため、ここでは特定の名前をあげていません。結果的に必要がないものまでインストールすることになるかもしれませんが、パッケージのサイズは大きくはありません。) 一般ユーザーとしてサーバーを起動するには:

```
$ vncserver
```

以下の方法でテストができます:

```
$ vncviewer localhost:2
```

(現在実行しているものや、マシンの構成方法によっては、1、3、4 ... など、2 以外の数字を指定する必要があるかもしれません。) リモートマシンから表示するには、少し変える必要があります:

```
$ vncviewer -via student@some_machine localhost:2
```

“color profile” のために認証する必要があるという奇妙なメッセージが表示され、パスワードが機能しない場合は次のようにサーバーマシンの **colord** デーモンを強制終了する必要があります。

```
$ sudo systemctl stop colord
```

これはバグで (機能ではない)、一部のディストリビューションおよび一部のシステムでのみ出現し、原因はよくわかっていません。

30.11 演習

デモ教材ビデオ

using_user_accounts_demo.mp4

課題 30.1: ユーザー アカウントの処理

1. /etc/passwd と /etc/shadow について、それぞれのファイルの、特に一般のユーザーのアカウントの、各フィールドを比較します。同じものと違うものは何でしょうか。
2. **useradd** を使って user1 アカウントを作成します。
3. ssh で user1 としてログインします。以下のようにします:

   ```
   $ ssh user1@localhost
   ```

   user1 のパスワードが必要なので、これは失敗します; まだ接続できていません。
4. user1 のパスワードを user1pw に設定し改めて user1 としてログインします。
5. /etc/shadow ファイルの新しく作られたレコードを調べます。
6. /etc/default/useradd ファイルを見て、現在のデフォルトが何であるかを確認します。また /etc/login.defs ファイルも調べます。
7. デフォルトのシェルとして **Korn** シェル（**ksh**）を利用する新しいユーザーアカウントuser2 を作成します。（**/bin/ksh** が無ければインストールするか、代わりに **C**（**/bin/csh**）を使ってください。）パスワードをuser2pw に設定します。
8. /etc/shadow を見てください。user1 アカウントの現在の有効期限はいつですか？
9. **chage** を使って、user1 の有効期限を 2013 年 12 月 1 日に設定してください。/etc/shadow を見てください。新しい有効期限はいつですか？
10. **usermod** を使って、user1 アカウントをロックします。/etc/shadow を見て user1 のパスワードに関して、変更された点を見つけてください。パスワードを userp1 にリセットして、この演習を終えてください。

解 30.1

1. ```
   $ sudo grep student /etc/passwd /etc/shadow
   ```

   ```
   /etc/passwd:student:x:1000:100:LF Student:/home/student:/bin/bash
   /etc/shadow:student:$6$jtoFVPICHhba$iGFFUU8ctrtOGoistJ4/30DrNLi1FS66qnn0VbS6Mvm
     luKI08SgbzT5.IcOHo5j/SOdCagZmF2RgzTvzLb11H0:16028:0:99999:7:::
   ```

   （studentの代わりに、一般のユーザーの任意の名前が利用できます。）ユーザー名欄だけがマッチします。
2. ```
   $ sudo useradd user1
   ```
3. ```
   $ ssh user1@localhost
   ```

   ```
   user1@localhost's password:
   ```

   最初に **sshd** サービスを起動しなければなりません:

   ```
   $ sudo service sshd restart
   ```

   または

   ```
   $ sudo systemctl restart sshd.service
   ```
   ```

4. `$ sudo passwd user1`

```
Changing password for user user1.
New password:
```

5. `$ sudo grep user1 /etc/passwd /etc/shadow`

```
/etc/passwd:user1:x:1001:100::/home/user1:/bin/bash
/etc/shadow:user1:$6$OBE1mPMw$CIc7urbQ9ZSnyiniVOeJxKqLFu8fz4whfEexVem2
   TFpucuwRN1CCHZ19XGhj4qVujslRIS.P4aCXd/y1U4utv.:16372:0:99999:7:::
```

6. 例として **RHEL** や **openSUSE** システムで説明します:

   `$ cat /etc/default/useradd`

```
# useradd defaults file
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

   `$ cat /etc/login.defs`

```
....
```

   ２つ目のファイルは長いので省略します。自分のシステム上で確認してください。

7. `$ sudo useradd -s /bin/ksh user2`
   `$ sudo passwd user2`

```
Changing password for user user2.
New password:
```

8. `$ sudo grep user1 /etc/shadow`

```
user1:$6$OBE1mPMw$CIc7urbQ9ZSnyiniVOeJxKqLFu8fz4whfEexVem2TFpucuwRN1CCHZ
   19XGhj4qVujslRIS.P4aCXd/y1U4utv.:16372:0:99999:7:::
```

   有効期限が設定されていません。

9. `$ sudo chage -E 2013-12-1 user1`
   `$ sudo grep user1 /etc/shadow`

```
    user1:$6$OBE1mPMw$CIc7urbQ9ZSnyiniVOeJxKqLFu8fz4whfEexVem2TFpucuwRN1CCHZ
  19XGhj4qVujslRIS.P4aCXd/y1U4utv.:16372:0:99999:7::16040:
```

10. `$ sudo usermod -L user1`

    `$ sudo passwd user1`

課題 30.2: 制約付きシェルとアカウント

1. 現在の端末で、制約付きシェルを実行します：

   ```
   $ bash -r
   ```

   パスのリセットやディレクトリの変更などの基本的な操作を実行してみてください。

2. 機能が制約されたアカウントを作成し、どのように動作が制約されているかを調べます。その後、消去します。

注目

- いくつかのディストリビューション、特に **Ubuntu** ベースのバージョンでは、正しく動作しないため本演習ができません。**RedHat** ベースのディストリビューションでは問題無く動作します。
- 以前に指摘したように制約付きシェルはセキュアではなく、より良い方法があるため不要となっています。しかしながら、まだ実行することができるためここで説明しました。

解 30.2

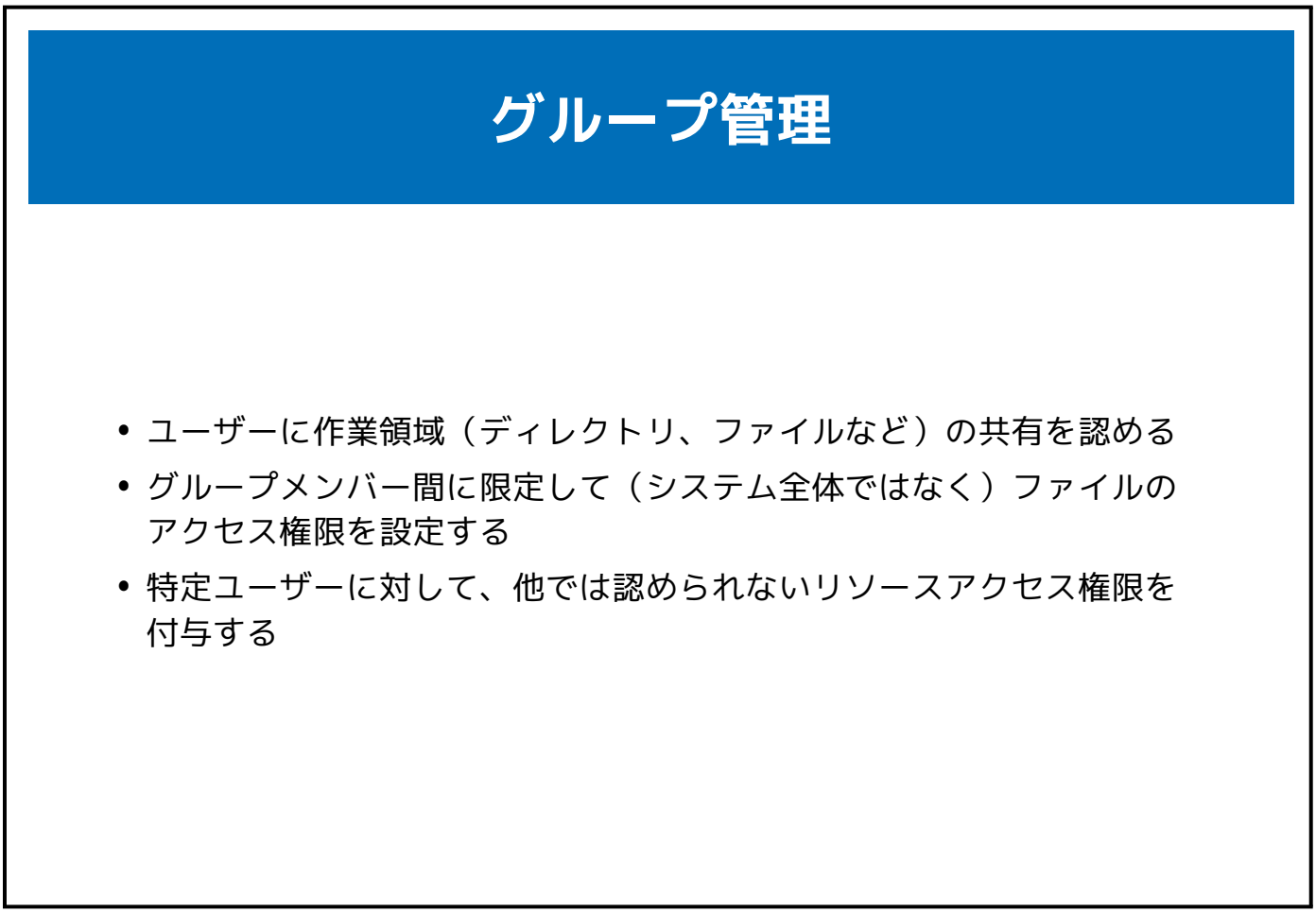
1.
```
c8:/tmp>bash -r
c8:/tmp>cd $OME
```
```
rbash: cd: restricted
```
```
c8:/tmp>PATH=$PATH:/tmp
```
```
rbash: PATH: readonly variable
```
```
c8:/tmp>exit
```
```
exit
```

2.
```
c8:/home/coop>sudo ln /bin/bash /bin/rbash
c8:/home/coop>sudo useradd -s /bin/rbash fool
c8:/home/coop>sudo passwd fool
```
```
Changing password for user fool.
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: all authentication tokens updated successfully.
```
```
c8:/home/coop>sudo su - fool
```
```
fool@c8~]$ c d /tmp
+-rbash: /usr/libexec/pk-command-not-found: restricted: cannot specify `/' in command names
```
```
[fool@c8~]$ cd /tmp
```
```
-rbash: cd: restricted
```
```
[fool@c8~]$ PATH=$PATH:/tmp
```
```
-rbash: PATH: readonly variable
```
```
[fool@c8~]$ exit
```
```
logout
```
```
c8/home/coop>sudo userdel -r fool
c8/home/coop>sudo rm /bin/rbash
```

章 31

グループ管理

31.1 グループ



Linux システムでは **グループ** と呼ばれるユーザーの集合を形成し、メンバに複数の共通の目的を持たせることができます。さらに特定のファイルとディレクトリを共有しいくつかの特権を共有します。このようにシステム上の他のユーザーとは分けられた集団によって **世界（the world）**が形成されます。グループは共同プロジェクトで非常に役立ちます。

ユーザーは１つ以上の **グループ**に属します。

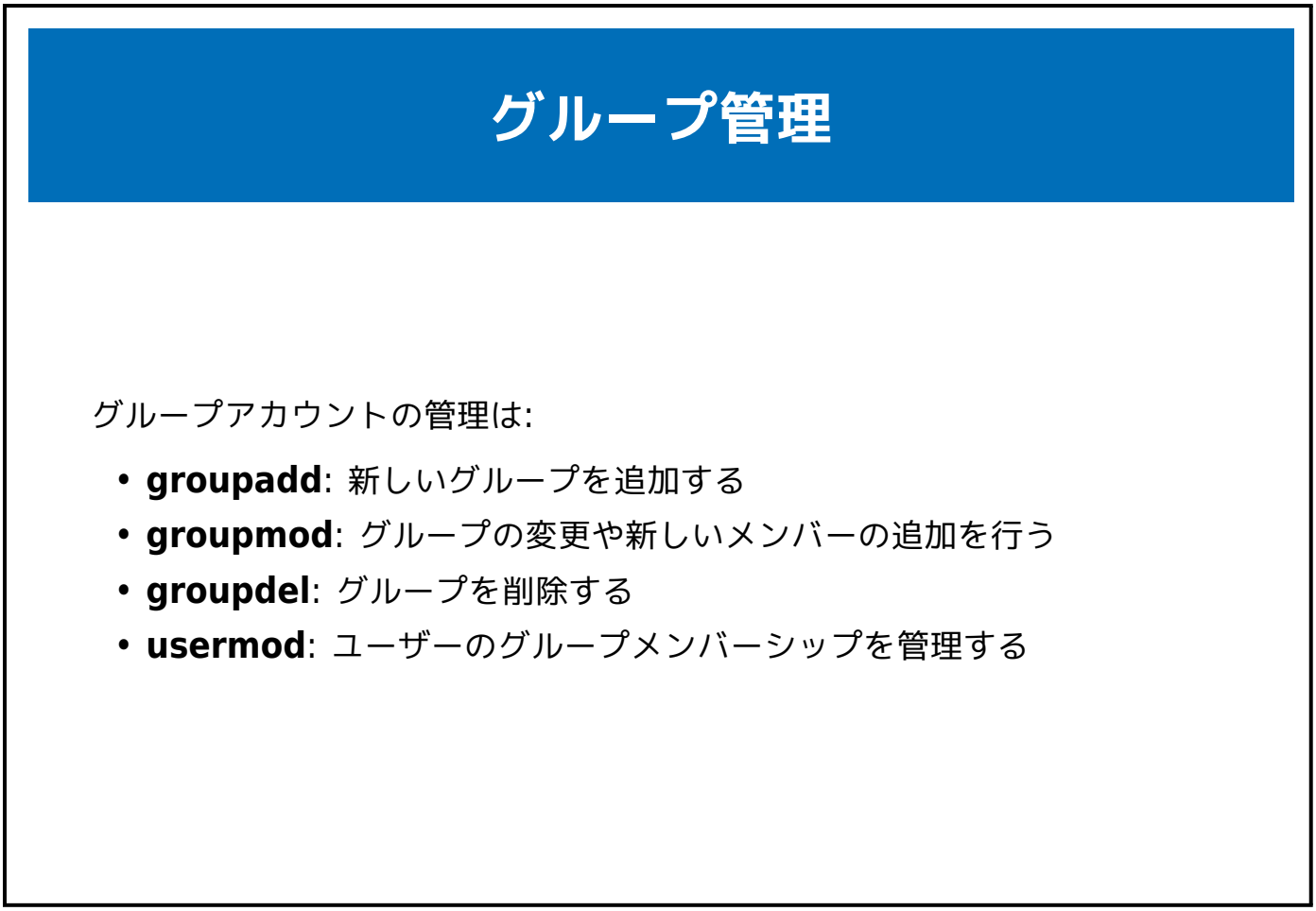
グループは `/etc/group` で定義されます。このファイルはユーザーに対して `/etc/passwd` が持っているのと同じ役割を果たします。ファイルの各行は次のようになります:

/etc/group

```
....
groupname:password:GID:user1,user2,...
....
```

- `groupname` はグループの名前です。
- `password` はパスワードのプレースホルダーです。グループパスワードは `/etc/gshadow` がある時だけ設定されます。
- `GID` はグループ識別子です。0 から 99 はシステムグループに属します。100 から `GID_MIN` (`/etc/login.defs` で定義され通常は `UID_MIN` と同じ) は特別なグループと見なされます。`GID_MIN` より大きな値は **UPG** (User Private Groups) となります。
- `user1,user2` はグループのメンバーとなるユーザーをカンマ区切りしたリストです。ユーザーにとってこのグループがプリンシパルグループ（つまり自分で作成したグループ）の場合は、ユーザーをここにリストする必要はありません。

31.2 グループ管理



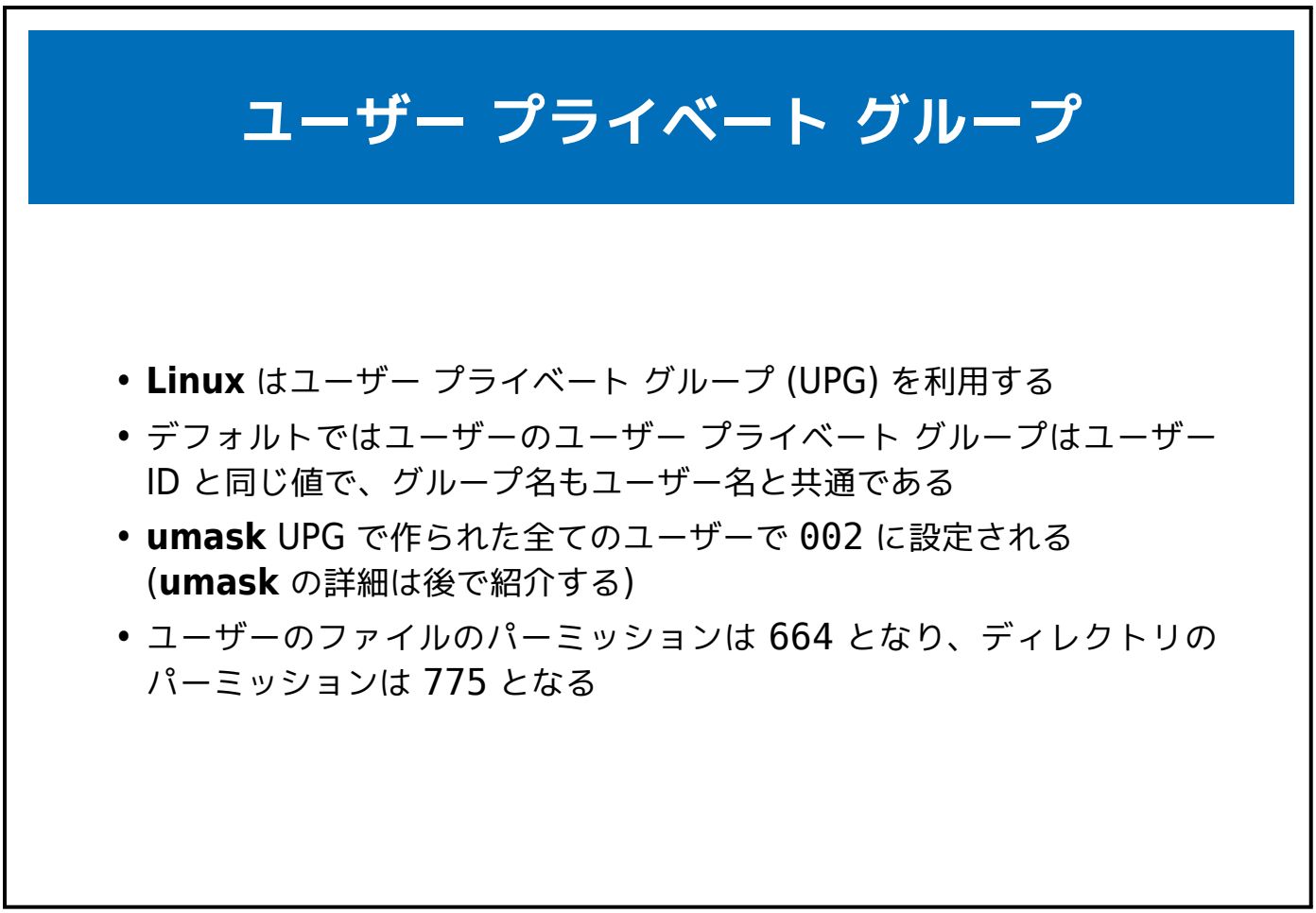
これらのグループ操作ユーティリティは `/etc/group` と（存在する場合は）`/etc/gshadow` を変更します。
この操作は root だけが実行できます。

使用例:

```
$ sudo groupadd -r -g 215 staff
$ sudo groupmod -g 101 blah
$ sudo groupdel newgroup
$ sudo usermod -G student,group1,group2 student
```

usermod -G コマンドを利用する時には、十分注意する必要があります。引数グループリストには、変更するものだけでなく完全なグループリストの指定が必要です。そうしないと、指定されていないグループは削除されてしまいます！新しいメンバを追加するときでも、既存のグループメンバシップが保持される -a オプションの使用が安全です。

31.3 ユーザー プライベート グループ



Linux は **ユーザー プライベート グループ（UPG）**機能を利用します。

UPG の背景にある考え方は、各ユーザーに自分のグループを持たせることです。ただし、UPG は個人専用であるとは **保証されません**。`/etc/group` で定義したプライベートグループに、他のメンバが追加できてしまうからです。

デフォルトでは **useradd** で作成されたアカウントを持つユーザーは、プライマリ GID = UID となっていて、グループ名もユーザー名と同じです。

`/etc/profile` で指定されているとおり UPG で作成されたすべてのユーザーの **umask** は `002` に設定されます。したがって、この方式ではユーザーのファイルはパーミッション 664 (`rw-rw-r----`)、ディレクトリは 775 (`rwxrwxr-x`) で作成されます。（**umask** は次のセクションで説明します。）

31.4 グループのメンバシップ

グループのメンバシップ

Linux ユーザーは:

- プライマリーグループを 1 つ持つ
  - プライマリーグループは `/etc/passwd` に定義されている
  - `/etc/group` にもリストされる
- 0 から 15 個のセカンダリーグループに所属することができる
- 以下のどちらかのコマンドで所属するグループを確認できる

```
$ groups [user1 user2 ...]
$ id -Gn [user1 user2 ...]
```

- 引数を与えないと、現在のユーザーについてレポートする（どちらのコマンドも共通）

プライマリグループの GID は、ユーザーがファイルまたはディレクトリを作成する時に常に使用されます。それ以外のセカンダリグループのメンバーシップは、ユーザーにパーミッションを追加で付与します。

グループのメンバシップは **groups** または **id -Gn** コマンドのいずれかを実行することで識別できます。

デフォルトグループは、ディストリビューションとインストールに依存します。

CentOS では:

```
[student@CentOS ~] groups
student
```

Ubuntu では:

```
student@ubuntu ~$ groups
student adm cdrom sudo dip plugdev lpadmin sambashare libvirt
```

31.5 演習

課題 31.1: グループに関連する作業

1. ２つの新しいユーザーアカウント（下記に示すように rocky と bullwinkle）を作成します。ホームディレクトリが作成されていることを確認します。
2. ２つの新しいグループ、friends と bosses （GID を 490 として）を作成します。/etc/group を確認してください。各グループに割り当てられた GID を確認してください。
3. ２つの新しいグループに rocky を追加してください。
   グループ friends に winkle を追加してください。
   /etc/group の変化を確認してください。
4. rocky としてログインしてください。ディレクトリ somedir を作成し、グループ所有者を bosses に設定します。（次セッションで説明する **chgrp** を使います。）
   （rockyのホームディレクトリにある全てに対して、実行権限を追加する必要があるでしょう。）
5. bullwinkle としてログインし、**touch** コマンドを使ってsomefile というファイルを /home/rocky/somedir に作成してください。実行できましたか？できませんね。それは、グループ所有権とディレクトリに chmod a+x を実行したからです。
6. bosses グループに bullwinkle を追加し、再実行してください。新メンバー追加に効力を持たせるために、一旦ログアウトしてログインする必要があります。以下を実行してください:

解 31.1

1. ```
   $ sudo useradd -m rocky
   $ sudo useradd -m bullwinkle
   $ sudo passwd rocky
   ```

   ```
   Enter new UNIX password:
   Retype new UNIX password:
   passwd: password updated successfully
   ```

   ```
   $ sudo passwd bullwinkle
   ```

   ```
   Enter new UNIX password:
   Retype new UNIX password:
   passwd: password updated successfully
   ```

   ```
   $ ls -l /home
   ```

   ```
   total 12
   drwxr-xr-x  2 bullwinkle bullwinkle 4096 Oct 30 09:39 bullwinkle
   drwxr-xr-x  2 rocky      rocky      4096 Oct 30 09:39 rocky
   drwxr-xr-x 20 student    student    4096 Oct 30 09:18 student
   ```

2. ```
   $ sudo groupadd friends
   $ sudo groupadd -g 490 bosses
   $ grep -e friends -e bosses /etc/group
   ```

   ```
   friends:x:1003:
   bosses:x:490:
   ```

3.
```
$ sudo usermod -G friends,bosses rocky
$ sudo usermod -G friends bullwinkle
```

```
$ grep -e rocky -e bullwinkle /etc/group
```

```
rocky:x:1001:
bullwinkle:x:1002:
friends:x:1003:rocky,bullwinkle
bosses:x:490:rocky
```

```
$ groups rocky bullwinkle
```

```
rocky : rocky friends bosses
bullwinkle : bullwinkle friends
```

4.
```
$ ssh rocky@localhost
$ cd ~
$ mkdir somedir
$ chgrp bosses somedir
$ ls -l
```

```
total 16
-rw-r--r-- 1 rocky rocky  8980 Oct  4  2013 examples.desktop
drwxrwxr-x 2 rocky bosses 4096 Oct 30 09:53 somedir
```

```
$ chmod a+x .
```

5.
```
$ ssh bullwinkle@localhost
$ touch /home/rocky/somedir/somefile
```

```
touch: cannot touch /home/rocky/somedir/somefile: Permission denied
```

```
$ exit
```

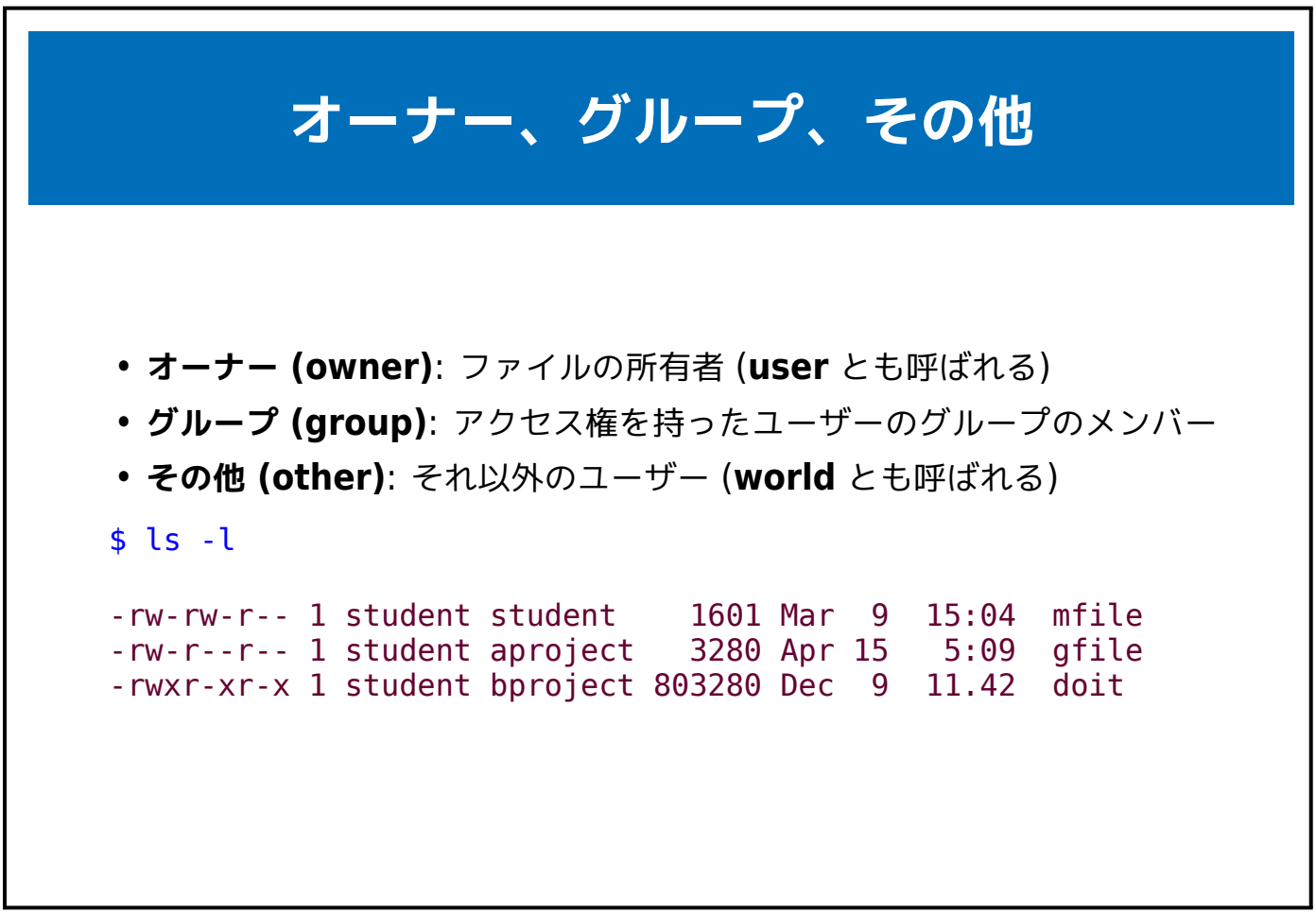
6.
```
$ sudo usermod -a -G bosses bullwinkle
$ ssh bullwinkle@localhost
$ touch /home/rocky/somedir/somefile
$ ls -al /home/rocky/somedir
```

(ファイルのオーナーを記録する)

章 32

ファイルのパーミッションと所有権

32.1 ファイルのパーミッションと所有権



以下を実行すると:

```
$ ls -l a_file
-rw-rw-r-- 1 coop aproject 1601 Mar 9 15:04  a_file
```

先頭の１文字（は、ファイルオブジェクトの種類を示しています。）の後に続く９文字に、ファイルを利用するであろう人に付与される **アクセス権** が表現されています。まず、利用者が３つのカテゴリー（**オーナー**、**グループ**、**その他** (これは **world** と同じ)）に分類され、それぞれに対して３文字が付与されています。

上記の出力リストでは、ユーザーは `coop` でグループは `aproject` です。

32.2 ファイルのアクセス権

ファイルのアクセス権

- 以下の例のように -l を付けてファイルリストを取得すると:

```
$ ls -l /usr/bin/vi
-rwxr-xr-x. 1 root root 910200 Jan 30  2014 /usr/bin/vi
```

  それぞれの３文字の組み合わせ（２番文字目から１０文字目まで）には、以下のどれかが設定される:

  - **r**: 読み込みを許可
  - **w**: 書き込みを許可
  - **x**: 実行を許可

- 以下の順に認証を試し、最初に成功したものを基準に権限が付与される:
  - ファイル所有者か
  - グループのメンバーシップか
  - その他ユーザー（=world）のパーミションで許可されるか

パーミッションが許可されない場合は、これらの文字の代わりに -（ダッシュ）が表示されます。

これに加えて **setuid/setgid** パーミッションなどカテゴリごとに他の特殊パーミッションが存在します。

これらの **ファイル アクセス パーミッション** は **Linux** セキュリティシステムの根幹をなすものです。全てのファイルアクセス要求に対して、要求したユーザーの権限とアカウント識別情報がファイルの所有者のものと比較されます。

この **認証** は、アクセスを要求したユーザーが、利用者カテゴリー３種類のどれに属しているかに従って与えられます。
以下の順序で確認されます:

1. オーナー（＝ファイル所有者）からの要求であれば、ファイル所有者のパーミッションを使用されます。
2. 要求者がオーナー以外で、ファイルに付与されたグループに属している場合は、グループのパーミッションを適用します。
3. どちらにも該当しない場合は、その他ユーザー (World) のパーミッションが適用されます。

32.3 chmod, chown と chgrp

chmod

- ファイルパーミションの変更には **chmod** を使う
- 例えば「オーナ」と「その他」に実行権を付与し、「グループ」からは書き込み権を剥奪するには:

```
$ ls -l a_file
-rw-rw-r-- 1 coop coop 1601 Mar  9 15:04 a_file
$ chmod uo+x,g-w a_file
$ ls -l a_file
-rwxr--r-x 1 coop coop 1601 Mar  9 15:04 a_file
```

u はユーザー（オーナー）の略、o はその他（world）の略、そして g はグループの略

- スーパーユーザーでない限り、パーミッションを変更できるのは自分が所有するファイルのみである
- パーミションは、しばしば 644 のように数字を使って指定される

パーミッションは、通常 8 進数ビットマップまたは記号で表現されます。
8 進数ビットマップでは `0755` のように、記号では `u+rwx,g+rwx,o+rx` のように記述します。

８進数ビットマップ表記

- **８進数表記**では、以下の各桁の設定の合計値を使う:
  - 4 （=3 ビット目）読み込み権を設定したい時
  - 2 （=2 ビット目）書き込み権を設定したい時
  - 1 （=1 ビット目）実行権を設定したい時

記号表記の構文は入力や覚えるのが煩雑なので、すべてのパーミッションを１ステップで設定できる８進数ビットマップ表記を使用することが多いのです。アルゴリズムは単純で、８進数１桁で各エントリーの３つのパーミション (r/w/x) を表現できます。

具体的には、7 なら読み取り／書き込み／実行、6 なら読み取り／書き込み、5 なら読み取り／実行、のパーミッションが付与されるという意味です。

chmod で 8 進数ビットマップ表記を利用する場合は３桁全てに値を指定します:

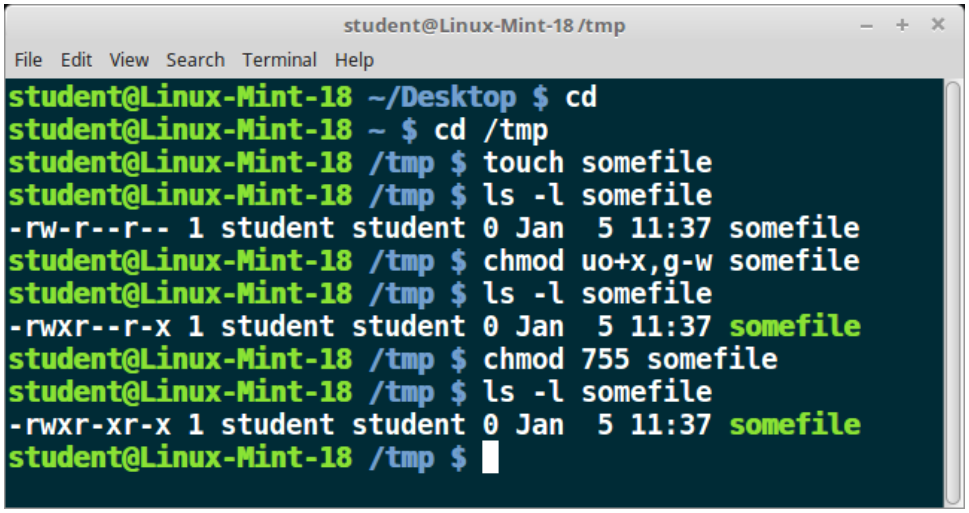


図 32.1: **chmod の利用**

chown と chgrp

- **chown** でファイル所有者を変更する（スーパーユーザーのみが可能）

```
$ sudo chown wally somefile
```

- **chgrp** でグループオーナーシップを変更する（グループメンバーのみが可能）

```
$ sudo chgrp cleavers somefile
```

- 両方を同時に変更する:

```
$ sudo chown wally:cleavers somefile
```

オーナーとグループはコロン（または、ピリオド）で並べて表記する

- -R オプションを使うと再帰的に変更する:

```
$ sudo chown -R wally:cleavers ./
$ sudo chown -R wally:wally subdir
```

ファイル所有者の変更は **chown** で行い、グループの変更は **chgrp** で行います。

ファイル所有者の変更ができるのはスーパーユーザーだけです。同様に、ファイルのグループオーナーシップを変更できるのはグループのメンバだけです。

-R オプションを使用すると、構文で指定した場所以下の全てのファイルとディレクトリに対してオーナーシップが変更されます。

32.4 umask

umask

- ファイルとディレクトリのデフォルトパーミションの設定に使われる
- **umask**: umask を設定するプログラムである
  - 付与しないパーミションを記述する
- umask のデフォルト値は `/etc/profile` に設定される
  - root の umask は 022
  - 誰でも設定できる

ファイル作成時に付与されるデフォルトパーミッションは、オーナー／グループ／その他ユーザー のすべてに 読み取り／書き込み許可 (0666) で、ディレクトリにはすべて 読み取り／書き込み／実行 (0777) です。しかし、以下を試すと:

```
$ touch   afile
$ mkdir adir
$ ls -l | grep -e afile -e adir
```

```
drwxrwxr-x 2  coop coop 4096 Sep 16 11:18 adir
-rw-rw-r-- 1  coop coop    0 Sep 16 11:17 afile
```

実際にはファイルのパーミッションは 664 で、ディレクトリのパーミションは 775 に変更されたことに気付くでしょう。これは、**umask** で定義された禁止パーミッション設定が反映された影響なのです。現在有効な **umask** 値を表示する方法は:

```
$ umask
```

```
0002
```

これはシステム管理者がユーザーに設定する最も一般的な値です。**umask** 値とファイル作成時のパーミション設定の論理積を計算することで最終的な値が決まります。つまり:

```
0666 & ~002 = 0664; つまり, rw-rw-r--
```

次のように **umask** コマンドを使用して、いつでも **umask** の値を変更できます。

```
$ umask 0022
```

32.5 ファイルシステムの ACL

ファイルシステムの ACL

- **POSIX ACL** (**A**ccess **C**ontrol **L**ists) は「オーナー／グループ／その他」という単純なユーザーカテゴリー区分を拡張する
- パーミション 777 を使わずに、ファイル／デフォルトを共有させる目的
- マウント時のオプション (`acl`) で利用可能になる
- インストール中のファイルシステム生成時に、デフォルトが設定される
- `getfacl/setfacl` を使って **ACL** の取得／設定を行う
- 例:

```
$ getfacl /home/stephane/file1
$ setfacl -m u:isabelle:rx /home/stephane/file1
$ setfacl -x u:isabelle    /home/stephane/file
```

オブジェクトまたはオブジェクトクラスにアクセスする、特定ユーザーまたはユーザーグループに、特別の権限を付与します。

Linux カーネルで **ACL** を有効にする時には、利用するファイルシステムにも **ACL** を実装する必要があります。最新の **Linux** ディストリビューションで使用される主要なファイルシステムには、全て **ACL 拡張** が組み込まれています。マウント時にオプション `-acl` を指定すれば **ACL** を利用できます。**ACL** のデフォルト設定は、システムインストール時に作成されます。

新しくファイルを作ると、ファイルが生成されたディレクトリから（設定されていれば）デフォルトの **ACL** が継承されます。また **mv** と **cp -p** は **ACL を継承** することにも注意してください。

ACL の削除は:

```
$ setfacl -x u:isabelle /home/stephane/file1
```

ディレクトリにデフォルトを設定するには:

```
$ setfacl -m d:u:isabelle:rx somedir
```

32.6 演習

デモ教材ビデオ

using_umask_demo.mp4

課題 32.1: chmod を使う

chmod を使うとき、8 進数表示か記号表記かのいずれかを使うことができます。記号表記について学びましょう。

パーミッションは直接指定するか、追加するか、削除するか、いずれの方法でも指定できます。例を見てみましょう:

```
$ chmod u=r,g=w,o=x afile
$ chmod u+w,g-w,o+rw afile
$ chmod ug=rwx,o-rw afile
```

各実行の後：

```
$ ls -l afile
```

を実行してパーミッションがどうなったかを見てみます。そして、他のオプションを指定してみましょう。

課題 32.2: umask

空のファイルを作成します:

```
$ touch afile
$ ls -l afile
```

```
1 -rw-rw-r-- 1 coop coop 0 Jul 26 12:43 afile
```

所有者とグループが読み書きできることがわかります。その他のユーザーは、読み出しのみ許可されています。

オペレーティングシステムレベルでは、ファイルやディレクトリを作成すると標準では所有者、グループが読み書きでき、**かつ** その他のユーザーは読む込だけが許可されています（0666)。**umask** コマンドで、標準値を変更できます。

引数なしで **umask** を実行すると現在の値を表示します:

```
$ umask
```

```
1 0002
```

ユーザーにとって一番良いと管理者が指定した値が設定されています。これは、ファイル作成時のパーミッションと組み合わせて使用されます:

```
0666 & ~002 = 0664; すなわち, rw-rw-r--
```

umask を変更してファイルを作成し、パーミッションがどうなるかを見てみましょう:

```
$ umask 0022
$ touch afile2
$ umask 0666
$ touch afile3
$ ls -l afile*
```

課題 32.3: アクセス コントロール リストを使う

1. ユーザー名をファイル名称としてファイルを作成し **getfacl** を実行し、プロパティを見ます。

2. 標準のプロパティで、新しいアカウントを作成します。(または、以前の演習で作成したものを利用します。)
3. ユーザーにログインし、最初に作成したファイルに一行追加します。これは失敗します。
4. **setfacl** を使って、新ユーザーの書き込みを許可します。
5. **setfacl** を使って、新ユーザーの書き込みを禁止します。
6. クリーンアップします。

解 32.3

端末を 2 つ開き、一つは自分のアカウント、他方は別のアカウントで作業します。

1. 端末 1 :

```
$ echo This is a file > /tmp/afile
$ getfacl /tmp/afile
```

```
getfacl: Removing leading '/' from absolute path names
# file: tmp/afile
# owner: coop
# group: coop
user::rw-
group::rw-
other::r--
```

2. 端末 2 :

```
$ sudo useradd fool
$ sudo passwd fool
...
```

3. 端末 2 :

```
$ sudo su - fool
$ echo another line > /tmp/afile
```

```
-bash: /tmp/afile: Permission denied
```

4. 端末 1 :

```
$ setfacl -m u:fool:rw /tmp/afile
$ getfacl /tmp/afile
```

```
getfacl: Removing leading '/' from absolute path names
# file: tmp/afile
# owner: coop
# group: coop
user::rw-
user:fool:rw-
group::rw-
mask::rwx
other::r--
```

端末 2 :

```
$ echo another line > /tmp/afile
```

5. 端末 1 :

```
$ setfacl -m u:fool:w /tmp/afile
```

端末 1 :

```
$ echo another line > /tmp/afile
-bash: /tmp/afile: Permission denied
```

6. クリーンアップ:

```
$ rm /tmp/afile
$ sudo userdel -r fool
```

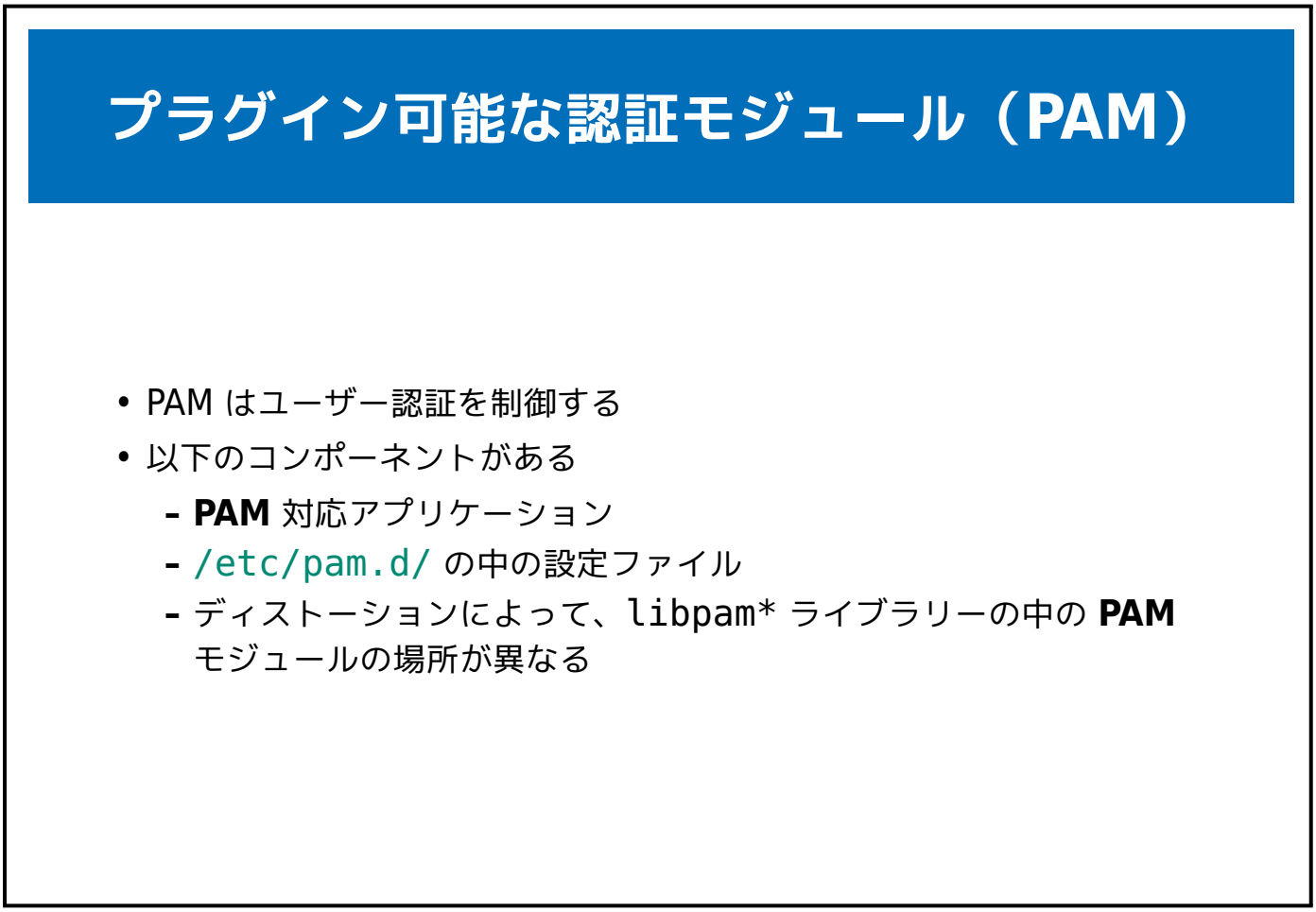
章 33

プラグイン可能な認証モジュール（PAM）

注目

** これらのセクションの一部または全体をオプション扱いとする場合があります。これらには補足資料、専門トピック、または高度な話題が含まれインストラクターが教室の状況や時間制約に応じてこれらの内容を紹介するかを判断します。

33.1 PAM (プラグイン可能な認証モジュール)



以前は、それぞれのアプリケーションが、別々にユーザーを認証していました。たとえば **su**、**login**、**ssh** は、それぞれがユーザーアカウントを認証して接続を確立させていました。

最新の **Linux** アプリケーションの多くは **PAM** (**P**luggable **A**uthentication **Modules**) を活用するようにコーディングされているので、**libpam** を使用した統一的な認証ができるようになっています。

libpam ライブラリの **PAM** モジュールは、柔軟で一貫性のある認証、パスワード、セッション、アカウントサービスを実現しています。

PAM 対応アプリケーション、またはサービスは、`/etc/pam.d` に用意された各アプリケーション毎の設定ファイルを編集することで、**PAM** の挙動を設定することができます。

33.2 認証プロセス

認証と構成ファイル

- ユーザーは **login**、**ssh**、**su** などの **PAM** 対応アプリケーションを起動
- 呼ばれたアプリケーションは **libpam** をコールする
- ライブラリーは **system-auth** を含め、起動すべき **PAM** モジュールが指定されたファイルを `/etc/pam.d` で探索する
- 選ばれたモジュールは、当該アプリケーションの認証設定ルールに従って認証を行う

`/etc/pam.d` 内のそれぞれのファイルは **サービス** に対応しており、ファイル内の（コメント行を除く）ひとつひとつの行に、認証ルールを指定していきます。ルールはスペースで区切られたトークンのリストです。最初の 2 つのトークンは大文字と小文字が区別されません:

```
type control module-path module-arguments
```

実例として **RHEL** システムの `/etc/pam.d/su` の内容を示します:

