ファイルシステム

佐藤一郎

2001年5月15日

ファイル

関連する情報を保存・管理し、それらにアクセスする手段を提供する仕組みをファイルシステムと呼ぶ

- 名前づけられた論理的単位
- 長期短期に保存可能で、必要なときに読み書き可能
- ユーザの便宜を図った記憶装置

コンピュータの記憶装置：

- 主記憶（メモリ）は高速に読み書きができるが、高価なためすべての情報を記憶することは困難。また、コンピュータ終了時には記憶が消失
- ２次記憶装置は高速読み書きはできないが、安価なため大容量記憶が可能で、コンピュータ終了後も保持される

ファイルシステムへの要求

ファイルはハードディスクなどの２次記憶装置に保存

- メモリと比較して読み書き速度は低速
- ディスク装置は機械的な動作を伴う
- ２次記憶装置は高価（メモリよりは安価）
- 記憶装置の容量を無駄なく利用
- 貴重なデータを取り扱うため信頼性が必要
- 記憶装置容量の拡大に円滑対応
- 特定の記憶装置によらない論理的なインターフェースを提供

ファイルの構造

各ファイルの構成方法として一般的に３つの方法がある

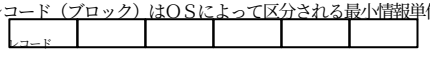
- バイトの並び

  特定のファイル構造を持たない

  UNIX、MS-DOS、Windows95/98/Me、Windows NT (FAT)

- レコードの並び

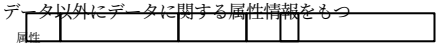
  レコード（ブロック）はＯＳによって区分される最小情報単位



  メインフレーム用OS、ミニコン用OS

- 属性の並び

  データ以外にデータに関する属性情報をもつ

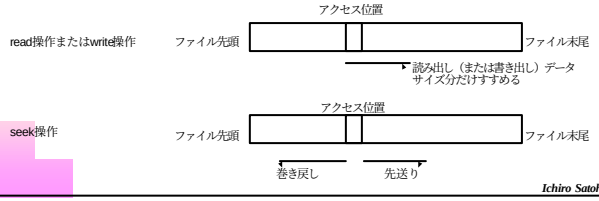


  MacOS、WindowsNT (NTFS)

ファイルの操作

ＯＳが提供するシステムコールまたはAPIによりファイルを操作

OPEN	ファイルアクセスに先立ち、ディスク上のファイルの位置を調べる。また、バッファ領域を含むファイル入出力に必要な情報領域（テーブル）を作成
CLOSE	ファイル入出力にテーブルの領域を解放
READ	ファイルからメモリにデータを読み出す
WRITE	メモリからファイルにデータを書き込む
SEEK	ファイルのアクセス位置を変更する



Ichiro Satoh

アクセス方法

ファイルのアクセス（読み出し・書き込み）の方法

順アクセス

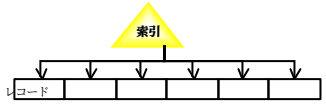
ファイルの先頭から末尾に向かってアクセス
（read操作、write操作により実現）

直接アクセス

ファイル上の任意の位置をアクセス
（seek操作、read操作、write操作により実現）

検索付き直接アクセス

キー値からアクセス対象となるレコードを取得・アクセス



Ichiro Satoh

ファイルの管理情報

代表的なファイル管理情報（ただしＯＳにより相違）

- **名前**
- **属性**
- **物理的位置（ブロック）**
- **大きさ（サイズ）**
- **保護情報（アクセス権）**
- **参照時間、生成時間**

