1. 講義資料：ファイルシステムの基礎

第1章 ファイルシステムの概要と抽象化

- **1-1. ファイルシステムとは何か？**
  - オペレーティングシステム（OS）は、ハードウェア（HDDなど）を直接操作するかわりに、「ファイル」や「ディレクトリ」といった**抽象化された概念**を提供します。
  - ユーザープログラムは「ファイルを読み書きする」といったシステムコールを発行し、OS内のファイルシステムがそれを解釈して、より低レベルな「セクタを読み書きする」といったI/O命令に変換し、ハードウェアを操作します。
  - このように、ファイルシステムはハードウェアとユーザープログラムの間に位置する重要な抽象化レイヤーです。
- **1-2. ファイルの概念**
  - **名前付け**: ファイルは通常、文字列で識別されます。OSによっては拡張子に特定の意味を持たせる場合があります。
  - **構造**: ファイルは単なるバイトの連なりとして扱われることもあれば、レコードや木構造といった内部構造を持つ場合もあります。
  - **種類**:
    - **通常ファイル**: ユーザーが作成する一般的なファイル（テキスト、実行可能バイナリなど）。
    - **特殊ファイル**: ディレクトリ情報や、UNIXにおけるデバイスファイルのように、システム資源に名前を付けるためのファイル。
  - **アクセス方法**:
    - **順アクセス（シーケンシャルアクセス）**: ファイルの先頭から順番にしか読み書きできない方式。磁気テープのようなデバイスに適しています。
    - **ランダムアクセス**: ファイル内の任意の位置に直接アクセス（シーク）して読み書きできる方式です。
  - **属性（メタデータ）**: ファイルの内容そのものではなく、ファイルに関する付加情報です。例えば、保護情報、サイズ、種類などがあり、「メタデータ」とも呼ばれます。
  - **操作**: ファイルには、`Create`, `Delete`, `Open`, `Close`, `Read`, `Write`, `Seek` といった一連の基本的な操作（API）が定義されています。
- **1-3. ディレクトリの概念**
  - **構造**: ファイルをグループ化して管理するための仕組みです。初期のシステムでは単純な1階層（フラット）でしたが、現代のOSでは**階層型ディレクトリシステム**が一般的です。
  - **パス名**: 階層構造の中でファイルやディレクトリの場所を一意に特定するための文字列です。
    - **絶対パス**: ルートディレクトリから始まる経路。
    - **相対パス**: カレントディレクトリ（現在の作業場所）から始まる経路。
  - **操作**: ディレクトリにも `Create`, `Delete`, `Opendir`, `Readdir` といった操作が定義されています。

第2章 ファイルシステムの実装

- **2-1. 実装の構成要素**
  - ファイルシステムの実装は、大きく2つの部分に分けられます。
    - **ディレクトリシステム**: ファイル名、階層構造、属性といったメタデータを管理します。
    - **ストレージシステム**: ファイルの内容（実体）をディスク上に格納・管理します。
- **2-2. ファイルの実装方法**
  - ファイルの実体（データ）をディスクブロックにどう割り当てるかには、いくつかの方式があります。
    - **連続割り当て**: ファイルを連続したディスクブロック群に格納します。順次アクセスは高速ですが、空き領域の断片化（フラグメンテーション）が問題になります。
    - **連結リスト割り当て**: ファイルを構成するブロックを連結リストで繋ぎます。ブロックの割り当てや解放は容易ですが、ランダムアクセスが非常に遅くなります。

        - **i-node方式**: UNIXで採用されている方式です。各ファイルに**i-node**という管理ブロックを割り当て、そこにファイル属性や、ファイルの実体を構成するディスクブロックへのポインタを格納します。ディレクトリはファイル名とi-node番号の対応のみを管理します。
- **2-3. 共有ファイル**
    - 同じファイルをディレクトリツリーの複数の場所から参照できるようにする仕組みです。i-nodeを共有する方法（ハードリンク）や、リンク型の特殊ファイルを用いる方法（シンボリックリンク）があります。

第3章 ファイルシステムの管理と最適化

- **3-1. ディスク空間管理**
    - **ブロックサイズ**: ファイルを格納する単位であるブロックのサイズは、性能と効率のトレードオフで決まります。
        - サイズが**大きい**と、転送効率は良いですが、小さなファイルでも大きな領域を占有してしまい、ディスク空間の無駄（内部フラグメンテーション）が増えます。
        - サイズが**小さい**と、空間効率は良いですが、一つのファイルが多数のブロックにまたがり、読み書きの際のシークや回転待ちが増え、転送速度が低下します。
    - **空きブロック管理**: 使用されていないブロックを管理する方法として、ビットマップ方式や連結リスト方式があります。
- **3-2. ファイルシステムの信頼性**
    - **バックアップ**: ディスク障害などに備え、ファイルシステムの情報をコピーしておくことは極めて重要です。
        - **物理ダンプ**: ディスク全体をブロック単位で丸ごとコピーします。
        - **論理ダンプ**: ファイルシステムの構造を復元できる必要な情報だけをコピーします。**差分ダンプ（Incremental Dump）** はこの一種で、前回のバックアップから変更されたファイルのみを対象とします。
        - 差分ダンプは、複数のレベル（例：月次フルバックアップ、週次差分、日次差分）を組み合わせることで、効率的な復元が可能になります。
    - **一貫性**: OSのクラッシュなどにより、管理情報に矛盾が生じることがあります（一貫性の喪失）。
        - **ブロック一貫性**: あるブロックは「使用中」か「空き」のどちらか一方に、ただ一度だけ現れるべきという原則です。
        - **ディレクトリー貫性**: ディレクトリからの参照数と、i-nodeに記録された被参照数（リンクカウント）が一致しているべきという原則です。
        - fsck のようなツールは、これらの矛盾を検出し、修復を試みます。
- **3-3. 性能向上**
    - **ブロックキャッシュ**: よく使われるディスクブロックをメモリ上に保持し、ディスクアクセスの回数を減らします。
    - **ブロックの先読み**: ファイルが順次アクセスされていることを検知し、要求される前に次のブロックを読み込んでおきます。
    - **ディスクアーム移動の削減**: 関連するブロックをディスク上で物理的に近い位置に配置することで、ヘッドの移動距離を短くします。