```
File  Edit  View  Search  Terminal  Help
c7:/tmp>cat /etc/pam.d/su
#%PAM-1.0
auth            sufficient      pam_rootok.so
# Uncomment the following line to implicitly trust users in the "wheel" group.
#auth           sufficient      pam_wheel.so trust use_uid
# Uncomment the following line to require a user to be in the "wheel" group.
#auth           required        pam_wheel.so use_uid
auth            substack        system-auth
auth            include         postlogin
account         sufficient      pam_succeed_if.so uid = 0 use_uid quiet
account         include         system-auth
password        include         system-auth
session         include         system-auth
session         include         postlogin
session         optional        pam_xauth.so
c7:/tmp>
```

図 33.1: **PAM 設定ファイル**

ここに階層構造（＝依存関係）があることに注意してください。su は system-auth などのローディングを要求します。

33.3 PAM を構成する

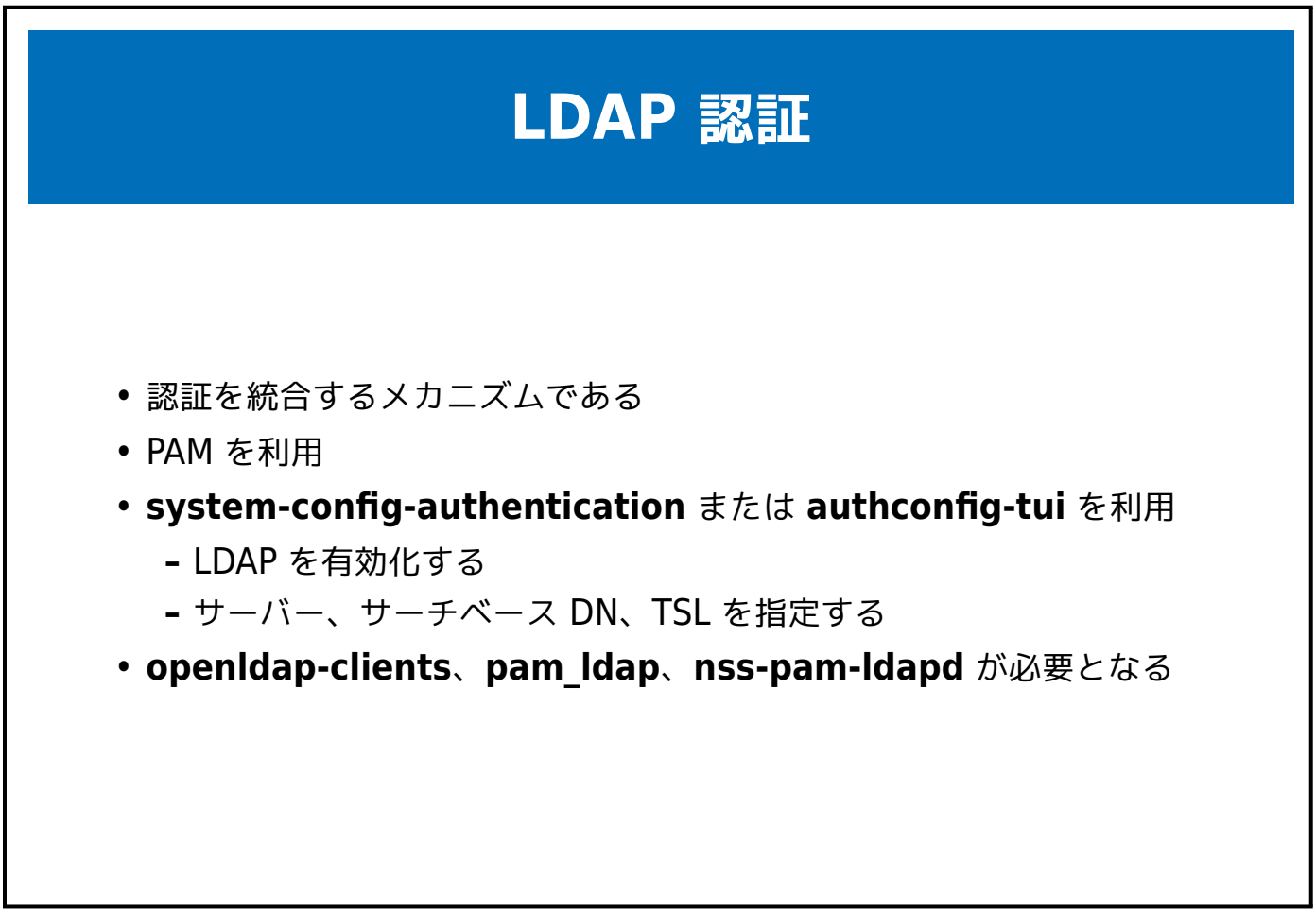
PAM ルール

- モジュールタイプ、コントロールフラグ、PAM モジュール
- モジュールタイプは、以下のどれかである:
  - auth
  - account
  - password
  - session
- コントロールフラグは、以下のどれかである:
  - required
  - requisite
  - optional
  - sufficient
- pam モジュール: 色々なモジュールがパラメータを持っている

モジュールタイプは、認証プロセスの手順をコントロールします。

auth: ユーザーに識別情報（ユーザー名、パスワードなど）を要求するように、アプリケーションに指示します。そして、認証情報を認定し権限を付与します。
account: パスワード有効期限、アクセス制御など、ユーザーのアカウントの状況を確認します。
password: ユーザー認証トークンの更新に責任を持ちます。通常はパスワードです。
session: セッションが確立される前後に行われる機能を提供するために使用されます。（環境のセットアップ、ログの記録など）
コントロールフラグは、モジュールの成功または失敗に対して認証プロセス全体がどう対処するのかを制御します。
required: サービスに権限を付与するためには、許可されたモジュールが ”success”（成功）を返す必要があります。そのモジュールがスタックの一部である場合、他のすべてのモジュールは引き続き実行されます。どのモジュールが失敗したかはアプリケーションに通知されません。
requisite: いずれかのモジュールがエラーで失敗すると、スタック内のほかのモジュールはそれ以上実行されず、戻り値がアプリケーションに送信されます。それ以外は required と同じです。
textttoptional: モジュールは要求されません。そのモジュールがサービスに関連付けられた唯一のモジュールである場合、その戻り値が失敗の原因になる可能性があります。
sufficient: このモジュールが成功すると、スタック内の後続のモジュールは実行されません。失敗した場合、スタック内の唯一のものでない限り、必ずしもスタックが失敗するわけではありません。
Pam-module /lib[64]/security にあるモジュール
これ以外の コントロールフラグ には include や substack があります。

33.4 LDAP 認証 **



LDAP (**L**ightweight **D**irectory **A**ccess **P**rotocol) は、ネットワーク上で分散ディレクトリサービスを使用および管理するための、業界標準プロトコルでオープン、かつベンダーニュートラル（= 特定ベンダーにロックインされない）なものです。

認証の統合に **LDAP** を使用する場合、各システム（またはクライアント）はユーザー認証のため集中 LDAP サーバーに接続します。**TLS** (**T**ransport **L**ayer **S**ecurity) は、安全性を高めるオプションになるため、使用をお勧めします。

LDAP は **PAM** と **system-config-authentication** または **authconfig-tui** を使用します。サーバー、検索のベース **DN** (domain name)、**TLS** を指定する必要があります。また **openldap-clients**、**pam_ldap**、**nss-pam-ldapd** も必要です。

LDAP 認証用にシステムを構成すると、以下の 5 つのファイルが変更されます:

```
/etc/openldap/ldap.conf
/etc/pam_ldap.conf
/etc/nslcd.conf
/etc/sssd/sssd.conf
/etc/nsswitch.conf
```

これらのファイルは、手動で編集するか、利用可能なユーティリティプログラム (**system-config-authentication** または **authconfig-tui**) を使って編集できます。

33.5 演習

本章に演習はありません。

章 34

ネットワークアドレス

34.1 IP アドレス

IP アドレス

- IP アドレスは、インターネット上のノードを一意に識別するもの
- IP アドレスは **ISP (Internet Service Providers)** で登録
- **IPv4** は **4 オクテット**で構成される 32 ビットアドレスである
  `148.114.252.10`
  (1 オクテット = 1 バイト)
- **IPv6** は 8 つの 16 ビットオクテットペアで構成される 128 ビットアドレスである
  `2003:0db5:6123:0000:1f4f:0000:5529:fe23`
- いくつか予約済アドレスがある
- ここではまだ主流と言える **IPv4** に焦点をあてる

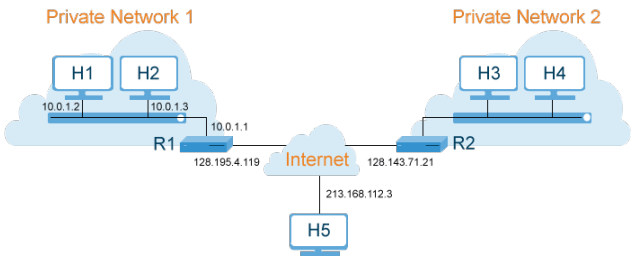
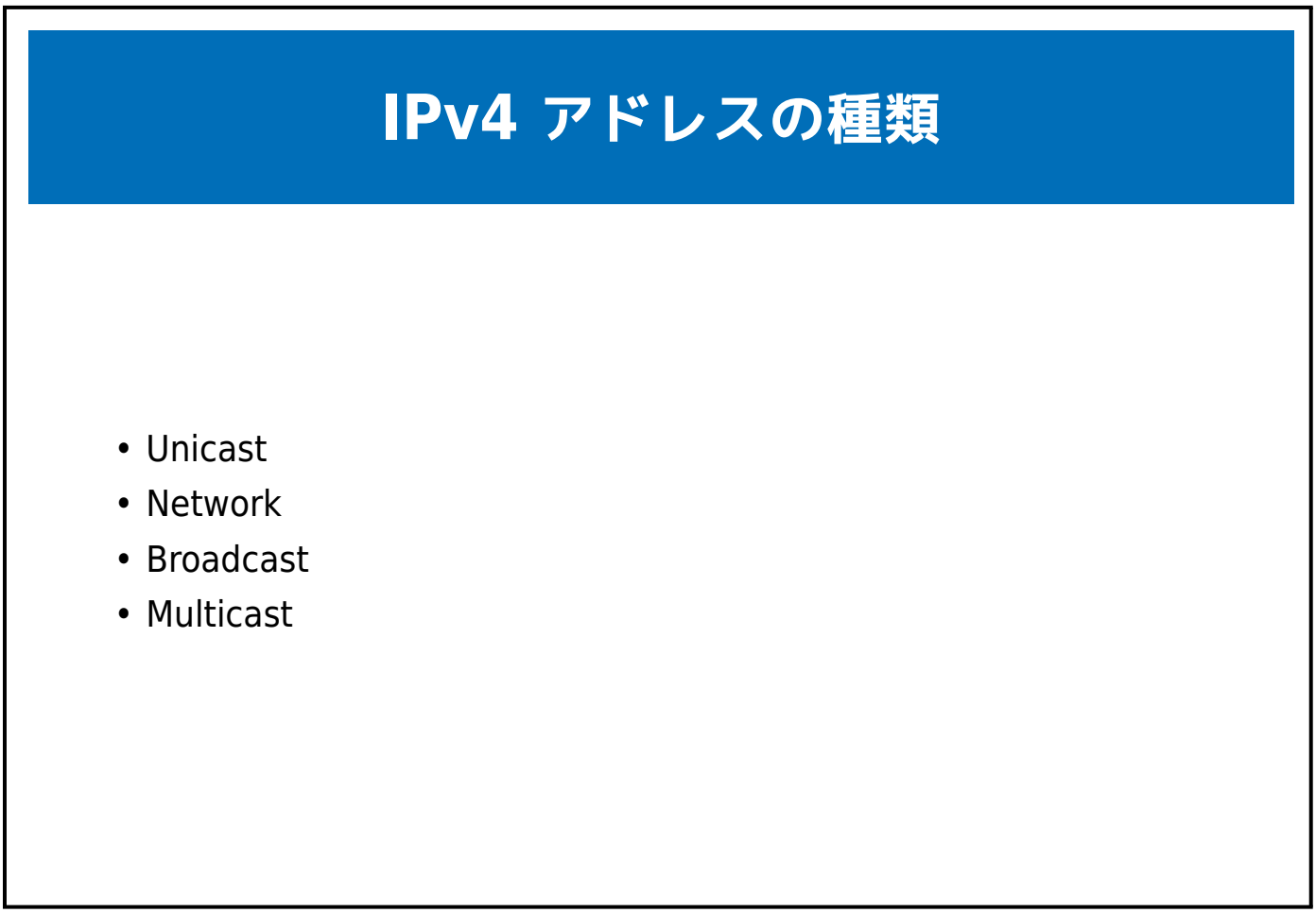


図 34.1: **IP アドレス**

34.2 IPv4 アドレスの種類



IPv4 アドレスの種類には、以下のものがあります:

- **ユニキャスト (Unicast)**:
  特定のホストに関連付けられたアドレスです。`140.211.169.4` や `64.254.248.193` のようになります。
- **ネットワーク (Network)**:
  ホスト部分 がすべて 2 進数のゼロに設定されているアドレスです。例えば：`192.168.1.0`（後で説明しますが、ホスト部分は最後の 1〜3 オクテットです。この例では最後のオクテットのみが該当します。）
- **ブロードキャスト (Broadcast)**:
  特定のネットワーク上にいるメンバに対して通信するアドレスです。`172.16.255.255` や `148.114.255.255` や `192.168.1.255` の例のように、ホスト部分はビットを全て立てるのでバイト表現では 255 となります。（最初の 2 つの例では、ホスト部分は最後の 2 オクテットで、3 番目の例では最後のオクテットのみです。）
- **マルチキャスト (Multicast)**:
  特定の複数のノードに対して通信するアドレスです。アドレス `224.0.0.2` はマルチキャストアドレスの例です。マルチキャストアドレスを認識するように設定された特定のノードだけが、そのマルチキャストグループへ投げられたパケットを解釈します。

予約済みアドレス

- ループバックインターフェイス:
  127.x.x.x
- アドレスが与えられていない状態:
  0.0.0.0
- 汎用 ブロードキャスト プライベートアドレス
  255.255.255.255
- その他
  10.0.0.0 - 10.255.255.255
  172.16.0.0 - 172.31.255.255
  192.168.0.0 - 192.168.255.255

特定のアドレスとアドレス範囲は、特別な目的のために予約されています:

- `127.x.x.x`

  ループバック（ローカルシステム）インターフェイス用に予約されています。`0 <= x <= 254` の値を取ります。通常は `127.0.0.1` と表されます。

- `0.0.0.0`

  まだ自分のアドレスが確定していない時に設定されます。**DHCP** や **BOOTP** などのプロトコルが（IP アドレスを取得するために）サーバーと通信しようとするときにこのアドレスを使用します。

- `255.255.255.255`

  内部使用のために予約されている、汎用のブロードキャストプライベートアドレスです。これらのアドレスは、誰にも割り当てられたり登録されたりすることはありません。通常、これらはルーティングできません。

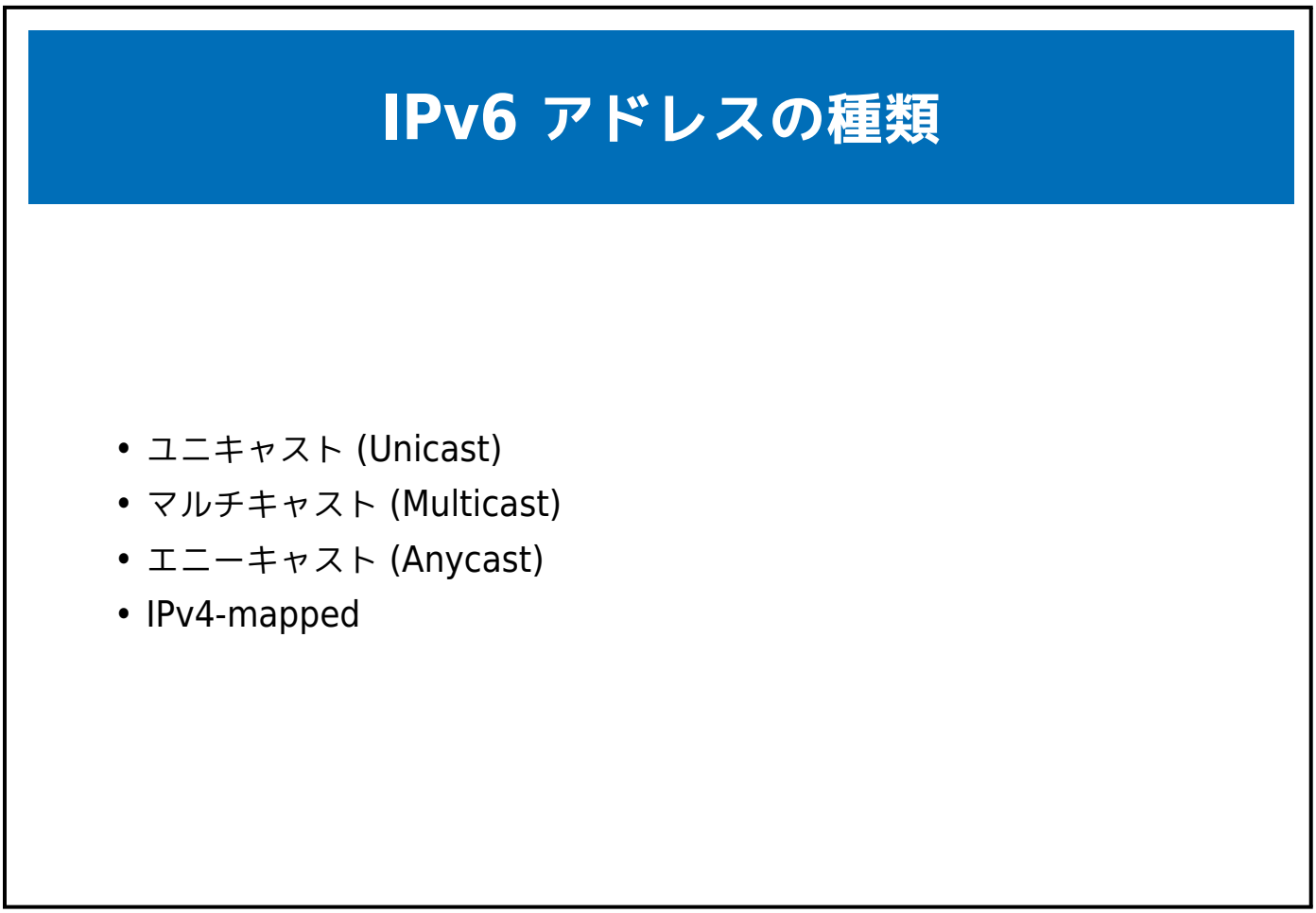
- その他の予約済みアドレスには、以下のものがあります:

  ```
  10.0.0.0 - 10.255.255.255
  172.16.0.0 - 172.31.255.255
  192.168.0.0 - 192.168.255.255
  ```

  これらにはそれぞれ目的があります。たとえば、なじみのあるアドレス範囲である `192.168.x.x` は、プライベートネットワーク内のローカル通信にのみ使用されます。

IPv4 と **IPv6** の予約済みアドレスの長いリストが https://en.wikipedia.org/wiki/Reserved_IP_addresses にあります。

34.3 IPv6 アドレスの種類



IPv6 アドレスの種類には、以下のものがあります:

- **ユニキャスト (Unicast)**:
  パケットは 1 つのインターフェイスに配信されます。
  - **Link-local**: 全てのインターフェイスに自動的に割り当てられます、ルーティングはできません。
  - **Global**: 動的、または手動で割り当てられるアドレスで、ルーティングも可能です。
  - ドキュメントの閲覧用などに予約されているアドレスです。
- **マルチキャスト (Multicast)**: パケットが複数のインターフェイスに対して配信されます。
- **エニーキャスト (Anycast)**:
  複数ノードのインターフェイスに同じアドレスを割り付けておき（ルーティング距離の観点から）、最も近いノードにパケットが転送されます。
- **IPv4-mapped**:
  IPv4 アドレスが埋め込まれた **IPv6** アドレスです。例えば `::FFFF:a.b.c.d/96` のようなものです。

さらに、**IPv6** には `::1/128` のように、`lo` インターフェイスに割り当てられるループバックなどの特別なアドレスがあります。

34.4 IPv4 アドレスクラス

IPv4 アドレスクラス

- Class A – 最上位のオクテットレンジが 1-127
  - ネットワーク数は 128、1 ネットワークあたり最大 16,777,214 のホストに対応、127.x.x.x はループバック用に予約されている
- Class B – 最上位のオクテットレンジが 128-191
  - ネットワーク数 16,384、最大 65,534 ホスト／ネットワーク
- Class C – 最上位のオクテットレンジが 192-223
  - ネットワーク数 2,097,152、最大 254 ホスト／ネットワーク
- Class D – 最上位のオクテットレンジが 224-239
  - マルチキャストアドレス
- Class E – 最上位のオクテットレンジが 240-254
  - 予約済みアドレス範囲である

歴史的に、IP アドレスはいくつかの定義クラスに分類されています。クラス A、B、C はアドレスのネットワーク部分とアドレスのホスト部分を区別するために使用されます。これはルーティングの目的で使用されます。

クラス A のアドレスは、アドレスのネットワーク部分に 8 ビット、ホスト部分に 24 ビットを使用します。クラス B のアドレスは、ネットワーク部分に 16 ビット、ホスト部分に 16 ビットを使用し、クラス C のアドレスは、ネットワーク部分に 24 ビット、ホスト部分に 8 ビットを使用します。

クラス D のアドレスはマルチキャストに使用されます。クラス E のアドレスは現在使用されていません。

34.5 ネットマスク

ネットマスク

- Class A ネットマスク
  - 255.0.0.0 (10 進数表記)
  - ff:00:00:00 (16 進数表記)
  - 11111111 00000000 00000000 00000000 (2 進数表記)
- Class B ネットマスク
  - 255.255.0.0 (10 進数表記)
  - ff:ff:00:00 (16 進数表記)
  - 11111111 11111111 00000000 00000000 (2 進数表記)
- Class C ネットマスク
  - 255.255.255.0 (10 進数表記)
  - ff:ff:ff:00 (16 進数表記)
  - 11111111 11111111 11111111 00000000 (2 進数表記)

これまで見てきたように、**ネットマスク** はネットワーク部分に使用されるアドレスの値と、ホスト部分に使用される値を決定するために使用されます。また、ネットワークとブロードキャストアドレスの決定にも使用されます。

- Class A アドレスは 8 ビットをネットワークのアドレス指定に、24 ビットをホストのアドレス指定に使います。
- Class B アドレスは 16 ビットをネットワークに、16 ビットをホストに使います。
- Class C アドレスは 24 ビットをネットワークに、8 ビットをホストに使います。
- Class D アドレスはマルチキャスト用に使われます。
- Class E アドレスは現在使われていません。

ネットワークアドレスは、ネットマスクと IP アドレスの論理積（AND 演算- &）を計算することで取得されます。同じメディアを介して接続し、同じネットワークアドレスを共有するノード群を構成するローカルネットワークを定義できる点でも、ネットワークアドレスは興味深いです。同じネットワーク上のすべてのノードは、お互いを直接見ることができます。

```
ex.  172.16.2.17  ip address
    &255.255.0.0  netmask
   -----------------
     172.16.0.0   network address
```

34.6 ホスト名

ホスト名の取得と設定

- 他ノードから見た、ネットワークデバイスを特定するためのラベル
- 以前は **ノード名 (nodename)** とも呼ばれていた
- インストール時に設定され、後からいつでも変更できる
- 誰でも、以下の方法でホスト名を取得できる:

```
$ hostname
wally
```

- ホスト名の変更には root 権限が必要である:

```
$ sudo hostname lumpy
lumpy
```

- 大部分の **Linux** ディストーションでは、現在のホスト名が `/etc/hostname` に保存されている
- リブート後にも変更が残るように **永続化**させるには、**systemd** インフラの一つである **hostnamectl** を使う

```
$ sudo hostnamectl set-hostname lumpy
```

DNS のために、ホスト名にはピリオド (dot) とドメイン名が追加されます。そのためホスト名が `antje` のマシンの完全修飾ドメイン名 **fully qualified domain name** (**FQDN**) は `antje.linuxfoundation.org` となります。

歴史的には、ホスト名の恒久的な変更には `etc` ディレクトリツリーにある設定ファイルの変更が必要でした。設定するファイルは **Red Hat** ベースのシステムでは `/etc/sysconfig/network`、**Debian** ベースのシステムでは `/etc/hostname` 、そして **SUSE** ベースのシステムでは `/etc/HOSTNAME` です。しかし、最近のシステムなら **hostnamectl** を使うべきです。

```
$ hostnamectl
   Static hostname: c8
         Icon name: computer-desktop
           Chassis: desktop
        Machine ID: ce0c82382a8a4c80bbd6931a917a2f1c
           Boot ID: 94207b3fbd9b4891b9a94e21762a47cb
  Operating System: Red Hat Enterprise Linux 8.2 (Ootpa)
            dracut-049-70.git20200228.el8 (Initramfs)
            Kernel: Linux 5.11.6
      Architecture: x86-64
```

and to see a usage message:

```
$ hostnamectl --help

hostnamectl [OPTIONS...] COMMAND ...

Query or change system hostname.

  -h --help              Show this help
     --version           Show package version
     --no-ask-password   Do not prompt for password
  -H --host=[USER@]HOST  Operate on remote host
  -M --machine=CONTAINER Operate on local container
     --transient         Only set transient hostname
     --static            Only set static hostname
     --pretty            Only set pretty hostname

Commands:
  status                 Show current hostname settings
  set-hostname NAME      Set system hostname
  set-icon-name NAME     Set icon name for host
  set-chassis NAME       Set chassis type for host
  set-deployment NAME    Set deployment environment for host
  set-location NAME      Set location for host
  set-location NAME      Set location for host
```

34.7 演習

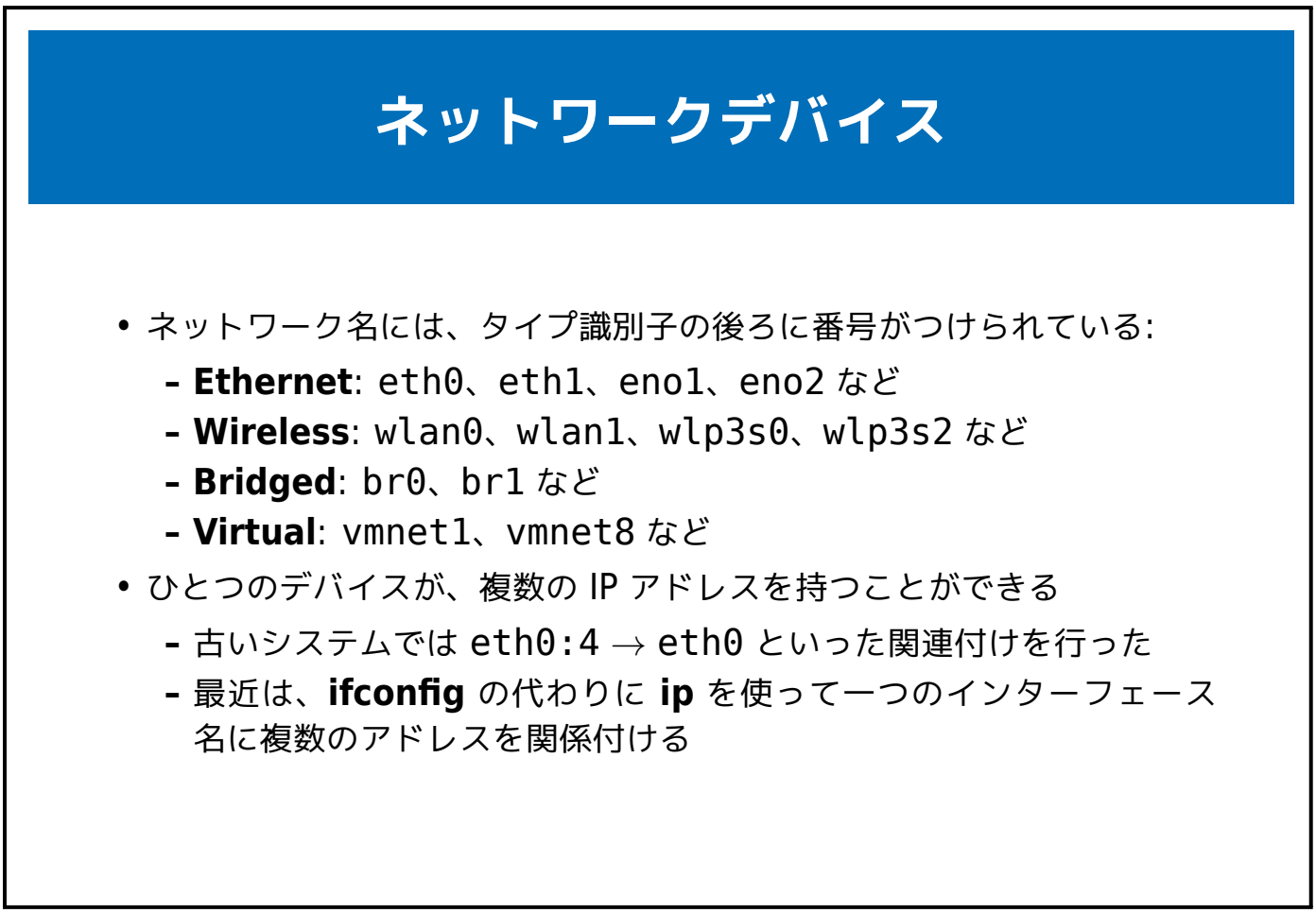
注目

本章には、演習はありません。ネットワーク構成に関する以下のセクションの準備です。

章 35

ネットワークデバイスと構成

35.1 ネットワークデバイス



ブロックデバイスやキャラクタデバイスとは異なり、ネットワークデバイスは **デバイスノード** とも呼ばれる特別な **デバイスファイル** とは関連付けられていません。`/dev` ディレクトリのエントリを関連付けるのではなく、名前で認識されています。

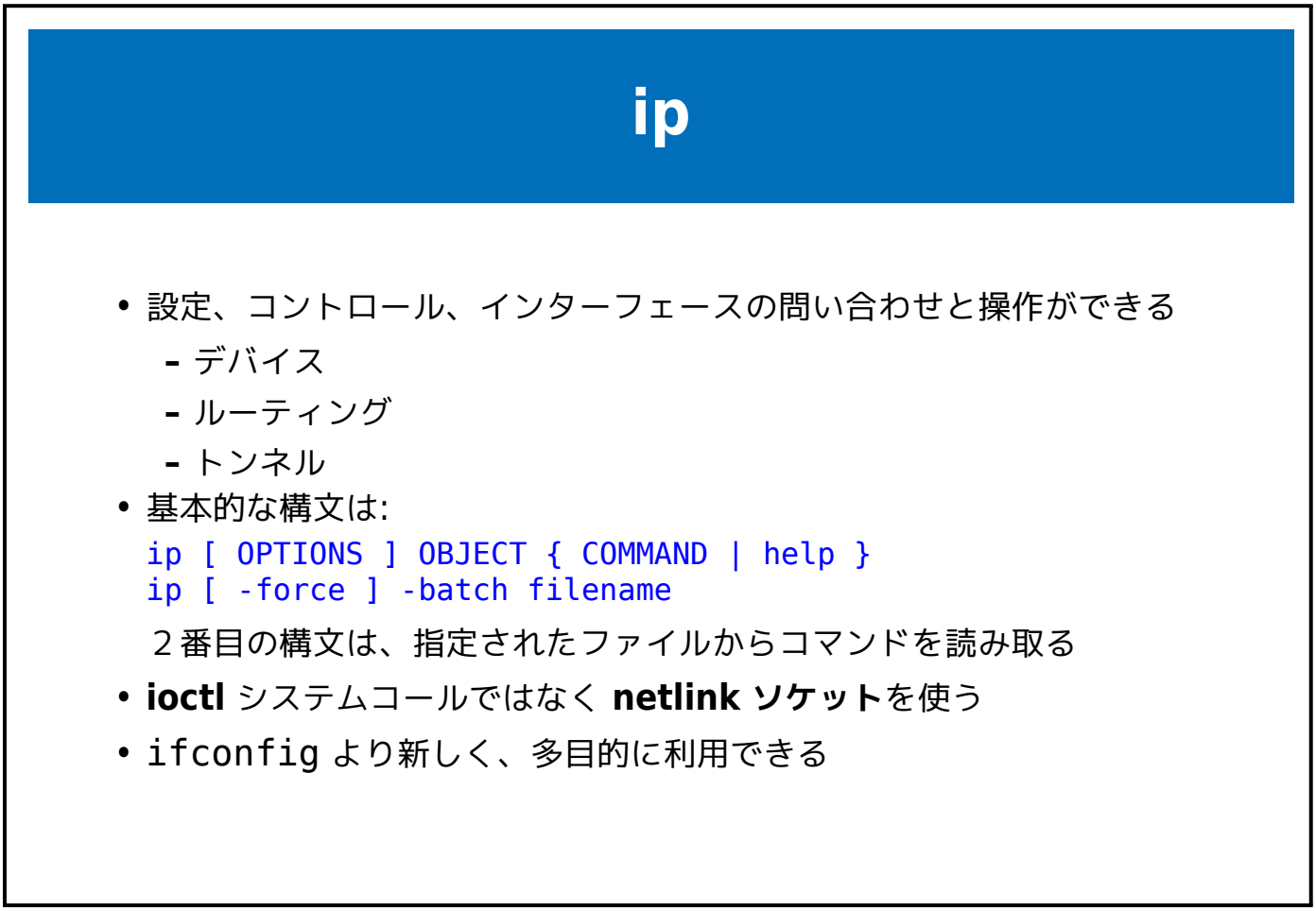
以前は、複数の仮想デバイスを単一の物理デバイスに関連付けることができました。これらは、コロンと数字を使って命名されていました。具体的には `eth0:0` は一番目の `eth0` デバイスのいう意味のエイリアス（別名）です。これは１つのネットワークカードで、複数の IP アドレスをサポートするために行われました。ただし **ifconfig** の代わりに **ip** を使用する場合この命名法は利用できないので、これ以上は説明しません。さらに、このやりかたは **IPV6** とは互換性がありません。

古典的な方法によるデバイスの命名は、特に同じ種類のインターフェイスが複数存在する場合に問題を発生させます。たとえば、同じネットワークカードが２枚あるとします。古典的な命名法では１つが `eth0`、もう１つは `eth1` という名前になります。では、それぞれの物理デバイスと名前の対応関係はどうなるでしょうか？

最も簡単な方法は、最初のデバイスを `eth0` に、２番目を `eth1` にすることです。ところが残念ながら、最新のシステムではデバイスの検出は確定論的ではなく、デバイスが予測できない順序で配置されたり接続されたりする場合があるのです。そのため、ローカルインターフェイスとインターネットインターフェイスの順序が入れ替わってしまう可能性がでてきます。カーネルのバージョンと構成によって、ハードウェアが変更されていなくてもインターフェイスの配置順序が変わることがわかっています。

多くのシステム管理者は、システム構成ファイルと起動スクリプトの中でハードウェア（MAC）アドレスとデバイス名との関連付けをハードコーディングするという方法で、この問題を解決しています。この方法は長年にわたって使われてきましたが、手動調整が必要なだけでなく、MAC アドレスが決まっていない場合には関連付けを定義できません。MAC アドレスが決まっていないとケースは、組み込みシステムと仮想化で発生する可能性があります。

35.2 ip



ip は、次に説明する以前からある **ifconfig** よりも推奨されるようになったコマンドラインユーティリティです。**ip** は **ioctl** システムコールではなく **netlink ソケット** を使うので、多目的なだけでなく効率的でもあります。

ip はさまざまな用途に使用することができます。デバイスの制御と表示、ルーティング、ポリシーベースのルーティング、トンネリングの操作などに使われます。

基本的な構文は:

```
ip [ OPTIONS ] OBJECT { COMMAND | help }
ip [ -force ] -batch filename
```

２番目の構文を使うと、指定されたファイルからコマンドを読み取ることができます。

ip は複合型ユーティリティです。OBJECT 引数には、実行されるアクションの種類を指定します。実際の COMMANDS が取り得る内容は選択された **OBJECT** に依存します。以下に代表的な OBJECT を示します。

Table 35.1: **ip**

OBJECT	機　能
address	**IPv4** または **IPv6** プロトコルのデバイスアドレス
link	ネットワークデバイス
maddress	マルチキャストアドレス
monitor	netlink メッセージの監視
route	ルーティングテーブルのエントリ
rule	ルーティングポリシーデータベースに登録されているルール
tunnel	IP トンネル

ip の使用の例

- すべてのネットワークインターフェイス情報を表示する:
  `$ ip link show`
- eth0 ネットワークインターフェイス情報を統計情報を含めて表示:
  `$ ip -s link show eth0`
- eth0 の IP アドレスを設定する:
  `$ sudo ip addr add 192.168.1.7 dev eth0`
- インターフェイス eth0 を停止する:
  `$ sudo ip link set eth0 down`
- インターフェイス eth0 の MTU を 1480 バイトに設定する:
  `$ sudo ip link set eth0 mtu 1480`
- ネットワークルートを設定する:
  `$ sudo ip route add 172.16.1.0/24 via 192.168.1.5`

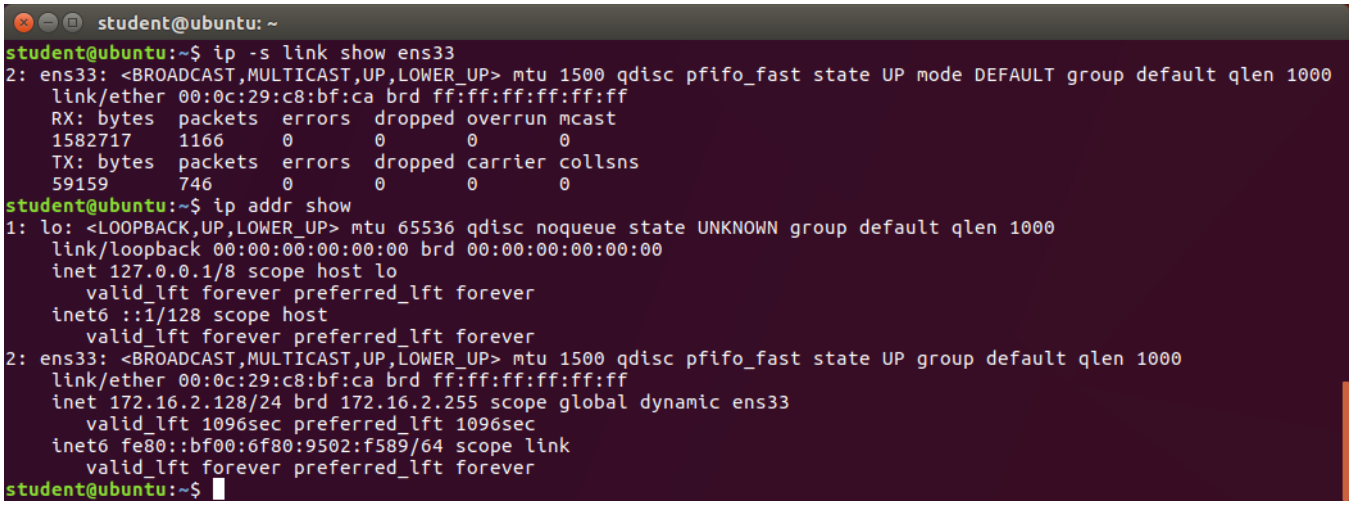


図 35.1: **ip の使用例**

35.3 ifconfig

ifconfig

- 設定、コントロール、インターフェースの問い合わせができる
- **ip** に置き換えられていく:
  - 新しい **Linux** ディストリビューションにはデフォルトでは インストールされていないことがある
- コマンドラインで操作する

```
$ ifconfig
$ ifconfig eth0
$ ifconfig eth0 192.168.1.50
$ ifconfig eth0 netmask 255.255.255.0
$ ifconfig eth0 down
$ ifconfig eth0 mtu 1480
```

- ネットワークスタートアップスクリプトとも呼ばれている

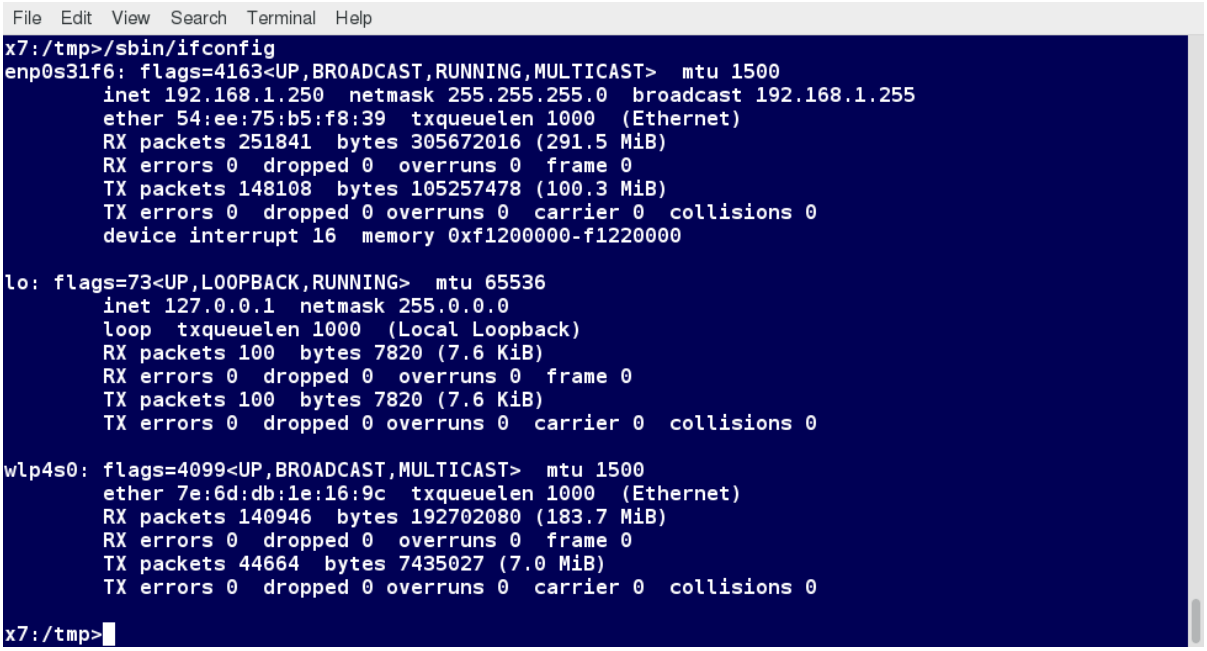


図 35.2: **ifconfig** の出力例

ifconfig の出力例

- 全て、ないし特定のインターフェースの情報を表示する:

```
$ ifconfig
$ ifconfig eth0
```

- eth0 に IP アドレス 192.168.1.50 を設定する:

```
$ sudo ifconfig eth0 192.168.1.50
```

- ネットマスクを 24-bit に設定する:

```
$ sudo ifconfig eth0 netmask 255.255.255.0
```

- eth0 を起動、ないし停止する:

```
$ sudo ifconfig eth0 up
$ sudo ifconfig eth0 down
```

- eth0 の **MTU** (**M**aximum **T**ransfer **U**nit) を 1480 バイトに設定する:

```
$ sudo ifconfig eth0 mtu 1480
```

ifconfig は **UNIX** ライクなオペレーティングシステムで長い間使用されてきたシステム管理ユーティリティで、コマンドラインまたはシステム構成スクリプトでネットワーク インターフェイス パラメータの構成、制御、照会を行うために使用されています。しばしば、ネットワーク スタートアップ スクリプトとも呼ばれています。
上のスライドではいくつかの使用例を紹介しています。

35.4 予測可能なネットワークインターフェイスデバイス名

予測可能なネットワークインターフェイスデバイス名

- **予測可能なネットワークインターフェイスデバイス名 (PNIDN)** (**Predictable Network Interface Device Names**) は **udev** と連携し **systemd** と統合されている
- デバイスに指定できる名前は 5 種類ある:
  1. **ファームウェア** や **BIOS** がオンボードデバイスに対して提供するインデックス番号を組み入れた名前　例：eno1
  2. **ファームウェア** や **BIOS** が提供している **PCI Express** ホットプラグスロットのインデックス番号を組み込んだ名前　例：ens1
  3. ハードウェア接続の物理的か地理的、もしくはその両方の場所を組み込んだ名前　例：enp2s0
  4. MAC アドレスを組み込んだ名前　例：enx7837d1ea46da
  5. 従来型のメソッドを使用した名前　例：eth0

2 つのオンボード **PCI** ネットワークインターフェイス eth0 と eth1 をもったマシンの例を示します:

```
$ ip link show | grep enp
2: enp4s2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN mode DEFAULT qlen 1000
3: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT qlen 1000

$ ifconfig | grep enp
  enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
  enp4s2: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
```

これらの名前は **PCI** システム上で、ボードがどこに接続されているかで決まります:

```
$ lscpi | grep Ethernet
02:00.0 Ethernet controller: Marvell Technology Group Ltd. 88E8056 PCI-E Gigabit Ethernet Controller (rev 12)
04:02.0 Ethernet controller: Marvell Technology Group Ltd. 88E8001 Gigabit Ethernet Controller (rev 14)
```

lspci 出力の各行の先頭の 3 組の数字は、バス番号、デバイス（またはスロット）番号、デバイス機能の番号です。これは、物理的な位置を反映したものです。
同様に、従来 wlan0 という名前で呼ばれていた無線デバイスを調べます。

```
$ ip link show | grep wl
3: wlp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DORMANT qlen 1000