第4章 様々なファイルシステム

- **ジャーナリングファイルシステム**: ディスクへの変更を行う前に、その内容を「ログ（ジャーナル）」に書き出す方式です。システムがクラッシュしても、再起動時にログを参照することで、処理を安全に完了または中断でき、迅速な復旧が可能です。
- **ログ構造化ファイルシステム**: ディスク全体を一つの大きなログとして扱い、書き込みを常にログの末尾への追記として行うことで、特に小さなランダム書き込みの性能を大幅に向上させることを目指したものです。
- **代表的な例**:
    - **MS-DOS (FAT)**: ブロック管理をディレクトリエントリから分離し、「次ブロック」を管理するテーブル（FAT）を用いたことが特徴です。

    - **UNIX V7**: ファイル名と属性・実体を管理するi-nodeを明確に分離した設計が特徴です。

---

2. 重要用語の抽出と説明

- **ファイルシステム (File System)**
    - OSが提供する、ストレージデバイス上のデータを管理するための仕組みであり、ユーザーやアプリケーションに対して「ファイル」や「ディレクトリ」といった抽象化されたインターフェースを提供します。
- **メタデータ (Metadata)**
    - ファイルの内容そのものではなく、ファイル名、サイズ、種類、保護情報、作成日時といったファイルに関する付加情報のことです。ファイル属性とも呼ばれます。
- **i-node (アイノード)**
    - 主にUNIX系のファイルシステムで用いられるデータ構造で、ファイルの種類、所有者、アクセス権、タイムスタンプ、そしてファイルの実体が格納されているディスクブロックへのポインタなどのメタ情報を保持します。
- **パス名 (Path Name)**
    - 階層型ディレクトリシステムにおいて、特定のファイルやディレクトリの位置を一意に示すための文字列です。ルートから始まる「絶対パス」と、カレントディレクトリを基準とする「相対パス」があります。
- **差分ダンプ (Incremental Dump)**
    - ファイルシステムのバックアップ手法の一つで、前回のバックアップ以降に変更があったファイルのみをコピーする方法です。これにより、バックアップ時間とストレージ容量を節約できます。
- **ファイルシステムの一貫性 (File-System Consistency)**
    - ファイルシステムを構成するメタデータ（例：空きブロックリスト、i-nodeのリンクカウントなど）に矛盾がない状態を指します。システムのクラッシュなどによって一貫性が失われることがあります。
- **ジャーナリングファイルシステム (Journaling File System)**
    - ディスク上のファイルシステムへの変更を実際に行う前に、その変更内容をログ（ジャーナル）として専用領域に記録するファイルシステムです。これにより、予期せぬシステム停止が発生しても、ログを元にファイルシステムを素早く一貫性のある状態に復旧できます。

---

3. 追加で学習すべき項目

提供された資料はファイルシステムの基本的な概念から実装、管理手法までを幅広く網羅していますが、さらに理解を深めるために以下の項目について学習することをお勧めします。

- **現代的なファイルシステムの具体的な実装**
    - **説明**: 資料ではMS-DOS (FAT) や UNIX V7といった古典的な例が挙げられていますが、現在主流の **NTFS (Windows), ext4 (Linux), APFS (Apple)** などのファイルシステムが、資料で述べられたジャーナリング、i-node、ディスク空間管理 といった概念をどのように発展させ、具体的に実装しているかを学ぶと、理論と実践の繋がりがより明確になります。
    - **学習のポイント**: コピーオンライト(Copy-on-Write)、エクステント(extent)ベースの割り当て、スナップショット機能など、現代的なファイルシステムが持つ高度な機能について調べると良いでしょう。
- **仮想ファイルシステム (VFS: Virtual File System)**
    - **説明**: 資料の目次には項目がありますが、詳細な説明は含まれていません。VFSは、OSが様々な種類の具体的なファイルシステム（例：ext4, NTFS, ネットワークファイルシステム）を**統一的なインターフェースで扱えるようにするための抽象化レイヤー**です。アプリケーションはVFSが提供する共通のAPI（`open`, `read`など）を呼び出すだけで、下層にあるファイルシステムの種類を意識する必要がなくなります。
    - **学習のポイント**: なぜVFSが必要なのか、そしてOSカーネル内でどのようにして異なるファイルシステムへの処理の振り分けを実現しているのか、その仕組みを学ぶことはOS全体の理解に不可欠です。
- **ストレージデバイスの進化とファイルシステム**
    - **説明**: 資料は主にハードディスク（HDD）を前提とした性能最適化（例：ディスクアームの移動削減）について言及しています。しかし、現代では**SSD (ソリッドステートドライブ)** が広く普及しています。SSD

にはシークタイムや回転待ちが存在しない一方で、書き込み回数に上限がある、書き込み単位が大きいといったHDDとは全く異なる特性があります。

- **学習のポイント**: SSDのこれらの特性（特に「書き込み増幅」問題）に対応するために、ファイルシステムやOSがどのような工夫（例：TRIMコマンド、ウェアレベリングとの協調）を行っているのかを学ぶことで、ハードウェアとソフトウェアの連携に関する深い知見が得られます。