$ lspci | grep Centrino
03:00.0 Network controller: Intel Corporation Centrino Advanced-N 6205 [Taylor Peak] (rev 34)
```

同じパターンが見られます。新しい命名スキームをオフにすれば簡単に古典的な名前に戻すことができます。古典的な命名は研究プロジェクトとして残す予定です。これ以降は、明確かつ単純に説明するためにおもに古典的な名前を使っていきます。

35.5 ネットワーク設定ファイル

ネットワーク設定ファイル

- 各ディストリビューションには、独自のファイルやディレクトリがある:
  - **Red Hat**
    `/etc/sysconfig/network`
    `/etc/sysconfig/network-scripts/ifcfg-ethX`
    `/etc/sysconfig/network-scripts/ifcfg-ethX:Y`
    `/etc/sysconfig/network-scripts/route-ethX`
  - **Debian**
    `/etc/network/interfaces`
  - **SUSE**
    `/etc/sysconfig/network`
- **systemd** が標準的になっている
- **Network Manager** を利用することが推奨される
- 最近のディストリビューションでは、これらの設定ファイルは存在しないか、中身がないことがある

新しい **Linux** ディストリビューションでは、これらの設定ファイルは存在しないか、空か、もしくはずっと小さくなっています。

35.6 ネットワークマネージャ

ネットワークマネージャ

- ネットワークインターフェースは、ソフトウエアもハードウエアも、基本的には（途中で変更されない）スタティックなものだった
- 最新システムの多くは、ダイナミックなコンフィギュレーションに対応している:
  - デバイスの場所が変わると、ネットワークも変更されることがある
  - 無線デバイスは、より多くのネットワークに接続される可能性がある
  - ハードウエア（無線デバイスなど）は、接続されたりオン／オフされるとデバイスが変わることがある
- これまで説明してきた設定ファイルは、スタティックな環境で使われることを想定したもので、ディストリビューション依存性も大きかった
- **ネットワークマネージャ** は、システムが異なってもほぼ共通である

昔はネットワーク接続はほぼすべて有線（イーサネット）で、大規模なシステム変更が無い限り（デバイス名やアドレスが）変更されることはありませんでした。

システムが起動すると、システムは `/etc` ディレクトリサブツリーのネットワーク構成ファイルを調べて、静的または動的（DCHP）アドレス構成やデバイスをブート時に開始すべきかどうかなどを決定しました。

複数のネットワークデバイスが存在する場合には、どの順序で起動するか、どのネットワークに接続するか、呼び名をどうするかなどポリシーを決定する必要がありました。

ワイヤレス接続（および **USB** アダプターなどのホットプラグ ネットワークデバイス）がより一般的になるにつれ、ハードウェアの一時的な性質と、接続されている特定のネットワークの性質の両方が原因で、設定ははるかに複雑になりました。

ディストリビューションに依存したインターフェースやコンフィギュレーションの排除には、大きなアドバンテージがありました。

ネットワークマネージャ には構成ファイルがありますが、それらを直接編集するのではなく通常は用意されているユーティリティ群を利用して操作や更新する方が良いでしょう。

ネットワーク マネージャ インターフェース

- **GUI**
  全ての **Linux** デスクトップ (**GNOME**, **KDE**, **XFCE**, etc) に備わっている
  通常トップバーのネットワークアイコンをクリックして起動する
- **nmtui**
  network **m**anager **t**ext **u**ser **i**nterface は **ncurses** を利用したシンプルなメニュー駆動式のコマンドラインインターフェースを持つ
- **nmcli**
  network **m**anager **c**ommand **l**ine **i**nterface は、抽象度の高いインターフェースから利用可能な低水準メソッドで、スクリプトに組み込む場合にも利用しやすい

ホテルの部屋やコーヒーショップでラップトップを使用している場合は、**Linux** ディストリビューションが提供する何らのグラフィカルインターフェースを使用していると思います。これを使用すれば、異なるネットワークを選択したり、セキュリティとパスワードを構成したり、デバイスのオン／オフを切り替えたりできます。

しばらく使い続けることを前提として構成を設定変更する時には、直感的に設定ファイルの編集ができる **nmtui** を使うと良いです。

スクリプトを使ってネットワーク構成を変更する必要がある場合は **nmcli** を使用します。またコマンドライン操作の方が好きな人も **nmtui** の代わりにこの **nmcli** を使用することもできます。

GUI が適切に設定されていれば、これらの３つの方法のいずれかを使用してもタスクを実行できるはずです。しかし、ここではディストリビューション中立で、設定ファイルに起因する違いを意識する必要がない **nmtui** と **nmcli** に焦点を当てます。

nmtui

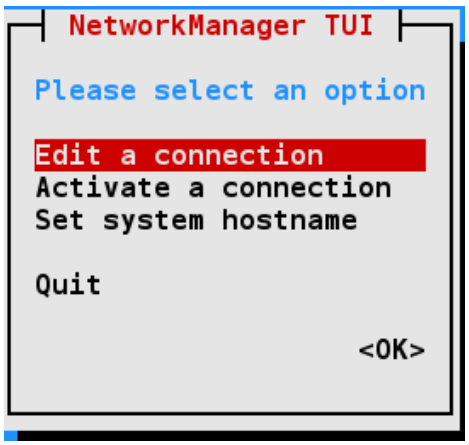


図 35.3: **nmtui メイン画面**

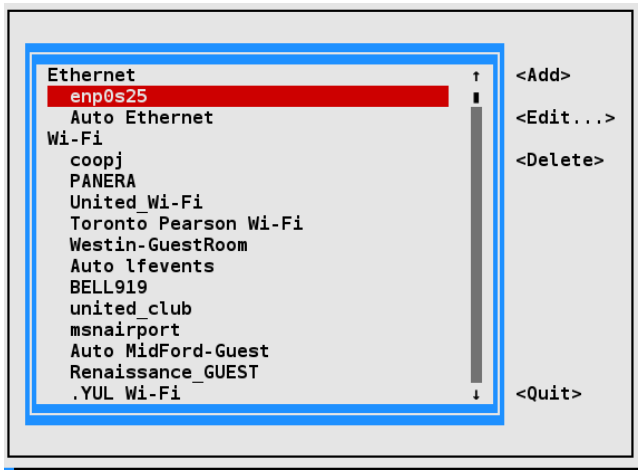


図 35.4: **nmtui 編集画面**

nmtui の利用はより直感的です。矢印キーや TAB キーを使ってメニューを操作します。

ネットワーク接続の編集や有効化以外に、システムのホスト名を設定することもできます。しかしこのような操作は、通常のユーザー権限では行えないので root パスワードの入力を求められます。

nmtui ワイヤレス接続設定

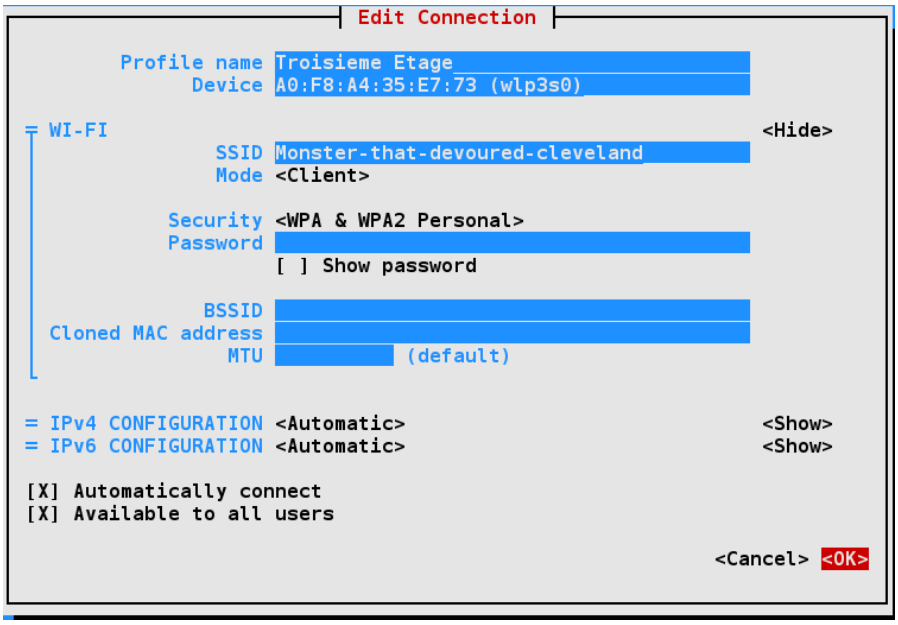


図 35.5: **nmtui ワイヤレス接続設定**

nmcli

- **ネットワーク マネージャー**に対するコマンドラインインターフェースである（**n**etwork **m**anager **c**ommand **l**ine **i**nterface）
- 対話モードも用意されている
- https://fedoraproject.org/wiki/Networking/CLI を参照
- `man nmcli-examples` も参照
- 演習で、いくつかの事例を紹介する

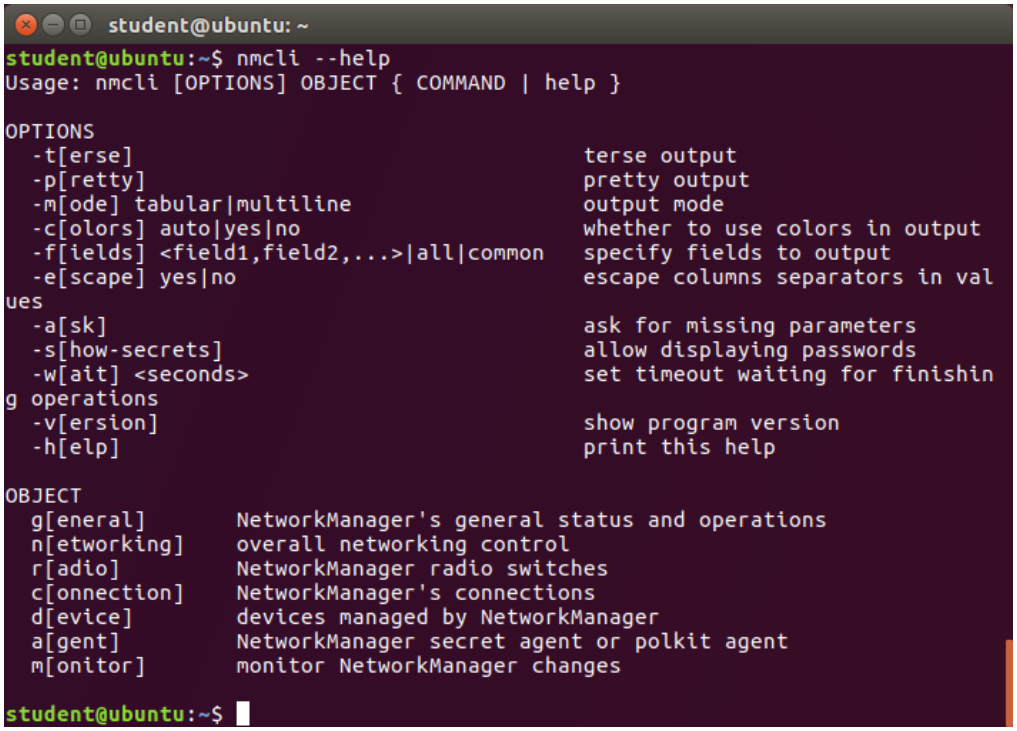


図 35.6: **nmcli**

35.7 ルーティング

ルーティング

- ネットワーク内、およびネットワーク間のパケットの経路を設定する
- ルーティングテーブル
  - 全てのネットワークとホストへのパスを定義する
  - （同一ネットワーク内の）ローカルパケットは直接送信する
  - リモートネットワーク向けのパケットはルータに送る

ルーティング は、ネットワークトラフィックを送信するネットワーク内のルート（パス）を選択するプロセスです。**ルーティングテーブル**は、システムが管理する他のネットワークへのルートのリストです。すべてのネットワークとホストへのルートを定義し、リモートトラフィックをルーターに送信します。

現在のルーティングテーブルを表示するには **route** または **ip** を使用します:

```
student@ubuntu: /etc
student@ubuntu:/etc$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.1.1     0.0.0.0         UG    100    0        0 ens33
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 ens33
192.168.1.0     0.0.0.0         255.255.255.0   U     100    0        0 ens33
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr0
student@ubuntu:/etc$ ip route
default via 192.168.1.1 dev ens33  proto static  metric 100
169.254.0.0/16 dev ens33  scope link  metric 1000
192.168.1.0/24 dev ens33  proto kernel  scope link  src 192.168.1.24  metric 100
192.168.122.0/24 dev virbr0  proto kernel  scope link  src 192.168.122.1 linkdown

student@ubuntu:/etc$
```

図 35.7: **route と ip route の利用**

デフォルト ルート

- ルーティングテーブルエントリーにマッチしない時に使用される
- DHCP を使って動的に取得することができる
- 手動で設定することもできる（スタティック設定）
  - **Red Hat** 系のシステム: GATEWAY=x.x.x.x `/etc/sysconfig/network` または `/etc/sysconfig/network-scripts/ifcfg-ethX` に含まれる
  - **Debian** 系のシステム: gateway=x.x.x.x `/etc/network/interfaces` に含まれる
- `/etc` 内のファイルの手による編集を避けるため **nmcli** を利用する

```
$ sudo  nmcli con mod virbr0 ipv4.routes 192.168.10.0/24 \
  +ipv4.gateway 192.168.122.0
$ sudo nmcli con up virbr0
```

デフォルトルート は、ネットワークのパスがルーティングテーブルに明示的に指定されていない場合のパケットの送信先です。
デフォルトゲートウェイはシステム稼働中にも設定が可能です:

```
$ sudo route add default gw 192.168.1.10 enp2s0
$ route
```

```
Kernel IP routing table
Destination     Gateway       Genmask         Flags Metric Ref Use Iface
default         192.168.1.10  0.0.0.0         UG    0      0      0 enp2s0
default         192.168.1.1   0.0.0.0         UG    1024   0      0 enp2s0
172.16.132.0    0.0.0.0       255.255.255.0   U     0      0      0 vmnet1
192.168.1.0     0.0.0.0       255.255.255.0   U     0      0      0 enp2s0
192.168.113.0   0.0.0.0       255.255.255.0   U     0      0      0 vmnet8
```

この変更により、ネットワーク接続が切れる可能性があるので注意してください！ ネットワークをリセットするか、上記の例で次のように変えれば復元できます:

```
$ sudo route add default gw 192.168.1.1 enp2s0
```

この変更は恒久的なものではなく、システムを再起動した時には消失します。

スタティックルート

- ルータ、ないしルートが複数ある場合のパケットのフローを制御する
- インターフェース毎に設定する
- 永続的な設定も、一時的な設定もできる
- **ip** を利用する (一時的な設定)
- 永続的な設定とするには:
  - **Red Hat** 系のシステム:
    `/etc/sysconfig/network-scripts/route-ethX`
  - **Debian** 系のシステム:
    `/etc/network/interfaces`
  - **SUSE** 系のシステム:
    `/etc/sysconfig/network/ifroute-eth0`
- `/etc` 内のファイルの手修正を避けるため **nmcli** を使います
  (演習で実際に試します)

システムに複数のルート、ないしルータがある場合や複数のインターフェースがある場合には、パケットの行き先に応じて選択的にルートをコントロールできると便利です。

一時的なルートの設定は **route** または **ip** コマンドを使って設定することができます:

```
$ sudo ip route add 10.5.0.0/16 via 192.168.1.100
```

Red Hat 系のシステムでは `/etc/sysconfig/network-scripts/route-ethX` を以下のように編集することで、永続的なルートの設定が可能です:

```
$ cat /etc/sysconfig/network-scripts/route-eth0
  10.5.0.0/16 via 172.17.9.1
```

Debian 系のシステムでは以下のように `/etc/network/interfaces` に行を追加する必要があります:

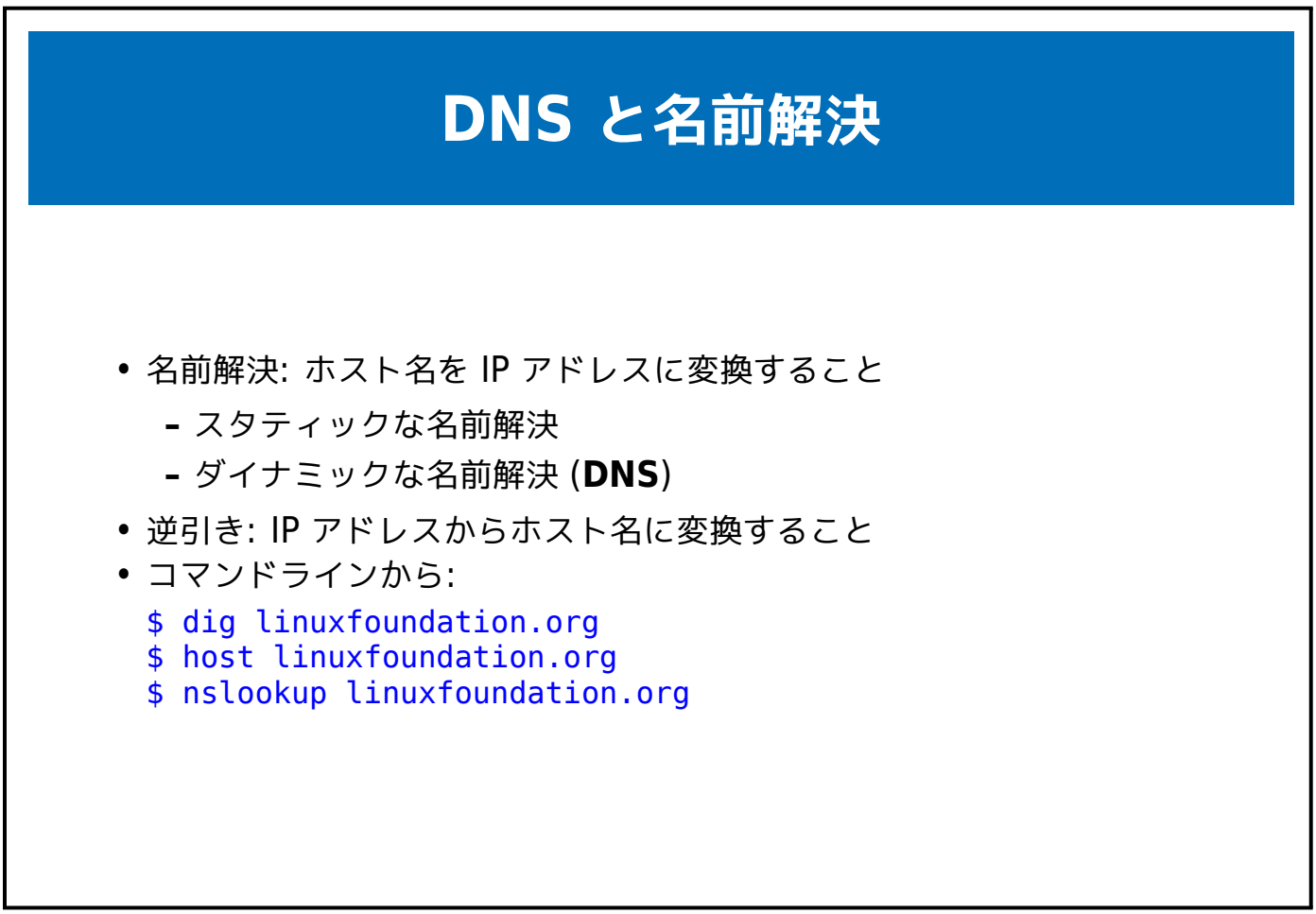
```
iface eth1 inet dhcp
  post-up route add -host 10.1.2.51 eth1
  post-up route add -host 10.1.2.52 eth1
```

SUSE 系のシステムでは以下のように `/etc/sysconfig/network/ifroute-eth0` に以下の行を追加、ないし作成する必要があります:

```
# Destination Gateway Netmask Interface [Type] [Options]
  192.168.1.150 192.168.1.1 255.255.255.255 eth0
  10.1.1.150 192.168.233.1.1 eth0
  10.1.1.0/24 192.168.1.1 - eth0
```

各フィールドは TAB で区切ります。

35.8 DNS と名前解決



名前解決 とは、ホスト名をホストの IP アドレスに変換することです。

たとえば、ブラウザまたは電子メールクライアントは `training.linuxfoundation.org` との間で送受信を行うために、`training.linuxfoundation.org` にサービスを提供しているサーバー群の名前を解決して IP アドレスに変換します。

名前解決は `/etc/hosts` を利用して **スタティック** に行うことも、**DNS** サーバーを利用してダイナミックに行うこともできます。

ホスト名の IP アドレス解決に使用するコマンドラインツールとして:

- **dig**: 情報を最も詳細に取得できるツールで、多くのオプションがあります。
- **host**: 簡潔に情報を取得できます。
- **nslookup**: 昔からあるツールです。

dig は最新のユーティリティです。他は非推奨とされる場合がありますが **host** の出力は読みやすく基本情報が含まれています。

逆引き が必要になる場合もあります。IP アドレスからホスト名に変換することです。上で紹介した 3 つのユーティリティに、ホスト名ではなく検索したいホストの IP アドレスを入力して、ホスト名が出力されるのを確認してください。

/etc/hosts

- ローカル、ないしスタティックなホスト名の解決に利用される
- 小規模で独立性の高いネットワークで有効である
- 通常は dns より前にチェックされる
- 参照優先順位を /etc/nsswitch.conf で設定できるが、最近はあまり使われない

```
student@ubuntu:/etc$ ls -l host*
-rw-r--r-- 1 root root  92 Oct 22  2015 host.conf
-rw-r--r-- 1 root root   7 Apr 21 08:46 hostname
-rw-r--r-- 1 root root 221 Apr 21 08:46 hosts
-rw-r--r-- 1 root root 411 Apr 20 17:14 hosts.allow
-rw-r--r-- 1 root root 711 Apr 20 17:14 hosts.deny
```

/etc/hosts は、ホスト名と IP アドレスのローカルデータベースを保持します。これには IP アドレスを対応するホスト名と別名（エイリアス名）にマッピングするレコードのセット（それぞれ 1 行ずつ）が含まれています。

典型的な /etc/hosts ファイルは次のようになります:

/etc/hosts

```
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.1.100 hans hans7 hans64
192.168.1.150 bethe bethe7 bethe64
192.168.1.2 hp-printer
192.168.1.10 test32 test64 oldpc
```

このような静的な名前解決は、主にローカルで小規模で分離されたネットワークで使用されます。通常 **DNS** によるアドレス解決の前に参照されます。参照の優先順位は /etc/nsswitch.conf で制御できます。（これは最近ではあまり使われていません）

/etc 内の他のホスト関連ファイルには:
/etc/hosts.deny, /etc/hosts.allow.

これらは自己文書化されており、その目的は名前を見れば分かります。allow ファイルが最初に検索され、クエリがそこに見つからない場合にのみ deny ファイルが検索されます。

/etc/host.conf には、一般的な構成情報が含まれています。ただしこれは殆ど使用されません。

DNS

- 動的な名前解決ができる
- /etc/resolv.conf
  - 検索対象とするドメインを指定することができる
  - 利用するネームサーバーの厳密な参照順序を指定する
  - 手動で構成を設定することも、**DHCP** (**D**ynamic **H**ost **C**onfiguration **P**rotocol) などのサービスから更新させることもできる

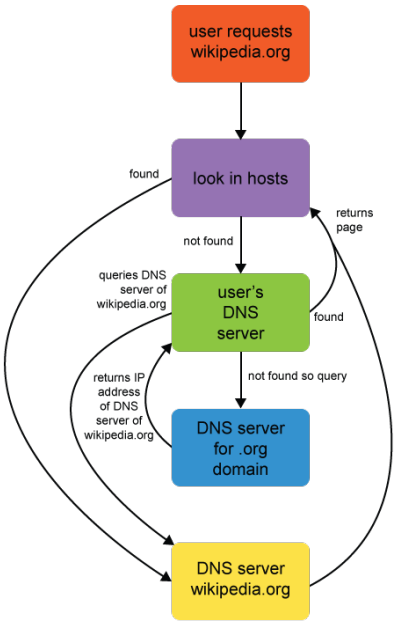


図 35.8: **DNS**

/etc/hosts を利用したローカルな名前解決が成功しない時、システムは **DNS** (**D**omain **N**ame **S**erver) に問い合わせます。

DNS は、動的に名前解決を行いクライアントが名前を検索するために使用するサーバーのネットワークを構成します。サービスは分散されます。各々の **DNS** サーバーは、そのサーバーの **権限ゾーン (zone of authority)** に帰属する情報だけを持っていますが、複数のサーバーが連携することでどんな名前でも解決できます。

DNS は /etc/resolv.conf で設定されます。これは伝統的に以下のようになっていました:

/etc/resolv.conf

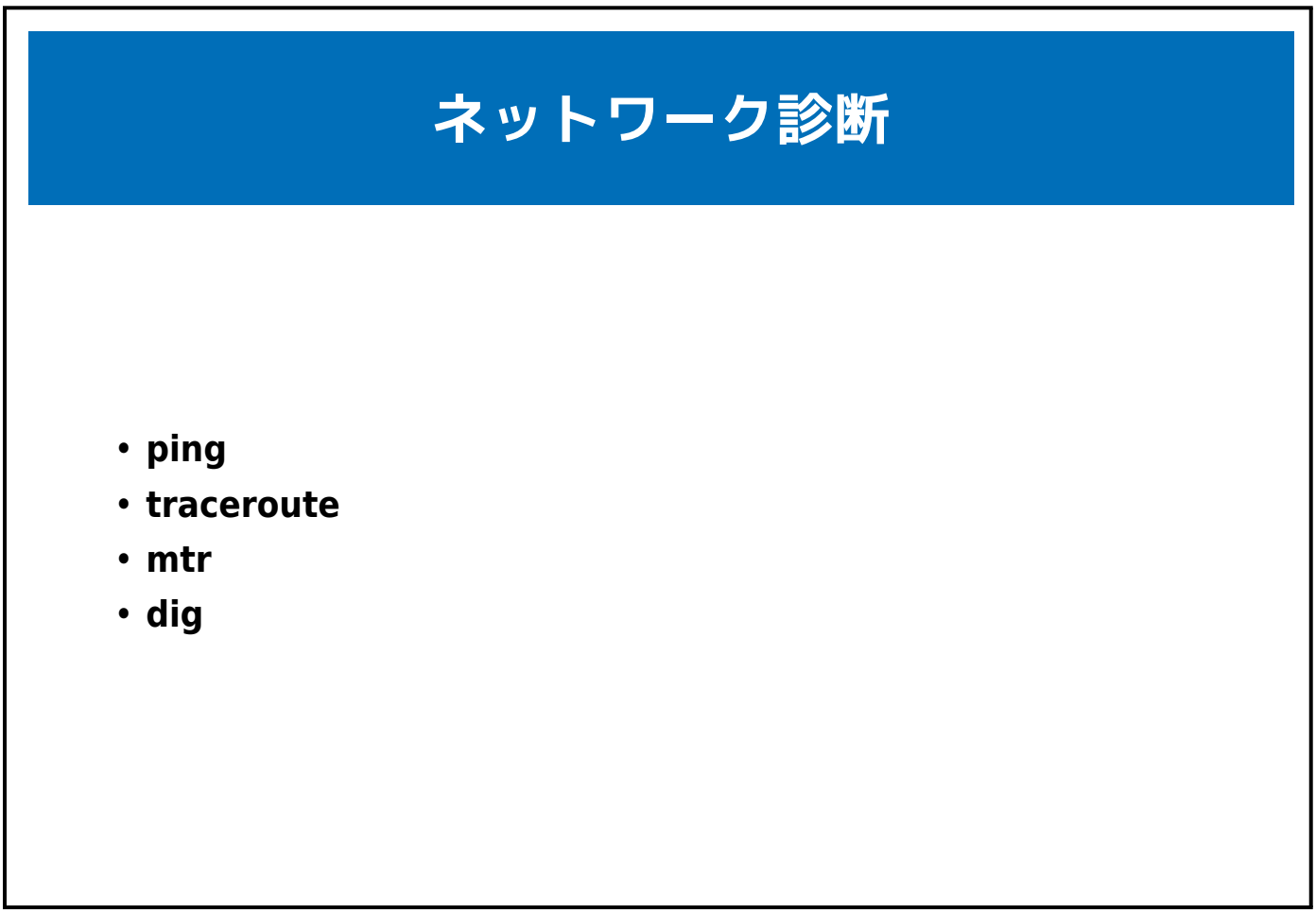
```
search example.com aps.org
nameserver 192.168.1.1
nameserver 8.8.8.8
```

最近のほとんどのシステムでは、次のように /etc/resolv.conf ファイルは自動生成されます:

```
# Generated by NetworkManager
1  192.168.1.1
```

これは、プライマリネットワークインターフェイスで **DHCP** を呼び出している **NetworkManager** が生成しています。

35.9 ネットワーク診断



全てのシステム管理者のツールボックスには、多くの基本的なネットワークユーティリティがあります。具体的には:

ping:
64 バイトのテストパケットを指定されたネットワークホストに送信し、(見つかった場合) 到達に必要だった時間 (ミリ秒単位)、失われたパケット、およびその他のパラメーターについての情報を報告します。正確な出力は対象となるホストによって異なりますが、少なくともネットワークが機能しておりホストまで到達できていることがわかります。

traceroute:
このユーティリティは、宛先へのネットワーク経路を調べるために使用されます。ホストに到達するために通過したルーターパケットフロー (経路) と、各 **ホップ** への到達にかかった時間を示します。

mtr:
ping と **traceroute** の機能を組み合わせて、**top** のように継続的に更新する表示を作成します。

dig:
DNS 機能のテストに役立ちます。**host** や **nslookup** を使用することもできます。これらは、ホストに関する **DNS** 情報を返す昔からあるプログラムです。

RHEL などいくつかの **Linux** ディストリビューションでは、最初の 3 つの診断ユーティリティを実行するために root 権限 (**sudo** でもよい) を要求します。

例:

```
$ ping -c 10 linuxfoundation.org
$ traceroute linuxfoundation.org
$ mtr linuxfoundation.org
```

ping の例

```
[student@fedora ~]$ ping -c 10 linuxfoundation.org
PING linuxfoundation.org (23.185.0.4) 56(84) bytes of data.
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=1 ttl=128 time=21.3 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=2 ttl=128 time=20.2 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=3 ttl=128 time=19.2 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=4 ttl=128 time=20.7 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=5 ttl=128 time=19.8 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=6 ttl=128 time=19.6 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=7 ttl=128 time=19.5 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=8 ttl=128 time=20.3 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=9 ttl=128 time=21.0 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=10 ttl=128 time=20.9 ms

--- linuxfoundation.org ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9012ms
rtt min/avg/max/mdev = 19.247/20.259/21.282/0.670 ms
[student@fedora ~]$
```

図 35.9: **ping**

traceroute の例

```
c8:/tmp>traceroute google.com
traceroute to google.com (172.217.9.78), 30 hops max, 60 byte packets
 1  router (192.168.1.1)  0.340 ms  0.425 ms  0.544 ms
 2  * * *
 3  096-034-031-180.biz.spectrum.com (96.34.31.180)  19.900 ms  19.553 ms  18.011 ms
 4  acr06ftbgwi-gbe-8-46.ftbg.wi.charter.com (96.34.30.244)  19.682 ms  20.144 ms  19.848 ms
 5  cts03alxnmn-tge-3-0-0.alxn.mn.charter.com (96.34.25.137)  26.237 ms  26.465 ms  26.214 ms
 6  crr01rochmn-bue-2.roch.mn.charter.com (96.34.25.166)  26.445 ms  23.845 ms  25.653 ms
 7  crr01stcdmn-bue-6.stcd.mn.charter.com (96.34.25.0)  27.146 ms  21.003 ms  28.613 ms
 8  bbr02slidla-tge-0-1-0-6.slid.la.charter.com (96.34.1.188)  35.569 ms  33.988 ms  31.121 ms
 9  bbr02chcgil-bue-1.chcg.il.charter.com (96.34.1.149)  46.788 ms  46.397 ms  39.472 ms
10  prr01chcgil-bue-4.chcg.il.charter.com (96.34.3.11)  44.474 ms  44.081 ms  43.731 ms
11  096-034-152-159.biz.spectrum.com (96.34.152.159)  42.068 ms 096-034-152-155.biz.spectrum.co
m (96.34.152.155)  42.878 ms 096-034-152-159.biz.spectrum.com (96.34.152.159)  44.781 ms
12  108.170.243.193 (108.170.243.193)  40.572 ms 108.170.243.174 (108.170.243.174)  39.112 ms 1
08.170.243.193 (108.170.243.193)  42.198 ms
13  72.14.239.123 (72.14.239.123)  42.209 ms  39.873 ms 72.14.232.192 (72.14.232.192)  45.645 m
s
14  72.14.239.123 (72.14.239.123)  43.850 ms 108.170.244.2 (108.170.244.2)  43.267 ms 108.170.2
44.15 (108.170.244.15)  42.362 ms
15  ord38s09-in-f14.1e100.net (172.217.9.78)  42.122 ms  41.888 ms  40.801 ms
c8:/tmp>
```

図 35.10: **traceroute**

mtr の例

```
                                My traceroute  [v0.92]
c8 (192.168.1.200)                                        2021-03-26T09:11:33-0500
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                                Packets               Pings
 Host                                         Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. router                                     0.0%    76    0.3   0.3   0.2   0.4   0.0
 2. ???
 3. 096-034-031-180.biz.spectrum.com           0.0%    76    9.5   9.9   8.2  27.5   2.4
 4. acr06ftbgwi-gbe-8-46.ftbg.wi.charter.com   0.0%    76   19.9  11.5   8.8  21.8   3.3
 5. crr01onlswi-bue-402.onls.wi.charter.com    0.0%    76   12.8  14.0  11.9  31.9   3.0
 6. crr01euclwi-bue-400.eucl.wi.charter.com    0.0%    76   14.2  16.0  13.4  32.6   3.2
 7. bbr01euclwi-bue-100.eucl.wi.charter.com    0.0%    76   18.1  18.9  14.2  27.4   2.8
 8. prr01mplsmn-bue-801.mpls.mn.charter.com    0.0%    75   20.8  28.8  19.7 505.9  55.9
 9. 10ge5-18.core1.msp1.he.net                 0.0%    75   20.9  26.3  19.3  44.5   7.6
10. 100ge11-2.core1.sea1.he.net                0.0%    75   89.3  77.4  71.3 101.7   6.6
11. 100ge15-1.core1.pdx1.he.net                0.0%    75   74.4  88.1  74.4 104.2   8.6
12. infinity-internet-inc.gigabitethernet2-11.core1  0.0%  75   74.5  77.0  74.5  89.2   2.9
13. xe-6-0-0.r-1.linuxfoundation.org           0.0%    75   77.1  77.0  74.5  88.1   3.0
14. kernel.org                                 0.0%    75   74.8  79.3  74.2  94.1   5.2
```

図 35.11: **mtr**

35.10 演習

課題 35.1: ネットワークインターフェースを、静的に構成する

注目

eth0 とは違うインターフェース名を使用する場合があります。**nmtui** を使うか、GUI を利用します。ここではコマンドラインによる方法を解説しますが、使用しているディストリビューションによって詳細部分が違うことに気をつけてください。

1. eth0 の現在の IP アドレス、デフォルトルートと **DNS** の設定を表示します。後で元に戻すので、どこかに書き留めておいてください。
2. eth0 を停止させ、先程書き留めた情報に基づいて **DCHP** ではなく静的アドレスを使用するように再構成します。
3. インターフェースを立ち上げ、先程書き留めた情報に基づいてネームサーバリゾルバを構成します。ホスト名を確認して **ping** します。
4. リブート後も設定したとおりに動作することを確認します。

全てを終えたら、もとの構成に戻します。

解 35.1

1. ```
   $ ip addr show eth0
   $ ip route
   $ cp /etc/resolv.conf resolv.conf.keep
   ```

   または、以下を実行します。

   ```
   $ ifconfig eth0
   $ route -n
   $ cp /etc/resolv.conf resolv.conf.keep
   ```

2. ```
   $ sudo ip link set eth0 down
   ```

   または、以下を実行します。

   ```
   $ sudo ifconfig eth0 down
   ```

On RedHat RedHat / CentOS では

Red Hat ベースのシステムでは、/etc/sysconfig/network-scripts/ifcfg-eth0 に以下のように格納されていることを確認します：

/etc/sysconfig/network-scripts/ifcfg-eth0 の内容

```
DEVICE=eth0
BOOTPROTO=static
ONBOOT=yes
IPADDR=noted from step 1
NETMASK=noted from step 1
GATEWAY=noted from step 1
```

On openSUSE OpenSUSE、SLES、Debian 系システムでは

SUSE ベースのシステムの場合 /etc/sysconfig/network の下のファイルを同様に編集します。
Debian ベースのシステムの場合 /etc/networking/interfaces に以下を追加してください:

/etc/sysconfig/network または /etc/networking/interfaces の追加内容

```
iface eth0 inet static
address noted from step 1
netmask noted from step 1
gateway noted from step 1
```

3. `$ sudo ip link set eth0 up`

   または、以下を実行します。

   ```
   $ sudo ifconfig eth0 up
   ```

   ```
   $ sudo cp resolv.conf.keep /etc/resolv.conf
   $ cat /etc/sysconfig/network
   $ cat /etc/hosts
   $ ping yourhostname
   ```

4. ```
   $ sudo reboot
   $ ping hostname
   ```

課題 35.2: 静的にホスト名を追加する

本演習では、ローカルのホストデータベースにエントリを追加します。

1. /etc/hosts を開き mysystem.mydomain を追加します。これは、ネットワークカードに割り当てられた IP アドレスを示します。
2. 次に ad.doubleclick.net への参照が、127.0.0.1 を示すようなエントリを追加します。
3. 追加の演習として http://winhelp2002.mvps.org/hosts2.htm または http://winhelp2002.mvps.org/hosts.txt から、host ファイルをダウンロードしインストールします。新旧の host ファイルで、ブラウザの動作に違いがありましたか？

解 35.2

1. ```
   $ sudo sh -c "echo 192.168.1.180    mysystem.mydomain >> /etc/hosts"
   $ ping mysystem.mydomain
   ```

2. ```
   $ sudo sh -c "echo 127.0.0.1       ad.doubleclick.net >> /etc/hosts"
   $ ping ad.doubleclick.net
   ```

3. ```
   $ wget http://winhelp2002.mvps.org/hosts.txt
   ```

```
1  --2014-11-01 08:57:12--  http://winhelp2002.mvps.org/hosts.txt
2  Resolving winhelp2002.mvps.org (winhelp2002.mvps.org)... 216.155.126.40
3  Connecting to winhelp2002.mvps.org (winhelp2002.mvps.org)|216.155.126.40|:80... connected.
4  HTTP request sent, awaiting response... 200 OK
5  Length: 514744 (503K) [text/plain]
6  Saving to: hosts.txt
7
8  100%[======================================>] 514,744      977KB/s   in 0.5s
9
10 2014-11-01 08:57:13 (977 KB/s) - hosts.txt saved [514744/514744]
```

```
$ sudo sh -c "cat hosts.txt >> /etc/hosts"
```

課題 35.3: nmcli を使い、ネットワークインターフェースに別名/アドレスを付与する

システムに恒久的に別 IPv4 アドレスを追加します。/dev を直接編集するのではなく **nmcli** を利用します。

1. まず、インターネットアドレスとインターフェース名を取得します:

```
$ sudo nmcli con
```

```
NAME           UUID                                  TYPE            DEVICE
Auto Ethernet  1c46bf37-2e4c-460d-8b20-421540f7d0e2  802-3-ethernet  ens33
virbr0         a84a332f-38e3-445a-a377-4363a8eb963f  bridge          virbr0
```

コネクション名称は Auto Ethernet であるとわかります。

```
$ sudo nmcli con show "Auto Ethernet" | grep IP4.ADDRESS
```

```
IP4.ADDRESS[1]:                         172.16.2.135/24
```

アドレスは 172.16.2.135 となっています。このコマンドはコネクションに関する全ての情報を表示し、以下のように NAME ではなく UUID を表示することに注意してください:

```
$ nmcli con show 1c46bf37-2e4c-460d-8b20-421540f7d0e2
```

2. 新しいアドレスを追加します:

```
$ sudo nmcli con modify "Auto Ethernet" +ipv4.addresses 172.16.2.140/24
```

3. インタフェースを活性化し、存在を確認します:

```
$ sudo nmcli con up "Auto Ethernet"
```

```
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection

$ ping -c 3 172.16.2.140
PING 172.16.2.140 (172.16.2.140) 56(84) bytes of data.
64 bytes from 172.16.2.140: icmp_seq=1 ttl=64 time=0.038 ms
64 bytes from 172.16.2.140: icmp_seq=2 ttl=64 time=0.034 ms
64 bytes from 172.16.2.140: icmp_seq=3 ttl=64 time=0.032 ms

--- 172.16.2.140 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.032/0.034/0.038/0.007 ms
```

4. 別名を削除します:

```
$ sudo nmcli con modify  "Auto Ethernet" -ipv4.addresses 172.16.2.140/24
...
$ sudo nmcli con up "Auto Ethernet"
...
```

課題 35.4: nmcli を使い、静的ルートを追加する

システムに、恒久的に静的 IPv4 ルートアドレスを追加します。/dev を直接編集するのではなく **nmcli** を利用します。

1. **route** と **ip** を使い、現在の経路表（ルーティングテーブル）を確認します:

```
$ route
```

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         172.16.2.2      0.0.0.0         UG    100    0        0 ens33
link-local      *               255.255.0.0     U     1000   0        0 ens33
172.16.2.0      *               255.255.255.0   U     100    0        0 ens33
192.168.122.0   *               255.255.255.0   U     0      0        0 virbr0
```

```
$ ip route
```

```
default via 172.16.2.2 dev ens33  proto static  metric 100
169.254.0.0/16 dev ens33  scope link  metric 1000
172.16.2.0/24 dev ens33  proto kernel  scope link  src 172.16.2.135  metric 100
192.168.122.0/24 dev virbr0  proto kernel  scope link  src 192.168.122.1 linkdown
```

2. **nmcli** を使って新しいルートを追加します:

```
$ sudo nmcli conn mod "Auto Ethernet" +ipv4.routes "192.168.100.0/24 172.16.2.1"
```

3. まだ、有効でないことに注意してください:

```
$ route
```

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         172.16.2.2      0.0.0.0         UG    100    0        0 ens33
link-local      *               255.255.0.0     U     1000   0        0 ens33
172.16.2.0      *               255.255.255.0   U     100    0        0 ens33
192.168.122.0   *               255.255.255.0   U     0      0        0 virbr0
```

4. インターフェースをリロードして有効化してください。結果を見ます:

```
$ sudo nmcli conn up "Auto Ethernet"
```

```
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection
```

```
$ route
```

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         172.16.2.2      0.0.0.0         UG    100    0        0 ens33
link-local      *               255.255.0.0     U     1000   0        0 ens33
172.16.2.0      *               255.255.255.0   U     100    0        0 ens33
192.168.100.0   172.16.2.1      255.255.255.0   UG    100    0        0 ens33
192.168.122.0   *               255.255.255.0   U     0      0        0 virbr0
```

5. リブートしてルートが有効であることを確認してください。(つまり **恒久的** になっているということです):
確認したら削除します:

```
$ route
```

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         172.16.2.2      0.0.0.0         UG    100    0        0 ens33
link-local      *               255.255.0.0     U     1000   0        0 ens33
172.16.2.0      *               255.255.255.0   U     100    0        0 ens33
192.168.100.0   172.16.2.1      255.255.255.0   UG    100    0        0 ens33
192.168.122.0   *               255.255.255.0   U     0      0        0 virbr0
```

```
$ sudo nmcli conn mod "Auto Ethernet" -ipv4.routes "192.168.100.0/24 172.16.2.1"
```

```
$ sudo nmcli conn up "Auto Ethernet"
```

```
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection
$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         172.16.2.2      0.0.0.0         UG    100    0        0 ens33
link-local      *               255.255.0.0     U     1000   0        0 ens33
172.16.2.0      *               255.255.255.0   U     100    0        0 ens33
192.168.122.0   *               255.255.255.0   U     0      0        0 virbr0
```

6. 以下のように、コマンドラインから **route** や **ip** を使いルートを設定できますが、これはリブート後には無効になります:

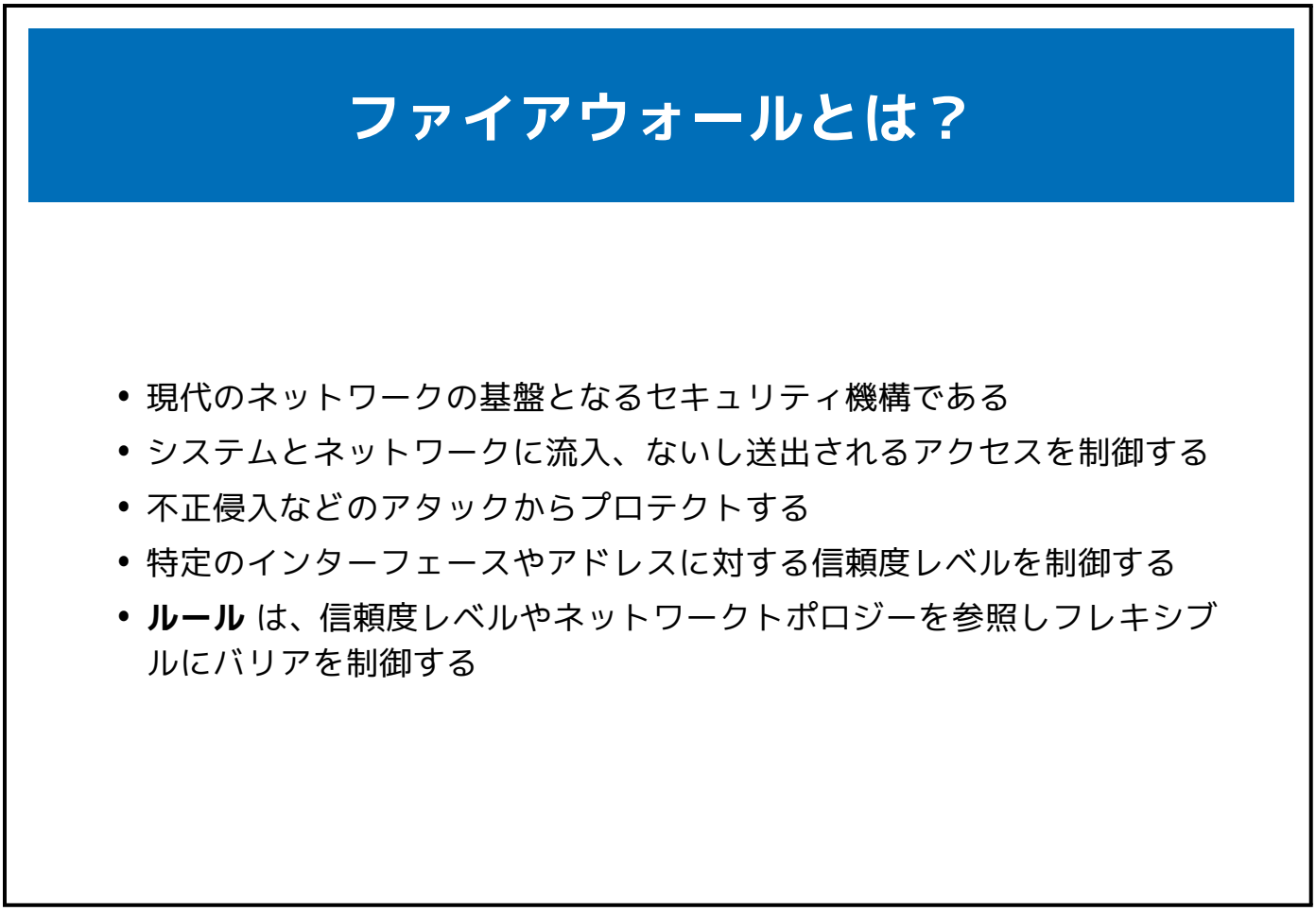
```
$ sudo ip route add 192.168.100.0/24 via 172.16.2.1
$ sudo route
....
```

この方法で設定したルートは、恒久的ではありません。

章 36

ファイアウォール

36.1 ファイアウォール



ファイアウォール は、すべてのネットワークトラフィックの監視と制御を行う、ネットワークセキュリティシステムです。受信と発信の両方のネットワーク接続とパケットに **ルール** を適用し、特定の接続の信頼レベルとネットワークトポロジに応じた柔軟なバリア（つまり、ファイアウォール）を構築します。

ファイアウォールはハードウェアベースでもソフトウェアベースでもかまいません。ファイアウォールはネットワークルーターだけでなく、個々のコンピュータやネットワークノード内に構築することもできます。多くのファイアウォールにはルーティング機能もあります。

パケット フィルタリング

- ファイアウォール構築の基礎となる技術である
- ネットワーク ストリーム上の全パケットがチェック対象となる
- フィルター後に、システムで設定した **ルール** が適用される
- アクションには 拒否、破棄、削除、リダイレクト などがある

ほとんどすべてのファイアウォールは **パケット フィルタリング** を行っています。

情報はネットワークを介してパケット形式で送信されます。これらのパケットには次のものが含まれます:

- ヘッダー (Header)
- ペイロード (Payload)
- フッター (Footer)

ヘッダーとフッターには宛先と送信元のアドレス、パケットの種類、準拠するプロトコルの種類、さまざまなフラグ、ストリーム内のパケット番号、そして転送に関するその他のあらゆるメタデータに関する情報が含まれています。

実際のデータは、ペイロードに格納されます。

パケットフィルタリングは、アプリケーション、トランスポート、ネットワーク、データリンクなど、ネットワーク伝送の複数の階層でパケットをインターセプトします。

ファイアウォールは、各パケットが次のことを行うための一連の **ルール** を確立します:

- 内容、アドレスなどに基づいて、承認または拒否する
- 何らかの方法で破棄する
- 別のアドレスにリダイレクトさせる
- セキュリティに関して調査する
- その他

パケット フィルタリングの結果を踏まえて実行されるルールとアクションを決定する、さまざまなユーティリティが存在します。

ファイアウォールの歴史

- **パケット フィルタリング**:
  パケットを監査し、パケットを破棄、拒否、転送する
- **ステートフル フィルター**:
  接続ステート を検査し、新規接続 か 既存接続の延長 か それ以外 かを判別する
- **アプリケーションレイヤー ファイアウォール**:
  アプリケーションが利用するプロトコルパケットを認識する

初期のファイアウォール（1980 年代後半に遡ります）は、**パケット フィルタリング** に基づくものでした。各ネットワークパケットのコンテンツを検査し破棄、拒否、または転送していました。パケットが含まれるストリームの通信状態に関する **接続ステート (connection state)** は考慮されていませんでした。

次の世代のファイアウォールは **ステートフル フィルタ** に基づいており、パケットの **接続ステート** も調査して、新しい接続であるか 既存の接続の延長なのか を判定材料に利用します。Dos 攻撃はこの種のファイアウォールを攻撃して過剰な負荷をかけようとします。

第３世代ファイアウォールは **アプリケーションレイヤー ファイアウォール** と呼ばれ、アプリケーションがどんなプロトコルで接続しているかを認識します。通常のフローと異なるものはすべて拒否できます。

36.2 インターフェイス

ファイアウォール の インターフェイス と ツール

- コマンドラインツール:
  - **iptables**
  - **firewall-cmd**
  - **ufw**
- グラフィカルインターフェース
  - **system-config-firewall**
  - **firewall-config**
  - **gufw**
  - **yast**
- **shorewall** など、これ以外にもユーティリティが存在する
- コマンドラインツールは、ディストリビューションによる差異が少ない

システムへのファイアウォール設定には、次の 2 つ進め方があります:

- コマンドラインから比較的低レベルのツールを利用し、併せて `/etc` 配下にある **iptables**、**firewall-cmd**、**ufw** などの各種構成ファイルを編集する。
- 堅牢なグラフィカルインターフェイスを利用する: **system-config-firewall**、**firewall-config**、**gufw**、**yast** など

私達は以下の理由から、低レベルのツールを使用することにします。

- 低レベルツールは、グラフィカルなツールほど頻繁に変更されない。
- 低レベルツールの方が、より多くの機能を持っていることが多い。
- 低レベルツールは、ディストリビューション毎の違いが少ない。**GUI** は 1 つのディストリビューション ファミリーに限定されディストリビューション毎にかなり異なる傾向がある

欠点は、最初は習得が難しいと感じることです。

firewall-cmd と **firewall-config** の両方を含む最新の **firewalld** パッケージに絞って説明します。デフォルトでこれらが含まれていないディストリビューションでも、ソースから比較的簡単にインストールすることができます。演習で必要になった時には我々もそうします。

iptables を使用しない理由

- 最近のファイアウォールの多くが **iptables** を使用している
- **iptables** は、これから説明していく **firewalld** と共通のカーネルファイアウォール実装インターフェースを利用している
- **iptables** を使って有用な設定ができるまで理解するにはとても時間がかかることから、今回は取り上げないことにした
- **iptables** は **Linux Foundation** のシステム管理者シリーズコースの **LFS311: Linux for System Engineers** で詳しく紹介している
- コースの詳細は https://training.linuxfoundation.org/linux-courses/system-administration-training/linux-for-system-engineers を参照

36.3 firewalld

firewalld と firewall-cmd

- **Dynamic Firewall Manager** (**firewalld**)
- **zones** と協調動作する
- **IPv4** と **IPv6** の両方をサポートする
- **一時的な変更** と **恒久的な変更** の設定を分離できる
- メインのツールは **firewall-cmd** である
- **iptables** ユーティリティの置き換えとなる
- **systemd** の構成要素である
- 徐々に主要なディストリビューションに採用されはじめた、ソースからのインストールも可能である

firewalld は **Dynamic Firewall Manager** です。ネットワークインターフェイスや接続の信頼レベルが定義されているネットワーク／ファイアウォールの **ゾーン** と協調動作します。これは **IPv4** と **IPv6** の両方のプロトコルをサポートします。

さらに構成情報の **ランタイム変更 (= 一時的な変更)** と **恒久的な**（持続的な）変更を分け、ファイアウォール ルールを追加するためのサービス、またはアプリケーションとのインターフェイスも持っています。

設定ファイルは `/etc/firewalld` と `/usr/lib/firewalld` に保存されます。`/etc/firewalld` 内のファイルは他のディレクトリの設定ファイルを上書きするものなので、システム管理者が編集する必要があります。

コマンドラインツールは **firewall-cmd** です。この先で説明していきます。

次に進む前に、以下を実行してみることをお勧めします:

```
$ firewall-cmd --help
```

```
Usage: firewall-cmd [OPTIONS...]
....
Status Options
  --state                Return and print firewalld state
  --reload               Reload firewall and keep state information
  --complete-reload      Reload firewall and loose state information
  --runtime-to-permanent Create permanent from runtime configuration
....
```

全部で約 200 行あるのでここに全部掲載できません。しかし殆どのオプションは、名前から何をするものか理解できるでしょう。

firewalld は **iptables** を置き換えるサービスです。両方のサービスを同時に実行するとエラーになります。

firewalld サービスの状態

- **firewalld** は他サービスと同様に 有効化／無効化、起動／停止 ができる

```
$ sudo systemctl [enable/disable] firewalld
$ sudo systemctl [start/stop] firewalld
```

- ステータの表示:

```
$ sudo systemctl status firewalld
$ sudo firewall-cmd --state
```

- ２つ以上のネットワークインターフェースがある場合は **IP フォワード** を有効化する必要がある:

```
$ sudo sysctl net.ipv4.ip_forward=1
$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

firewalld はファイアウォールの使用と構成を実行するサービスで、上に示した一般的なやり方で 有効化／無効化、開始／停止ができます。次のいずれかの方法で、現在の状態を表示できます:

```
$ sudo systemctl status firewalld
```

```
firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled)
   Active: active (running) since Tue 2015-04-28 12:00:59 CDT; 5min ago
 Main PID: 777 (firewalld)
...
```

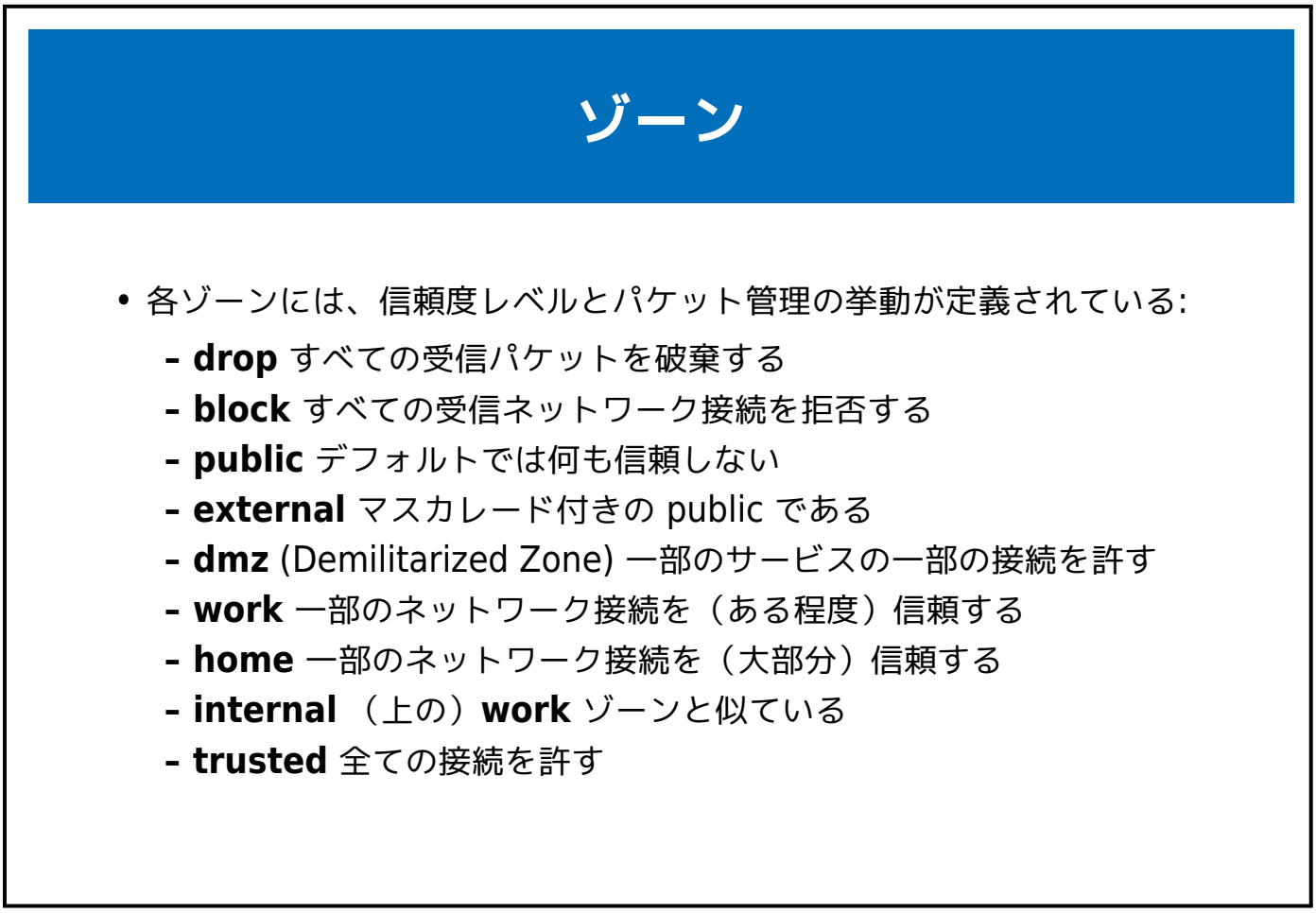
```
$ sudo firewall-cmd --state

running
```

IPv4 を使っている時に、２つ以上のネットワークインターフェイスがある場合には **IP フォワード** を有効にする必要があります。ランタイムで（動作中に）これを行うには、次のいずれかを実行します:

```
$ sudo sysctl net.ipv4.ip_forward=1
$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

(**echo** を正しく実行させるには、2 番目のコマンドを root として実行する必要があります。) ただし、これは恒久的ではありません。恒久化するには `net.ipv4.ip_forward=1` という行を `/etc/sysctl.conf` に追加して再起動するか `sudo sysctl -p` とタイプして再起動無しで設定変更を読み込ませる必要があります。

36.4 ゾーン



firewalld は **ゾーン** と連携して動作します。それぞれのゾーンには、信頼レベルと受信／発信パケットに対する特定の動作が定義されています。各インターフェイスは特定のゾーンに属します（通常は **NetworkManager** が該当するゾーンを **firewalld** に通知します）。ゾーンは **firewall-cmd** または **firewall-config** GUI で変更できます。ゾーンには:

- **drop**: すべての受信パケットは応答せずに破棄されます。発信接続のみが許可されます。
- **block**: すべての受信ネットワーク接続が拒否されます。許可される接続はシステム内からの接続のみです。
- **public**: ネットワーク上のコンピュータを信頼しません。意識的に選択された特定の着信接続のみが許可されます。
- **dmz**: (Demilitarized Zone) 一部の（すべてではない）サービスへのアクセスを公開する場合に使用します。特定の受信接続のみが許可されます。
- **external**: ルーターなどで **マスカレード** が使用されている場合に使用されます。信頼レベルは **public** ゾーンと同じです。
- **work**: 接続されたノードを（完全ではありませんが）無害であると信頼します。特定の受信接続のみが許可されます。
- **home**: ほとんどの場合他のネットワークノードを信頼しますが、許可する受信接続を選択します。
- **internal**: **work** ゾーンに似ています。
- **trusted**: すべてのネットワーク接続が許可されます。

システムインストール時に、すべてではないにしてもほとんどの **Linux** ディストリビューションは、すべてのインターフェイスのデフォルトに **public** ゾーンを選択します。説明したいくつかのゾーンの違いは明確ではありません。ここでは詳細に説明しませんが、必要以上にオープンなゾーンの使用は避けてください。

ゾーン管理の例 1

- デフォルトゾーンの取得する:

```
$ sudo firewall-cmd --get-default-zone
public
```

- 現在使用されているゾーンのリストを取得する:

```
$ sudo firewall-cmd --get-active-zones
public
  interfaces: eno16777736
```

- 利用可能な全てのゾーンをリストする:

```
$ sudo firewall-cmd --get-zones
block dmz drop external home internal public trusted work
```

```
$ firewall-cmd --help
....
Zone Options
  --get-default-zone   Print default zone for connections and interfaces
  --set-default-zone=<zone>
                       Set default zone
  --get-active-zones   Print currently active zones
  --get-zones          Print predefined zones [P]
  --get-services       Print predefined services [P]
  --get-icmptypes      Print predefined icmptypes [P]
  --get-zone-of-interface=<interface>
                     Print name of the zone the interface is bound to [P]
  --get-zone-of-source=<source>[/<mask>]
                 Print name of the zone the source[/mask] is bound to [P]
  --list-all-zones  List everything added for or enabled in all zones [P]
  --new-zone=<zone>    Add a new zone [P only]
  --delete-zone=<zone> Delete an existing zone [P only]
  --zone=<zone>  Use this zone to set or query options, else default zone
                       Usable for options maked with [Z]
  --get-target         Get the zone target [P] [Z]
  --set-target=<target>
                       Set the zone target [P] [Z]
```

ゾーン管理の例 2

- デフォルトゾーンを **trusted** 変更し、更に変更を元に戻す:

```
$ sudo firewall-cmd --set-default-zone=trusted
success
$ sudo firewall-cmd --set-default-zone=public
success
```

- 一時的にインターフェースを特定のゾーンに割り当てる:

```
$ sudo firewall-cmd --zone=internal --change-interface=eno1
success
```

- それを恒久化する:

```
$ sudo firewall-cmd --permanent --zone=internal --change-interface=eno1
success
```

`/etc/firewalld/zones/internal.xml` が作成される

全ての **firewalld** の制御は **firewall-cmd** プログラムを介して行います。より詳細な情報は、次の方法で取得できます:

```
man firewalld-cmd
```

ゾーン管理の例 3

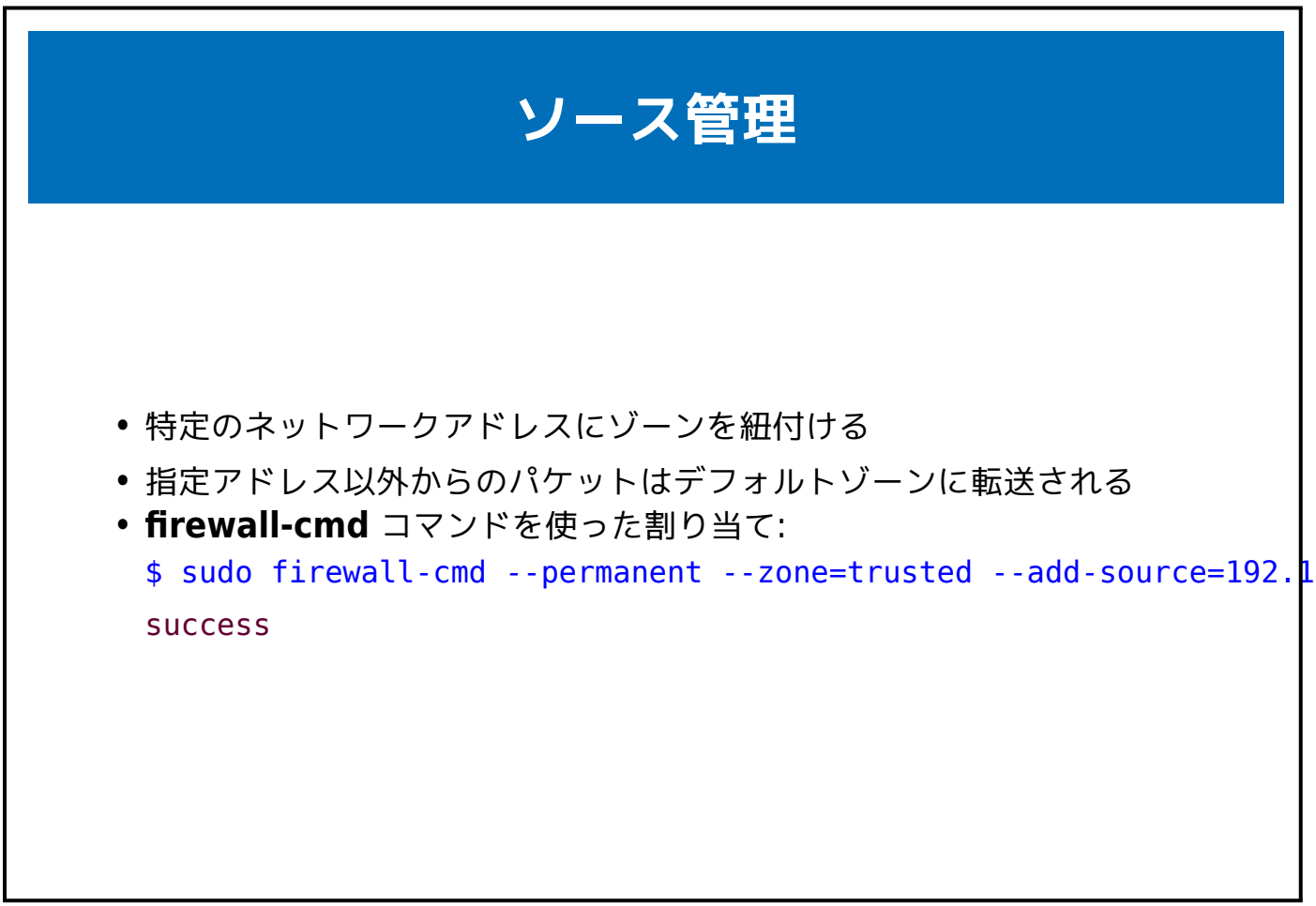
- 特定のインターフェースに関連付けられたゾーンを突き止めるには:

```
$ sudo firewall-cmd --get-zone-of-interface=eno1
public
```

- 特定のゾーンに関する、全ての詳細情報を取得するには:

```
$ sudo firewall-cmd --zone=public --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: eno1
  sources:
  services: chromecast libvirt libvirt-tls nfs nfs3 rsyncd ssh
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

36.5 ソース管理



どのゾーンも、ネットワークインターフェイスだけでなく、特定のネットワークアドレスに結び付けることもできます。
次の場合、パケットはそのゾーンに割り当てられます:

- 既にそのゾーンに結び付けられているアドレスからパケットを取得している場合。もしそうでなければ、
- ゾーンに結び付けられたインターフェイスから取得している場合

上記の取得方法に当てはまらないパケットは、デフォルトゾーン（つまり、通常は **public**）に割り当てられます。
ソースをゾーンに（恒久的に）設定するには:

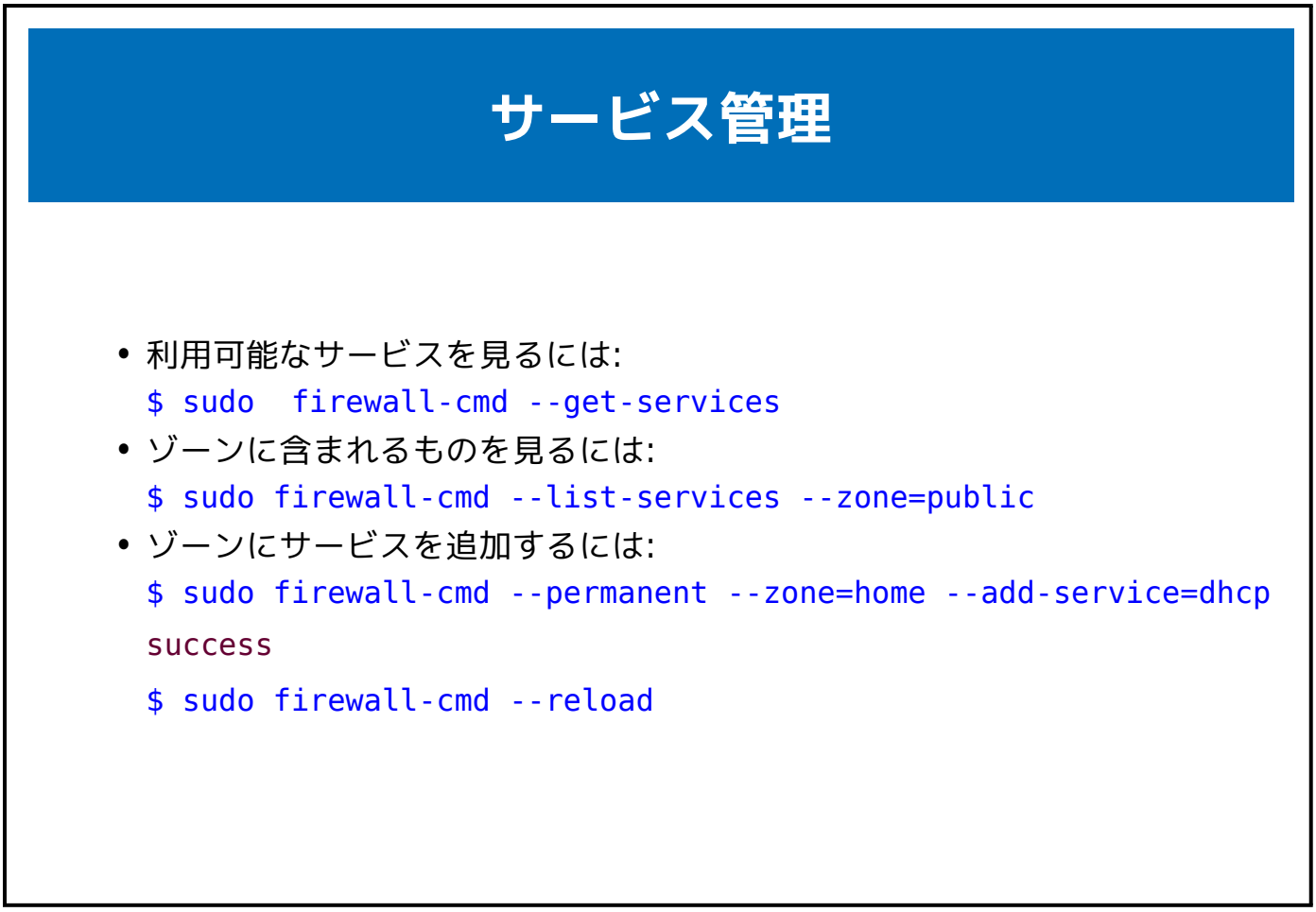
```
$ sudo firewall-cmd --permanent --zone=trusted --add-source=192.168.1.0/24
success
```

これは `192.168.1.x` の IP アドレスを持つソースをすべて **trusted** ゾーンに追加することを意味します。
`--remove-source` オプションを使用すれば、以前に割り当てられたソースをゾーンから削除することができます。
また `--change-source` を使用すれば、ゾーンを変更できることも覚えておいてください。
ゾーンに結び付けられたソースをリストするには:

```
$ sudo firewall-cmd --permanent --zone=trusted --list-sources 192.168.1.0/24
```

上記のどちらのコマンドも `--permanent` オプションを省略すると、現在のランタイム動作のみに適用されます。

36.6 サービスとポートの管理



これまで、特定のインターフェイスやアドレスをゾーンに割り当てる方法を説明しましたが、ゾーン内でアクセスできる **サービス** や **ポート** については説明していませんでした。利用可能なすべてのサービスを表示するには:

```
$ sudo  firewall-cmd --get-services
RH-Satellite-6 amanda-client bacula bacula-client dhcp dhcpv6 dhcpv6-client dns ftp high-availability http
   https imaps ipp ipp-client ipsec kerberos kpasswd ldap ldaps libvirt libvirt-tls mdns mountd ms-wbt\
    mysql nfs ntp openvpn pmcd pmproxy pmwebapi pmwebapis pop3s postgresql proxy-dhcp radius rpc-bind samb
    samba-client smtp ssh telnet tftp tftp-client transmission-client vnc-server wbem-https
```

特定のゾーン内で現在利用可能なサービスを見るには:

```
$ sudo firewall-cmd --list-services --zone=public
dhcpv6-client ssh
```

ゾーンにサービスを追加するには:

```
$ sudo firewall-cmd --permanent --zone=home --add-service=dhcp
success
```

```
$ sudo firewall-cmd --reload
```

２番目のコマンドによる変更を有効にするためには `--reload` の指定が必要です。`/etc/firewalld/services` のファイルを編集して新しいサービスを追加することもできます。

ポート管理

- ポート管理:

```
$ sudo firewall-cmd --zone=home --list-ports
$ sudo firewall-cmd --zone=home --add-port=21/tcp
```

ポート管理は、サービス管理に非常に似ています:

```
$ sudo firewall-cmd --zone=home --add-port=21/tcp
success
```

```
$ sudo firewall-cmd --zone=home --list-ports
21/tcp
```

`/etc/services` を見ると port 21 が **ftp** に対応していることが確認できます:

```
$ grep " 21/tcp" /etc/services
ftp             21/tcp
```

36.7 演習

課題 36.1: firewalld のインストール

ほとんどの **Linux** ディストリビューションには **firewalld** パッケージ（多目的に使える **firewall-cmd** ユーティリティを含む）が含まれています。まだインストールされていない場合もあります。

まず、次のコマンドでインストール済みかチェックします。

```
$ which firewalld firewall-cmd
```

```
/usr/sbin/firewalld
/usr/bin/firewall-cmd
```

プログラムが見つからなかったら、ディストリビューションに応じた以下のいずれかの方法でインストールします:

```
$ sudo dnf install firewalld
$ sudo zypper install firewalld
$ sudo apt-get install firewalld
```

インストールできない場合には **firewalld** パッケージがディストリビューションに含まれていません。ソースからインストールする必要があります。

ソースからインストールするために、https://fedorahosted.org/firewalld/ に行き、**git** ソースのレポジトリから入手するか、最新の tar ボールををダウンロードします。

以下の手順でソースからインストールします（最新バージョンを利用してください）:

```
$ tar xvf firewalld-0.3.13.tar.bz2
$ cd firewalld-0.3.13
$ ./configure
$ make
$ sudo make install
```

このソースには **アンインストール** 機能も含まれています: 不要になった場合は、

```
$ sudo make uninstall
```

`./configure` を実行してライブラリなどで不足するものを見つけたら、パッケージからのインストールの場合はディストリビューションが対応してくれますが、ソースからのインストールの場合は少々面倒です。**Linux Foundation** の **ready-for.sh** スクリプトを実行すれば、ほとんど問題はおきません。

課題 36.2: firewall-cmd を使う

firewalld の使用方法の表面的なことだけを見てみます。わかりやすいオプションを持ち、さまざまなタスクを実行してくれる **firewall-cmd** を実行することで大半のことが行えます。

これを確認するためには以下を実行します：

```
$ firewall-cmd --help
```

```
Usage: firewall-cmd [OPTIONS...]
....
Service Options
  --new-service=<service>
                        Add a new service [P only]
  --delete-service=<service>
                        Delete and existing service [P only]
....
```

RHEL 8 システムでは、409 行もあるので省略します。

もっと詳細な説明に興味がある場合は、**man firewall-cmd** を実行すれば、概要、/etcにある構成ファイル、**zones** と **services** に関する **man** ページを見つけることができます。

課題 36.3: ゾーンにサービスを追加する

パブリックゾーンに **http** と **https** サービスを追加し、正しく表示されるか確認します。

解 36.3

```
$ sudo firewall-cmd  --zone=public --add-service=http
```

```
1 success
```

```
$ sudo firewall-cmd  --zone=public --add-service=https
```

```
1 success
```

```
$ sudo firewall-cmd --list-services --zone=public
```

```
1 dhcpv6-client http https ssh
```

注意：以下を実行していれば、

```
$ sudo firewall-cmd --reload
$ sudo firewall-cmd --list-services --zone=public
```

```
1 dhcpv6-client ssh
```

新サービス追加後リストから消えます。ちょっと奇妙ですが、これは新しいサービスを追加した時に `--permanent` フラグを指定しなかったためです。`--reload` オプションは永続化されたサービスのみを表示するものだからです。

課題 36.4: firewall GUI を使う

各ディストリビューションは、それぞれのファイアウォール用にグラフィカルなインターフェースを持っています。**Red Hat** ベースのシステムでは **firewall-config**、**Ubuntu** では **gufw**、そして **openSUSE** では **yast** に組み込まれています。

ディストリビューションに依存しないように、コマンドラインについて説明しました。しかし、比較的容易なファイアウォールを構成する場合は、GUI を利用することで効率的に進めることができるでしょう。

ファイアウォールの GUI 構成ツールを使って、**パブリック** ゾーンに **http** と **https** を追加してみてください。その効果を実感してください。

GUI の理解にも時間を割いてください。

章 37

システムの起動とシャットダウン

37.1 ブートシーケンスの理解

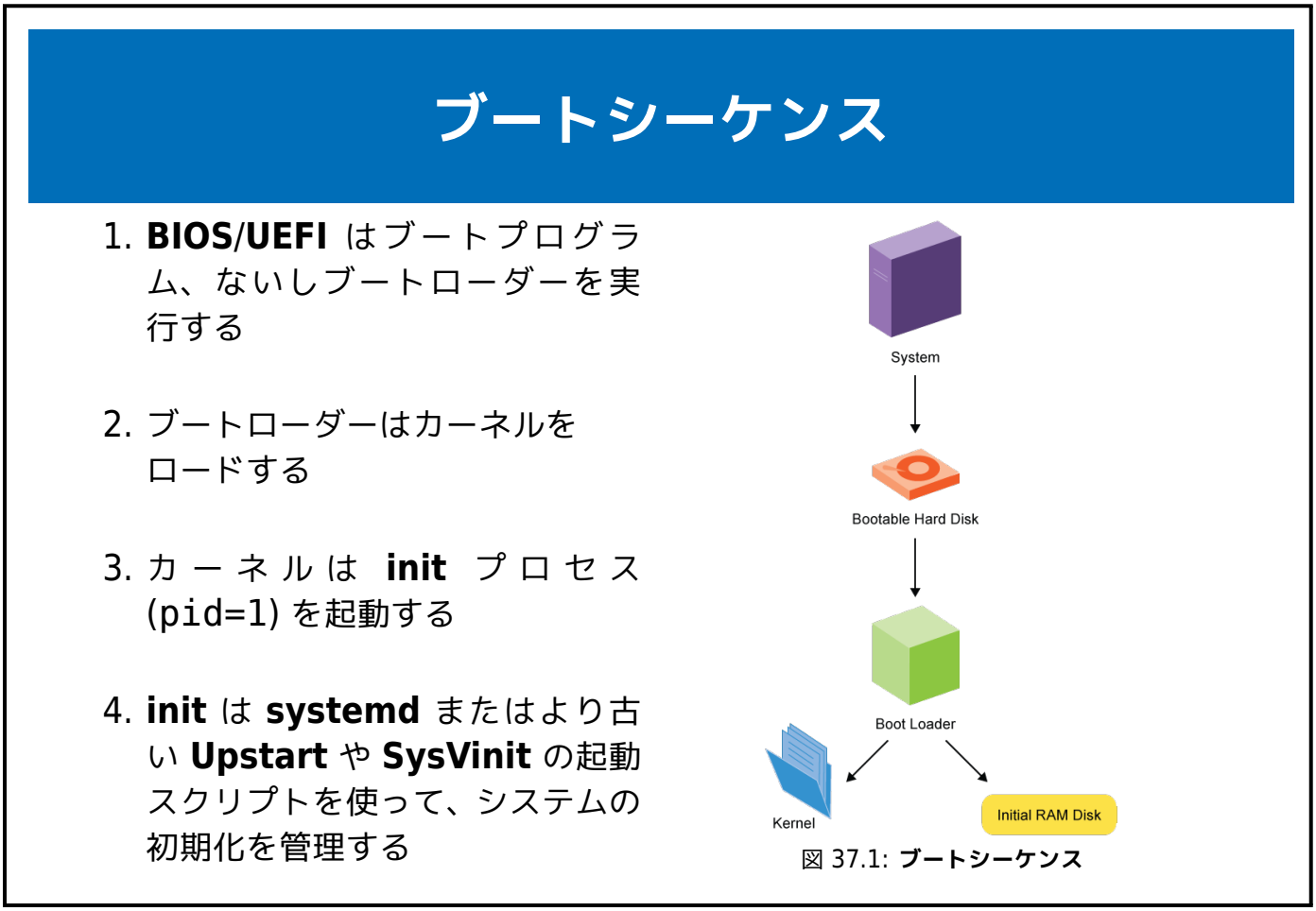


図 37.1: **ブートシーケンス**

コンピュータに電源が供給されると、コンピュータは **BIOS** (**B**asic **I**nput **O**utput **S**ystem) が命令した操作を実行します。

まず **BIOS** は **POST** (**P**ower **O**n **S**elf **T**est) を実行します。**POST** はメモリーとハードウェアをチェックし、特定の場所またはデバイスからブートプログラムを探します。通常ブートプログラムは、デバイスの **MBR** (**M**aster **B**oot **R**ecord) もしくは **UEFI** にあります。コンピュータの制御は、このブートプログラム（通常は **GRUB**）に移ります。

次に、ブートプログラムがカーネルをメモリーにロードし実行します。**x86** プラットフォーム（および他の多くのプラットフォーム）では、最初にカーネルを解凍する必要があります。次にハードウェアチェックを実行し、重要な周辺機器ハードウェアにアクセスし、最終的に **init** プロセスを実行します。**init** プロセスはシステム起動を引き継いで **SysVinit** が使用されている場合、もしくは **Upstart** や **systemd** で管理されている場合は、（その中で定義されている）**init** スクリプトを実行します。

最近のコンピュータは **BIOS** の代替である **UEFI** に移行していますが **BIOS** の大部分の機能が **UEFI** に継承されています。

BIOS

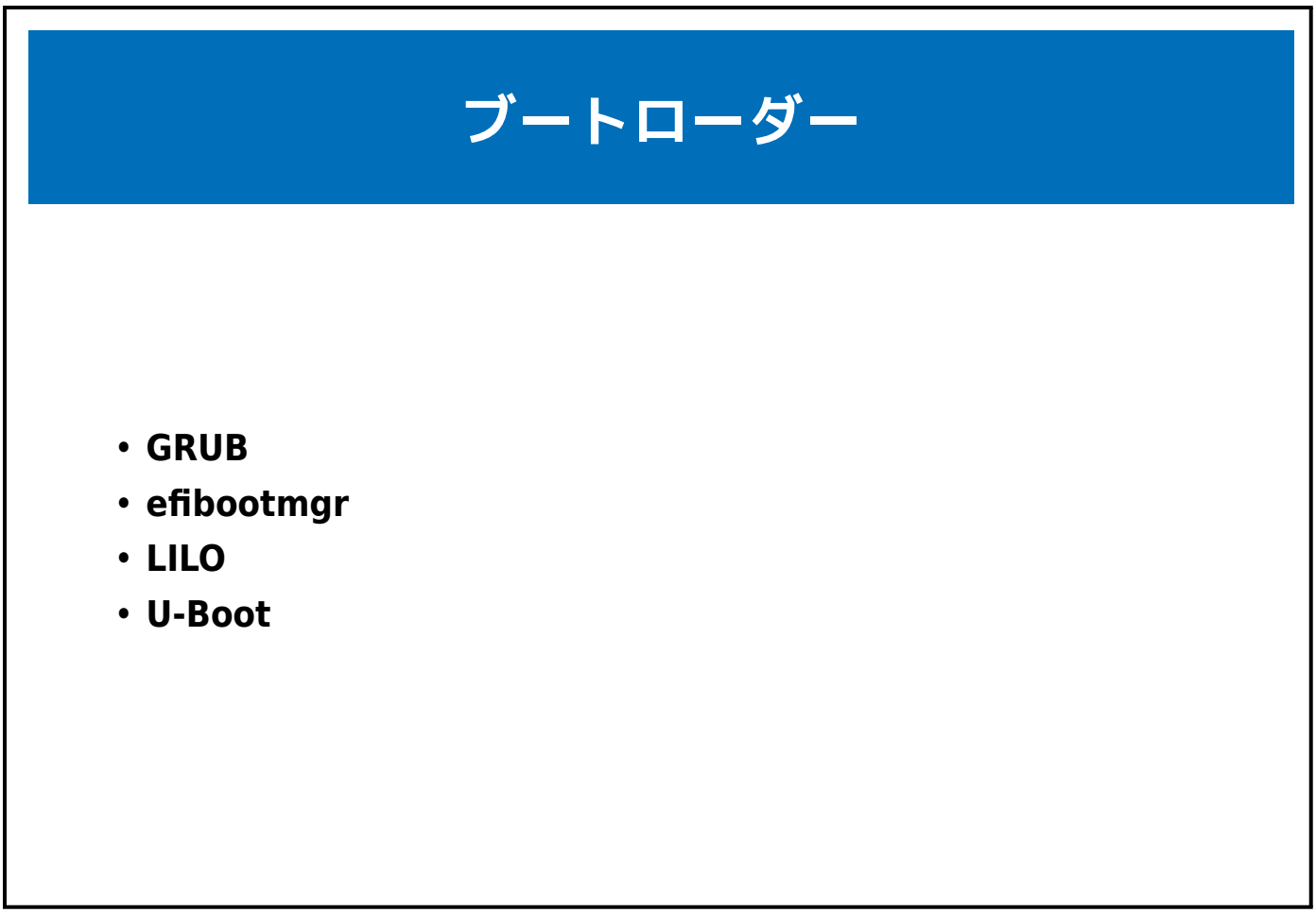
- 通常は **ROM** チップに書かれている
- キーボード、ディスプレイ、ディスクを制御するコードが含まれる
- ブート中にブートローダーをロードする

x86 アーキテクチャでは **BIOS** にはキーボード、ディスプレイ、ディスクドライブ、シリアル通信、およびその他の機能を利用するための最小限の初期化コードが含まれています。OS による完全なシステム初期化処理の中で、これらのデバイスのほとんどは専用のデバイスドライバを再ロードして、デバイスの全機能が利用できるようにします。

BIOS は通常コンピュータ内の **ROM** チップに書き込まれています。（**BIOS ROM** とも呼ばれます）**ROM** なので特別な初期化の必要はなく、仮にディスク障害が起こっても **BIOS** が損傷することはありません。また、他のデバイスの助けを借りずにシステムを起動させることが可能です。

ブートプロセス中に **BIOS** はブートローダーをロードします。

37.2 ブートローダー



事実上（組み込み系以外の）すべての最新 **Linux** ディストリビューションは **GRUB** (**GR**and **U**nified **B**oot Loader) を使用します。**GRUB** には複数のオペレーティングシステムを起動する機能、シリアルケーブル接続でも使いやすいグラフィカルおよびテキストベースのインターフェイス、インタラクティブな起動設定も可能な強力なコマンドラインインターフェイス、ネットワークベースのディスクレスブート、およびその他の高度な機能が含まれます。

efibootmgr は実際にはブートローダーではなくブートマネージャーです。マルチブートの EFI システムで **GRUB** と組み合わせて使用されます。

Linux Loader（**LILO**）は古くて時代遅れです。

U-Boot は組み込みシステムで広く使われています。他にも **bareboot** などいくつかのブートローダーがありますが、このコースでは組み込みシステムは対象にしていません。

37.3 /etc のシステム構成ファイル

/etc の構成ファイル

- **Linux** ディストリビューションは特定ファイルについては、決まった場所に配置しようとする
- システム横断的な設定ファイルは、通常 `/etc` のサブディレクトリに配置される
- ユーザー固有の設定ファイルは、各ユーザーのホームディレクトリに配置される
- **Red Hat** 系システムは全て `/etc/sysconfig` を広範囲に利用する
- **Debian** 系システムは皆 `/etc/default` を使う
- **RHEL** と **SUSE** は両方を利用する

構成情報の保管に関する一般的なルールは:

- システム全体の構成ファイルは通常 `/etc` に置かれる
- ユーザー固有の構成ファイルは `/home/[username]` に置かれる

ただし、このルールが当てはまらない場合もあります。例えば、デフォルトの構成情報は `/usr/lib/systemd` に保存されているでしょうが `/etc/systemd` 内のファイルによって上書きされる可能性があるのです。

`/etc/` 下にはテキストファイルだけが置かれます、バイナリ形式やデータは置かれません。

/etc/sysconfig

- このディレクトリとサブディレクトリにあるファイルは、さまざまなシステムユーティリティサービスに利用される
- これらのファイルの内容は、システムがサービスを起動／停止する時やステータスを問い合わせた時に調査、引用される
- 例えば **RHEL** システムでは:

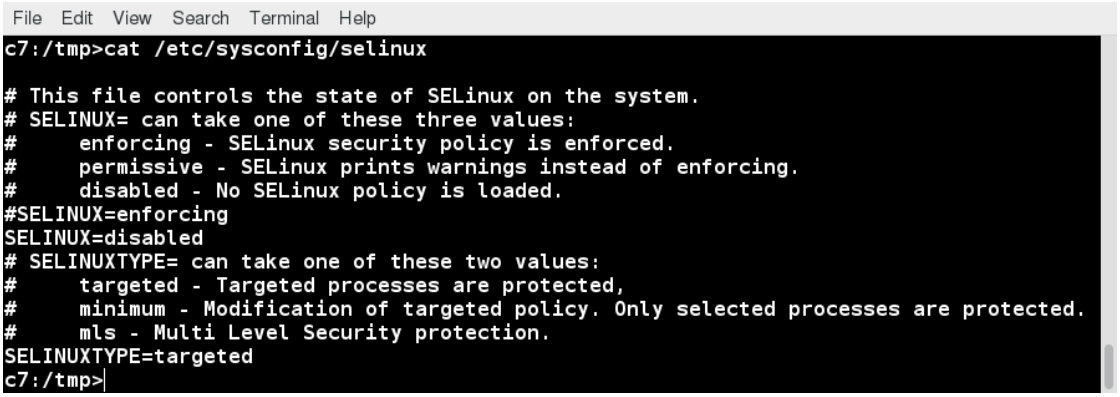


図 37.2: **/etc/sysconfig/selinux の構成事例**

RHEL システムでは:

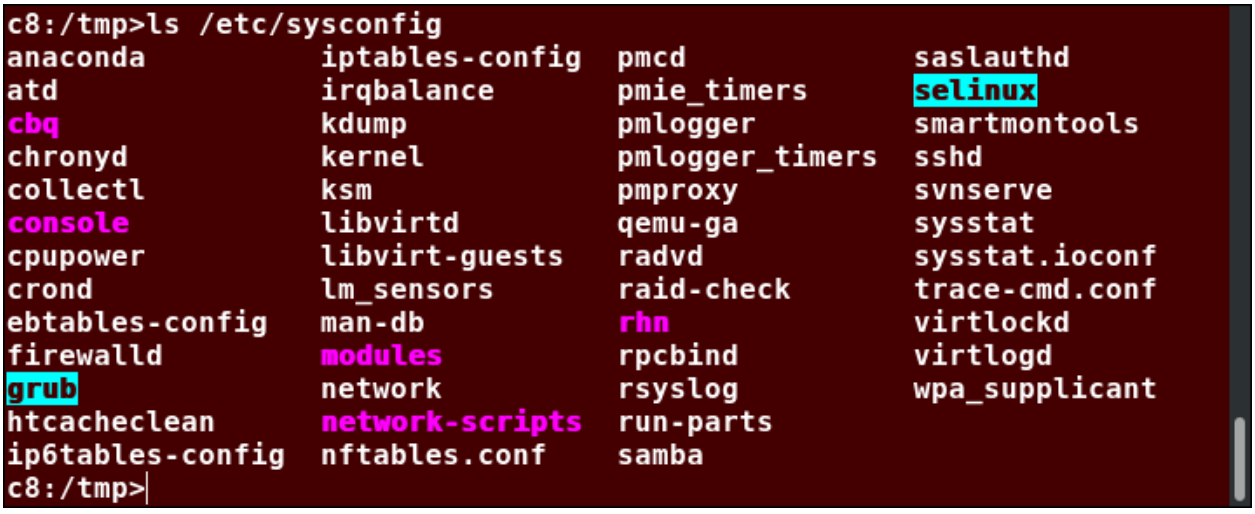


図 37.3: **RHEL 8 の /etc/sysconfig**

/etc/default

- **Red Hat** における `/etc/sysconfig` と似たような使われ方である
- ファイルにはサービス起動時の追加的なオプションが定義されている
- 環境変数を設定するコードが含まれている
- 例えば `/etc/default/useradd` には、ユーザーを新規作成した時のデフォルト値が含まれている
- 現在では **Red Hat** 系を含む全ての主要な **Linux** ディストリビューションは `/etc/default` を持っている（まだ `/etc/sysconfig` が残っていたとしても）

Ubuntu システムでは:

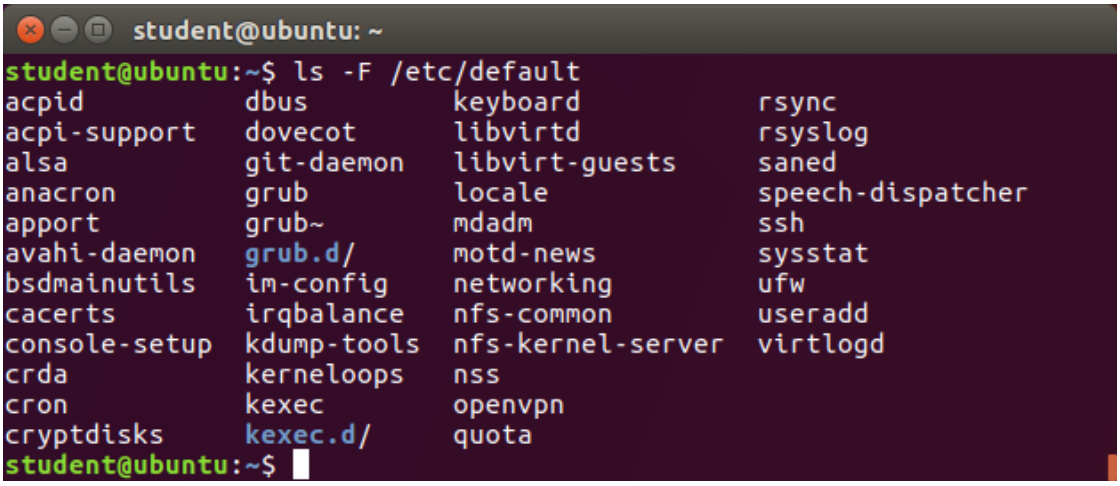


図 37.4: **/etc/default**

37.4 シャットダウンと再起動

シャットダウンと再起動

- **シャットダウン**:
  - システムを、安全に停止させることである
  - 全てのユーザーに、未完了のアクションを警告する
  - 停止と再起動ができる
  - オプションを付けない時のデフォルトは、完全にパワーオフさせる
    - **Ubuntu** などいくつかのディストリビューションは、代わりにシングルユーザーモードに移行させる
- **reboot**、**halt**、**poweroff** といった歴史あるコマンドが今でも頻繁に利用されている

shutdown コマンドは、システムを安全な方法で停止するために使用します。システムが停止することをすべてのユーザーに通知してから、正常かつ安全な方法で停止します。シャットダウン後にはシステムは停止または再起動されます。

例:

```
$ shutdown -h +1 "Power Failure imminent"
$ shutdown -h now
$ shutdown -r now
$ shutdown now
```

ベテランユーザーが頻繁に使用する **reboot**、**halt**、**poweroff** などの歴史のあるコマンドもあります。

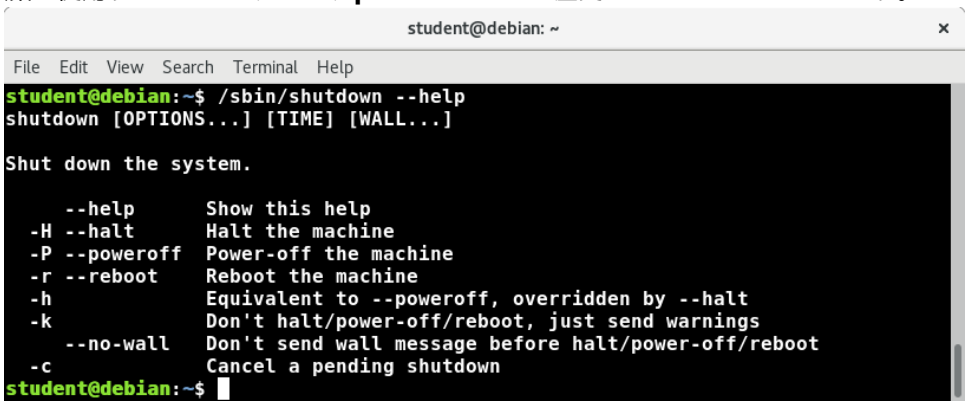


図 37.5: **シャットダウンコマンド**

37.5 演習

課題 37.1: シャットダウン VS. 停止（Halt）VS. リブート

注意：本演習はコンソールから実行してください。（つまり、SSH を利用したネットワーク経由では実行しないでください。）

1. **shutdown** を使って、システムをリブートします。
2. **shutdown** を使って、システムの電源を切ります。
3. システムの電源を入れます。

解 37.1

1. `$ sudo shutdown -r now`
2. `$ sudo shutdown -h now`
3. 電源ボタンを入れるか、仮想マシンを再スタートします。

章 38

GRUB

38.1 GRUB (The Grand Unified Boot Loader)

GRUB (The Grand Unified Boot Loader)

- 起動を選択する
  - オペレーティングシステムを選択する
  - **Linux** カーネルバージョンを選択する
  - **initramfs** イメージを選択する
- 引数を渡す
  - メニュー編集モードを利用して、既存のパラメータを変更する
  - **GRUB** コマンドライン経由で、対話的に起動コマンドを発行する
- **RHEL 6** 系以外の主要な **Linux** ディストリビューションは、全て バージョン 1 から 2 に更新されている

事実上（組み込み系以外の）**x86** 系のすべての最新 Linux ディストリビューションは、システム起動の初期フェーズの制御に **GRUB** (**GR**and **U**nified **B**ootloader) を使用します。他のプラットフォームでは **IA64** (**Itanium**) など **EFI** システムで使用される **ELILO** や、組み込み機器で多く使用される **U-Boot** などのローダーがあります。

GRUB の重要な機能:

- ブート時にオペレーティングシステムを選択します。
- 選択されたオペレーティングシステムに対して、カーネルや初期 RAM ディスク（initramfs）を選択します。
- 事前の構成ファイルなどの編集をしていなくても、ブート時に起動パラメータを指定できます。

GRUB のバージョン

- **GRUB 2**
  - 全ての最新 **Linux** ディストリビューションが採用している
  - 設定ファイル:
    `/boot/grub/grub.cfg`
    `or /boot/grub2/grub.cfg`
    `or /boot/efi/EFI/redhat/grub.cfg`
    `/boot/grub2/grubenv`
    `/etc/default/grub`
    `/etc/grub.d`
- **GRUB 1**
  - 主要ディストリビューションで利用しているのは **RHEL 6** だけ
  - 設定ファイル:
    `/boot/grub/grub.conf`
  - 考え方は共通だが、詳細部分はかなり違う
- バージョン 1 は既に過去のものなので、ここでは **GRUB 2** を説明する

GRUB 2 と古い Version 1 の間には詳細部分での差異はありますが、基本思想は共通です。

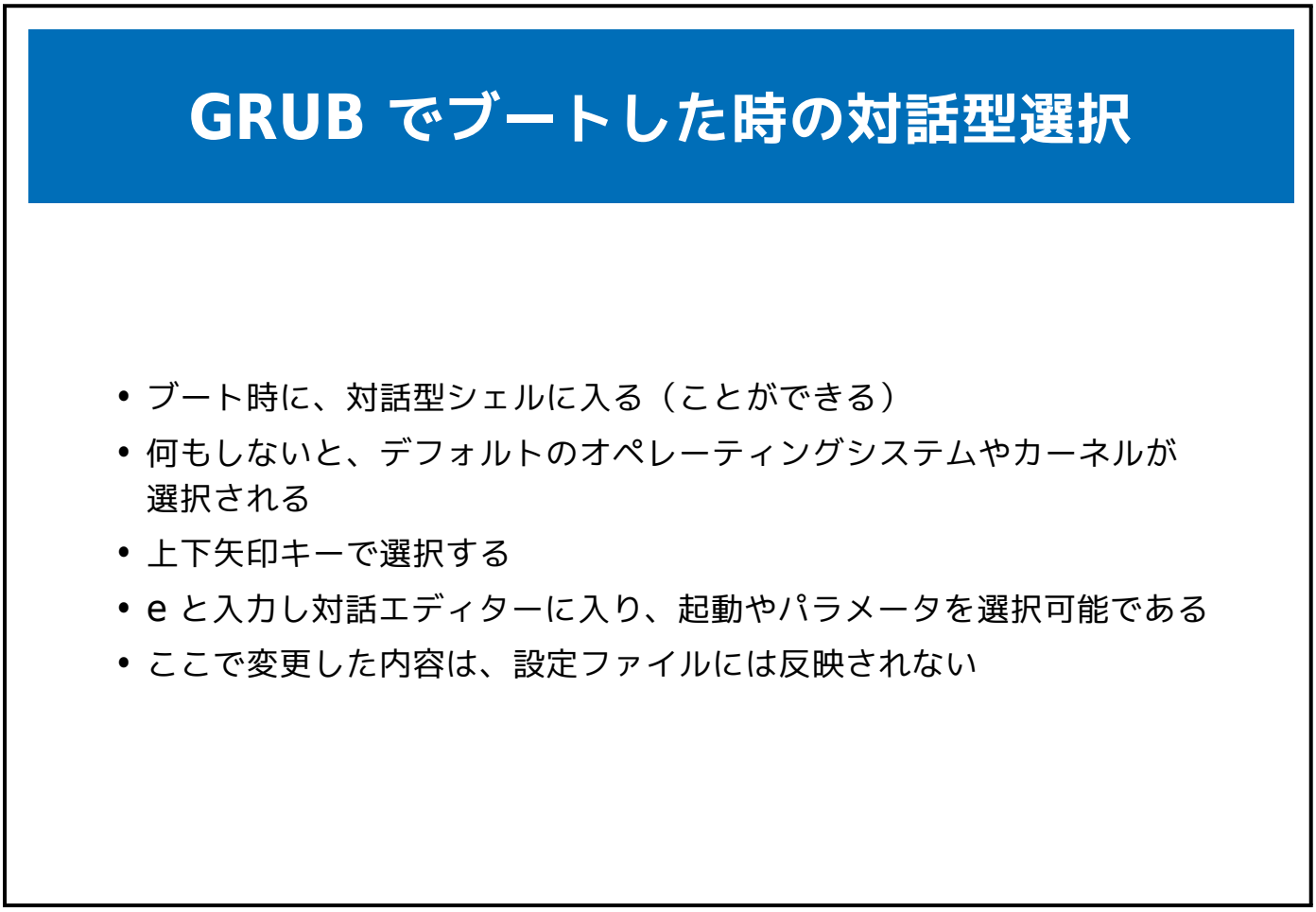
ブート時に基本構成ファイルを読む:

`/boot/grub/grub.cfg`
又は
`/boot/grub2/grub.cfg`
又は
`/boot/efi/EFI/redhat/grub.cfg`

このファイルは `/etc/grub.d` ディレクトリと `/etc/default/grub` にある構成ファイルに基づいて **update-grub** (または **grub2-mkconfig**) によって自動生成されるので、手動で編集すべきではありません。通常これらのユーティリティは、ディストリビューションが提供する **Linux** カーネル更新やコンパイルに使用するスクリプトから実行します。

繰り返しますが、このファイルを直接編集するのは止めた方が良いです。とはいえ編集しても大抵は変更は保存されるでしょう。(自動生成による上書きで消されないでしょう。)

38.2 GRUB でブートした時の対話型選択



システムがブートし最初の POST と BIOS のステージが終わると、**GRUB** が開始されメニューが表示されます。メニューにはブート可能な 1 つ以上の **Linux** ディストリビューション、またはオペレーティングシステムのイメージのリストが含まれています。ここにサブメニューによる選択オプションがあるかもしれません。

上矢印、下矢印、`Enter` キーを使用して正しいブートオプションを選択するか、デフォルトの選択肢が選択されるまで待つこともできます。（待機時間は設定可能です）

これだけではありません。エントリを選択した後 `e` と入力するとインタラクティブシェルに入り、対話型で（起動コマンドの）編集ができます。このシェルでは、特定のブートオプションが記述された構成ファイル **スタンザ** を変更します。通常これは **カーネルコマンドライン** の変更のために使います。たとえば、コマンドラインの最後に `single` という単語を追加すると、その修正が反映され **シングルユーザーモード** でシステムが起動するので、プロンプトに対して必要な修復などを行うことができます。目的の変更が完了したら正しいキーを押してシステムをブートさせます。

操作に使うショートカットキーが画面の一番下に表示されるので、キー割り付けを覚える必要はありません。

構成に加えた変更は **恒久的ではない** ため、次回起動時には失われます。変更を恒久的にするには、適切なユーティリティを使用しマシン上の設定ファイルを変更する必要があります。

特定のスタンザを編集するのではなく、純粋なシェルを使用することもできます。さまざまなコマンドを実行することが可能で **GRUB** の再インストール、または修復もできます。構成ファイルが見つからないなどの重大な問題がある場合には **GRUB** がコマンドラインモードに戻るので、レスキューメディアを使わなくてもシステムを復旧できる可能性があります。

38.3 GRUB のインストール

GRUB のインストール

- **GRUB** ブートローダーは、インストール時にインストールされる
- ディストリビューションにより **grub2-install** か **grub-install** を利用して、何時でも再インストールができる
- インストールには別の意味もある:
  - GRUB の実行に必要な設定ファイルを用意する
  - 必要なユーティリティをインストールする

```
$ sudo grub2-install /dev/sda
```

- EFI マルチブートシステムでは **efibootmgr** の実行も必要となる
  多少複雑度が増す。詳細は **man** ページ参照

GRUB では **インストール**という単語がいくつかの異なる意味で使われます:

1. **grub** プログラムと関連ユーティリティを適切な場所へインストールすること。**GRUB 1** には実際に **grub** と呼ばれるプログラムがありましたが **GRUB 2** には **grub2-*** や **grub-*** といった名前のユーティリティがたくさんあります。
2. **GRUB** でブートさせる為に必要なファイルを `/boot/grub` または `/boot/grub2` の下にインストールすること。これは **Linux** カーネルが必要とする `/boot` ディレクトリ下のファイル (`vmlinuz-*, initramfs-*`) とは別のものです。
3. **GRUB** を **ブートローダー** としてシステムにインストールすること。通常ハードディスク全体の先頭セクタ（MBR）にインストールしますがパーティション内にインストールすることもできます。

再インストールする時の正確な手順は **GRUB** のバージョンによって異なります。以下のように簡単です:

```
$ sudo grub2-install /dev/sda
```

これらのコマンドを実行する前に **man page** を注意深く読んでください。多くのオプションがあり、その設定を間違うと **GRUB** を使ってシステムを起動できなくなります。

GRUB デバイスの命名規則

- ブート時は、ファイルシステムがまだマウントされていない状況下でファイルにアクセスする必要がある
- 最初のハードドライブは hd0、2 番目は hd1 のように命名される
- しかし、パーティション名は 1 からスタートする:
  - sda1 は (hd0,1)
  - sdc4 は (hd2,4)

構成ファイル内の各スタンザには **ルートパーティション** を指定する必要があります。これはシステムのルートディレクトリとは違います。カーネル自体を（/boot ディレクトリに）含んでいるパーティションという意味です。別パーティションに /boot を配置するのは一般的で、ここではカーネルを /dev/sda1 に置くことにします。

この場合 kernel (hd0,0)/vmlinuz ... を実行しても問題ありません。代わりに root の行を省略します。

grub.cfg を見ると明快にだと思います。

38.4 GRUB 構成ファイルのカスタマイズ

GRUB 構成ファイル

- /boot/grub/grub.cfg (ないし /boot/grub2/grub.cfg) を直接変更してはいけない
- /etc/default/grub と /etc/grub.d にあるファイルを編集する
- grub.cfg は **update-grub** が実行される度に書き換えられる

```
c8:/tmp>cat /etc/default/grub
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="crashkernel=auto rhgb quiet"
GRUB_DISABLE_RECOVERY="true"
GRUB_ENABLE_BLSCFG=true
c8:/tmp>
```

図 38.1: **RHEL 8 の /etc/default/grub**

- **grub2-mkconfig** または **grub-mkconfig** によって新しいファイルが生成される

GRUB 2 では /etc ディレクトリの中に注意が必要な場所が２つあります。これらは、システムのカーネルを更新したり（**update-grub** や **grubby** などの）更新プログラムを手動実行した際に grub.cfg を再構築するために使用されます。

１つ目は /etc/default/grub です。これに加え /etc/grub.d も関係します。**Ubuntu 19.10** では:

```
student@ubuntu: ~
student@ubuntu:~$ ls -l /etc/grub.d
total 80
-rwxr-xr-x 1 root root  9783 Mar 30 16:45 00_header
-rwxr-xr-x 1 root root  6258 Nov  1  2016 05_debian_theme
-rwxr-xr-x 1 root root 12676 Mar 30 16:45 10_linux
-rwxr-xr-x 1 root root 11281 Mar 30 16:45 20_linux_xen
-rwxr-xr-x 1 root root  1992 Jan 28  2016 20_memtest86+
-rwxr-xr-x 1 root root 12059 Mar 30 16:45 30_os-prober
-rwxr-xr-x 1 root root  1418 Mar 30 16:45 30_uefi-firmware
-rwxr-xr-x 1 root root   214 Mar 30 16:45 40_custom
-rwxr-xr-x 1 root root   216 Mar 30 16:45 41_custom
-rw-r--r-- 1 root root   483 Mar 30 16:45 README
student@ubuntu:~$
```

図 38.2: **/etc/grub.d の内容**

２つのファイルは、構成ファイルが更新された時に昇順で実行されます。

38.5 Boot Loader Specification 構成ファイル（BLSCFG）

Boot Loader Specification 構成ファイル（BLSCFG）

- **blscfg** (Boot Loader Specification Configuration) **grub** メソッドは **Fedora 30** と **RHEL 8** で紹介された
- 通常ユーザーには透過的（従来と変わらない操作性）だが **生成された設定ファイル** の内容は全く異なる:
  - `/boot/loader/entries`
  - `/boot/grub2/grubenv` (`/boot/grub2/grub.cfg` にリンク)
- https://systemd.io/BOOT_LOADER_SPECIFICATION/ を参照
- 利用可能なら **grub2-switch-to-blscfg** で切り替えが可能
- 他の **Linux** ディストリビューションが採用するかは未知数である

BLSCFG で構成されたシステムでも、カーネルのインストールや更新などの際には通常の **grub** コマンドを使用します。
ブートしても同じ対話型の **grub** の画面が表示されるので、いつものように使用できます。

ただし、いくつか重要な違いがあります。たとえば `/boot/grub2/grub.cfg` は存在しますが、選択できる各カーネルの詳細情報とオプションはそこにはありません。それらは `/boot/loader/entries` にあり、選択肢ごとにファイルが存在します。

grub2-switch-to-blscfg プログラムを実行すると新しいスキームへの切り替えが可能です。
`GRUB_ENABLE_BLSCFGS=[true|false]` を変更することでも新スキームのオン／オフを変更できます。
たとえば **RHEL 8** の場合:

/etc/default/grub

```
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="crashkernel=auto rhgb quiet"
GRUB_DISABLE_RECOVERY="true"
GRUB_ENABLE_BLSCFG=true
```

/boot/loader/entries

```
$ sudo ls -l /boot/entries
```

```
-rw-r--r--. 1 root root 408 Jun 21  2019 ce0c82382a8a4c80bbd6931a917a2f1c-0-rescue.conf
-rw-r--r--  1 root root 381 Feb 17 07:09 ce0c82382a8a4c80bbd6931a917a2f1c-4.18.0-240.15.1.el8_3.x86_64.conf
-rw-r--r--  1 root root 285 Mar  9 06:41 ce0c82382a8a4c80bbd6931a917a2f1c-5.11.5.conf
-rw-r--r--  1 root root 285 Mar 11 13:02 ce0c82382a8a4c80bbd6931a917a2f1c-5.11.6.conf
-rw-r--r--  1 root root 305 Mar 15 07:23 ce0c82382a8a4c80bbd6931a917a2f1c-5.12.0-rc3.conf
```

/boot/loader/entries/*5.12.0-rc3.conf

```
title dracut (5.12.0-rc3) 8.2 (Ootpa) dracut-049-70.git20200228.el8
version 5.12.0-rc3
linux /boot/vmlinuz-5.12.0-rc3
initrd /boot/initramfs-5.12.0-rc3.img $tuned_initrd
options $kernelopts $tuned_params
id dracut-20210315122247-5.12.0-rc3
grub_users $grub_users
grub_arg --unrestricted
grub_class kernel
```

Boot Loader Specification 構成のスキームを使用すると、ブート可能な各カーネルは /boot/loader/entries にエントリを取得します。

/boot/grub2/grubenv にはいくつかの環境変数を設定できます、具体的には:

- kernelopts
- tuned_initrd
- tuned_params
- grub_users

/boot/grub2/grubenv

```
# GRUB Environment Block
kernelopts=root=UUID=6921b738-1e36-429a-89be-8b97cf2f0556 ro
boot_success=0
boot_indeterminate=0
saved_entry=ce0c82382a8a4c80bbd6931a917a2f1c-5.11.6.0~custom
################################################################################....
```

38.6 演習

デモ教材ビデオ

using_grub_demo.mp4

課題 38.1: GRUB を用いた非グラフィカル モードへのブート

注目

本演習ではコンソールを利用します（**SSH** ではありません）。

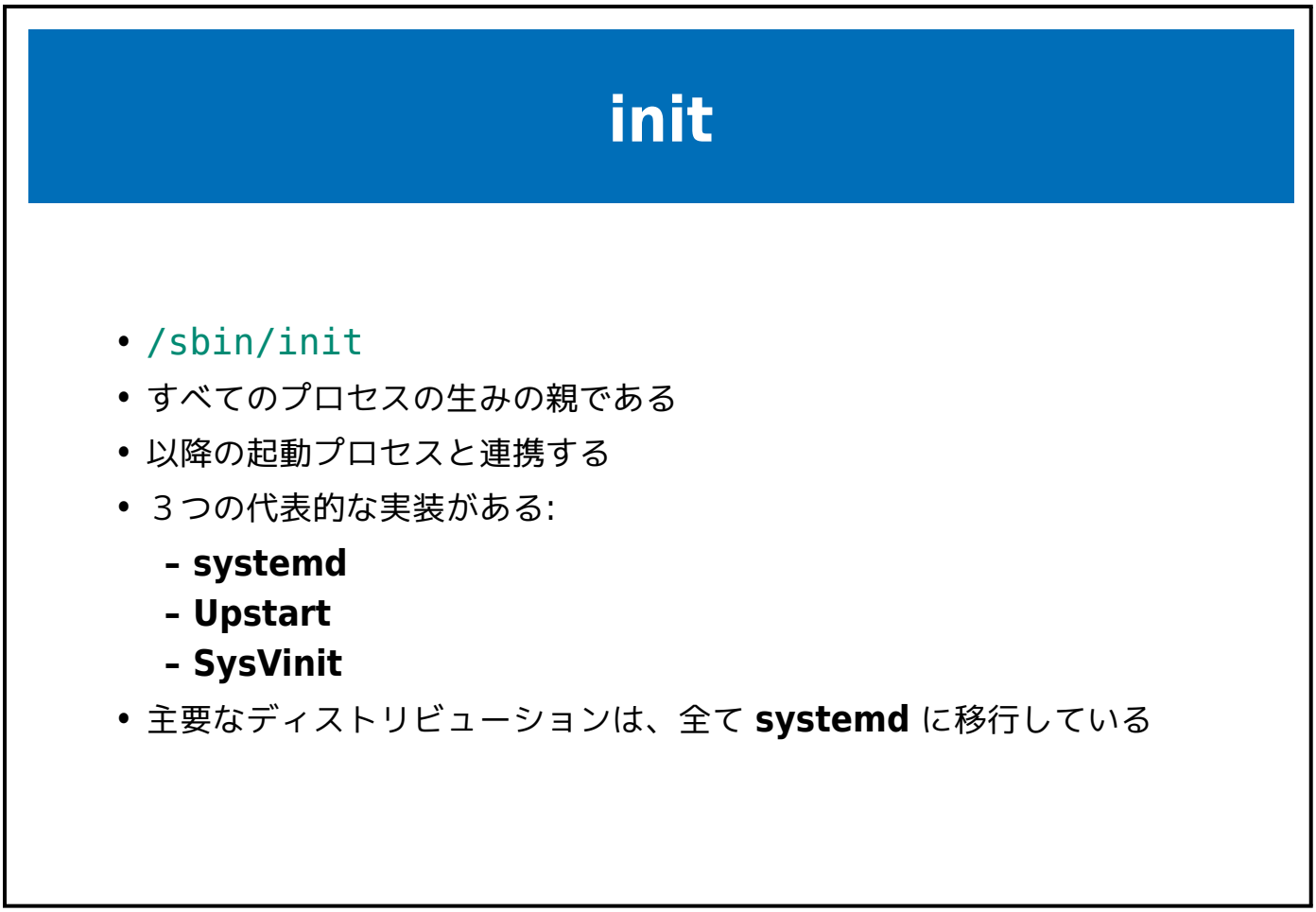
1. システムをリブートして **e**（または、スクリーンに表示されている別のキー）を入力し **GRUB** の対話シェルに入ります。
2. 非グラフィカルモードにブートします。やり方はシステムに依存します。**ランレベル**（これについては、次セッションで説明します）を使用する古典的なシステムでは **GRUB** メニュー内から選択したカーネルコマンド行に 3 を追加します。この方法は **SysVinit** のランレベルをエミュレートする **systemd** システムでも有効です。他のシステムでは、上記の代わりに `text` を追加します。
3. ブートを継続するため、適切なキーを押下します。
4. システムが非グラフィカルモードで実行可能になったら、グラフィカルモードを立ち上げます。システムによりますが、以下のコマンドのいずれかを実行します:

```
$ sudo systemctl start gdm
$ sudo systemctl start lightdm
$ sudo telinit 5
$ sudo service gdm restart
$ sudo service lightdm restart
```

章 39

System Init について：systemd、SystemV と Upstart

39.1 init プロセス



`/sbin/init`（通常は単に **init** と呼ばれます）は、システムで実行される最初のユーザーレベルのプロセス（またはタスク）で、システムがシャットダウンされるまで実行を続けます。歴史的には、すべてのユーザープロセスの **親** と見なされてきましたが、一部のプロセスはカーネルから直接開始されるため、技術的に厳密に言えばこれは正しくありません。

init 以降の起動処理と連携し、諸々の環境を設定しシステムへのログインに必要なプロセスを開始します。**init** は、プロセス終了後にクリーンアップを行う際にはカーネルとも密接に連携します。

歴史的には、ほぼすべてのディストリビューションが **UNIX** の由緒ある **SysVinit** の **init** プロセスをベースにしていました。しかしながら **SysVinit** は、数十年前に、今とはかなり異なる状況下で開発されたものでした:

- マルチユーザー メインフレームシステムがターゲットでした。（PC、ラップトップなどのデバイスではありませんでした）
- シングルプロセッサ システムがターゲットでした。
- そのため起動（およびシャットダウン）時間は重要ではありませんでした。物事を正しく動作させる方がはるかに重要でした。

起動は、一連の（**ランレベル** と呼ばれる）順次的なステージに分割された **連続プロセス** と見なされます。次のステージに進むためには、前のステージを完結させる必要があります。このため、起動処理に複数プロセッサまたはコアによる並列処理を活用するのは簡単ではありませんでした。

さらに、シャットダウン／再起動 は比較的まれなイベントと見なされており、正確にどのくらい時間がかかるかは重要ではありませんでした。現代の **Linux** システムは数秒で起動します。

最新のシステムには、機能を強化した新しいメソッドが求められました。

39.2 SysVinit 以外の起動方法

SysVinit 以外の起動方法

- **Upstart**
  - **Ubuntu** によって開発され 2006 年から導入された
  - **Fedora 9** (2008) に採用され、**RHEL 6** とそのクローンが導入した
  - 多くの組み込み機器やモバイルデバイスにも使われている
- **systemd**
  - メジャディストリビューションでは **Fedora** が 2011 年に初採用
  - **RHEL** と **SUSE** が続いた
  - **Ubuntu 16.04** で **Upstart** から **systemd** に切り替えた
  - 現在では､全ての重要な **Linux** ディストリビューションは **systemd** をベースにしている

SysVinit 固有の制約に対処するため、新しいシステム起動制御メカニズムが開発されました。他にもありますがエンタープライズディストリビュータは **Upstart** と **systemd** という２つの主要なスキームを採用しました。

当初 **Upstart** を選択した **Ubuntu** を例外とすれば、それ以外の主要な **Linux** ディストリビューションは全て **systemd** をデフォルトの起動方法に採用しました。

systemd への移行はかなり複雑で、バグや欠落している機能が無効にされる可能性もあるため、レガシーソフトウエアとの基本互換レイヤーが導入され、これは今でも残っています。このため **SysVinit** ユーティリティの互換ラッパーは現在も存続しています。

systemd の開発と採用の歴史は一筋縄ではありませんでした。開発者の変化にとんだ個性によって、必ずしもすべての議論が技術的かつ友好的に進められたわけではありませんでした。しかしこの神聖な戦いも収束し、もう心配はなくなっています。

今後は **systemd** に絞って説明します。もはやほとんど使用されなくなった **SysVinit** と **Upstart** は説明しません。

39.3 systemd

systemd の機能

- 以前の init ベースのシステムより、起動が高速である
- 積極的に並列実行を活用する
- サービスの起動に **socket** と **D-Bus** アクティベーションを利用する
- 従来のシェルスクリプトはプログラムに置き換える
- デーモンのオンデマンド起動機能を提供する
- **cgroups** を使用してプロセスを追跡する
- マウントと自動マウントのポイントを保持する
- 精巧なトランザクション依存関係に基づくサービス制御ロジックを実装している
- **SysVinit** を一時的に代替し **SysVinit** スクリプトとの互換性を持つ

systemd と **Linux** のセッションマネージャは、今やすべての主要なディストリビューションが採用しています。

systemd は **SysVinit** との後方互換性があり、ランレベルの概念は **ランレベルターゲット** 経由でサポートされることに注意してください。**telinit** プログラムは、ランレベルで動作するようにエミュレートされます。

bash スクリプトの代わりに **systemd** は `.service` ファイルを使用します。さらに **systemd** はすべてのデーモンを独自の **Linux cgroups**（制御グループ）に分類します。

systemd の構成ファイル

- `/etc/hostname` は以下を置き換える:
  - **Red Hat** 系システムの `/etc/sysconfig/network`
  - **SUSE** 系システムの `/etc/HOSTNAME`
  - **Debian** 系システムの `/etc/hostname` (が、元々標準だった)
- これ以外のファイルには:
  - `/etc/vconsole.conf`: デフォルトキーボードマップとコンソールフォント
  - `/etc/sysctl.d/*.conf`: カーネル **sysctl** パラメータの集約
  - `/etc/os-release`: ディストリビューション ID

systemd は標準化された構成ファイルのセットを使用することを推奨していますが、代替としてディストリビューション依存のレガシー構成ファイルを使用することもできます。

各構成ファイルは、各ディストリビューションがそれぞれどのように設定するかによって異なります。たとえば、仮想端末のデフォルトを構成する `/etc/vconsole.conf` は **Ubuntu** システムには存在しません。

SysVinit との互換性

systemd は **SysVinit** との後方互換性があるため、古いコマンドもほとんどの場合機能します。ランレベルターゲットのメカニズムを介して概念的にはランレベルの使用をサポートしています。さらに **telinit** はランレベルで動作するようにエミュレートされます。

39.4 systemctl

systemctl

```
$ systemctl [options] command [name]
```

- **systemd** がコントロールする全てのステータスを表示:
  ```
  $ systemctl
  ```
- 利用可能な全てのサービスを表示:
  ```
  $  systemctl list-units -t service --all
  ```
- 現在アクティブなサービスだけを表示:
  ```
  $ systemctl list-units -t service
  ```
- 一つ以上の **units** の起動（活性化）:
  ```
  $ sudo systemctl start foo
  $ sudo systemctl start foo.service
  $ sudo systemctl start /path/to/foo.service
  ```
  unit はサービスでもソケットでも良い

systemctl は、サービスを管理するための主要なユーティリティです。一部の **systemctl** コマンドは root ユーザー以外でも実行できますが、それ以外のコマンドは root または **sudo** で実行する必要があります。

ほとんどのコマンドでは、サービス名に付加された `.service` を省略できます。

SysVinit から **systemd** への移行方法について非常に良くまとまったサマリーが **SysVinit to systemd Cheatsheet** として https://fedoraproject.org/wiki/SysVinit_to_Systemd_Cheatsheet に公開されているので参考にしてください。

systemctl (続き)

- 停止 (非活性化):

```
$ sudo systemctl stop foo.service
```

- サービスの有効化／無効化:

```
$ sudo systemctl enable sshd.service
$ sudo systemctl disable sshd.service
```

これらのコマンドは、今ここでサービスを起動／停止するのではなく
次回以降のシステム起動時にサービスを起動するかどうかを設定する

ほとんどのコマンドでは、サービス名に付加された `.service` を省略できます。

39.5 演習

課題 39.1: systemd を使って新たなスタートアップサービスを追加する

スタートアップファイルを追加するには root ユーザーで /etc/systemd/system か、その下のどこかに、ファイル・ディレクトリを作成する必要があります。この際、ディストリビューションごとに異なるテイストがあります。たとえば、最小限のファイル /etc/systemd/system/fake2.service の内容（ダウンロードした SOLUTIONS から fake_service として取り出すことができます）は、以下のようになっています:

fake2.service

```
[Unit]
Description=fake2
After=network.target

[Service]
ExecStart=/bin/sh -c '/bin/echo I am starting the fake2 service ; /bin/sleep 30'
ExecStop=/bin/echo I am stopping the fake2 service

[Install]
WantedBy=multi-user.target
```

unit ファイルにはさまざまな内容が含まれています。After=network.target は、ネットワーク起動後に本サービスを実行すべきこと、WantedBy=multi-user.target は、マルチユーザーモードになったら実行すべきことを意味しています。これは **SysVinit** の runlevel 2 と 3 と同じ意味です。graphical.target は runlevel 5 と関係します。

サービスを開始、終了して、ステータスチェックをするために、以下のコマンドを実行します:

```
$ sudo systemctl start fake2.service
$ sudo systemctl status fake2.service
$ sudo systemctl stop fake2.service
```

これを実行中（＝サービスの停止、起動を行っている間に）**unit** ファイルをいじった場合、システムが警告を出すので、以下のコマンドでリロードする必要があります:

```
$ sudo systemctl daemon-reload
```

以下のコマンドで出力を確認してください:

```
$ sudo tail -f /var/log/messages
```

（**Ubuntu** の場合には /var/log/syslog を見てください）（ログ出力の確認は）バックグラウンド行うか、サービスを実行しているウィンドウとは別のウィンドウで見てください。

システムブート時に、サービスを起動するかどうかを設定するには:

```
$ sudo systemctl enable fake2.service
$ sudo systemctl disable fake2.service
```

実際に設定出来たかを確認するために、システムをリブートして確かめてください。

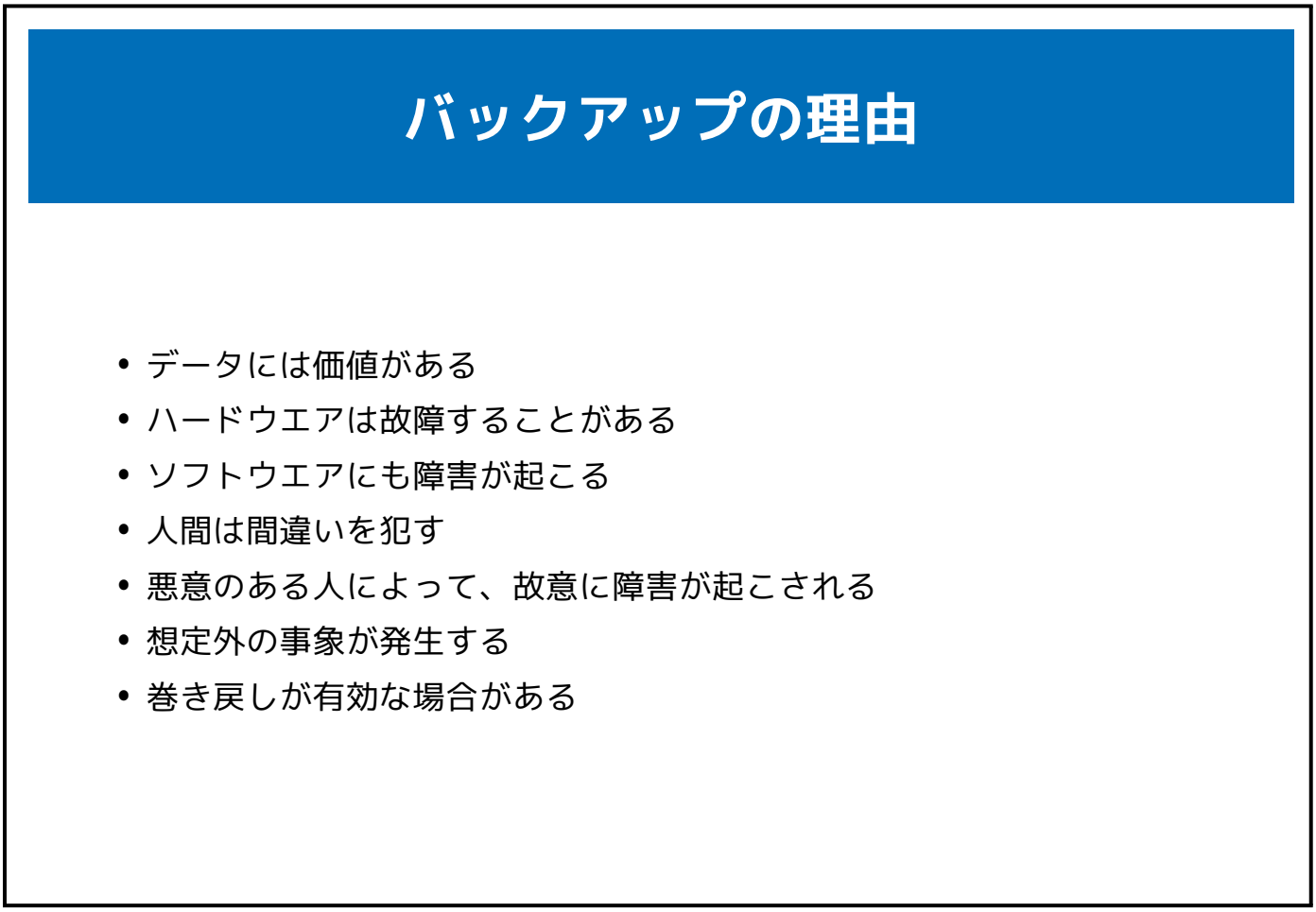
章 40

バックアップとリカバリの方法

注目

** これらのセクションの一部または全体をオプション扱いとする場合があります。これらには補足資料、専門トピック、または高度な話題が含まれインストラクターが教室の状況や時間制約に応じてこれらの内容を紹介するかを判断します。

40.1 バックアップの基本



１つのパーソナルシステムのみを管理している場合でも、多数のマシンのネットワークを管理している場合でも、システムバックアップは非常に重要です。

ディスク上のデータは、重要な作業成果物であるため保護すべきです。失われたデータを再度作成するのは、時間と費用がかかります。一部のデータは他にない唯一のものであり、再度作成する方法がない場合があります。

ストレージメディアの信頼性は向上していますが、ドライブ容量も増大しています。１バイトあたりの故障率は低下していますが、それでも予測できない故障は発生します。ドライブには「故障したドライブ」と「故障する可能性のあるドライブ」の２種類しかない、という言い方は悲観的かもしれませんが真実です。**RAID** はデータ保護には役立ちますが、それでもバックアップは必要です。

完璧なソフトウェアはありません。バグによってデータが破壊または破損される可能性があります。長期間使用している安定したプログラムでも問題が発生する場合があります。

誰もが、隣のパーティションから（または自分の口から）OOPS！(しまった！)、またはもっとひどい言葉をを聞いたことがあるはずです。時には単純な入力ミスだけで、ファイルとデータが大規模に破壊されることがあります。

それは普通に不満を抱く従業員か、もしくは損害を起こすポイントを知っている外部のハッカーかもしれません。セキュリティの問題とバックアップ機能は非常に強く関連しています。

ファイルは、どのように、誰が、いつ起こったかを知らなくてもただ消えてしまう可能性があります。

場合によっては、システムのすべてまたは一部を以前の **スナップショット** に復元することが要求されます。

バックアップが必要なものは何か？

- 絶対に必要なもの:
  - 業務に関連したデータ
  - システム設定ファイル
  - ユーザーのファイル (通常は /home 以下のファイル).
- 多分必要なもの:
  - スプーラーディレクトリ (印刷、メールなど)
  - ログファイル (/var/log やそれ以外の場所にある).
- おそらく必要がないもの:
  - 簡単に再インストールできるソフトウエア、適切に管理されたシステムでは大部分がこの範疇に該当するだろう
  - /tmp ディレクトリ
- 絶対に必要ないもの:
  - /proc、/dev、/sys や swap など疑似ファイルシステム

組織に不可欠なファイルのバックアップが必要なのは明らかです。頻繁に変更される構成ファイルも、各個人ユーザーのファイルと共にバックアップが必要です。

システムの履歴を調査する必要がある場合ログファイル（の保全）は重要です。侵入やその他のセキュリティ違反を検出するためには、ログファイルは特に重要です。

簡単に再インストールできるものをバックアップする必要はありません。また、スワップパーティション（またはファイル）と /proc ファイルシステムは（/tmp ディレクトリと同様に）、データが基本的に一時的なものなのでバックアップの必要がないか、バックアップしても無意味です。

40.2 バックアップ vs. アーカイブ

バックアップ vs. アーカイブ

- 全てのバックアップメディアには読み取り不能となる寿命の限界がある
- 標準的な想定寿命は:
  - 磁気テープ: 10-30 年
  - CD や DVD: 3-10 年
  - ハードディスク: 2-5 年
- 実際の寿命は以下に大きな影響を受ける:
  - 環境条件（温度、湿度など）
  - 記録メディアの品質
  - 現在のオペレーティングシステムとハードウエアで、データを読み出せるソフトが存在するか
- バックアップには十分だが、恒久的なデジタルアーカイブには使えない

通常のバックアップ周期よりも長く保存させたい場合には、複数のコピーを作成したり時々新しいメディアにコピーすることで対応できます。

超長期間（つまり数十年、数世紀など）の保存は、以下のような標準的な保存手段が全て時代遅れになってしまうので簡単ではありません。

- ハードウエア
- ソフトウエアとドキュメントのフォーマット（書式）
- メディア

安価なデジタル記録メディアの寿命は、どれも（ワインのように、適切に保管され継続的に手入れされていたとしても）紙やフィルムほどは長くはありません。

これは人々が深刻に懸念している問題で、すべてが失われる前に良い解決策を考えるべきです。

テープドライブ

- 比較的低速である
- シーケンシャルアクセスのみである
- 以前のようには一般的ではなくなった
- 今日では、メインのバックアップには使われない
- それでも「オフサイトストレージ」や「アーカイブ」には有効である

テープドライブは以前ほど一般的ではなくなりました。比較的低速で順次（シーケンシャル）アクセスしかできません。最近は、プライマリバックアップにテープを使用することはほとんどありません。長期間参照できるようなアーカイブ目的のオフサイトストレージとしてテープが使用されることがあります。ただし、磁気テープドライブには物理的劣化やデータ損失がなくても、その寿命には限界があります。

最近のテープドライブはほとんどが **LTO** (**L**inear **T**ape **O**pen) で、1990 年代後半にオープンスタンダードの代替として最初のバージョンが登場しました。初期は、ほとんどが独自フォーマットのものでした。初期バージョンの記録容量は最大 100GB でした。新しいバージョンでは、同じサイズのカートリッジで 2.5TB 以上を記録できます。

日々のバックアップには、通常は何らかの **NAS** (**N**etwork **A**ttached **S**torage)、または **クラウドベースのソリューション** が使われるため、新しいテープベースのインストールはますます魅力的ではなくなっています。それでもまだテープを使う場面もあるので、システム管理者は対処する必要があるかもしれません。

以降は、バックアップメディアの物理的な形式ではなく、抽象度の高い（記録メディアに依存しない）テーマを説明をします。

40.3 バックアップ方法と戦略

バックアップ方法

- **完全バックアップ (Full)**:
  システム上の全ファイルをバックアップする
- **増分バックアップ (Incremental)**:
  最後に行った増分、または完全バックアップ以降に変更された全ファイルをバックアップする
- **差分バックアップ (Differential)**:
  最後の完全バックアップ以降に変更された全ファイルをバックアップ
- **複数のレベルの増分バックアップ (Multiple level incremental)**:
  同じレベル、または以前のレベルにおいて、最後に取られたバックアップ以降に変更された全ファイルをバックアップする
- **ユーザー (User)**:
  特定のユーザーのディレクトリ内のファイルのみをバックアップする

多くの場合、複数の異なるバックアップ方法を互いに連携して使用できます。

すべてのバックアップは、保護対象システムと同じ物理的な場所に保管しないでください。そうでないと、火災またはその他の物理的損傷により全てが失われる可能性があるからです。過去には「バックアップとは磁気テープを安全な場所に物理的に輸送すること」を意味していました。現在は「バックアップファイルをインターネット経由で別の物理的な場所に転送すること」を意味しています。適切な暗号化とその他のセキュリティ予防措置を使用して、安全な方法でデータを転送する必要があるのは明らかです。

バックアップ戦略

テープを使った有用な戦略（「テープ」を他のメディアに置き換えられる）

1. 金曜日の完全バックアップにはテープ１を使用
2. 月曜日から木曜日の増分バックアップにはテープ２～５を使用
3. 第 2 金曜日の完全バックアップにはテープ６を使用
4. 2 番目の月曜日から木曜日の増分バックアップにはテープ２～５を使用
5. テープ６の完全バックアップが完了するまでテープ１を上書きしない
6. テープ６に完全バックアップした後、災害復旧時のためにテープ１を外部の場所に移動
7. 次の完全バックアップ（次の金曜日）にはテープ６の代わりにテープ１を使用

バックアップ方法は、対応する **レストア (復元)** 方法がないと役に立たないことに注意してください。バックアップの戦略を選択する際には、両方向（つまり、バックアップとレストア）の堅牢性、明瞭性、および使いやすさを考慮する必要があります。

目安として、少なくとも２週間のバックアップを用意するようにしてください。

最も単純なバックアップ戦略は、すべてを一度完全バックアップしてから、その後変更されるすべての増分バックアップを実行することです。完全バックアップには多くの時間がかかる場合がありますが、増分バックアップからの復元はもっと困難で、時間がかかる場合があります。したがって、両方を組み合わせれば時間と労力を最適化できます。

バックアップ関連ユーティリティ

- **cpio**
- **tar**
- **gzip, bzip2, xz**
- **dd**
- **rsync**
- **dump/restore**
- **mt**

cpio と **tar** はファイルのアーカイブの作成と抽出を行います。

ほとんどの場合、アーカイブは **gzip**、**bzip2**、**xz** で圧縮されています。アーカイブファイルはディスク、磁気テープ、またはファイルを保持できる他のデバイスに書き込むことができます。アーカイブは、ひとつのファイルシステムやマシンから別のファイルシステムやマシンにファイルを転送するのに非常に便利です。

dd は強力なユーティリティで、メディア間で生データを転送する時によく使用します。パーティション全体、またはディスク全体をコピーできます。

rsync も強力なユーティリティでネットワーク全体で、またはローカルマシン上の異なるファイルシステムの間でディレクトリサブツリーまたはファイルシステム全体を同期できます。

dump と **restore** は古くからあるユーティリティで、バックアップ専用に設計されました。ファイルシステムから直接読み込みます。（それがより効率的な方法です。）ただし、元のファイルシステムと同じ種類のファイルシステムに復元しなければなりません。今はもっと新しい選択肢があります。

mt はバックアップと復元を実行する前に、テープの状態を問合せをして位置調整するのに役立ちます。

40.4 tar

tar を使用したバックアップ

- アーカイブの作成: -c または単に c を利用する

```
$ tar cvf  /dev/st0 /root
$ tar -cvf /dev/st0 /root
```

- マルチボリュームオプションで作成: -M を利用する

```
$ tar -cMf /dev/st0 /root
```

テープを挿入するよう促される

- 比較オプションでファイルを検証: -d 又は --compare を利用する

```
$ tar --compare --verbose --file /dev/st0
$ tar -dvf /dev/st0
```

- 各オプションには短縮形 (- と一文字) と長い形式 (-- で始まる) がある
- **tar** のデフォルトは再帰的である

tar アーカイブを作成する場合、引数としてディレクトリを指定するとそのディレクトリのすべてのファイルとサブディレクトリがアーカイブに含まれます。復元するとき、必要に応じてディレクトリを再構成します。

さらに、増分バックアップを実行できる --newer オプションもあります

Linux で使用される **tar** のバージョンは、１本のテープまたは使用するデバイスに納まらない量のバックアップも処理できます。

-f または --file オプションを付加してデバイスやファイルを指定できます。

バックアップ作成後、検証オプションを使用して完全かつ正確であることを確認できます。

デフォルトでは **tar** は、全てのサブディレクトリを再帰的にアーカイブ内に格納します。

アーカイブを作成すると **tar** は絶対パス名から先頭のスラッシュを削除する旨のメッセージを出力します。これによりファイルをどこにでも復元できますがデフォルトのこの挙動を変更することもできます。

ほとんどの **tar** オプションは、ダッシュを１つ付けた短い形式、または２つ付けた長い形式で指定できます。-c は --create と完全に同じです。

また（短い表記を使用する場合）オプションを組み合わせることができますが、その時にはすべてのダッシュを入力する必要がないことに注意してください。

さらにシングルダッシュの **tar** オプションは、ダッシュ付きでもダッシュなしでも使用できます。
例えば tar cvf file.tar dir1 と tar -cvf file.tar dir1 は同じ結果になります。

tar を使用したファイルの復元

- アーカイブからの復元: -x 、または --extract を利用する

```
$ tar --extract --same-permissions --verbose --file /dev/st0
$ tar -xpvf /dev/st0
$ tar  xpvf /dev/st0
```

- 復元するファイル名の指定ができる

```
$ tar -xvf /dev/st0 somefile
```

- tar バックアップのコンテンツをリストする

```
$ tar --list --file /dev/st0
$ tar -tf /dev/st0
```

-x または --extract オプションは、デフォルトでアーカイブからファイルを抽出します。特定のファイルだけを指定すると、抽出するファイルを絞り込むことができます。ディレクトリを指定すると、そこに含まれるすべてのファイルとサブディレクトリも抽出されます。

-p または --same-permissions オプションは、ファイルを元の権限で復元します。

-t または --list オプションは、アーカイブ内のファイルをリストしますが抽出はしません。

tar を使用した増分バックアップ

- `--newer` または `--after-date` オプションを利用する
- 日付（だけ）をファイル抽出の基準にする
- 指定した日付以降に変更されたファイルをバックアップし圧縮する:

```
$ tar --create --newer '2011-12-1' -vzf backup1.tgz /var/tmp
$ tar --create --after-date '2011-12-1' -vzf backup1.tgz /var/tmp
```

どちらも、2011 年 12 月 1 日以降に変更された `/var/tmp` 内の
すべてのファイルのバックアップアーカイブを作成する

`-N` (もしくは同じ意味の `--newer`) または `--after-date` オプションを使用すれば **tar** で増分バックアップを実行できます。どのオプションでも、日付または対象（基準）のファイル名のいずれかを指定する必要があります。

tar はファイルの日付のみを調べパーミッションやファイル名などファイルに対する他の変更は考慮しません。これらが変更されたファイルを増分バックアップに含めるには **find** コマンドで対象となるファイルを検索してバックアップするファイルのリストを作成します。

Note: `--newer` などのオプションが後に続く場合 `-vzf` などのオプションにはダッシュを使用する必要があります。そうしないと **tar** が混乱します。この種のオプション指定の混乱は **ps** や **tar** のような古い **UNIX** ユーティリティで発生することがあります。そこには、さまざまな **UNIX** の系統が関係する複雑な歴史があります。

40.5 圧縮: gzip、bzip2、xz とバックアップ

アーカイブの圧縮方法

- ファイル **圧縮** はディスクスペース／ネットワーク転送時間を節約する
- 圧縮率を高めて効率を稼ごうとすると、圧縮時間は余計にかかる
  - **gzip**: Lempel-Ziv コーディング（LZ77）を使用し `.gz` ファイルを生成する
  - **bzip2**: Burrows-Wheeler ブロックソートテキスト圧縮アルゴリズムと Huffman コーディングを使用し `.bz2` ファイルを生成する
  - **xz**: `.xz` ファイルを生成、従来の `.lzma` 形式もサポートする
- 現代のマシンではしばしば **圧縮** → **転送** → **解凍** した方が単純に非圧縮のファイルを転送（またはコピー）するより時間が短くなる
- https://www.kernel.org からの **Linux** カーネルのダウンロードには今は **xz** 形式だけが使われる

圧縮ユーティリティは非常に簡単に（そして頻繁に）**tar** と組み合わせて使用されます:

```
$ tar zcvf source.tar.gz  source
$ tar jcvf source.tar.bz2 source
$ tar Jcvf source.tar.xz  source
```

これらは圧縮アーカイブを作成します。最初のコマンドは、以下のコマンドとまったく同じ結果になります:

```
$ tar cvf source.tar source ; gzip -v source.tar
```

このコマンドは、中間ファイルストレージを使わないのでより効率的です。アーカイブと圧縮は並列にパイプライン処理されます。
解凍する場合:

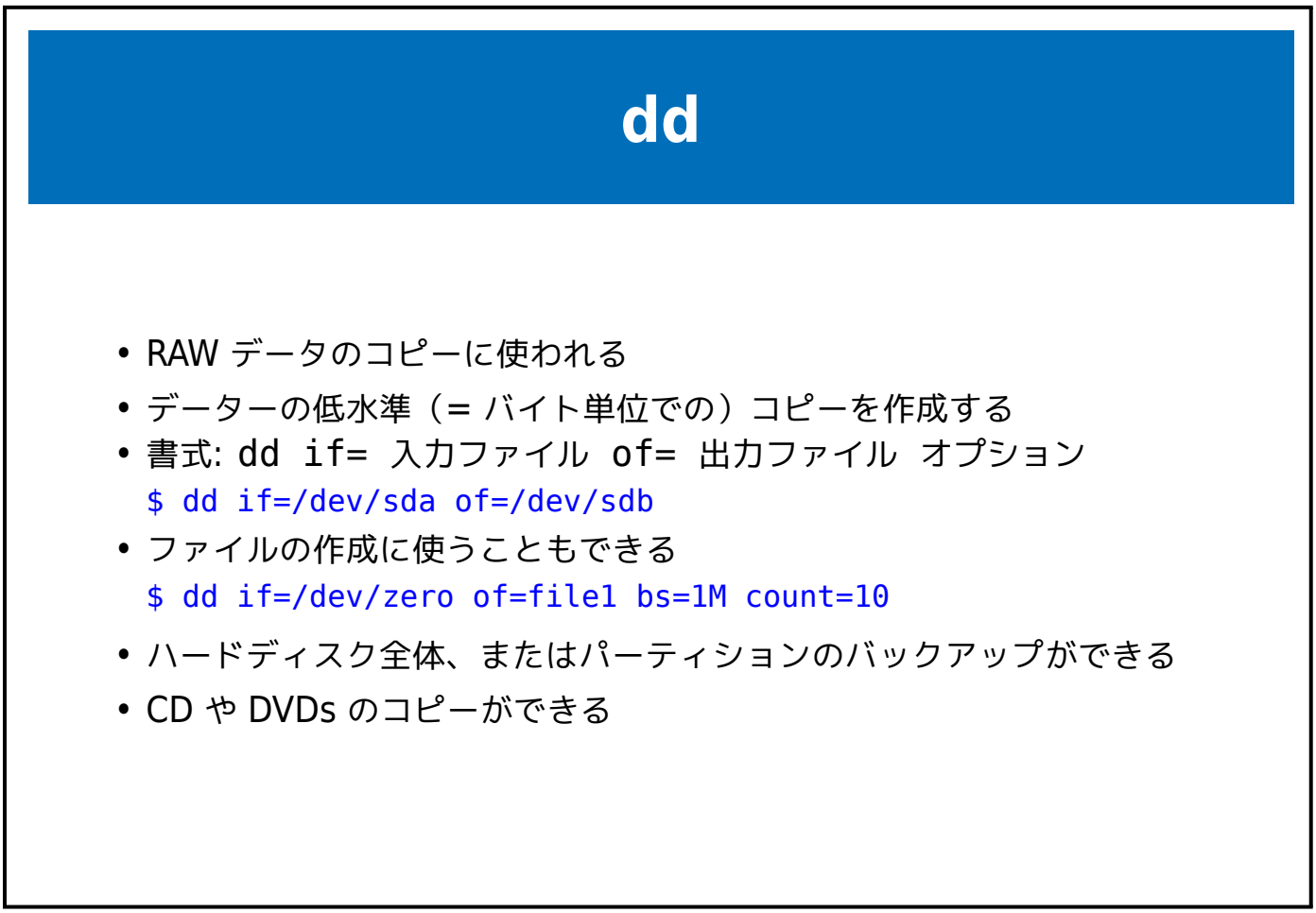
```
$ tar xzvf source.tar.gz
$ tar xjvf source.tar.bz2
$ tar xJvf source.tar.xz
```

または単純に:

```
$ tar xvf source.tar.gz
```

tar の最新バージョンは、圧縮の方法を検出して自動的に処理します。
`.jpg` 画像や `.pdf` ファイルなど、既に圧縮されているアーカイブを更に圧縮する価値は無いのは明らかです。

40.6 dd



dd は一般的な **UNIX** ベースのプログラムで、主な目的は生データの変換や低レベルのコピーや RAW データの変換です。**dd** は、指定した数のバイトやブロックをコピー、バイトオーダー変換（＝エンディアン変換）の逐次実行、さらにデータの変換にも使われます。さらに raw デバイスファイルの領域コピーにも使われます。例えば、ハードディスクのブートセクターのバックアップ、もしくは `/dev/zero` や `/dev/random` のようなスペシャルファイルから決まった量のデータを読み出すのにも使われます。

ハードドライブ全体を、別のディスクにバックアップします。

```
$ dd if=/dev/sda of=/dev/sdb
```

ハードディスクイメージの作成します。

```
$ dd if=/dev/sda of=sdadisk.img
```

パーティションをバックアップします。

```
$ dd if=/dev/sda1 of=partition1.img
```

CD-ROM をバックアップします。

```
$ dd if=/dev/cdrom of=tgsservice.iso bs=2048
```

40.7 rsync

rsync を使ったバックアップ

- 使い方:
  ```
  $ rsync [options] sourcefile destinationfile
  ```
- ローカルマシンとネットワークターゲット間:
  ```
  $ rsync file.tar someone@backup.mydomain:/usr/local
  ```
- 本番前に `--dry-run` でテスト実行する:
  ```
  $ rsync -r --dry-run /usr/local /BACKUP/usr
  ```

rsync (**r**emote **sync**hronize)（リモート同期）は、次のようにネットワークを介して（または同じマシンの異なる場所の間で）ファイルを転送するために使用されます。

同期元（ソース）と 同期先（デスティネーション）は、`target:path` の形式を取ることができます。ここで `target` は `[user@]host` の形式にすることができます。`user@` の部分はオプションであり、リモートユーザーがローカルユーザーと異なる場合に使用されます。したがって、以下の **rsync** コマンドはすべて実行可能です:

```
$ rsync file.tar someone@backup.mydomain:/usr/local
$ rsync -r --dry-run /usr/local /BACKUP/usr
```

rsync を使う時には、場所の正確な指定（特に `--delete` オプションを使用する場合）について、特に注意する必要があります。そのため、最初は `--dry-run` オプションを使用し、期待通りの動作になっていることを確認してから本番を実行することを強くお勧めします。

rsync は非常に賢いです。ローカルファイルとリモートファイルを小さなチャンクで照合します。これにより、１つのディレクトリを類似のディレクトリにコピーする場合に差分のみをネットワーク経由でコピーするので非常に効率的です。多くの場合 **rsync** に `-r` オプションを併用し、ディレクトリツリーを再帰的にたどり ソースファイル としてリストされているディレクトリから下の、すべてのファイルとディレクトリをコピーします。この方法は、プロジェクトディレクトリのバックアップをするときに非常に便利です:

```
$ rsync -r project-X archive-machine:archives/project-X
```

明快な（そして非常に効果的で高速な）バックアップ戦略は、**rsync** コマンドを使用してネットワーク全体でディレクトリ、またはパーティションを単純に複製しそして頻繁に更新することです。

40.8 cpio **

cpio を使ったバックアップ

- -o または --create を使ってアーカイブを作る:
  ```
  $ ls | cpio --create -O /dev/st0
  ```
- -i または --extract を使ってアーカイブを解凍する:
  ```
  $ cpio -i somefile -I /dev/st0
  ```
- -t または --list を使ってアーカイブのコンテンツをリストする:
  ```
  $ cpio -t -I /dev/st0
  ```

cpio (**c**o**p**y **i**n and **o**ut) は **UNIX** の初期の頃から使用されているファイルアーカイバユーティリティで、当初はテープバックアップ用に設計されたものです。新しいアーカイブプログラム（**tar** など、正確にはあまり新しくありませんが）が **cpio** に関連する多くのタスクを実行するために開発されてはいますが **cpio** はまだ生き残っています。

rpm2cpio ユーティリティを使って **RPM** パッケージを **cpio** アーカイブに変換して解凍することができます。また **Linux** カーネルは **cpio** のあるバージョンを内部的に使用していて、ブート中に **initramfs** と **initrd** の初期 RAM ディスクとファイルシステムを処理します。**cpio** が生き残っている理由の１つは、たとえそれが多少堅牢ではないとしても tar や他の後継機能よりも軽いことです。

コマンドラインで入力（-I デバイス）または出力（-O デバイス）、またはリダイレクションを指定することが可能です。

-o または --create オプションは **cpio** にファイルをアーカイブにコピーするよう指示します。**cpio** は標準入力からファイル名のリスト（１行に１つ）を読み取りアーカイブを標準出力に書き出します。

-i または --extract オプションは **cpio** に標準入力からアーカイブを読み取り、そのアーカイブからファイルをコピーするよう指示します。コマンドラインでファイル名を（*.c のように）パターンとしてリストすると、パターンに一致するアーカイブ内のファイルのみがアーカイブからコピーされます。パターンが指定されていない場合すべてのファイルが抽出されます。

-t または --list オプションは **cpio** にアーカイブの内容をリストするように指示します。-v or --verbose オプションを追加すると、長いリストが生成されます。

40.9 バックアッププログラム **

独自のアプリケーションやストレージベンダーが提供するアプリケーション、オープンソースアプリケーションなど **Linux** で利用可能なバックアッププログラムスイートがいろいろあります。特に良く知られているものには:

- **Amanda** (**A**dvanced **M**aryland **A**utomatic **N**etwork **D**isk **A**rchiver) は、ネイティブユーティリティ（**tar** と **dump** を含む）を使用しますが、はるかに堅牢で制御しやすいです。

  Amanda は、一般的に標準リポジトリから入手してエンタープライズ **Linux** システムで利用できます。詳細な情報は http://www.amanda.org で確認できます。

- **Bacula** は、異種ネットワーク上の自動バックアップ用に設計されています。かなり複雑なので、経験豊富な管理者のみが使用することを（作成者は）推奨しています。

  Bacula は、通常標準リポジトリを通じてエンタープライズ **Linux** システムで利用できます。詳細な情報は https://www.bacula.org/7.0.x-manuals/en/main/Main_Reference.html で確認できます。

- **Clonezilla** は、非常に堅牢な **ディスククローンプログラム** です。ディスクのイメージを作成して展開しバックアップを復元したり、**ゴースト化（＝クローン作成）**に利用して多くのマシンのインストールに使用できるイメージを提供できます。詳細な情報は https://clonezilla.org で確認できます。

  プログラムには２つのバージョンがあります。単一マシンのバックアップとリカバリに適した **Clonezilla Live** と、同時に多くのコンピュータにクローンを作成できるサーバーエディションの **Clonezilla SE, server edition** です。

  Clonezilla の使用はそれほど難しくなく非常に柔軟で、（**Linux** だけでなく）多くのオペレーティングシステム、ファイルシステムの種類、ブートローダーをサポートしています。

40.10 演習

課題 40.1: tar を使ったバックアップ

1. backup ディレクトリを作成、その下に /usr/include 下の全ファイルを圧縮し、上位ディレクトリの include を含む **tar** アーカイブを作成します。圧縮形式には **gzip**、**bzip2** または **xzip** のどれを使っても構いません。
2. アーカイブ中のファイルをリストします。
3. restore というディレクトリを作成し、その下にアーカイブを解凍します。
4. オリジナルのディレクトリ下の内容とアーカイブを解凍した内容を比較します。

解 40.1

1. ```
   $ mkdir /tmp/backup
   $ cd /usr ; tar zcvf /tmp/backup/include.tar.gz include
   $ cd /usr ; tar jcvf /tmp/backup/include.tar.bz2 include
   $ cd /usr ; tar Jcvf /tmp/backup/include.tar.xz include
   ```

   または

   ```
   $ tar -C /usr -zcf include.tar.gz include
   $ tar -C /usr -jcf include.tar.bz2 include
   $ tar -C /usr -Jcf include.tar.xz include
   ```

   3 つの圧縮方式の効率を比較します：

   ```
   $ du -sh /usr/include
   ```

   ```
   55M        /usr/include
   ```

2. ```
   $ ls -lh include.tar.*
   ```

   ```
   c8:/tmp/backup>ls -lh
   total 17M
   -rw-rw-r-- 1 coop coop 5.3M Jul 18 08:17 include.tar.bz2
   -rw-rw-r-- 1 coop coop 6.7M Jul 18 08:16 include.tar.gz
   -rw-rw-r-- 1 coop coop 4.5M Jul 18 08:18 include.tar.xz
   ```

3. ```
   $ tar tvf include.tar.xz
   ```

   ```
   qdrwxr-xr-x root/root         0 2014-10-29 07:04 include/
   -rw-r--r-- root/root     42780 2014-08-26 12:24 include/unistd.h
   -rw-r--r-- root/root       957 2014-08-26 12:24 include/re_comp.h
   -rw-r--r-- root/root     22096 2014-08-26 12:24 include/regex.h
   -rw-r--r-- root/root      7154 2014-08-26 12:25 include/link.h
   .....
   ```

   解凍の際は j、J または z オプションは不要です: **tar** は自動的に判断します。

4. ```
   $ cd .. ; mkdir restore ; cd restore
   $ tar xvf ../backup/include.tar.bz2
   ```

   ```
   include/
   include/unistd.h
   include/re_comp.h
   include/regex.h
   include/link
   .....
   $ diff -qr include /usr/include
   ```

課題 40.2: cpio を使ったバックアップ

先の演習と同じですが **tar** の代わりに **cpio** を利用します。内容は、若干の変更を行うことで容易に利用できるようにします。

1. backup ディレクトリを作成、その下に /usr/include 下の全ファイルを圧縮し、上位ディレクトリの include を含む **tar** アーカイブを作成します。圧縮形式には **gzip**、**bzip2** または **xzip** のどれを使っても構いません。
2. アーカイブ中のファイルをリストします。
3. restore というディレクトリを作成し、その下にアーカイブを解凍します。
4. オリジナルのディレクトリ下の内容とアーカイブを解凍した内容を比較します。

解 40.2

1. `$ (cd /usr ; find include | cpio -c -o > /home/student/backup/include.cpio)`

```
82318 blocks
```

   または、圧縮形式で作成します:

   `$ (cd /usr ; find include | cpio -c -o | gzip -c > /home/student/backup/include.cpio.gz)`

```
82318 blocks
```

   `$ ls -lh include*`

```
total 64M
-rw-rw-r-- 1 coop coop  41M Nov  3 15:26 include.cpio
-rw-rw-r-- 1 coop coop 6.7M Nov  3 15:28 include.cpio.gz
-rw-rw-r-- 1 coop coop 5.3M Nov  3 14:44 include.tar.bz2
-rw-rw-r-- 1 coop coop 6.8M Nov  3 14:44 include.tar.gz
-rw-rw-r-- 1 coop coop 4.7M Nov  3 14:46 include.tar.xz
```

2. `$ cpio -ivt < include.cpio`

```
drwxr-xr-x  86 root     root            0 Oct 29 07:04 include
-rw-r--r--   1 root     root        42780 Aug 26 12:24 include/unistd.h
-rw-r--r--   1 root     root          957 Aug 26 12:24 include/re_comp.h
-rw-r--r--   1 root     root        22096 Aug 26 12:24 include/regex.h
.....
```

   入力がリダイレクションされていることに注意してください。引数としてアーカイブを指定しないやり方もあります:

```
$ cd ../restore
$ cat ../backup/include.cpio | cpio -ivt
$ gunzip -c include.cpio.gz | cpio -ivt
```

3.
```
$ rm -rf include
$ cpio -id < ../backup/include.cpio
$ ls -lR include
```

   または、

   `$ cpio -idv < ../backup/include.cpio`

   `$ diff -qr include /usr/include`

課題 40.3: rsync を使ったバックアップ

1. **rsync** を利用して /usr/include のコピーをバックアップディレクトリに作成します:

```
$ rm -rf include
$ rsync -av /usr/include .
```

```
sending incremental file list
include/
include/FlexLexer.h
include/_G_config.h
include/a.out.h
include/aio.h
.....
```

2. ２回目のコマンド実行の結果を見ます:

```
$ rsync -av /usr/include .
```

```
sending incremental file list

sent 127398 bytes  received 188 bytes  255172.00 bytes/sec
total size is 41239979  speedup is 323.23
```

3. **rsync** で混乱しやすいのは、以下のコマンド形式が誤りであることです:

```
$ rsync -av /usr/include include
```

```
sending incremental file list
...
```

このコマンドを実行すると、新しいディレクトリ include/include が作成されます！

4. 不要なファイルを削除するためには --delete オプションを用います:

```
$ rsync -av --delete /usr/include .
```

```
sending incremental file list
include/
deleting include/include/xen/privcmd.h
deleting include/include/xen/evtchn.h
....
deleting include/include/FlexLexer.h
deleting include/include/

sent 127401 bytes  received 191 bytes  85061.33 bytes/sec
total size is 41239979  speedup is 323.22
```

5. 他のシンプルな演習として、バックアップコピーからサブディレクトリを削除し --dry-run オプションをつけた場合と、つけない場合で **rsync** を実行:

```
$ rm -rf include/xen
$ rsync -av --delete --dry-run /usr/include .
```

```
sending incremental file list
include/
include/xen/
include/xen/evtchn.h
include/xen/privcmd.h

sent 127412 bytes  received 202 bytes  255228.00 bytes/sec
total size is 41239979  speedup is 323.16 (DRY RUN)
```

```
$ rsync -av --delete  /usr/include .
```

6. **rsync** を適切なオプションの組み合わせで使うスクリプトを紹介します:

script using rsync

```
#!/bin/sh
set -x

rsync --progress -avrxH --delete $*
```

ローカルマシン上でも、ネットワーク越えでも同様に実行できます。-x オプションを用いると、ファイルシステム境界を越えた **rsync** の実行を抑止します。

追加課題

複数のマシンを利用することができるならば、ソースとバックアップ先に異なるマシンを指定して実行します。

章 41

Linux Security Modules

41.1 Linux Security Modules

Linux Security Modules（LSM）とは

- **LSM** (**L**inux **S**ecurity **M**odule) **mandatory access controls** は、要求に応じてカーネルに作用する
- **LSM** の使用:
  - カーネルへの変更が最少となること
  - カーネルへのオーバーヘッドが最小となること
  - 柔軟性があり実装に対して選択の自由度があること、それらは **LSM** (**L**inux **S**ecurity **M**odule) の中で自己完結している

最新のコンピュータシステムはセキュリティで保護する必要がありますが、そのセキュリティ要件はデータの機密性、アカウントを持つユーザーの数、外部ネットワークへの露出、法的要件、その他の要因によって異なります。適切なセキュリティ制御を使用可能にする責任は、アプリケーション設計者と **Linux** カーネル開発者とメンテナにあります。もちろんユーザーも適切な手順に従う必要があります。しかし適切に実行されているシステムでは、非特権ユーザーはシステムをセキュリティ侵害にさらすことがないように操作権限が非常に限られているはずです。このセクションでは **Linux** カーネルが **Linux Security Modules** フレームワークを使用してセキュリティを強化する方法、特に **SELinux** を中心に説明します。

基本的な考え方は **システムコールをフック** することです。強化された権限で作業を実行するために、アプリケーションがカーネル（システム）モードへの移行を要求する部分にコードを挿入します。このコードはパーミッションが有効であること、悪意のある意図から保護されていることなどの確認を行います。これは、システムコールがカーネルによって実行される前後にセキュリティ関連の機能ステップを呼び出す形で行われます。

主要な LSM の選択

- **SELinux**: https://selinuxproject.org/page/Main_Page
- **AppArmor**: https://apparmor.net
- **Smack**: http://schaufler-ca.com
- **TOMOYO**: https://tomoyo.osdn.jp

複数 LSM の同時利用について

2019 年以降は、適切な順序に従えば複数の **LSM** を組み合わせる（スタックする）ことが可能になっています。

- https://www.starlab.io/blog/a-brief-tour-of-linux-security-modules にセキュリティメカニズム紹介と選択に関する優れたレビューがある。

長い間、強化セキュリティモデル実装は **SELinux** だけでした。2001 年にこのプロジェクトが最初にアップストリームでカーネルに組み込まれた時、セキュリティ強化のアプローチの選択肢が 1 種類に限定されることに対して反論がありました。

その結果 **SELinux** 以外のモジュールも使用できる **LSM** アプローチが採用され、開発成果は 2003 年に Linux カーネルに組み込まれました。

当初は **Linux** カーネルの同じ部分を変更する可能性があるため、一度に使用できる **LSM** は 1 つだけでした。

しかし現在では複数 **LSM** のスタックも可能です。スタックさせる時は、モジュールに対し **major** と **minor** を指定します。

ユーザーの利用度の多い順に最初に **SELinux** を説明し、次に **AppArmor** を説明します。

41.2 SELinux

SELinux の概要

- NSA によって開発された
- 保護のためのメソッドが追加された
- 利用するのは:
  - コンテキスト
  - ルール
  - ポリシー
- デフォルトでは、明示的に許可されていないアクションは全て拒否する
- オンライン上の追加情報:
  SELinux User's and Administrator's Guide:
  https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/selinux_users_and_administrators_guide/index

SELinux は、もともと米国 **NSA** (**N**ational **S**ecurity **A**dministration) によって開発されたもので、非常に長い間 **RHEL** に統合され多くの利用事例がありました。

運用という観点から見ると **SELinux** はプロセスがシステム上のどのファイル、ディレクトリ、ポート、およびその他のアイテムにアクセスできるかを決定するセキュリティルールのセットです。

３つの基本概念:

1. **コンテキスト:** ファイル、プロセス、ポートに付与されたラベル。例には **SELinux** ユーザー、ロール、タイプがあります。
2. **ルール: コンテキスト**、**プロセス**、**ファイル**、**ポート**、**ユーザー** 等のアクセス制御を記述したものです。
3. **ポリシー: SELinux** が行うシステム全体のアクセス制御の判断が何かを記述する **ルール** のセットです。

SELinux コンテキストは、ユーザー、プロセス、ファイル、およびポートが相互に対話する方法を定義するルールによって使用されるラベルです。デフォルトのポリシーはアクセス拒否なので、システムで許可させたいアクションをルールで定義します。

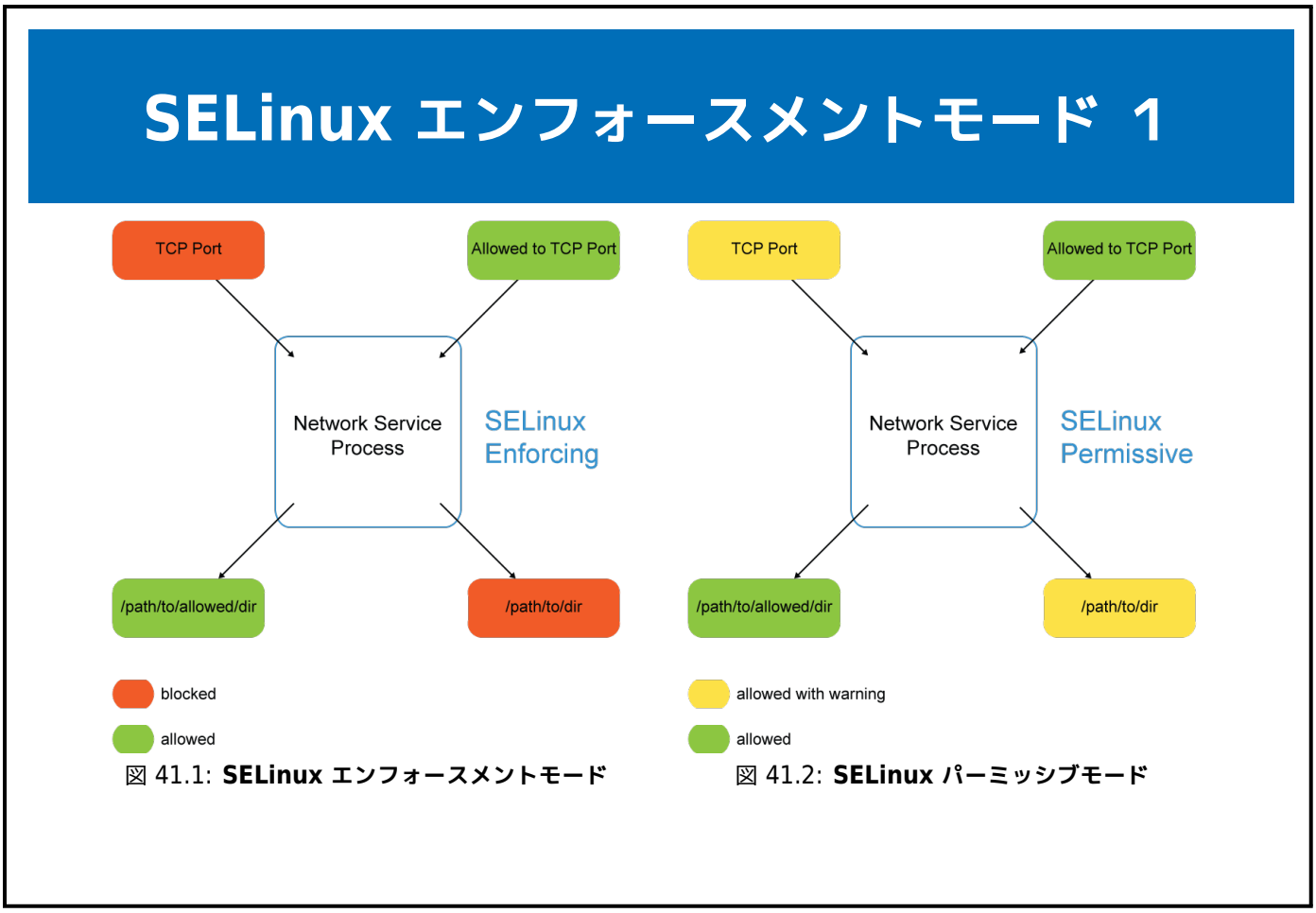


図 41.1: **SELinux エンフォースメントモード**

図 41.2: **SELinux パーミッシブモード**

これらのモードは、ファイル（通常は `/etc/selinux/config`）で選択（および説明）され、その場所はディストリビューションによって異なります。（多くの場合 `/etc/sysconfig/selinux` もしくはそこからリンクされています）ファイルは読むだけで内容がわかるように自己文書化されています。

SELinux エンフォースメントモード 2

- **SELinux** には３つのモードがある
  - エンフォーシング (Enforcing)
  - パーミッシブ (Permissive)
  - 無効 (Disabled)
- 定義ファイルは `/etc/sysconfig/selinux` (**CentOS** と **openSUSE**) または `/etc/selinux/config` (**Ubuntu**)
- **getenforce** と **setenforce** で、現在のモード確認、設定を行う

- **エンフォーシング: (Enforcing)** 全ての **SELinux** は有効で、ポリシーに従ってアクセスは拒否されます。すべての違反を監査しログに記録します。
- **パーミッシブ: (Permissive) SELinux** は有効ですが、アクセスを拒否せずエンフォーシングモードでは拒否される動作についての監査と警告のみが行われます。
- **無効: (Disabled) SELinux** カーネルとアプリケーションコードを完全に無効にして、システムを保護しません。

sestatus ユーティリティは、現在のモードとポリシーを表示します。

getenforce と setenforce

- 現在のモードの確認と設定:
  - **getenforce**

```
$ getenforce
Enforcing
```

  - **setenforce**

```
$ sudo setenforce Permissive
$ getenforce
Permissive
```

setenforce を使用して、システムの稼働中に enforcing モードと permissive モードを切り替えることができます。ただし、この方法では disabled モードへの切り替えは行えません。

setenforce を使って **Permissive** と **Enforcing** モードを切り替える事ができますが **SELinux** を完全に無効にすることはできません。**SELinux** を無効にする方法は少なくとも２つあります。

- **設定ファイル: SELinux** の設定ファイル (通常は `/etc/selinux/config`) を編集し `SELINUX=disabled` を指定する。これがデフォルトのやり方で **SELinux** を恒久的に無効化するために利用されます。
- **カーネルパラメータ:** 再起動時に `selinux=0` をカーネルパラメータに追加します。

ただし、**SELinux** を再度有効にする可能性のあるシステムでは **SELinux** を無効にすることはお勧めできません。**SELinux** を無効にするのではなく、パーミッシブモードを使用することをお勧めします。こうすれば、時間のかかるファイルシステム全体を対象としたラベルの再割り付けを回避できるからです。

SELinux ポリシー

- **targeted:**
  SELinux の **デフォルトポリシー**で、プロセスを対象（=targeted）にかなりの制限を課す。ユーザープロセスと **init** プロセスは対象外だがネットワークサービスプロセスは対象となる。**SELinux** は **すべてのプロセス** にメモリー制限を適用する。これにより、バッファオーバーフロー攻撃に対する脆弱性が軽減される。
- **minimum:**
  選択されたプロセスのみが保護される targeted ポリシーの一種である。
- **MLS:**
  マルチレベルセキュリティ（MLS: **M**ulti-**L**evel **S**ecurity）は、一番制限が厳しいポリシーである。すべてのプロセスは、特定のポリシーを持つきめ細かいセキュリティ ドメインに置かれる。

同じ構成ファイル（通常は `/etc/sysconfig/selinux` です）に **SELinux ポリシー** を設定します。複数のポリシーを定義できますが、一度にアクティブにできるのは１つだけです。ポリシーを変更するには、システムの再起動とファイルシステムの内容の時間のかかる再ラベル付けが必要になる場合があります。各ポリシーは `/etc/selinux/[SELINUXTYPE]` の下にインストールされるファイルに設定されています。

コンテキスト ユーティリティ

- コンテキストとコンテキスト ユーティリティ
  - ユーザー (User)
  - 役割 (Role)
  - タイプ (Type)
  - レベル (Level)
- コンテキストの参照と変更:
  - -Z オプションを使ってコンテストを見る

```
$ ls -Z
$ ps auZ
```

  - **chcon** を使って、コンテキストを変更する

```
$ chcon -t etc_t somefile
$ chcon --reference somefile someotherfile
```

前述したように、コンテキストはファイル、ディレクトリ、ポート、プロセスに適用されるラベルです。**SELinux** には **User**、**Role**、**Type**、**Level** という４つのコンテキストがあります。

ここでは、最も一般的に使用されるコンテキストである **type** に注目します。ラベルの命名規則により `kernel_t` のように type コンテキストのラベルは `_t` で終わる必要があります。

```
$ ls -Z
-rw-rw-r--. dog dog unconfined_u:object_r:user_home_t:s0 somefile
```

```
$ chcon -t etc_t somefile
$ ls -Z
-rw-rw-r--. dog dog unconfined_u:object_r:etc_t:s0   somefile
```

```
$ ls -Z
-rw-rw-r--. dog dog unconfined_u:object_r:etc_t:s0   somefile
-rw-rw-r--. dog dog unconfined_u:object_r:user_home_t:s0 somefile1
```

```
$ chcon --reference somefile somefile1
$ ls -Z
-rw-rw-r--. dog dog unconfined_u:object_r:etc_t:s0   somefile
-rw-rw-r--. dog dog unconfined_u:object_r:etc_t:s0   somefile1
```

SELinux と標準コマンド

- **ls**、**ps**、**cp** など基本ユーティリティの多くが **SELinux** と連携するために拡張されている
- 通常は Z オプションを付加する:

```
$ ls -Z ...
$ ps axZ
```

- **SELinux** が無効化されていると、拡張オプションは機能しない

ls や **ps** などの多くの標準コマンドラインコマンドは **SELinux** をサポートするように拡張され、**man** ページに詳細な説明が追加されています。パラメータ Z が次のように標準のコマンドラインツールに渡されます:

```
$ ps axZ
```

```
LABEL PID TTY STAT TIME COMMAND
system_u:system_r:init_t:s0 1 ? Ss 0:04 /usr/lib/systemd/systemd --switched-root ...
system_u:system_r:kernel_t:s0 2 ? S 0:00 [kthreadd]
...
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 2305 ? D 0:00 sshd: jimih@pts/0
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 2306 pts/0 Ss 0:00 -bash
...
system_u:system_r:httpd_t:s0 7490 ? Ss 0:00 /usr/sbin/httpd -DFOREGROUND
system_u:system_r:httpd_t:s0 7491 ? S 0:00 /usr/sbin/httpd -DFOREGROUND
...
```

```
$ ls -Z /home/ /tmp/
```

```
/home/:
drwx------. jimih jimih unconfined_u:object_r:user_home_dir_t:s0 jimih
/tmp/:
-rwx------. root root system_u:object_r:initrc_tmp_t:s0 ks-script-c4ENhg
drwx------. root root system_u:object_r:tmp_t:s0 systemd-private-0ofSvO
-rw-------. root root system_u:object_r:initrc_tmp_t:s0 dnf.log
```

SELinux をサポートするために拡張されたツールには、他に **cp**、**mv**、**mkdir** があります。**SELinux** を無効にした場合、これらのユーティリティの関連フィールドには有用な情報は表示されないことに注意してください。

コンテキストの継承

- 新しいファイルは、親ディレクトリからコンテキストを継承する
- ファイルを移動しても、元のディレクトリコンテキストが保存される
- コンテキストを変更しないと問題が発生する場合がある

新しく作成されたファイルは親ディレクトリからコンテキストを継承しますが、ファイルを移動しても保存されているのは移動前のソースディレクトリのコンテキストであるため、移動後のディレクトリで問題を引き起こす可能性があります。

前の例で言えばファイルを `/tmp` から `/home/jimih` に移動しても `tmpfile` のコンテキストは変更されません。

```
$ cd /tmp/
$ touch tmpfile
$ ls -Z tmpfile
```

```
1 -rw-rw-r--. jimih jimih unconfined_u:object_r:user_tmp_t:s0 tmpfile
```

```
$ cd
$ touch homefile
$ ls -Z homefile
```

```
1 -rw-rw-r--. jimih jimih unconfined_u:object_r:user_home_t:s0 homefile
```

```
$ mv /tmp/tmpfile .
$ ls -Z
```

```
1 -rw-rw-r--. jimih jimih unconfined_u:object_r:user_home_t:s0 homefile
2 -rw-rw-r--. jimih jimih unconfined_u:object_r:user_tmp_t:s0 tmpfile
```

ファイルの移動で **SELinux** の問題が発生する典型的な例は **httpd** サーバーの `DocumentRoot` ディレクトリへのファイル移動です。**SELinux** が有効化されたシステムでは、ウェブサーバーは正しいコンテキストラベルを持つファイルにのみアクセスできます。ファイルを `/tmp` に作成し、それを `DocumentRoot` ディレクトリに移動した場合、そのファイルの **SELinux** コンテキストが調整されるまで **httpd** サーバーからファイルにアクセスできなくなります。

restorecon

- 親ディレクトリと調和するように、ファイルコンテキストをリセット:
  `$ restorecon -Rv /home/jimih`

restorecon は、親ディレクトリの設定に基づいてファイルコンテキストをリセットします。

次の例では **restorecon** はホームディレクトリのすべてのファイルのデフォルトラベルを再帰的にリセットします:

```
$ ls -Z
-rw-rw-r--. jimih jimih unconfined_u:object_r:user_home_t:s0 homefile
-rw-rw-r--. jimih jimih unconfined_u:object_r:user_tmp_t:s0 tmpfile

$ restorecon -Rv /home/jimih
restorecon reset /home/jimih/tmpfile context \
           unconfined_u:object_r:user_tmp_t:s0->unconfined_u:object_r:user_home_t:s0

$ ls -Z
-rw-rw-r--. jimih jimih unconfined_u:object_r:user_home_t:s0 homefile
-rw-rw-r--. jimih jimih unconfined_u:object_r:user_home_t:s0 tmpfile
```

`tmpfile` のコンテキストが、ホームディレクトリで作成されたファイルのデフォルトコンテキストにリセットされていることに注目してください。タイプが `user_tmp_t` から `user_home_t` に変更されました。

semanage

- 新しいディレクトリに対する、デフォルトコンテキストを設定する
- デフォルト値の設定だけで、既存のディレクトリの設定は変更されない
- 既存コンテキストの設定変更は **restorecon** を続けて実行する

別の課題として、新しく作成されたディレクトリのデフォルトコンテキストの設定方法があります。**semanage fcontext** (**policycoreutils-python** パッケージによって提供される) は、ファイルおよびディレクトリのデフォルトコンテキストの変更と表示ができます。**semanage fcontext** はデフォルト設定のみを変更することに注意してください。既存のオブジェクトにすぐには適用されません。これをするには、その後に **restorecon** を実行する必要があります。例えば:

```
[root@rhel7 /]# mkdir /virtualHosts
[root@rhel7 /]# ls -Z
...
drwxr-xr-x. root root unconfined_u:object_r:default_t:s0 virtualHosts

[root@rhel7 /]# semanage fcontext -a -t httpd_sys_content_t /virtualHosts
[root@rhel7 /]# ls -Z
...
drwxr-xr-x. root root unconfined_u:object_r:default_t:s0 virtualHosts

[root@rhel7 /]# restorecon -RFv /virtualHosts
restorecon reset /virtualHosts context \
          unconfined_u:object_r:default_t:s0->system_u:object_r:httpd_sys_content_t:s0

[root@rhel7 /]# ls -Z
drwxr-xr-x. root root system_u:object_r:httpd_sys_content_t:s0 virtualHosts
```

したがって `default_t` から `httpd_sys_content_t` へのコンテキスト変更は **restorecon** 実行後に適用されます。

SELinux のブール値の使い方

- **SELinux** ポリシーの挙動を変更する
- 有効化／無効化の設定は
  - **getsebool**: ブール値を表示する
  - **setsebool**: ブール値を設定する
  - **semanage boolean -l**: 恒久的にブール値を設定する

```
$ setsebool allow_ftpd_anon_write on
$ getsebool allow_ftpd_anon_write
  allow_ftpd_anon_write -> on
$ semanage boolean -l | grep allow_ftpd_anon_write
$ allow_ftpd_anon_write -> off
```

これは、恒久的な設定ではありません。

```
$ setsebool -P allow_ftpd_anon_write on
$ semanage boolean -l | grep allow_ftpd_anon_write
  allow_ftpd_anon_write          -> on
```

これで恒久化されました。

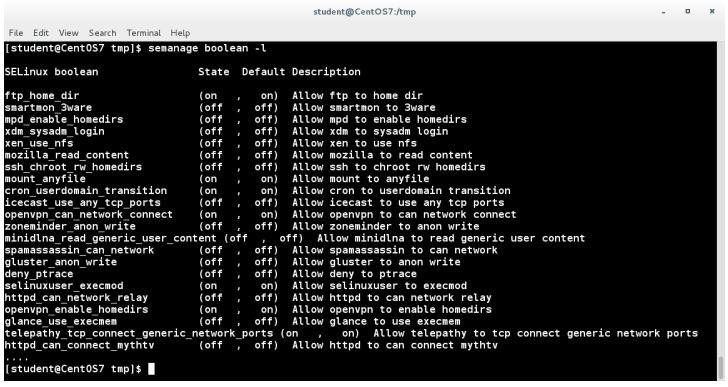


図 41.3: **semanage**

SELinux アクセスの監視

- **setroubleshoot-server** を使って **SELinux** のエラーを追跡する
  - インストール後に **auditd** サービスを再起動する
- RAW（= 生）メッセージは `/var/log/audit/audit.log` に保存
- メッセージを `/var/log/messages` 移動する
- **sealert** を利用して詳細メッセージを見る

SELinux には、実行時に問題を収集し、これらの問題をログに記録し、同じ問題の再発を防ぐソリューションを提案するツール群が付属しています。これらのユーティリティは **setroubleshoot-server** パッケージで提供されます。それらの使用例を次に示します:

```
[root@rhel7 ~]# echo 'File created at /root' > rootfile
[root@rhel7 ~]# mv rootfile /var/www/html/
[root@rhel7 ~]# wget -O - localhost/rootfile
--2014-11-21 13:42:04--  http://localhost/rootfile
Resolving localhost (localhost)... ::1, 127.0.0.1
Connecting to localhost (localhost)|::1|:80... connected.
HTTP request sent, awaiting response... 403 Forbidden
2014-11-21 13:42:04 ERROR 403: Forbidden.

[root@rhel7 ~]# tail /var/log/messages
Nov 21 13:42:04 rhel7 setroubleshoot: Plugin Exception restorecon
Nov 21 13:42:04 rhel7 setroubleshoot: SELinux is preventing /usr/sbin/httpd from getattr access on the file .
For complete SELinux messages. run sealert -l d51d34f9-91d5-4219-ad1e-5531e61a2dc3
Nov 21 13:42:04 rhel7 python: SELinux is preventing /usr/sbin/httpd from getattr access on the file .
.....
Do allow this access for now by executing:
# grep httpd /var/log/audit/audit.log | audit2allow -M mypol
# semodule -i mypol.pp

Additional Information:
Source Context system_u:system_r:httpd_t:s0
Target Context unconfined_u:object_r:admin_home_t:s0
Target Objects [ file ]
Source httpd
Source Path /usr/sbin/httpd
......
```

41.3 AppArmor

AppArmor

- 強制アクセス制御 (**MAC**) 機能を提供する
- 管理者は、機能を制限したいプログラムにセキュリティプロファイルを関連付けることができる
- **SELinux** より使い方が簡単と考えられている（全面的とは言えないが）
- ファイルシステム中立と考えられている (セキュリティラベル付けが必要ないので).

AppArmor は **SELinux** の代替となる **LSM** です。2006 年以降 **Linux** カーネルにサポートが組み込まれています。**SUSE**、**Ubuntu** およびその他のディストリビューションで使用されています。

AppArmor は強制アクセス制御 (= **M**andatory **A**ccess **C**ontrol (**MAC**)) を提供して、従来の **UNIX** の任意アクセス制御 (=**D**iscretionary **A**ccess **C**ontrol (**DAC**)) モデルを補完します。

手動によるプロファイル指定に加えて **AppArmor** には学習モードがあります。このモードでは、プロファイルの違反はログに記録されますが、プログラムの実行は阻止されません。典型的なプログラム動作に基づくログからプロファイルに変換できます。

状態の確認

- **AppArmor** はデフォルトでロードされ、有効化される
- **Linux** カーネルのコンパイル時に組み込んで有効化する必要がある
- 起動、停止など:
  `$ sudo systemctl [start|stop|restart|status] apparmor`
- 起動時にロードするかどうかの設定:
  `$ sudo systemctl [enable|disable] apparmor`
- 現在のステータスの詳細を見る:
  `$ sudo apparmor_status`
- **Profiles** と **processes** には **enforce** と **complain** モードがある
  - **SELinux** における **enforcing** と **permissive** モードに相当する

AppArmor を採用しているディストリビューションは、デフォルトでそれを有効にしてロードすることが多いです。**Linux** カーネル（= **AppArmor** カーネルモジュール）も同様に有効にする必要があります。そして多くの場合一度に実行できる **LSM** は１つだけです。**AppArmor** カーネルモジュールが利用可能である場合 **systemd** が採用されたシステムでは次のことが可能です:

```
$ sudo systemctl [start|stop|restart|status] apparmor
```

現在の動作ステータスを変更、ないし問い合わせします。または、実行します:

```
$ sudo systemctl [enable|disable] apparmor
```

起動時にロードする／しないを指定します。
現在のステータスを見るには:

```
$ sudo apparmor_status
```

```
 apparmor module is loaded.
25 profiles are loaded.
25 profiles are in enforce mode.
   /sbin/dhclient
....
```

Profiles と **processes** には **enforce** ないし **complain** モードがあります。**SELinux** の **enforcing** と **permissive** に相当するものです。
プロセスのリストでは **PID** が与えられます:

```
$ ps aux | grep libvirtd
```

```
root         787  0.0  0.9 527200 35936 ?        Ssl  10:54   0:00 /usr/sbin/libvirtd
student     3346  0.0  0.0  13696  2204 pts/16   S+   11:42   0:00 grep --color=auto libvirtd
```

モード と プロファイル

- モード:
  - **エンフォース (Enforce) モード**　制限されたアクションは阻止され、試行が記録される。**aa-enforce** で設定する
  - **コンプレイン (Complain) モード**　ポリシーは実施されないが、違反の試行は記録される。**aa-complain** で設定する
- `/etc/apparmor.d` の中に設定された **ポリシー** によってユーティリティやソフトウエアパッケージの詳細設定ができる

プロファイルによって `/usr/bin/evince` などのパス名を持つシステム上の実行プログラムの使用方法を制限します。
プロセッサーは下記のどちらかのモードで実行されます:

- **エンフォース (Enforce) モード**
  アプリケーションにはポリシー遵守が強制され、制限された動作は阻止されます。違反の試行がシステムログファイルに記録されます。これがデフォルトのモードです。プロファイルをこのモードに設定するには **aa-enforce** を使用します。
- **コンプレイン (Complain) モード**
  ポリシー遵守の強制はありませんが、ポリシー違反の試行は記録されます。これは学習モードとも呼ばれます。プロファイルをこのモードに設定するには **aa-complain** を使用します。

Linux ディストリビューションには、あらかじめパッケージ化されたプロファイルが付属しており、通常は、その特定のパッケージがインストールされたときにプロファイルが一緒にインストールされます。もしくは **apparmor-profiles** などの **AppArmor** パッケージと一緒にプロファイルがインストールされます。これらのプロファイルは `/etc/apparmor.d` に保存されます。

新しいソフトウェアをインストールするときにはパッケージ内の実行可能ファイルに紐付いた新しいプロファイルを作成できます。

システムにインストールされる正確な **AppArmor** プロファイルは、ソフトウェアパッケージ選択によって異なります。
たとえばある **Ubuntu** システムの場合:

```
student@ubuntu:/etc/apparmor.d$ ls
abstractions              usr.lib.dovecot.anvil     usr.lib.telepathy
apache2.d                 usr.lib.dovecot.auth      usr.sbin.avahi-daemon
bin.ping                  usr.lib.dovecot.config    usr.sbin.cups-brows
....
```

これらのファイルで何ができるかについての完全なドキュメントは `man apparmor.d` とすれば入手できます。

AppArmor ユーティリティ

Table 41.1: **AppArmor ユーティリティ**

プログラム	用　途
apparmor_status	すべてのプロファイルおよびプロファイルを含むプロセスのステータスを表示
apparmor_notify	Show a summary for **AppArmor** log messages **AppArmor** のログメッセージの概要を表示
complain	特定のプロファイルを complain モードに設定
enforce	指定したプロファイルを強制モードに設定します
disable	現在のカーネルから指定されたプロファイルをアンロードし、システム起動時にロードされないようにする
logprof	ログファイルをスキャンし既存のプロファイルでカバーされていない **AppArmor** イベントが記録されている場合は考慮に入れる方法を提案し、承認されている場合は制限するように変更して再ロードする
easyprof	プログラムの基本的な **AppArmor** プロファイルの生成を支援します。簡易なインターフェイスを提供

AppArmor には、監視と制御のための管理ユーティリティが数多くあります。たとえば **openSUSE** システムの場合:

```
$ rpm -qil apparmor-utils | grep bin
```

```
/usr/bin/aa-easyprof
/usr/sbin/aa-audit
/usr/sbin/aa-autodep
/usr/sbin/aa-cleanprof
/usr/sbin/aa-complain
/usr/sbin/aa-decode
/usr/sbin/aa-disable
/usr/sbin/aa-enforce
/usr/sbin/aa-exec
......
/usr/sbin/complain
/usr/sbin/decode
/usr/sbin/disable
/usr/sbin/enforce
....
```

多くのユーティリティは、短縮された名前を使っても完全な長い名前でも起動することができます。例えば:

```
linux-llgn:/etc/apparmor.d # ls -l /usr/sbin/*complain
```

```
-rwxr-xr-x 1 root root 1442 Oct 25 07:37 /usr/sbin/aa-complain*
lrwxrwxrwx 1 root root   11 Nov 11 13:02 /usr/sbin/complain -> aa-complain*
linux-llgn:/etc/apparmor.d #
```

41.4 演習

課題 41.1: SELinux：コンテキスト

注目

本演習は（**RHEL** のような）**SELinux** がインストールされているシステムでのみ実行可能です。**Ubuntu** のような **Debian** ベースのディストリビューションでもインストールは可能ですが容易ではなく、しばしば失敗します。

1. **getenforce** と **sestatus** を実行して **SELinux** がイネーブルになっていることと **enforcing** モードであることを確認してください。なっていなければ /etc/selinux/config を編集してからリブートして再確認してください。

2. **httpd** パッケージをインストールしてください（まだ、していなければ）。これは **Apache** ウェブサーバです。立ち上がっていることを確認してください:

```
$ sudo dnf install  httpd
$ elinks http:/localhost
```

（ここから先のステップでは、ブラウザとして **lynx** や **elinks** なども使うことができますし、グラフィカル ブラウザとして **firefox** や **chrome** を利用することもできます。）

3. スーパーユーザーで /var/www/html に小さなファイルを作成します:

```
$ sudo sh -c "echo file1 > /var/www/html/file1.html"
```

4. このファイルが見えることを確認します:

```
$ elinks -dump http://localhost/file1.html
```

```
file1
```

root のホーム ディレクトリに別の小さなファイルを作成し、それを /var/www/html に **move** します。（コピーではありません。移動してください。）見てみます:

```
$ sudo cd /root
$ sudo sh -c "echo file2 > file2.html"
$ sudo mv file2.html /var/www/html
$ elinks -dump http://localhost/file2.html
```

```
                                 Forbidden

   You don't have permission to access /file2.html on this server.
```

5. セキュリティコンテキストを調べます:

```
$ cd  /var/www/html
$ ls -Z file*html
```

```
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 file1.html
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 file2.html
```

6. offending コンテキストを変更し、あらためて調べます:

```
$ sudo chcon -t httpd_sys_content_t file2.html
$ elinks http://localhost/file2.html
```

```
file2
```

課題 41.2: apparmor セキュリティを使う

注目

本演習は、（**Ubuntu** のような）**AppArmor** がインストールされているシステムでのみ実行可能です。

以下は **Ubuntu** 上でテストしています。しかし、他の **AppArmor** が実行可能なシステム、たとえば **openSUSE** などでも、**apt-get** コマンドを **zypper** に置き換えるだけでできるはずです。

Ubuntu では **/bin/ping** ユーティリティは、SUID されて実行されます。本演習では **ping** を **ping-x** にコピーして、プログラムが機能するようにケーパビリティを調整します。

次に **AppArmor** プロファイルをビルド、インストールして何も変わっていないことを確認します。**AppArmor** プロファイルを変更、ケーパビリティの追加を行いプログラムの機能性を高めます。

1. 必要なパッケージがインストールされていることを確認します:

```
student@ubuntu:~$ sudo apt-get install apparm*
```

2. **ping** のコピーを作成し（ここでは、**ping-x** とします）、特別なパーミッションやケーパビリティを持っていないことを確認します。さらに、**student**、一般ユーザーでは起動できないことを確認します:

```
student@ubuntu:~$ sudo cp /bin/ping /bin/ping-x
student@ubuntu:~$ sudo ls -l /bin/ping-x

-rwxr-xr-x 1 root root 64424 Oct 17 10:12 /bin/ping-x

student@ubuntu:~$ sudo getcap /bin/ping-x
student@ubuntu:~$

student@ubuntu:~$ ping-x -c3 -4 127.0.0.1
ping: socket: Operation not permitted
```

3. **capabilities** をセットして **ping-x** を再度実行します:

```
student@ubuntu:~$ sudo setcap cap_net_raw+ep  /bin/ping-x

student@ubuntu:~$ ping-x  -c3 -4 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.092 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.093 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.086 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2034ms
rtt min/avg/max/mdev = 0.086/0.090/0.093/0.008 ms
```

変更した **ping-x** プログラムが動作するようになりました。

4. **ping-x** 用の **AppArmor** プロファイルが無く、**ping** 用のプロファイルしかなかったことを確認します。現在の **ping** プログラムの状態を調べます:

```
student@ubuntu:~$ sudo aa-status
```

aa-status の出力は長いため **grep** を使って関心のある部分を抜き出します：

```
student@ubuntu:~$ sudo aa-status | grep -e "^[[:alnum:]]" -e ping

apparmor module is loaded.
87 profiles are loaded.
51 profiles are in enforce mode.
   ping
36 profiles are in complain mode.
17 processes have profiles defined.
6 processes are in enforce mode.
```

```
11 processes are in complain mode.
0 processes are unconfined but have a profile defined.
```

ping がプロファイルを持ち、実行（enforce）されていることがわかります。

5. 次に **ping-x** 用にプロファイルを作成します。複数の端末を利用します。

   一番目の端末（**端末 1**）で、**aa-genprof** コマンドを実行します。これは /var/log/syslog から **AppArmor** のエラーをスキャンすることで、**AppArmor** プロファイルを生成します。

   二番目の端末（**端末 2**）では **ping-x** を実行します。（より詳細な情報は、**aa-genprof** の **man** ページを見てください。）

   端末 1：

```
student@ubuntu:~$ sudo aa-genprof /bin/ping-x
Writing updated profile for /bin/ping-x.
Setting /bin/ping-x to complain mode.

Before you begin, you may wish to check if a
profile already exists for the application you
wish to confine. See the following wiki page for
more information:
http://wiki.apparmor.net/index.php/Profiles

Please start the application to be profiled in
another window and exercise its functionality now.

Once completed, select the "Scan" option below in
order to scan the system logs for AppArmor events.

For each AppArmor event, you will be given the
opportunity to choose whether the access should be
allowed or denied.

Profiling: /bin/ping-x

[(S)can system log for AppArmor events] / (F)inish
```

   端末 2：

```
student@ubuntu:~$ ping-x  -c3 -4 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.099 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.120 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.114 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2041ms
rtt min/avg/max/mdev = 0.099/0.111/0.120/0.008 ms
```

   端末 1：

   ping-x コマンドの実行が完了したら、**aa-genprof** を使ってプロファイルに必要な情報を追加します。必要な全ての情報を収集するためには何回か **スキャン** が必要です。

   S を入力してスキャンを実行します:

```
Reading log entries from /var/log/syslog.
Updating AppArmor profiles in /etc/apparmor.d.
Complain-mode changes:

Profile:    /bin/ping-x
Capability: net_raw
Severity:   8

 [1 - capability net_raw,]
(A)llow / [(D)eny] / (I)gnore / Audi(t) / Abo(r)t / (F)inish
```

A を入力して、ケーパビリティを許可してください:

```
Adding capability net_raw, to profile.

Profile:        /bin/ping-x
Network Family: inet
Socket Type:    raw

 [1 - network inet raw,]
(A)llow / [(D)eny] / (I)gnore / Audi(t) / Abo(r)t / (F)inish
```

A を入力してネットワークファミリーを許可してください:

```
Adding network inet raw, to profile.

Profile:        /bin/ping-x
Network Family: inet
Socket Type:    dgram

 [1 - #include <abstractions/nameservice>]
  2 - network inet dgram,
(A)llow / [(D)eny] / (I)gnore / Audi(t) / Abo(r)t / (F)inish
```

A を入力してプロファイルにソケットタイプのデータグラムを追加してください:

```
Adding #include <abstractions/nameservice> to profile.

= Changed Local Profiles =

The following local profiles were changed. Would you like to save them?

 [1 - /bin/ping-x]
(S)ave Changes / Save Selec(t)ed Profile / [(V)iew Changes] / View Changes b/w (C)lean profiles / Abo(
```

S を入力して新しいプロファイルを格納してください:

```
Writing updated profile for /bin/ping-x.

Profiling: /bin/ping-x

[(S)can system log for AppArmor events] / (F)inish
```

F を入力して終了してください:

```
Setting /bin/ping-x to enforce mode.

Reloaded AppArmor profiles in enforce mode.

                Please consider contributing your new profile!
See the following wiki page for more information:
http://wiki.apparmor.net/index.php/Profiles

Finished generating profile for /bin/ping-x.
```

6. `/etc/appamor.d/bin.ping-x` に格納されているプロファイルを見てみます。

```
student@ubuntu:~$ sudo cat /etc/apparmor.d/bin.ping-x
# Last Modified: Tue Oct 17 11:30:47 2017
#include <tunables/global>

/bin/ping-x {
  #include <abstractions/base>
  #include <abstractions/nameservice>

  capability net_raw,
```

```
  network inet raw,

  /bin/ping-x mr,
  /lib/x86_64-linux-gnu/ld-*.so mr,

}
```

7. **aa-genproc** ユーティリティは、新しいポリシーをインストールし活性化します。すぐに使うことができ **systemctl reload apparmor** コマンドによって、必要に応じてリロードされます。余計な問題を避け、変更が保存されていることを確認するためシステムをリブートします。

   システムを再起動したら、student として新しいプロファイルのもとで **ping-x** が機能することを確認します。ip アドレスを使ってローカルホストに ping します:

```
student@ubuntu:~$ ping-x -c3 -4 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.057 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.043 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.095 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2027ms
rtt min/avg/max/mdev = 0.043/0.065/0.095/0.021 ms
```

8. これは期待通りに動作するはずです。プロファイルは限定的で、**AppArmor** は指定されたパラメータ以上は機能的に許可しません。**AppArmor** がアプリケーションを保護していることを確認するため、ローカルホストの **IPV6** アドレス宛に ping してください。

   失敗します:

```
student@ubuntu:~$ ping-x -c3  -6 ::1
ping: socket: Permission denied
```

   （-6 オプションは **IPv6** のみの使用を意味し、::1 は、**IPv6** のローカルホストを意味します。）

   出力を見るとソケットに関する問題があることがわかります。システムログを見ると **AppArmor** が **ping-x** プログラムに対して **IPv6** アクセスを許可しないことがわかります:

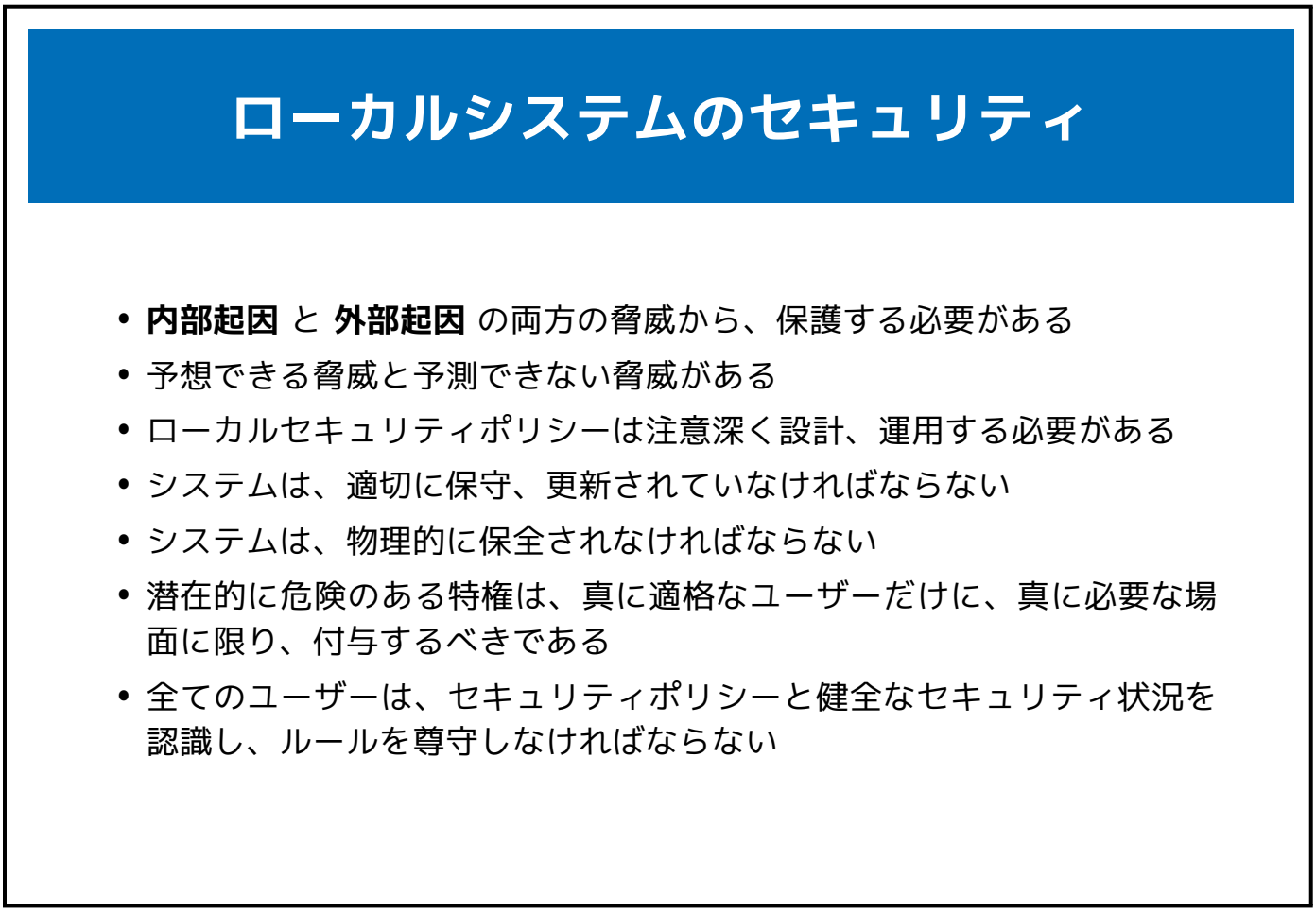
```
766:104): apparmor="DENIED" operation="create" profile="/bin/ping-x"
pid=2709 comm="ping-x" family="inet6" sock_type="raw" protocol=58
requested_mask="create" denied_mask="create
```

9. これを解決するために、以前のように **aa-genprof** を実行し、**端末 2** で **IPv6** ループバックにオプションを追加して ping します。

章 42

ローカルシステムのセキュリティ

42.1 ローカルシステムのセキュリティ



コンピュータは本質的に安全ではなく、侵入したり攻撃したりする人々からの保護が必要です。侵入や攻撃は、システムに害を与える、サービスを拒否する、情報を盗む、ただそれだけのために行われます。

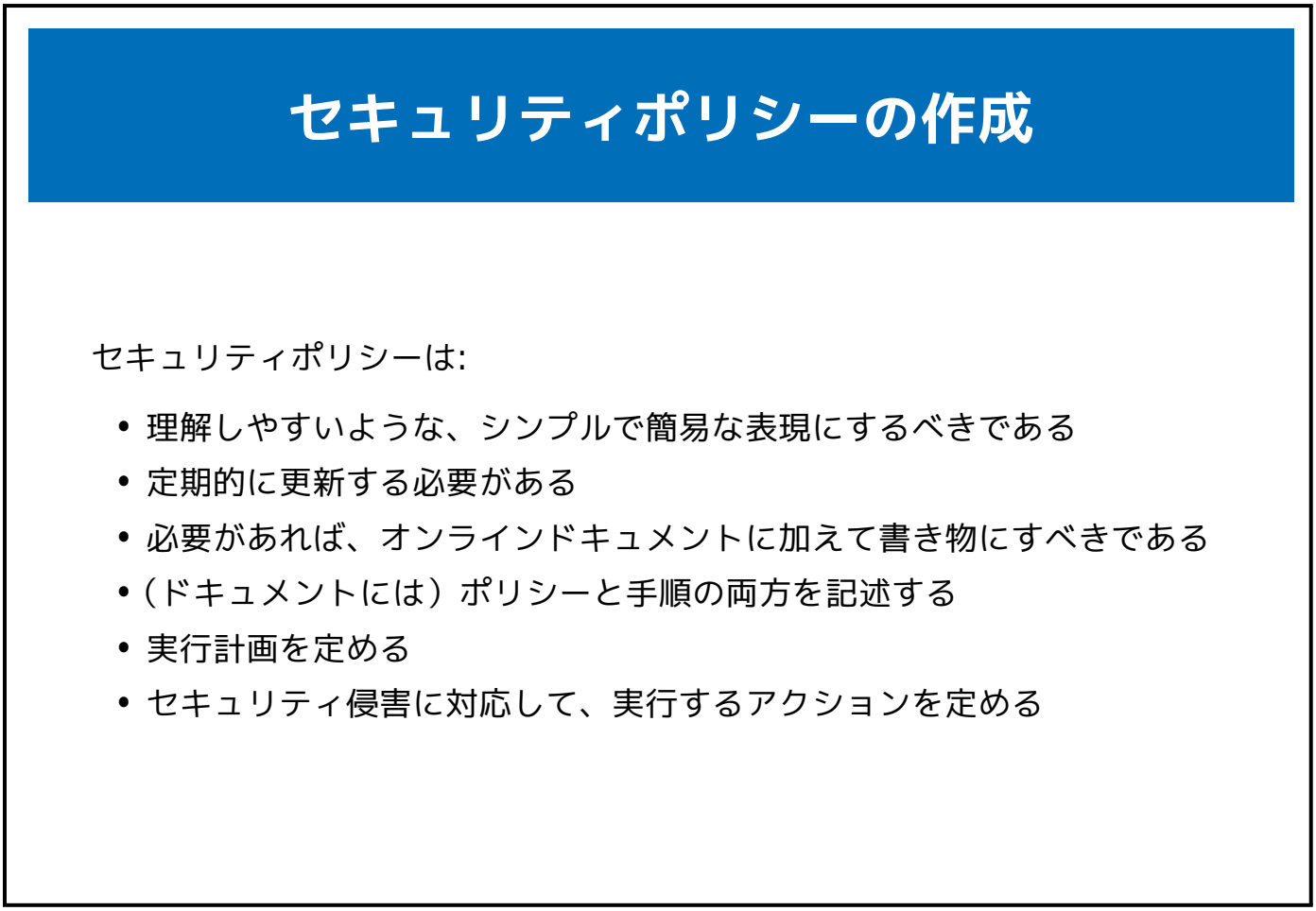
絶対に安全なコンピュータはありません。私たちにできることは、侵入者の動きを鈍化させるか阻止することです。そうすれば侵入者がもっと簡単に侵入できるターゲットを探しに立ち去るか、または、私たちが侵入の動きを捉えて適切に処置できます。

セキュリティとは、システムが本来すべきことを定期的に実行する能力と定義できます。システムの整合性と正確性を実現し、使用を許可された人だけがシステムを使用できるようにすることです。

セキュリティに関する最大の課題は、セキュリティと生産性の適切なバランスを見つけることです。セキュリティ制限が厳しく、解りにくく、使いにくい場合、特に効果のない対策では、ユーザーは使用を避けるでしょう。

保護する必要がある対象は４種類あります。physical（物理的なもの）、local（ローカル）、remote（リモート）、personnel（人的なもの）です。このセクションでは、ネットワークセキュリティではなくローカルのセキュリティを説明します。

42.2 セキュリティポリシーの作成



ポリシーは、理解しやすいような一般的な表現にする必要があります。必要なデータを保護し、要求されたサービスへのアクセスを拒否し、ユーザーのプライバシーを保護する、必要があります。

これらのポリシーは、定期的に更新する必要があります。要件と同様にポリシーも変更する必要があります。古くなったポリシーを持つことはポリシーがないことよりも悪い場合があります。

ポリシーに含めるもの

- 守秘義務
- データの整合性
- 可用性
- 一貫性
- 制御
- 監査

セキュリティポリシーには、権限のない担当者による情報の読み取りやコピーから情報を保護する方法を含める必要があります。また、所有者の許可なく情報が変更または削除されないように保護する必要があります。すべてのサービスを利用できるように保護されるべきで、許可の無い者によるいかなる方法によっても傷つけられることのないようにする必要があります。

データが正しいこと、およびシステムが期待どおりに動作することを確認する必要があります。システムへのアクセス権を誰に付与するかを決定する有効なプロセスがあるはずです。人的な要因は、セキュリティチェーンの中でも最も弱いリンクです。継続的な監査を通じて最も注意を払う必要があります。

評価すべきリスクとは

- 何を保護したいか？
- 何から保護するのか？
- 適切な保護を提供するには、どれだけの時間、人員、費用が必要か？

何を保護するのか（財産の特定）、何から保護するのか（脅威の特定）、システムの保護には何が必要になるか、がリスク分析の基本となります。リスク分析はコンピュータセキュリティの第一歩です。

システムを保護する方法を決定するには、何を何から保護するかを知る必要があります。これにより、システムを保護するためのポリシーとその手続きを計画できます。

これはコンピュータのセキュリティポリシーを構築するための最初のステップです。そして、システムを保護するためのポリシーと手続きを計画・実施するための前提条件です。

セキュリティ方針の選択

- ほとんどのコンピューティング環境では 2 つの基本的な方針が使われる
  - 明示的に許可されていないものは拒否
  - 明示的に禁止されていないものは許可
- 思い出すべき事実:
  1. 人的要因は最も弱いリンクである
  2. 脆弱性の無いコンピュータは無い
  3. 被害妄想でちょうどよい

一番目の方針を選択する時は、より厳格なものにします。特権なしの実行許可が明確かつ明示的に指定されている場合のみ、実行が許されます。これは最も一般的に使用される方針です。

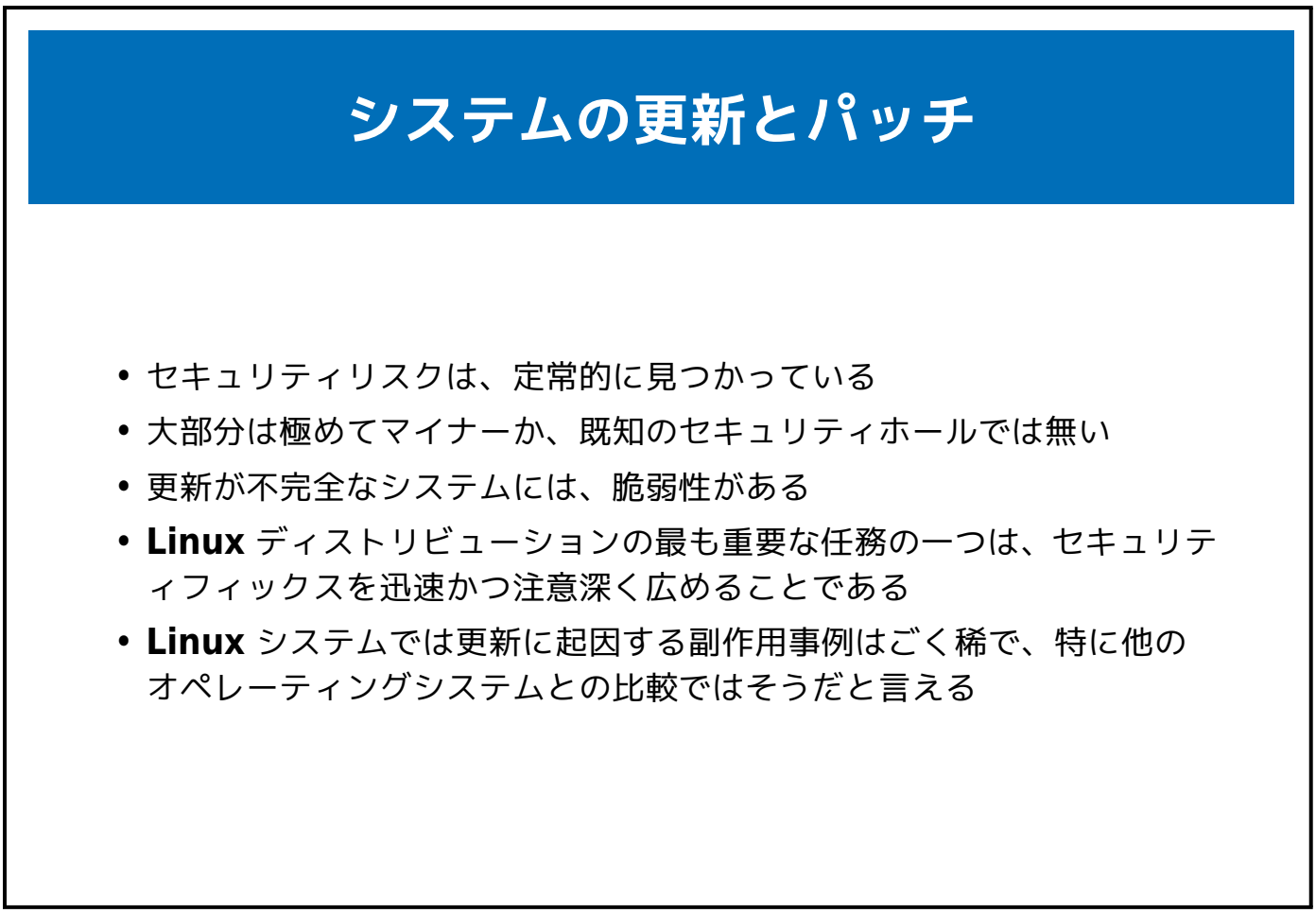
二番目の方針を選択すれば、ユーザーは明示的に禁止されていること以外は何でも実行できるより自由な環境が構築されます。この方針の選択は想定される信頼度が高いことを暗示していますが、よほど明確な理由が無い限りこの方針は採用されません。

セキュリティ方針を作る際に覚えておくべき一般的なガイドラインは:

1. **人的要因は最も弱いリンクである**
   ユーザーを啓蒙し満足させる必要があります。一番大きな比率を占めるのは内部からの攻撃であり、多くの場合悪意のあるものではありません。
2. **脆弱性の無いコンピュータは無い**
   何にも接続されておらず、安全な部屋に閉じ込められていて電源がオフになっているシステムだけが、唯一安全と言えるシステムです。
3. **被害妄想でちょうどよい**
   コンピュータを保護するときは、疑い深く、用心深く、忍耐強く行動してください。保護とは、ずっと注意を払い続けていく継続的な作業です。プロセスとユーザーを確認し、普段と異なる事象に目を光らせます。

パスには現在のディレクトリを絶対に入れないでください。つまり `~/.bashrc` で `PATH=./:$PATH` のようなことをしないでください。これは重大なセキュリティリスクです。悪意のある人物が有害なプログラムを作成し、既存のプログラムと同名に置換している可能性があります。たとえば次の行だけを含む **ls** という名前のスクリプトを考えてみてください: `/bin/rm -rf $HOME` このファイルが含まれているディレクトリに移動して **ls** と入力するとホームディレクトリが消去されてしまします！

42.3 システムの更新とセキュリティ

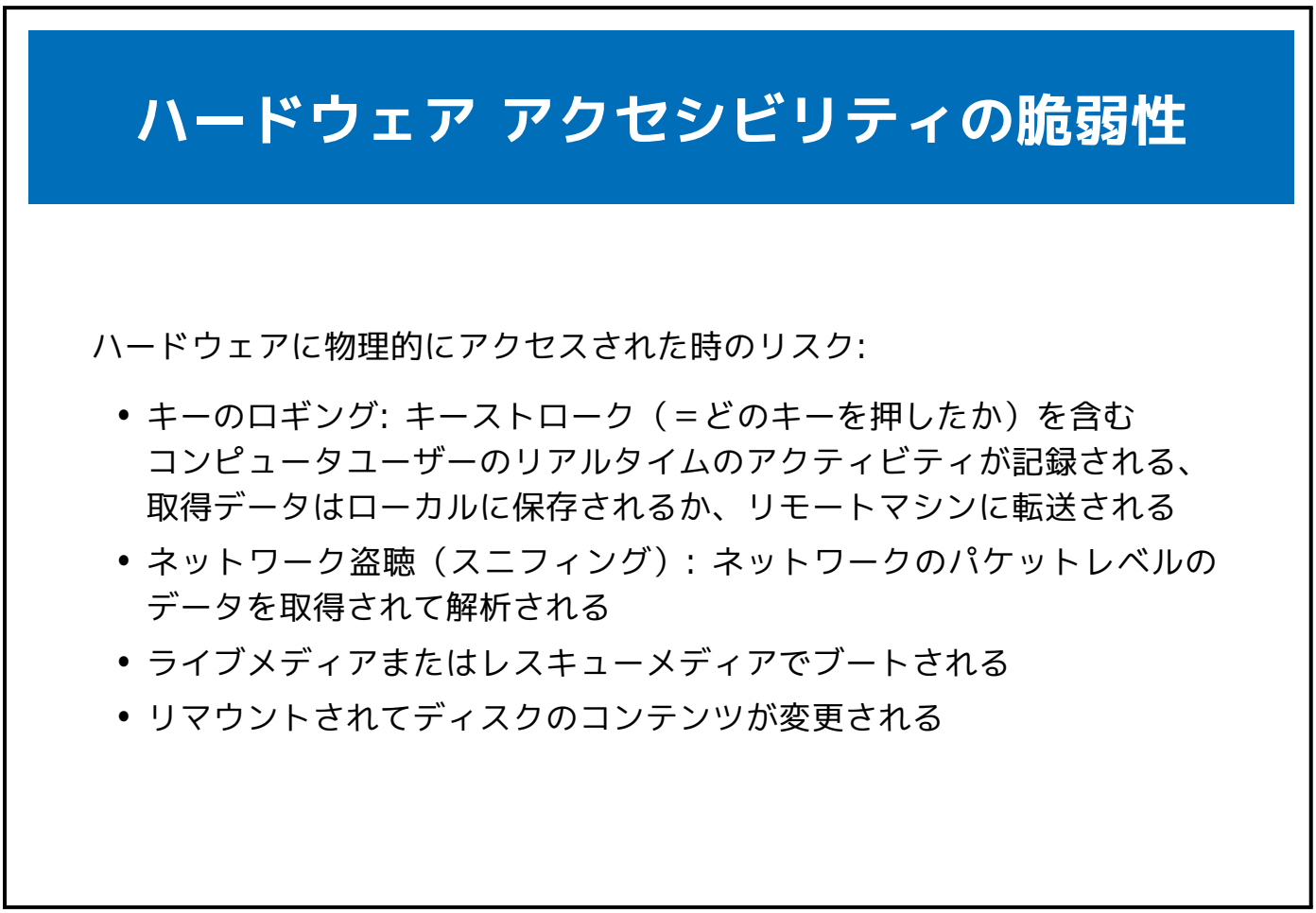


Linux ディストリビュータが提供する更新とアップグレードに注意を払い、できるだけ早く適用することが重要です。

ほとんどの攻撃は既知のセキュリティホールを悪用し、問題が明らかになってからパッチが適用されるまでの間に行われます。この **ゼロデイ攻撃** は実際には非常に珍しいです。通常攻撃者は、まだ発見されていないか、修正がリリースされていないセキュリティホールを使用します。

システム管理者は、修正よりも多くの問題を引き起こすプロプライエタリなオペレーティングシステムベンダーとの苦い経験から、リリース直後の修正を適用することに消極的です。ただし、**Linux** ではこのような **リグレッション（= デグレード）** は非常に稀なので、セキュリティパッチの適用を遅らせることで生じる危険性をおそらく正当化できません。

42.4 物理的なセキュリティ



システムへの物理的アクセスができてしまえば、攻撃者は複数の攻撃ベクトル（＝手段）を簡単に利用できてしまいます。
そうなると、オペレーティングシステムレベルの助言はすべて意味のないものになります。

このため、セキュリティポリシーはサーバーおよびワークステーションへの物理アクセスの保護の要件定義から始めるべきです。

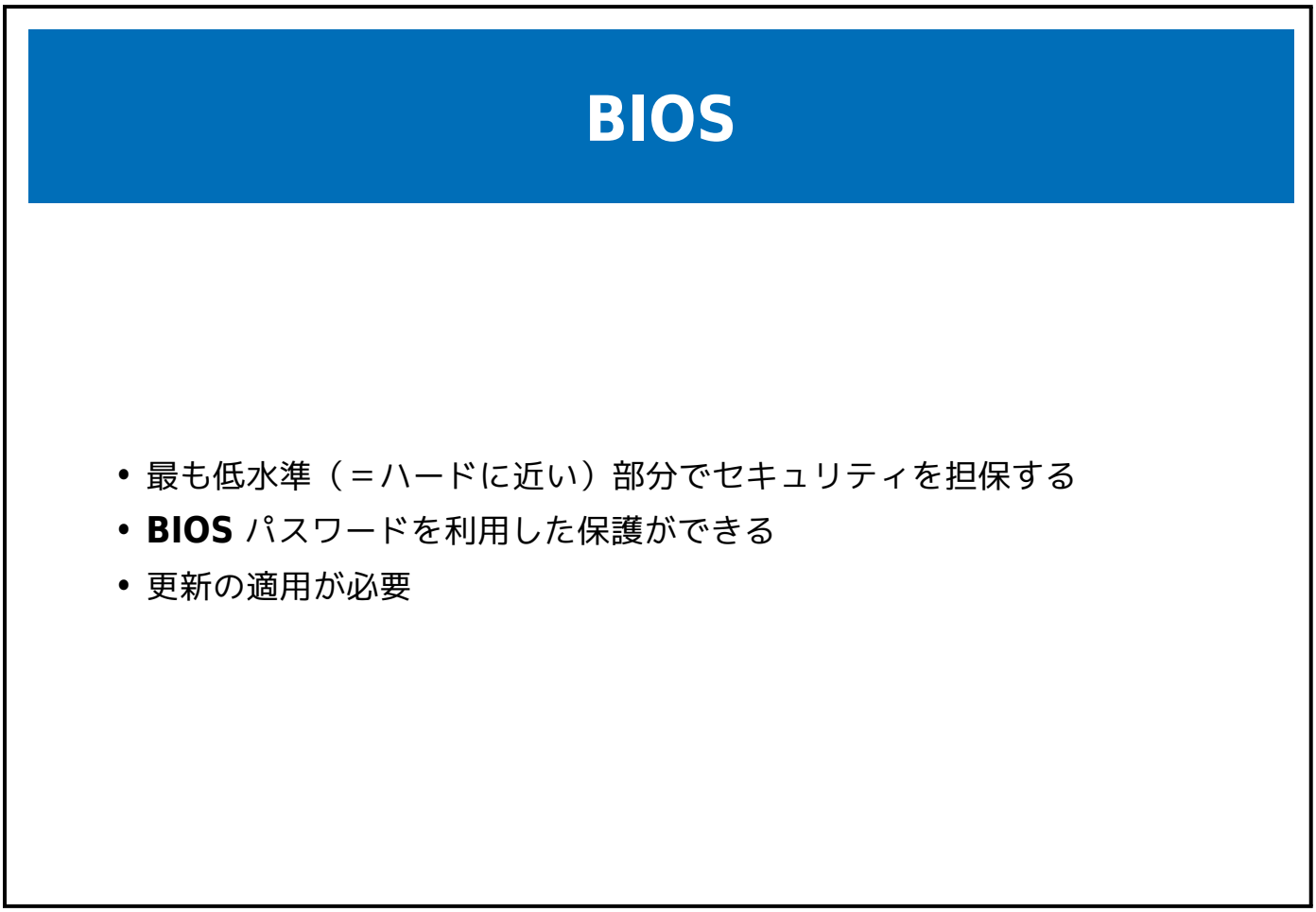
ハードウェア アクセスのガイドライン

必要な保護の手順は次の通りである:

- ワークステーションとサーバーを封鎖する（立ち入り禁止にする）
- 信頼できない人がネットワークにアクセス出来ないようにする
- パスワード入力に使うキーボードが（盗聴を仕込まれるなどの）改ざんを受けないようにキーボードを保護する
- ライブまたはレスキューの **CD/DVD** や **USB** キーでシステムを起動できないように **BIOS** をパスワードで保護

シングルユーザーコンピュータ、および家庭環境のコンピュータの場合で、上記の対応の一部（リムーバブルメディアからの起動禁止など）が過剰であるなら、それらを適用しないこともできます。ただし、機密情報がシステム上にあるなど慎重に保護する必要がある場合は、そこに機密情報を置かないようにするか、上記のガイドラインに従った保護を強化する必要があります。

42.5 BIOS

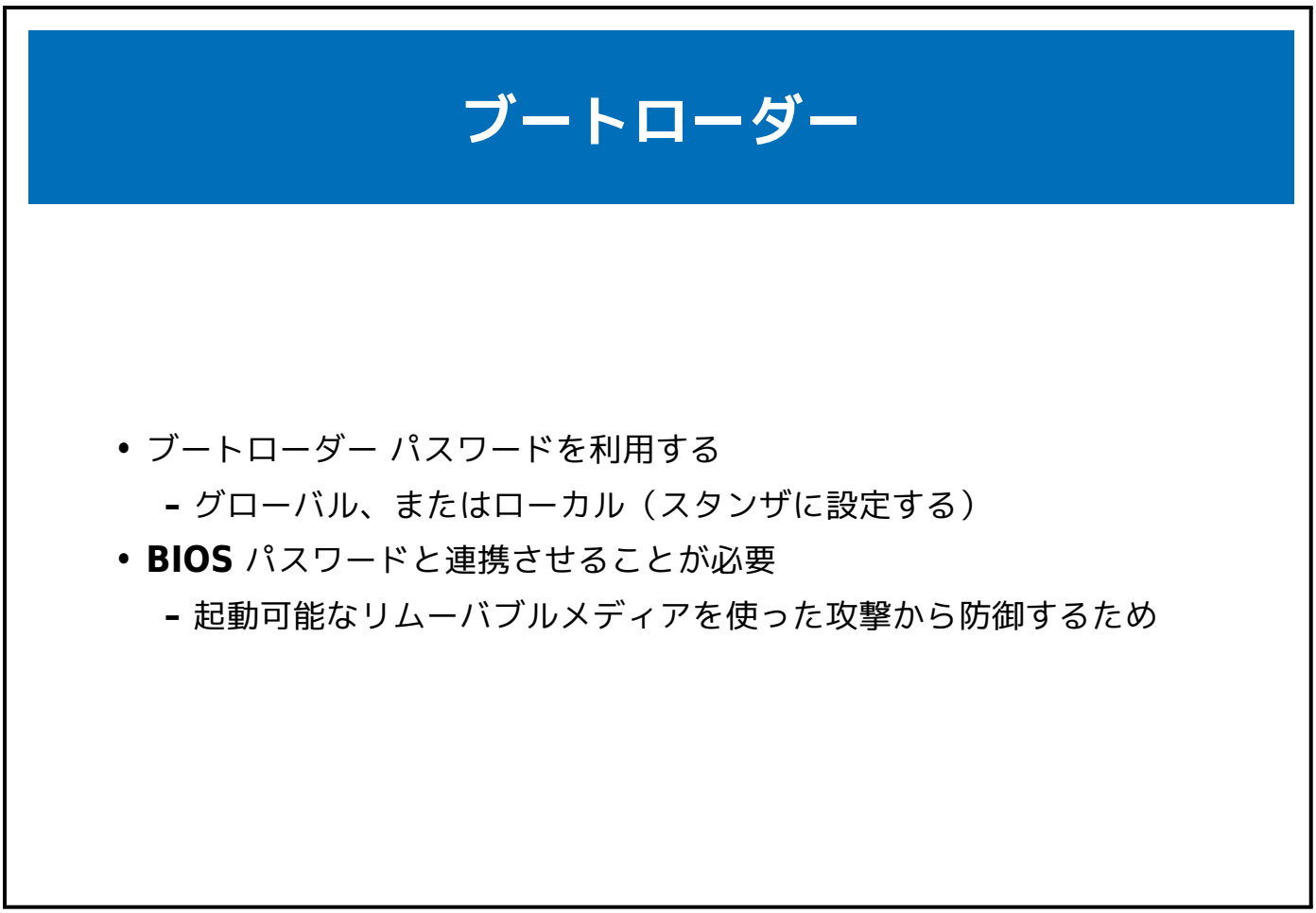


BIOS は、システムを構成または操作する最低レベル（＝最もハードウエアに近いという意味）のソフトウェアです。ブートローダーは **BIOS** にアクセスしてマシンを起動する方法を決定します。

BIOS パスワードを設定すると、権限のない人物が起動オプションを変更してシステムにアクセスすることを防ぐことができます。ただし、ここで問題になるのは誰かがマシンに直接ローカルにアクセスできてしまう可能性がある場合だけです。

また、一般的には **BIOS** に常にパッチを当ててファームウェアを最新バージョンにしておくことが推奨されます。ただし、ほとんどの **BIOS** アップデートはセキュリティとは関係がないもので、万一 **BIOS** コードにバグが含まれていると、本来は不要なアップデートによってシステムが起動できなくなってしまう可能性もあるので、システム管理者は新しい **BIOS** の適用は慎重に行うように教育されています。

42.6 ブートローダー



BIOS パスワードを設定すると、権限のない人物が起動オプションを変更してシステムにアクセスすることを防ぐことができます。ただしこれを実行するにはマシンを直接操作する必要があるので、問題になるのはマシンが物理的にアクセスされてしまう場合だけです。ブートローダーのパスワードだけを使用した場合、ユーザーがブートプロセス中にブートローダーの設定を編集することはできなくなりますが、ユーザが光ディスクやペンドライブなどの代替ブートメディアからブートすることを **妨ぐことはできません。** したがって、完全に保護するには **BIOS** パスワードと組み合わせて設定する必要があります。

以前の **GRUB 1** では **grub** パスワードの設定するのは比較的簡単でしたが、今主流の **GRUB 2** の場合は少し複雑になっています。ただし個々のユーザーに固有のパスワードの設定ができる（通常のログインパスワードでもよい）という柔軟性があります。

繰り返しになりますが `grub.cfg` を直接編集しないでください。代わりに `/etc/grub.d` のシステム構成ファイルを編集してから **update-grub** または **grub2-mkconfig** を実行して新しい構成ファイルを保存します。

この関連の説明のひとつが https://help.ubuntu.com/community/Grub2/Passwords にあります。

42.7 ファイルシステムのセキュリティ

安全なマウントオプションの使用

- nodev
  ファイルシステム上のキャラクタデバイスやブロックデバイスなどのスペシャルデバイスを利用できないようにする
- nosuid
  set-user-identifier ビットや **set-group-identifier** ビットを無効にする
- noexec
  マウントされたファイルシステム上のバイナリの直接実行を制限する
- ro
  ファイルシステムを **読み取り専用モード** でマウントする

ファイルシステムをマウントする場合 **mount** コマンドを使用してコマンドラインで、または `/etc/fstab` に設定して自動的にさまざまなオプションを指定してセキュリティを強化できます。

例えば ro オプションを使ってファイルシステムを読み取り専用モードでマウントします。

```
$ mount -o ro,noexec,nodev /dev/sda2 /edsel
```

`/etc/fstab` の中で

```
/dev/sda2 /edsel ext4 ro,noexec,nodev 0 0
```

42.8 setuid / setgid ビット

setuid / setgid ビット

- 実行ファイルに対して
  - プログラムを所有している（＝実行しているのではなく）オーナーまたはグループの権限でプロセスが実行される

```
$ chmod u+s somefile
$ chmod g+s somefile
```

- ディレクトリに対して
  - 共有ディレクトリの作成に利用される
  - **sgid** ディレクトリ内に作られたファイルはディレクトリのグループ所有者のグループ権限で実行される

```
$ chmod g+s somedir
```

- スクリプトで **setuid** ビットを変更しても何も起こらない（効果が無い）
  - **bash** 自体の **setuid** ビットを変更する必要があるが、これは非常に不味いやり方である

通常プログラムは、プログラムを実行するユーザーの権限で実行されます。これは実行中のバイナリファイルが誰の所有物であってもプロセスは依然として特権が制限されていることを意味します。

場合によっては、ネットワークインターフェイスの起動や停止や、スーパーユーザーが所有するファイルの編集など、通常は一般ユーザーに許可されないことができるようにしたい場面があります。

実行可能なファイルに **setuid** (**s**et **u**ser **ID**) ビットを設定すると、プログラムを **実行するユーザー** ではなくプログラムの **所有者** の権限でプログラムを実行するように動作を変更できます。

さらに **setgid** ビットを設定するとファイルを実行しているグループの特権ではなくファイルを所有しているグループの特権でプロセスが実行されます。

これは一般的には **悪い考え** であり、ほとんどの状況では使用すべきでないと強調しておきます。この種の設定が必要な場合には、多くの場合より権限をより限定した **デーモンプログラム** を作成する方が適切です。一部の **Linux** ディストリビューションではこの設定機能が完全に無効になっています。

デフォルトではファイルがディレクトリに作成されると、そのファイルはそのファイルを作成したユーザーとユーザーのグループに所有されます。ディレクトリで **setgid** 設定を使用すると、ディレクトリで作成されたファイルがそのディレクトリのグループ所有者によって所有されるように変更されます。これによって同じグループに所属するユーザーがファイルを共有できる共有ディレクトリを作成できます。

42.9 演習

課題 42.1: セキュリティとマウントのオプション

ファイルシステムにあるプログラムを実行させないため noexec オプションをつけて、パーティションまたはループバックデバイスをマウントします。既存のパーティションを使うこともできますが、パーティションをマウントしているときは、その動作を変更することができません。そのためここでは影響を限定的にするためにループバックデバイスを用います。

1. 空のファイルを用意し、ファイルシステムを格納してマウントします。
2. 何か実行可能なファイルをコピーしてきます。そのファイルが新しい場所で実行できるか確認します。
3. アンマウントし noexec オプションをつけて再マウントします。
4. 実行可能かテストします。noexec マウントオプションを付けているので、エラーが出るはずです。
5. 後片付けをします。

解 42.1

1. ```
   $ dd if=/dev/zero of=image bs=1M count=100
   $ sudo mkfs.ext3 image
   $ mkdir mountpoint
   $ sudo mount -o loop image mountpoint
   ```

2. ```
   $ sudo cp /bin/ls mountpoint
   $ mountpoint/ls
   ```

3. ```
   $ sudo umount mountpoint
   $ sudo mount -o noexec,loop image mountpoint
   ```

   or

   ```
   $ sudo mount -o noexec,remount image mountpoint
   ```

4. ```
   $ mountpoint/ls
   ```

5. ```
   $ sudo umount mountpoint
   $ rm image
   $ rmdir mountpoint
   ```

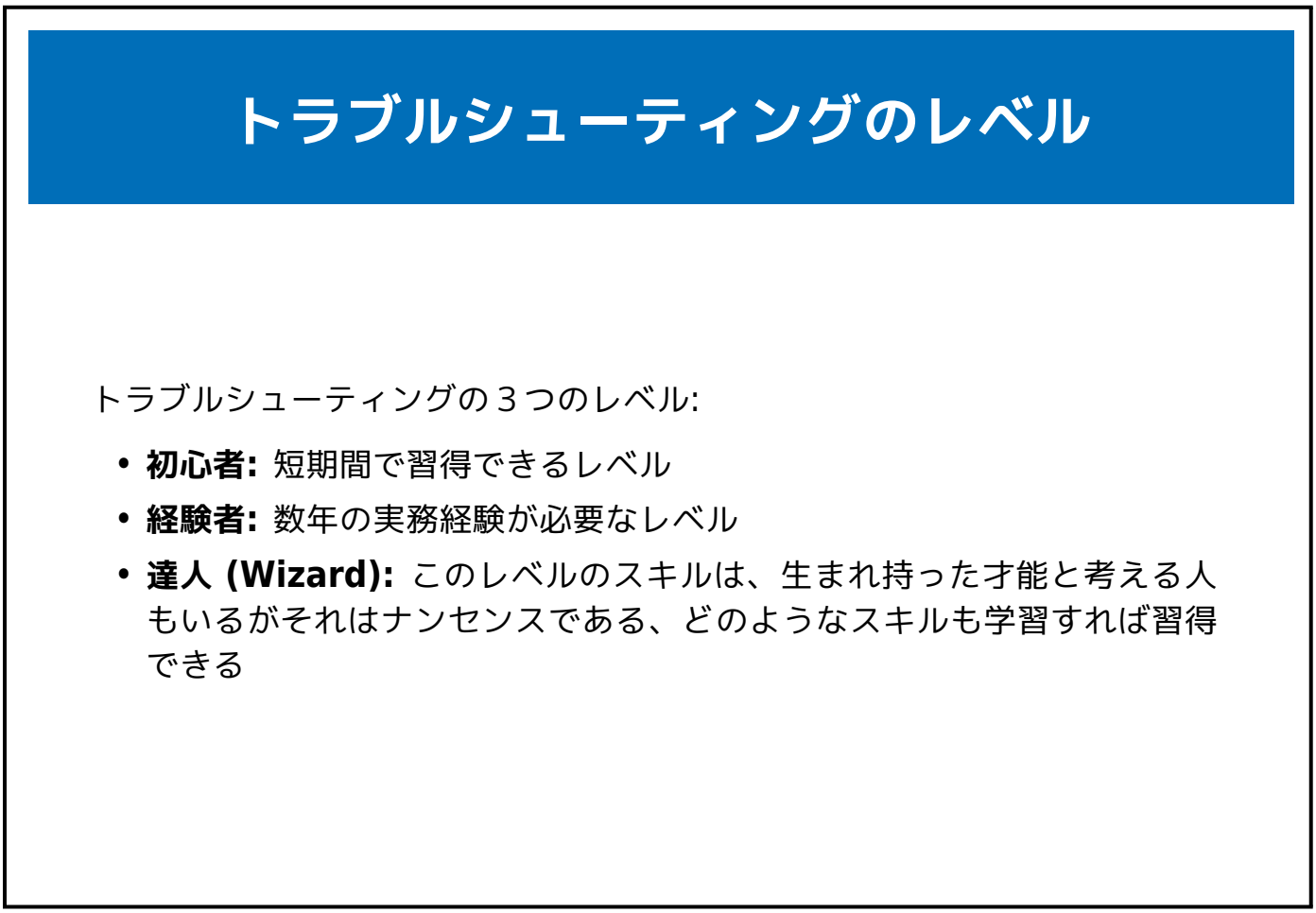
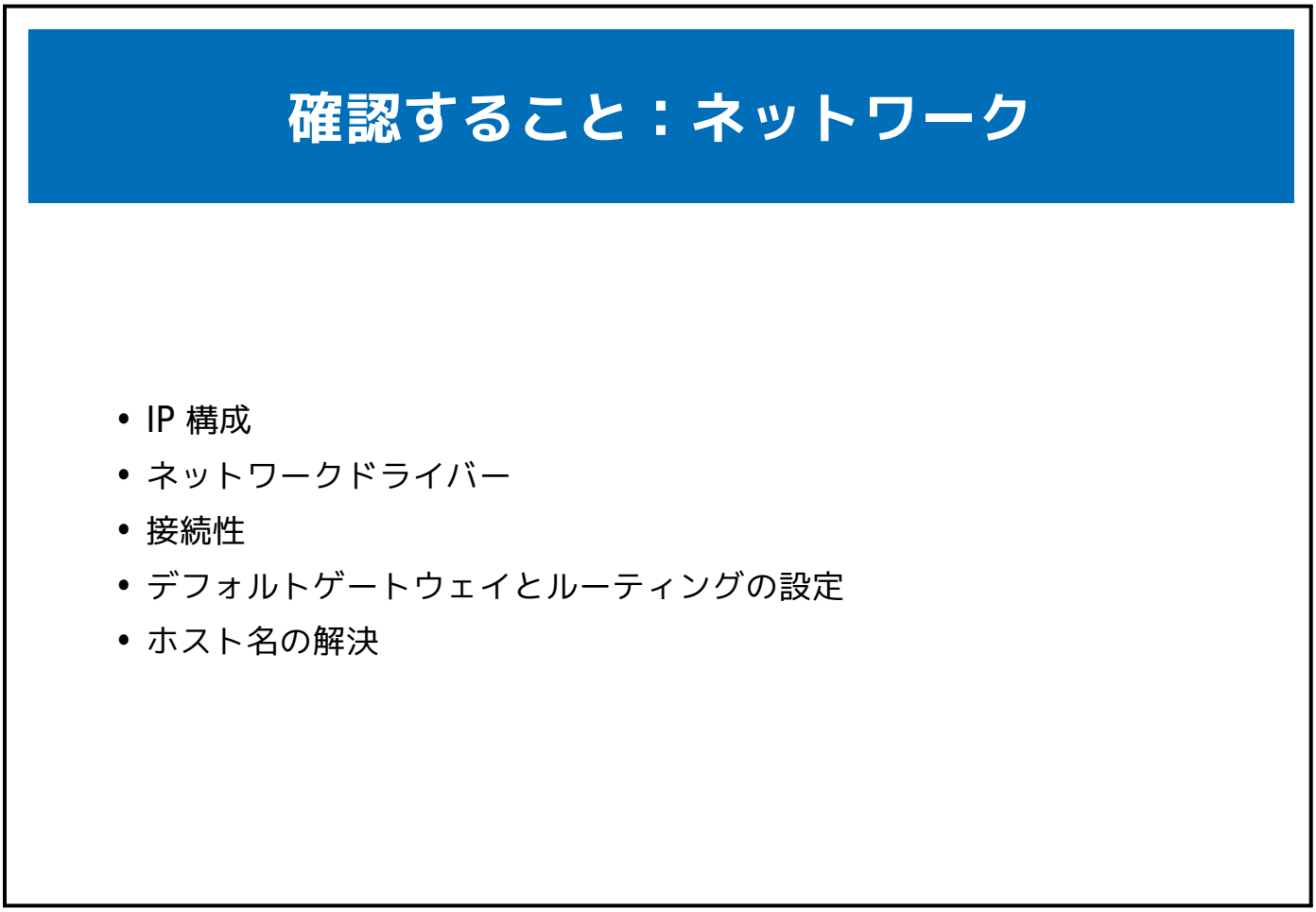
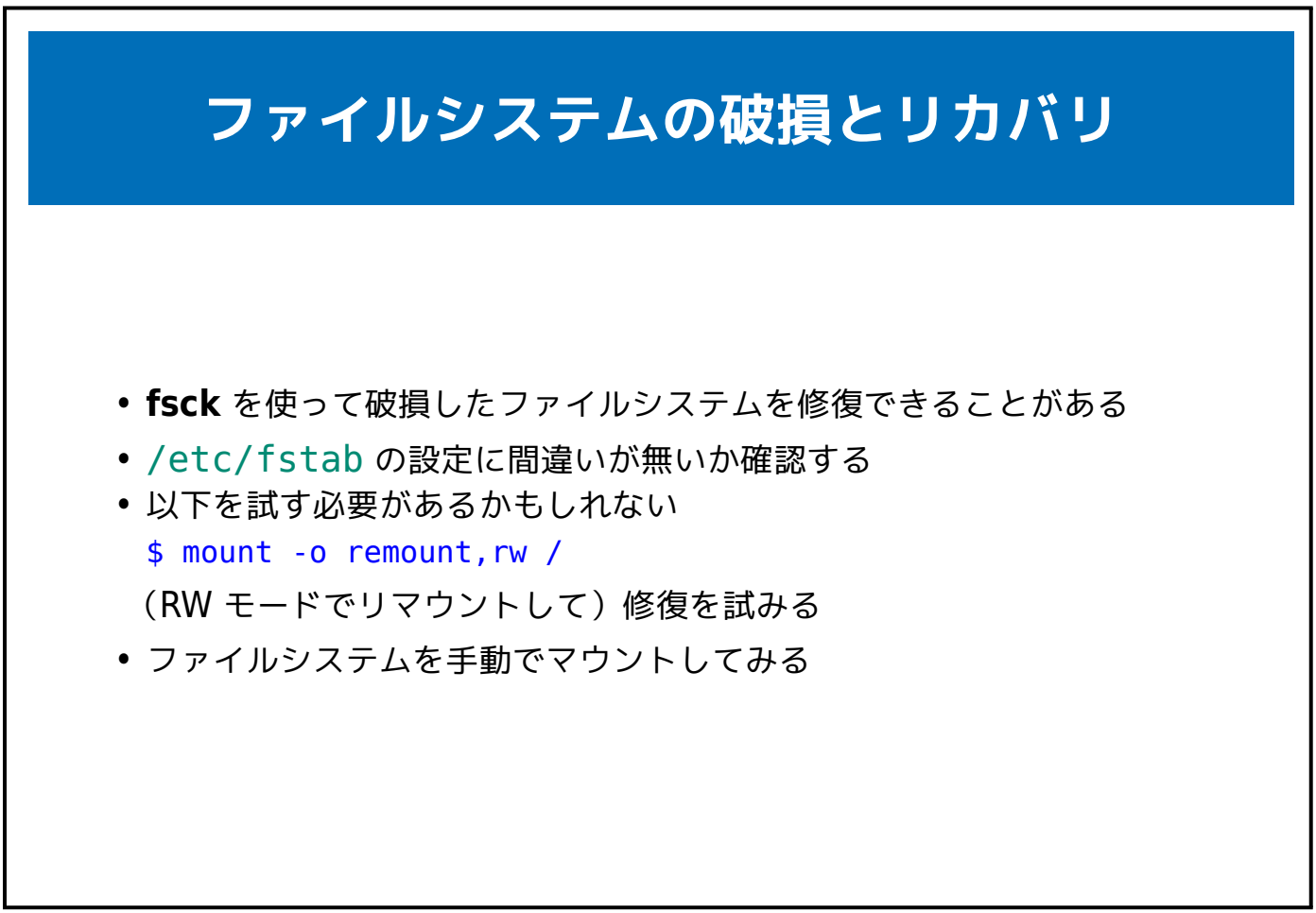
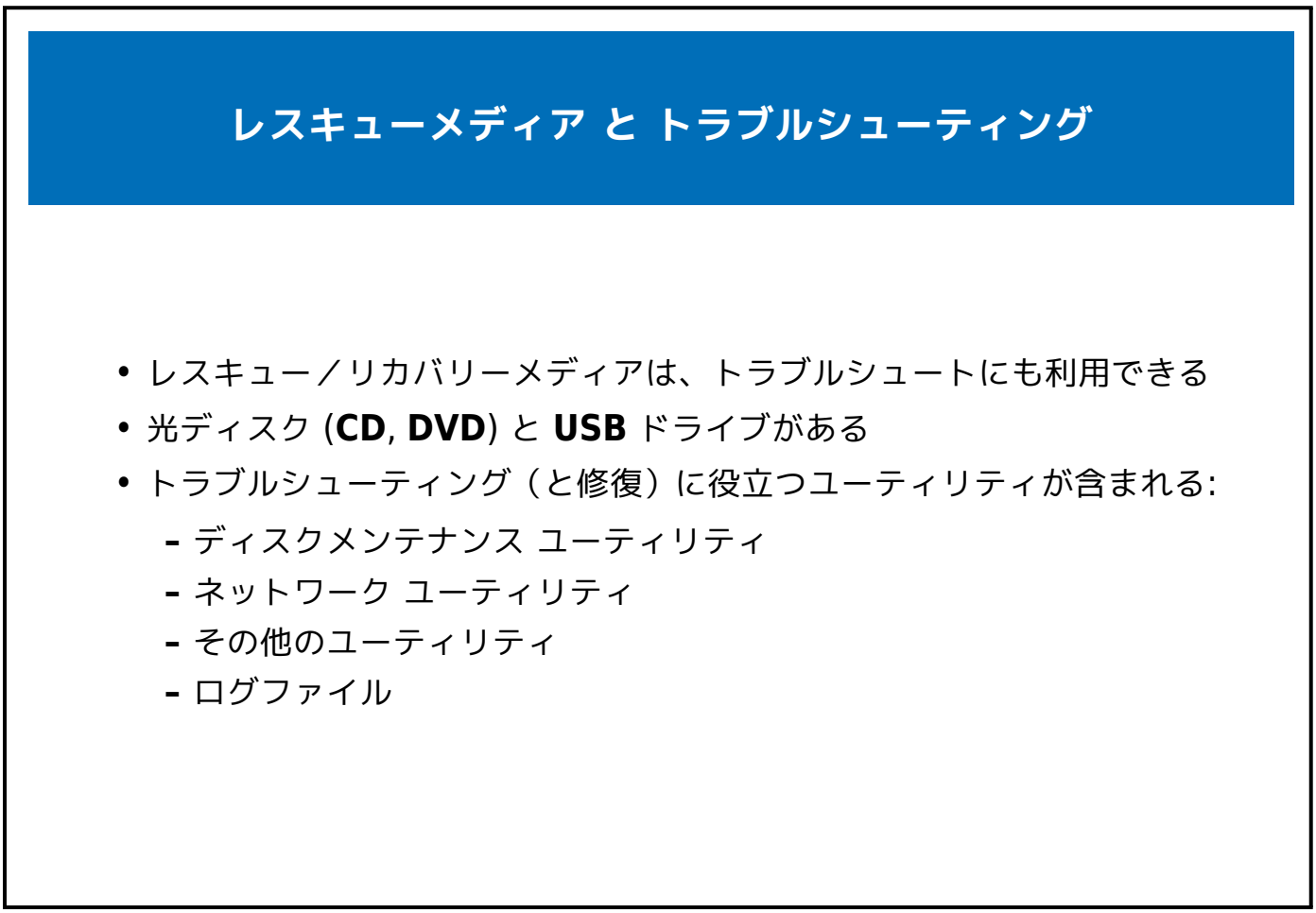
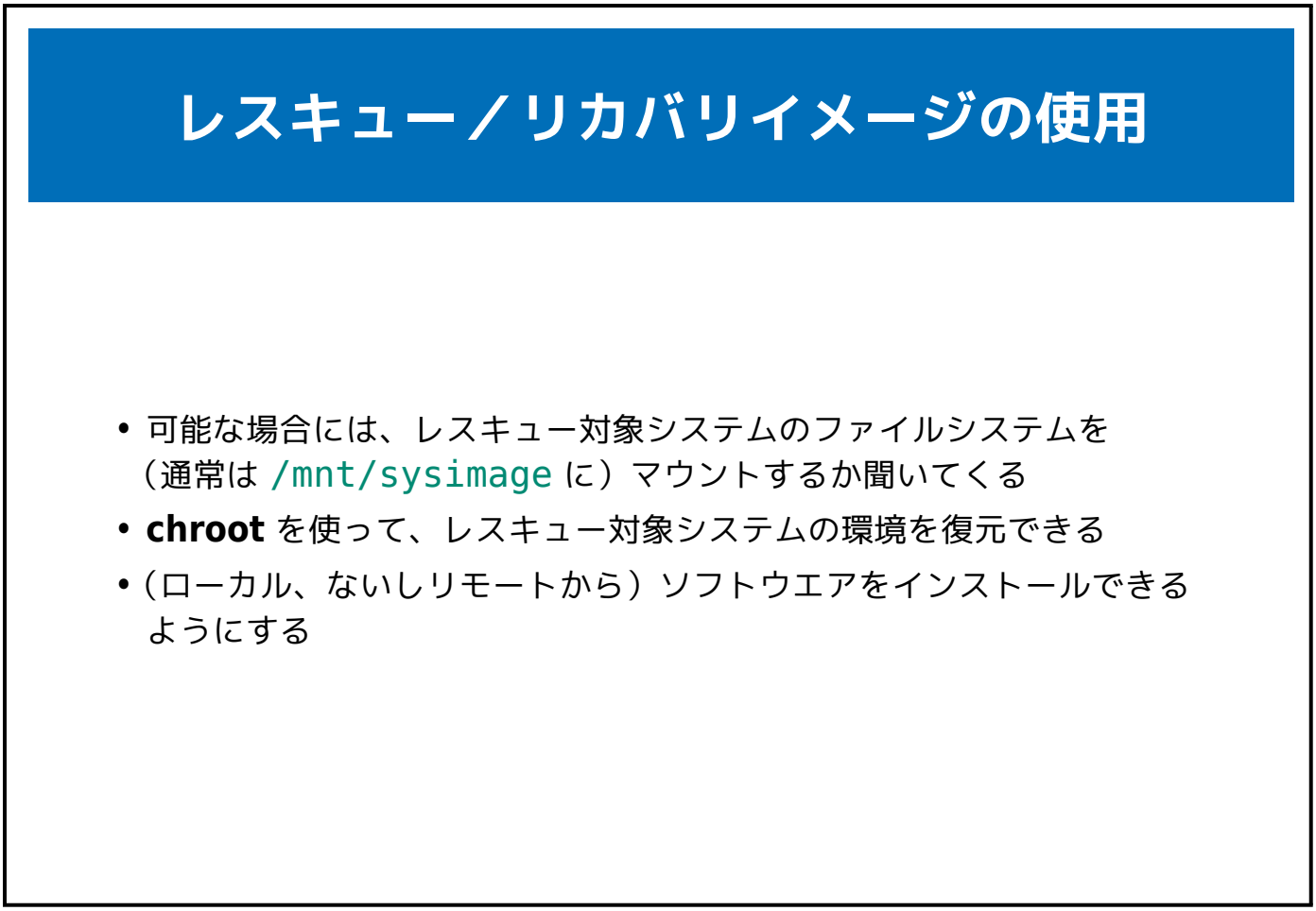
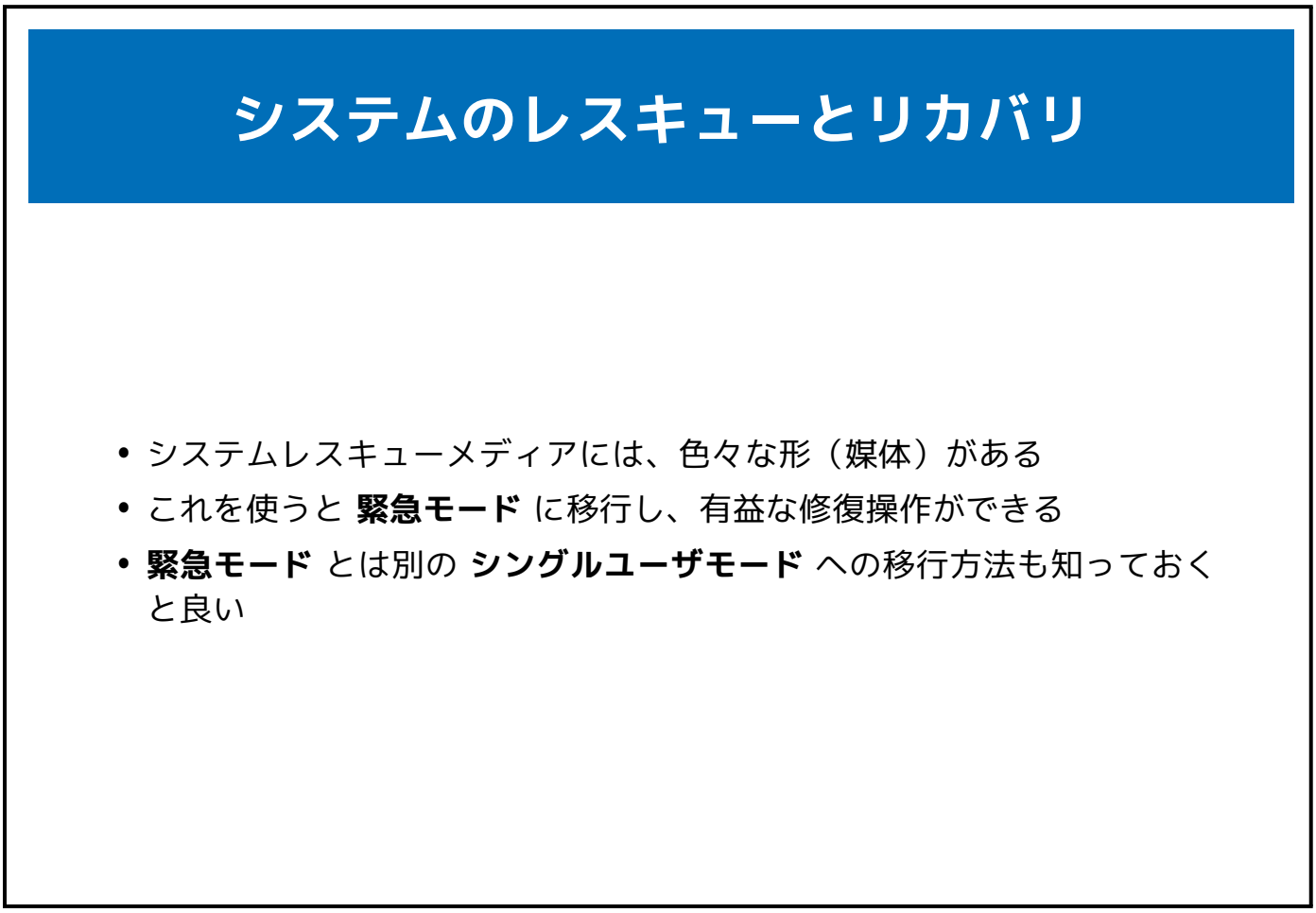
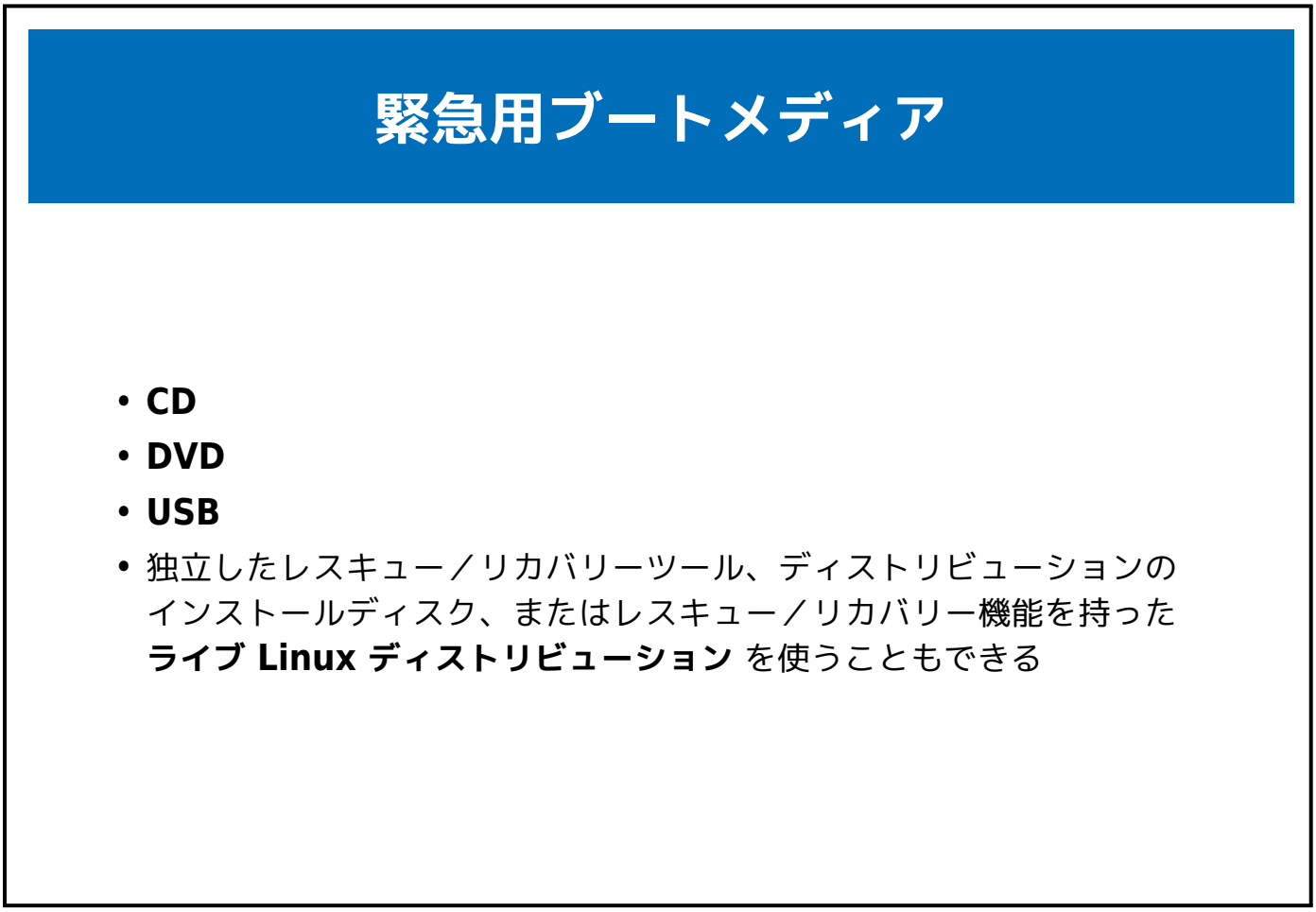
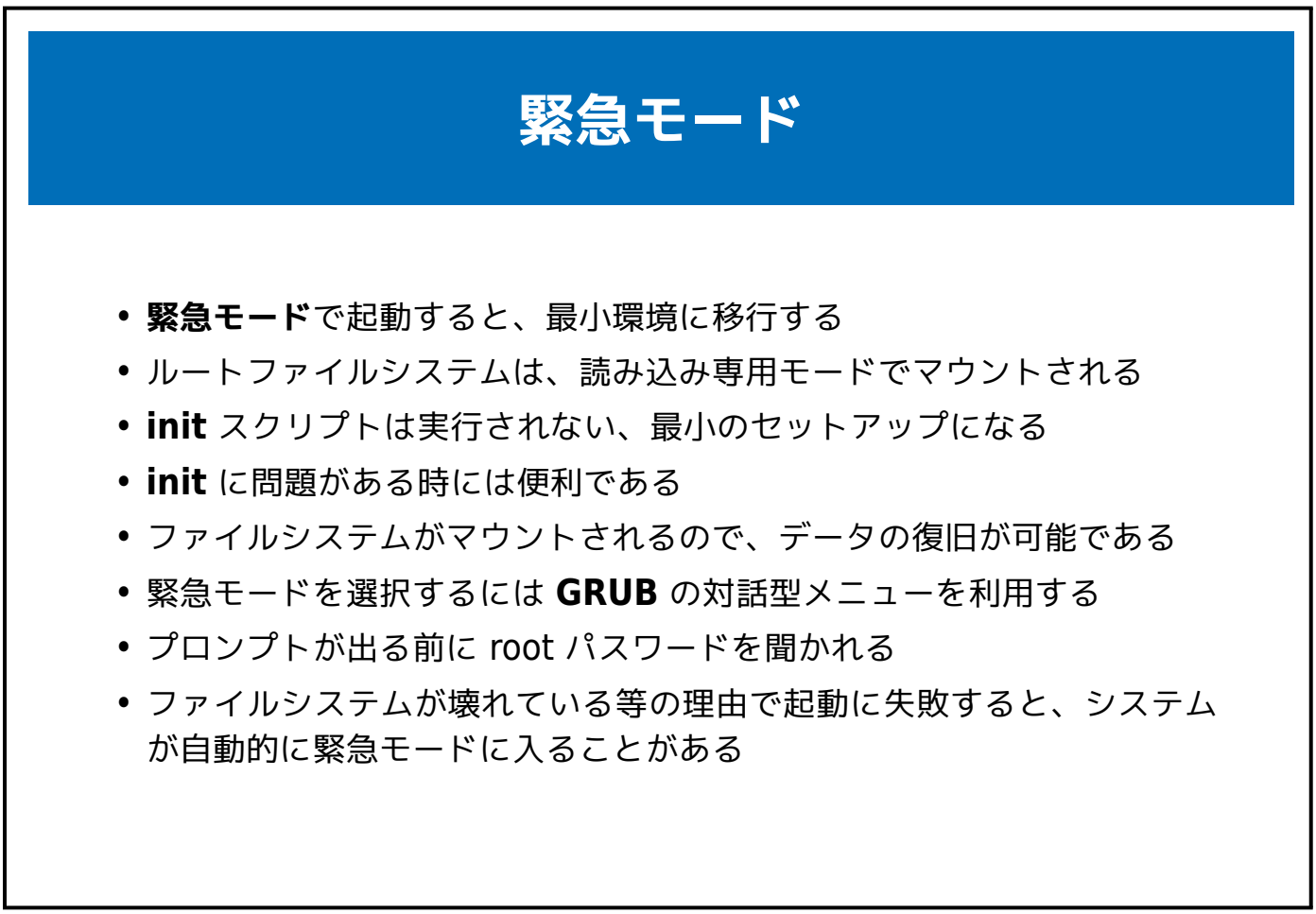
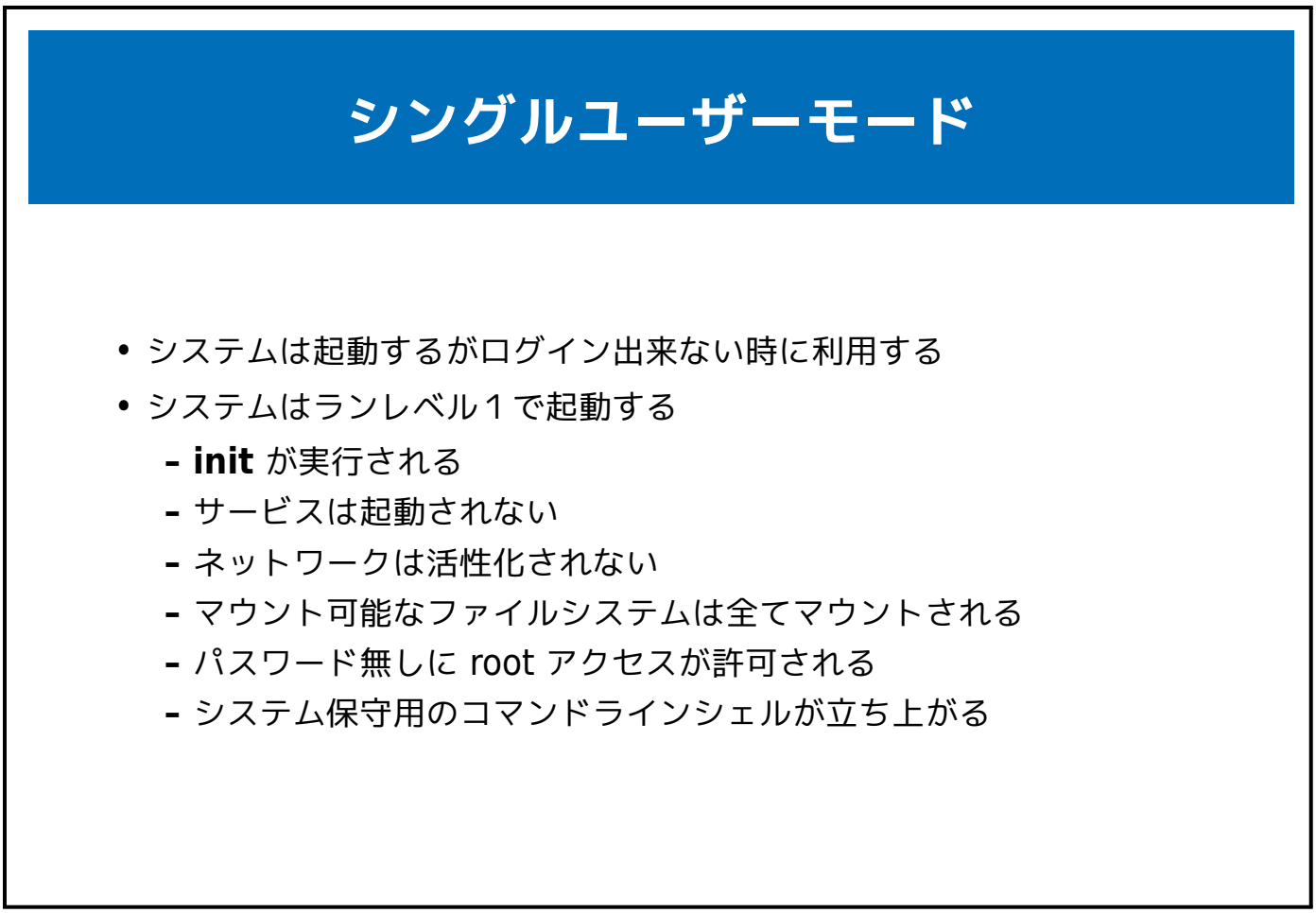
これは恒久的な対応ではありません。恒久的にしたい場合は /etc/fstab に次の行を追加します:

/etc/fstab の追加内容

```
/home/student/image  /home/student/mountpoint    ext3    loop,rw,noexec 0 0
```

課題 42.2: setuid に関する追記とスクリプト

現在のディレクトリにある afile ファイルに書き込む **C** プログラム(./writeit.c)があります。このプログラムは SOLUTIONS ファイルから writeit.c として取り出すことができます。
```

writeit.c

```
/*
@*/
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
#include <sys/stat.h>

int main(int argc, char *argv[])
{
        int fd, rc;
        char *buffer = "TESTING A WRITE";
        fd = open("./afile", O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
        rc = write(fd, buffer, strlen(buffer));
        printf("wrote %d bytes\n", rc);
        close(fd);
        exit(EXIT_SUCCESS);
}
```

writeit.c をコンパイルします:

```
$ make writeit
```

または、以下のようにしても同じことができます。

```
$ gcc -o writeit writeit.c
```

root が所有者のファイルに対して、（一般のユーザーの権限で）このプログラムを実行すると以下のようにエラーになります:

```
$ sudo touch afile
$ ./writeit
wrote -1 bytes
```

しかし root で実行すると:

```
$ sudo ./writeit
wrote 15 bytes
```

つまり root ユーザーは自分が所有するファイルには書き込めますが、一般のユーザーはできません。

writeit の所有者を root にしても意味がありません：

```
$ sudo chown root.root writeit
$ ./writeit
wrote -1 bytes
```

まだ afile を変更する権限がないからです。

setuid ビットをセットすることで、一般のユーザーにその権限を追加できます:

```
$ sudo chmod +s writeit
$ ./writeit
wrote 15 bytes
```

注目

何故この操作を行うスクリプトを書くのではなく、プログラムを書いて実行可能にコンパイルするのでしょうか？

Linux ではスクリプトの **setuid** ビットを変更しても、シェル（たとえば、**bash**）の **setuid** ビットを変更しなければ、何も変わりません。しかしこの変更は大きな間違いです; シェルの **setuid** ビットを変更すると、これ以降実行されるプログラムが全て特権を持つことになるからです。

章 43

トラブルシューティングの基本

43.1 トラブルシューティングのレベル



最高水準で管理されている **Linux** システムでも、問題は発生します。

トラブルシューティング によって、問題がソフトウェアまたはハードウェアに起因するのか、システムローカルなのか、ローカルネットワークまたはインターネット内から発生するのかを特定することができます。

トラブルシューティングを適切に行うには、判断力と経験が必要です。この作業にはある程度芸術的な要素も含まれるのですが、それでも適切に系統だった手順に従うことで、将来再利用が可能な問題の原因切り分け手法の確立につながります。

43.2 トラブルシューティングテクニック

基本的なトラブルシューティングテクニック

- 問題を特定する
- 問題を再現させる
- いつも、簡単にできることから試す
- 原因となる可能性のなる要因を、一つずつ排除していく
- 一度に変更するのは１つだけとする、もしそれで解決しなければその変更は元に戻す
- 詳細な情報を取得するためにシステムログを確認する (`/var/log/messages` 、`/var/log/secure` など)

（トラブルシュートの進め方の）ルール策定の方針と解析手段は、十分に確立された手順に従うことが求められます、直感に基づいて結論に飛びつくことはお勧めできません。チェックリストと統一された手順を使用する動機は、ウィザード（達人）への依存を回避し、良く知られた手順に忠実に従うだけで最終的にシステム管理者が問題解決できるようにするためです。そうしなければ、もしウィザードが組織を離れてしまった時に、難しい問題の解決に必要なスキルを持った人が誰もいなくなります。

一方、直感をベースに解析を進める時は、生産性を基準にどこまで直感的な方法を続けるかを判定できるデータを十分な速さで取得できることを担保するべきです。

直感を無視すると問題の解決に時間がかかる場合もあるので、トラブルシューター（＝解析者）がこれまで直感に頼る方法でどれだけ問題を解決できたかの実績に基づいて、この方法にリソースを投入するか決めるべきでしょう。別の言い方をすると「的を得た直感」は魔法などではなく経験値の積み重ねだという事です。

43.3 ネットワーク



ネットワークの問題は、ソフトウェア起因またはハードウェア起因のいずれの可能性もあり、デバイスドライバの読み込みや、ネットワークケーブル接続など簡単な原因もありえます。ネットワークがアップ（＝活性化）しているけれどもパフォーマンスが悪い場合はトラブルシューティングではなくパフォーマンス チューニングの対象になります。問題の原因がマシン以外に起因する場合や、バッファサイズなどのさまざまなネットワーク パラメータの調整が必要な場合があります。

ネットワークに問題がある時には、以下の項目を確認する必要があります:

- **IP 構成:**
  ifconfig または **ip** を使用して、インターフェイスがアップ（活性化）しているか、アップしているなら設定されているかを確認します。
- **ネットワークドライバー:**
  インターフェイスを起動できない時には、ネットワークカードに正しいデバイスドライバがロードされていない可能性があります。**lsmod** でネットワークドライバがカーネルモジュールとしてロードされているかを確認します。もしくは `/proc` や `/sys` など、関連する疑似ファイルで `/proc/interrupts` や `/sys/class/net` などの値を確認します。
- **接続性:**
  ping を使用して対象のネットワークに到達可能かどうかを確認し、応答時間とパケット損失があるかを確認します。**traceroute** を使うとネットワーク内でパケットが通過した経路を追跡できますが、**mtr** を使えばこれを連続的に実施できます。これらのユーティリティを使用すると問題がローカル側にあるのかインターネット側にあるのかの切り分けが可能です。
- **デフォルトゲートウェイとルーティング構成:**
  route -n を実行してルーティングテーブルが適切か確認します。
- **ホスト名の解決:**
  対象 URL に対して **dig** または **host** を実行し **DNS** が正しく機能しているかどうかを確認します。

43.4 確認すること：ファイルの整合性

確認すること：ファイルの整合性

- 構成ファイルやバイナリーファイルの破損をチェックする
- 色々な方法が利用できる
  - **rpm**
    `$ rpm -V some_package`
    一つのパッケージのを検査する
    `$ rpm -Va`
    全てのパッケージを検査する
  - **aide**: 侵入検出
    `$ aide --check`
  - **debsums**: **Debian**
    `$ debsums options some_package`

破損した構成ファイルとバイナリをチェックする方法はいくつかあります。一つの方法は `rpm -V` コマンドを使って、特定のパッケージを確認する方法です。また `rpm -Va` を使えば全てのパッケージの整合性を確認することができます。

aide は、ファイルの変更をチェックするもう一つの方法です。`aide --check` コマンドは、全てのファイルをスキャンして前回のスキャン結果と比較します。

Debian では、整合性チェックの方法は **debsums** だけです。`debsums somepackage` はそのパッケージ内のファイルのチェクサムを確認します。しかし、全てのパッケージのチェックサムは保存されていないので役に立つとは限らないでしょう。

43.5 ブートプロセスの失敗

システムが適切にまたは完全に起動しない場合、問題の原因を特定するには各段階で何が起きているかを理解することが重要です。**BIOS** ステージまで進んだなら、以下の何れかの状態でしょう:

ブートローダーの画面がでない:
GRUB の設定ミスかブートセクターが壊れていないか確認します。ブートローダーの再インストールが必要かもしれません。

カーネルのロードに失敗する:
ブートプロセス中に **カーネルパニック** が発生する場合は、おそらくカーネルの構成ミスかカーネルイメージの破損、あるいはカーネルコマンドラインで指定された **GRUB** 構成ファイルのパラメータが誤っていると想定されます。過去にそのカーネルが正常に起動していたのであれば、カーネルが破損したかカーネルに間違ったパラメータが渡されています。どちらかであるかに応じてカーネルを再インストールするか、ブート時にインタラクティブな **GRUB** メニューに入り、最小限のコマンドラインパラメータを使用して修正を試みることができます。またはレスキューイメージで起動することもできます。

カーネルはロードできるが、ルートファイルシステムのマウントに失敗する:
主な原因は次のとおりです。

1. **GRUB** 構成ファイルの記述に間違いがあります。
2. `/etc/fstab` の記述に間違いがあります。
3. カーネルに組み込まれたルートファイルシステムの種類、もしくは **initramfs** イメージがサポートされていません。

init プロセス実行中の失敗:
init が開始されるとさまざまな要因で障害が起こる可能性があります。停止する直前に表示されたメッセージをよく見てください。以前には別のカーネルでは問題がなかったとすれば、これは大きな手掛かりとなります。ファイルシステムの破損、起動スクリプトのエラーなどに注意してください。3（グラフィックなし）や1（シングルユーザーモード）などの低いランレベルでブートしてみてください。

43.6 ファイルシステムの破損とリカバリ



ブートプロセス中に、１つ以上のファイルシステムのマウントに失敗した場合 **fsck** を使用して修復を試みることができます。ただし、それを行う前に `/etc/fstab` の記述が誤っていたり破損していないことを確認する必要があります。繰り返しますが、実行中のカーネルがファイルシステムの種類に対応していない場合は、それが（マウントに失敗した）原因である可能性があります。

ルートファイルシステムがマウントできたなら、ファイルを調べることができます。ただし `/` は読み取り専用としてマウントされている可能性があります。ファイルを編集して修正するには次のコマンドを実行します:

```
$ sudo mount -o remount,rw /
```

書き込み権限を付けてリマウントする

`/etc/fstab` が正しいと思われる場合は **fsck** に進むことができます。まず、次を試してください:

```
$ sudo mount -a
```

これはすべてのファイルシステムをマウントしようとします。これが完全に成功しなかった場合には、問題があったものを手動でマウントしてみてください。最初に **fsck** を調査のために実行すべきです。その後、見つかったエラーを修正して再度実行します。

43.7 仮想コンソール

仮想コンソールの使用

- **Linux** にはデフォルトで 12 個の **仮想コンソール**（**仮想端末** とも呼ばれる）がある
  - 最初の６つはログインテキストコンソールである
  - ほとんどのディストリビューションが、コンソール１をシステムコンソールとして使用する
  - 通常コンソール７はグラフィカルコンソール、ただし（**RHEL** を含む）一部のディストリビューションは、コンソール１を使用する
- `Ctrl-Alt-FX` を使ってコンソールを切り替える
  - 例: `Ctrl-Alt-F5` はコンソール５に切り替える

デフォルトでは、**Linux** はシステムへのローカルアクセスが可能な 12 個の仮想コンソール（仮想端末とも呼ばれる）を定義しています。通常、最初の６つはログインテキストコンソールです。７番目（がある場合には、それは）はグラフィカルコンソールです。

`Ctrl-Alt-FX`（X はコンソールの番号）を使用してコンソール間を移動できます。

43.8 演習

注目

本章には、演習はありません。システムの構成と監視について、いままでの章で説明した内容のまとめです。そして、この後にある演習を含むシステム復旧のための、追加内容を含んでいます。

章 44

システムの救出（レスキュー）

44.1 レスキューメディア と トラブルシューティング



システムのレスキュー（救済）とリカバリー（修復）について説明しますが、ここではレスキューメディアを使ったトラブルシュートについて述べます。

何を選択するかは **Linux** ディストリビューションによって異なりますが、インストールメディアまたはライブ **CD/DVD** ないし **USB** ドライブから起動する時に `Rescue Installed System` などの名前がついた起動オプションの選択が可能でしょう。

レスキューディスクには、便利なプログラムが多く含まれています:

- パーティション作成、**RAID** デバイス管理、論理ボリューム管理、ファイルシステム作成のためのディスクユーティリティ:
  fdisk、**mdadm**、**pvcreate**、**vgcreate**、**lvcreate**、**mkfs**、これ以外にも多数あります。
- ネットワークのデバッグとネットワーク接続のためのネットワークユーティリティ:
  ifconfig、**route**、**traceroute**、**mtr**、**host**、**ftp**、**scp**、**ssh**
- これ以外にも、利用可能なユーティリティが多数あります:
  bash、**chroot**、**ps**、**kill**、**vi**、**dd**、**tar**、**cpio**、**gzip**、**rpm**、**mkdir**、**ls**、**cp**、**mv**、**rm** など

44.2 レスキュー／リカバリイメージの使用



レスキューイメージを起動すると、いくつか質問をしてきます。その１つは（それが可能な場合に）ファイルシステムをマウントするかどうかです。

マウントを指定した場合どこかに（通常は /mnt/sysimage に）マウントされます。そのディレクトリに移動すればファイルにアクセスできます。もしくは次のコマンドでその環境に（＝リカバリー対象の元々のディレクトリーレイアウトに）変更できます。

```
$ sudo chroot /mnt/sysimage
```

ネットワークベースのレスキューの場合 /mnt/source をマウントするよう求められる場合もあります。

chroot した環境の中からソフトウェアパッケージをインストールすることができます。または chroot した環境の外部からインストールすることもできます。たとえば **RPM** ベースのシステムでは --root オプションを使用してルートディレクトリの場所を指定できます:

```
$ sudo rpm -ivh --force --root=/mnt/sysimage /mnt/source/Packages/vsftpd-2*.rpm
```

44.3 システムのレスキューとリカバリ



システムは、遅かれ早かれ適切にブートできないとか、ファイルシステムのマウントに問題がでるとか、デスクトップ表示の開始に問題があるなどといった重大な障害に遭遇してしまう可能性があります。光ディスクまたはポータブル **USB** ドライブ形式の **システムレスキューメディア** を使用してこれらの症状を解決できます。**緊急モード** または **シングルユーザーモード** でブートすると一連の **Linux** ツールを使用してシステムを通常の機能に修復できます。

44.4 緊急用ブートメディア



緊急用ブートメディアは、ファイルの欠落、構成の誤り、ファイルの破損、サービスの構成の誤りなどが原因でシステムがブートしない時に役立ちます。

何らかの理由で root パスワードが失われたり、変に改ざんされたりしてリセットする必要がある場合にも、レスキューメディアが役立つことがあります。

ほとんどの **Linux** ディストリビューションのインストール用のメディア（**CD**, **DVD**, **USB**）や **ライブメディア** は、レスキューディスクとしても利用できるようになっています。これは非常に便利です。専用のレスキューディスクも利用できます。

（任意の形式の）ライブメディアは、メモリー内だけで完結した、完全で起動可能なオペレーティングシステムを提供します。ディスクからはロードされません。ユーザーは、実際に（ディスクに）インストールしたり、コンピュータ上の既存のオペレーティングシステムに変更を加えたりすることなく、オペレーティングシステムや **Linux** ディストリビューションを体験および評価できます。

ライブリムーバブルメディアは、ハードディスクドライブなどのセカンダリストレージが欠けているコンピュータや、またはハードディスクドライブやファイルシステムが破損しているコンピュータでも実行できるという優れた点があり（ディスクがオフラインとなっている間に）ユーザーはデータをレスキューできます。

44.5 レスキューメディアの使用

レスキューメディアの使用

- 事実上全ての **Linux** ディストリビューションの **GRUB** のメニューにはレスキューイメージの選択がある。もしあなたのシステムにこの選択肢がありレスキューオプションが働くなら、レスキューメディアをマウントする前にこれを試すべきである。
- システム起動時にメニュー選択から `rescue` を選ぶか、または `boot:` プロンプトに以下のようにタイプする必要がある:

  ```
  Linux rescue
  ```

- 言語の選択やその他（ネットワークを初期化するかなど）の質問をうけるかもしれない
- また、おそらくどのファイルシステムをマウントするかなども質問されるだろう

ライブ、インストール、レスキューのいずれのメディアを使用している場合でも、レスキューとリカバリのために特別なオペレーティングシステムを開始する手順は同じで、紹介したように１つのメディアが３つの目的すべてを果たします。

システムはリムーバブルメディアからブートすると、ブートメニューのオプションからレスキュー／リカバリモードにアクセスできます。多くの場合次のように `rescue` と入力する必要があります:

```
boot: Linux rescue
```

各ディストリビューションには多少の違いがありますが、手順を確認するのは簡単ですのですべての手順を説明しません。

次に使用する言語などのいくつかの質問と、いくつかのディストリビューションに依存する選択があります。そして、正しいレスキューイメージの場所を選択するように求められます: **CD/DVD**、**ハードディスク**、**NFS**、**FTP**、**HTTP** など。

選択した場所には、正しいインストールツリーが含まれている必要があり、インストールツリーはレスキューディスクと同じ **Linux** バージョンのものである必要があります。リムーバブルメディアを使用している場合、インストールツリーはメディアの作成元と同じものでなければなりません。ベンダーからダウンロードした `boot.iso` イメージを使用している場合は、ネットワークベースのインストールツリーも必要です。

ファイルシステムのマウントについても質問されます。それらが見つかると `/mnt/sysimage` の下にマウントされます。その後シェルプロンプトが表示され、システムに適切な修正を行うためのさまざまなユーティリティにアクセスできます。

chroot はルート（`/`）ファイルシステムへのアクセスを改善するために使用できます。

レスキュー用の USB キー

- 現在の全てのディストリビューションでは **USB** インストール／ライブ／レスキューディスクイメージが利用可能である
- **USB** ドライブへの転送（リムーバブルドライブが /dev/sdX なら）：
  `$ dd if=boot.iso of=/dev/sdX`
- **BIOS** で **USB** ドライブからの起動が有効になっていることを確認する
- **livecd-tools** や **liveusb-creator** などのツールを使うとこの作業がとても簡単になり、ディスクのカスタマイズもできる

多くのディストリビューションは、ダウンロード用の boot.iso イメージファイルを提供しています（名前は異なる場合があります）。次のように **dd** を使用して、これを USB キードライブに置くことができます:

```
$ dd if=boot.iso of=/dev/sdX
```

上記はシステムがリムーバブルドライブを /dev/sdX として認識すると仮定しています。これによりドライブの既存のコンテンツが消去されることに注意してください！

システムに **USB** メディアから起動する機能があり、**BIOS** がそうするように構成されている場合、**USB** ドライブから起動できます。その後は、レスキュー **CD** や **DVD** と同じように機能します。ただし **USB** ドライブにはインストールツリーが存在しないことに注意してください。したがってこの方法では、必要に応じてネットワークベースのインストールツリーが必要になります。

livecd-tools や **liveusb-creator** などの便利なユーティリティを使用すると、インストールイメージを取得する場所としてローカルドライブまたはインターネットのいずれかを指定できます。起動可能なイメージを作成してリムーバブルドライブに書き込むという面倒な作業をすべて実行できます。これは非常に便利であり、実際すべての **Linux** ディストリビューションで機能します。

44.6 緊急モード



緊急モードでは最小限の環境でブートします。ルートファイルシステムは読み取り専用でマウントされ init スクリプトは実行されないのでほとんど何もセットアップされません。

シングルユーザーモード（次で説明します）よりも緊急モードが優れている主な点は、**init** が破損しているか機能していない場合でもファイルシステムをマウントして再インストール中に失われたデータをリカバリできることです。

緊急モードに入るには **GRUB** ブートメニューからエントリを選択し e を押して起動メニューを編集する必要があります。そして、システムにブートを指示する前に `emergency` という単語をカーネルコマンドラインに追加します。シェルプロンプトを表示する前に root のパスワードの入力を求められます。

また、ファイルシステムの破損などさまざまな理由でブートが失敗した場合にも、緊急モードに入ることができます。

44.7 シングルユーザーモード



システムブート時にブートが完了してもログインできない場合は **シングルユーザーモード** を試してください。

このモードではシステムは **ランレベル**１（**SysVinit** 言語の場合）で起動します。シングルユーザーモードは自動的にファイルシステムをマウントしようとするため、ルートファイルシステムを正常にマウントできない場合、または **init** 構成が破損している場合は使用できません。

シングルユーザーモードで起動するには１つの例外を除いて緊急モードで説明したのと同じ方法を使用します。例外はキーワードを `emergency` から `single` に置き換えることです。

44.8 演習

課題 44.1: 救出・復旧（レスキュー/リカバリ）メディアの準備

重要

本演習では、意図的にシステムを破壊しレスキューメディアを使って復旧します。そのため、何か実行する前には必ずレスキューメディアから確実にブートできることを確認してください。そこで、まずレスキューメディアを確認します。それは、専用のレスキュー/リカバリメディアでも良いですし、光学ディスクや USB ドライブにインストールしたライブイメージでも構いません。

メディアからブートして、システムを強制的にレスキューメディアからブートする方法（**BIOS** の設定に詳しくなければなりません）、システムがブートした後にレスキューモードを選択できることを確認します。

注目

仮想マシンを利用しているときは、２つの違いを除いて論理的に同じ操作です：

- **BIOS** に入る方法は、使用しているハイパーバイザーによっては難しいものになります。あるものは素早くキー入力する必要があったりします。そのため、ドキュメントを読みやり方を確認しておきます。
- 仮想マシンがマウントできるように設定することで、物理的な光学ディスクやドライブを使用することもできます。**USB** を使用する場合は、もう１つやることがあります。仮想マシンが物理デバイスを認識できるようにすることです。`.iso`イメージ ファイルを直接仮想マシンに接続するのが容易な方法です。

仮想マシンで作業することで、危険性が少なくなります。修復不可能なやり方でシステムを壊す恐れがあるときは、本演習を始める前に、仮想マシンのバックアップを取ることで、いつでも、イメージを復元できます。

重要

レスキュー/リカバリ メディアからブートできることを確認するまでは、以降の演習を始めないでください

課題 44.2: 破壊された GRUB 設定を復旧する

1. (`/boot/grub/grub.cfg`、`/boot/grub2/grub.cfg`、`/boot/grub/grub.conf`) の **GRUB** 設定ファイルを編集します。`kernel` 行の UUID フィールド中の最初の文字を削除します。レスキューモードで元に戻すため、消した文字を記録しておきます。(もし、ルートファイルシステムをラベルまたはディスクデバイスノードで識別している場合も、同様の方法で行ってください)。オリジナルのバックアップを撮ってください。

   BLSCFG システムの場合

   かわりに `/etc/grub2/grubenv` を集することで、コマンド行を破壊することもできます。

2. マシンをリブートしてください。システムは `No root device` was found. メッセージを出して、ブートを停止します。`panic` occurred メッセージも出力されます。
3. マシンに **インストール** か **ライブ DVD CD** または **USB** ドライブ（または、インストール用のサーバがあればネットワークブートメディア）をセットします。リブートします。ブートメニューが表示されたら、レスキューモードを選択します。
4. 別の方法として、大半のディストリビューションが提供している **GRUB** メニューから、レスキューイメージを選択します。レスキューメディアと同様にできるでしょう。しかし、必ずしも動作するとは限りません。たとえば、ルートファイルシステムが破壊されていれば、何もできません。

5. レスキューモードでファイルシステム検索を求められたら、同意してください。プロンプトが出たら、シェルを開いて **mount** と **ps** のようなユーティリティを実行して、レスキューを始めます。
6. ファイルを編集したり、バックアップから戻したりして **GRUB** 設定を修正し、破壊されたシステムを修復します。
7. exit を入力してインストーラーに戻ります。ブートメディアを取り出します。リブートの手順を確認します。リブートします。通常通り立ち上がるはずです。

課題 44.3: パスワード失敗からの復旧

1. root（**sudo** ではなく）で、パスワードを変更します。新パスワードを知らないと仮定します。
2. ログアウトし、古いパスワードでログインします。当然、失敗します。
3. レスキューメディアでブートし、オプション選択時に Rescue を選択します。ファイルシステムをマウントし、コマンドラインシェルを起動します。
4. **chroot** します（システムにいつも通りアクセスできます）:

   ```
   $ chroot /mnt/sysimage
   ```

   root のパスワードを元にリセットします。
5. 終了し、レスキューメディアを取り出しリブートします。通常通りログインできます。

課題 44.4: パーティション テーブルの破壊から復旧する

重要

本演習はかなり危険で、システムを使用不能にします。内容を完全に理解してから、演習を開始してください。

注目

以下の方法は、**MBR** を採用したシステム用です。**GPT** を利用しているときは、パーティション分割の章で説明したように --backup-file と --load-backup オプションを指定した **sgdisk** を使います。

1. root でログインして **MBR** をセーブします:

   ```
   $ dd if=/dev/sda of=/root/mbrsave bs=446 count=1
   1+0 records in
   1+0 records out
   446 bytes (446 B) copied, 0.00976759 s, 45.7 kB/s
   ```

   注意してください: コマンドを正しく打ち込んだか確認してください、そして、セーブしたファイルの長さが正しいかも確認してください。

   ```
   $ sudo ls -l /root/mbrsave
   -rw-r--r-- 1 root root 446 Nov 12 07:54 mbrsave
   ```

2. **MBR** を次のように消去します:

   ```
   $ dd if=/dev/zero of=/dev/sda bs=446 count=1
   1+0 records in
   1+0 records out
   446 bytes (446 B) copied, 0.000124091 s, 3.6 MB/s
   ```

3. リブートします。失敗しますね。
4. レスキュー環境でリブートします。**MBR** を元に戻します：

   ```
   $ dd if=/mnt/sysimage/root/mbrsave of=/dev/sda bs=446 count=1
   ```

5. レスキュー環境を終了しリブートします。システムは元通りブートするはずです。

課題 44.5: インストール イメージを利用した復旧

注目

本演習は、特に **Red Hat** ベース システム用となっています。他のディストリビューション ファミリーでも、簡単に置き換えることができると思います。

1. **zsh** パッケージを削除します（インストールされている場合）：

   ```
   $ dnf remove zsh
   ```

   または、

   ```
   $ rpm -e zsh
   ```

   簡単にするために、他に依存しないパッケージを選択しました。もし他に依存するパッケージを選択した場合は、必要に応じて削除したパッケージを再インストールする必要があります。

2. レスキュー環境でブートします。
3. レスキュー環境内で、**zsh** を再インストール（またはインストール）します。まず、インストールメディアを `/mnt/source` にマウントします:

   ```
   $ mount /dev/cdrom /mnt/source
   ```

   パッケージを再インストールします:

   ```
   $ rpm -ivh --force --root /mnt/sysimage /mnt/source/Packages/zsh*.rpm
   ```

   `--force` オプションを指定し、**rpm** が依存情報他を指定ディレクトリから決定することを指定します。もし、システムが何度もアップデートされていてインストールイメージがかなり古くなっていた場合、本手順は失敗します！

4. 終了しリブートします。
5. **zsh** が再インストールされていることを確認します:

   ```
   $ rpm -q zsh
   ```

   ```
   1 zsh-5.0.2-7.el7.x86_64
   ```

6. ```
   $ zsh
   ```

   ```
   1 ....
   2 [coop@q7]/tmp/LFS201%
   ```
   ```

章 45

Closing and Evaluation Survey

45.1 Evaluation Survey

Evaluation Survey

Thank you for taking this **Linux Foundation** course.

Your comments are important to us and we take them seriously, both to measure how well we fulfilled your needs and to help us improve future sessions.

- Please Evaluate your training using the link your instructor will provide.
- Please be sure to check the spelling of your name and use correct capitalization as this is how your name will appear on the certificate.
- This information will be used to generate your **Certificate of Completion**, which will be sent to the email address you have supplied, so make sure it is correct.

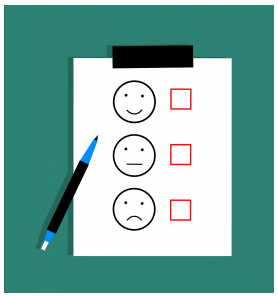


図 45.1: **Course Survey**