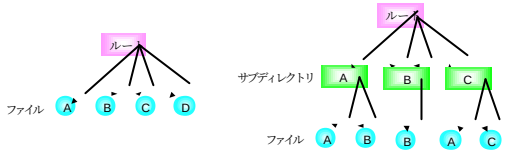
ファイルの管理情報とファイル本体を結ぶ部分をディレクトリと呼ぶ

- **ディレクトリに管理情報そのものを保持、または**
- **ディレクトリに管理情報のある領域（ブロック）のポインタを保持**

Ichiro Satoh

ディレクトリ構造

２つ以上のファイルの名前が重なる（衝突）と区別できない
名前の有効範囲を限定　→　ディレクトリ階層

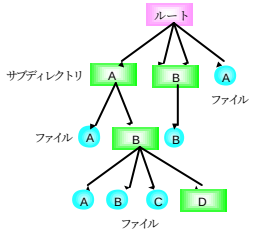


OSによりディレクトリをフォルダーと呼ぶことがある

Ichiro Satoh

ディレクトリ階層

ディレクトリ及びファイルにより木構造を構成



- 木の根（ルート）をルートディレクトリと呼ぶ
- 木の葉はディレクトリまたはファイル
- 節はディレクトリ

- ファイル名の指定では根から葉にいたる経路（パス）を利用

ファイルの実装

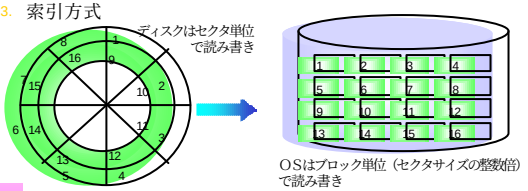
磁気テープ

先頭から順にファイルを配置するしかない

磁気ディスク

ランダムアクセス可能な記憶媒体へのファイル配置方法は多様

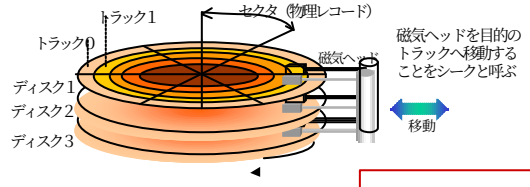
1. 連続方式
2. （連結）リスト方式
3. 索引方式



ハードディスク

セクタ単位で読み書きが可能

- **任意のトラック･セクタを指定可能**
- **セクタ内の読み書きは順次的**



詳細図は教科書88ページ参照

ヘッドを移動することなく読めるトラックの組をシリンダと呼ぶ

ハードディスクへのアクセス時間内訳：
- **6割：シーク時間**
- **3割：サーチ時間**
- **1割：データ読み書き時間**

ブロック

多くのOSがディスクをセクタを要素とする１次元配列として管理

セクタの指定：
シリンダ番号、ヘッド番号（ディスク番号）、セクタ番号の３項組

セクタサイズ：
システムに依存（代表的サイズ:512B）

ブロック：
ＯＳがハードディスクを読み書きする際のデータ単位

ブロックサイズ：
ＯＳに依存。ただし、512B、1024B、2048B、4096B、8192Bなど

注意：ブロックサイズが大きいと：
○まとめて入出力処理が可能なため高速化
×ファイルサイズがブロックサイズより小さいと無駄が多い

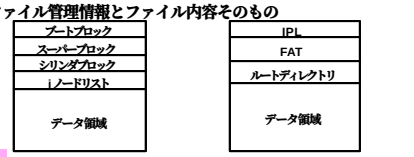
ボリューム

記憶媒体（OSではボリュームと総称）：

磁気ディスクや光ディスク、磁気テープなどの記憶する媒体

ボリュームの構成：

- 初期プログラムローダ(IPL)
  システム起動時に自動的に読み込まれるプログラム
- ボリュームの管理情報
  ボリュームの名前、領域割り当てサイズなどのボリューム管理に必要な情報
- ファイル
  ファイル管理情報とファイル内容そのもの



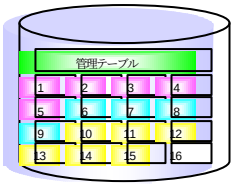
UNIXのボリューム　　Windows (FAT) またはMS-DOSのボリューム

ファイルの割り当て 連続方式

ファイルはディスク上の連続したブロックに隙間なく格納
○必要ブロック数と開始ブロック番号だけを管理すればよい（簡単）
○連続ブロックを順にアクセスするため高速
×ファイル生成時にファイルサイズを固定化
×断片化（フラグメンテーション）により無駄な領域が発生しやすい

ボリューム管理テーブル

ファイル名	ブロック数	開始ブロック
ファイル1	5	1
ファイル2	4	6
ファイル3	6	10



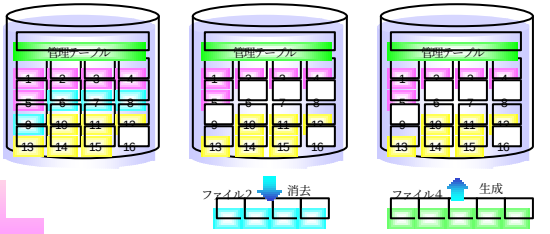
ファイルの割り当て 連続方式

ブロックの獲得と解放を繰り返すと未使用領域が増える

ファイル名	数	開始
ファイル1	5	1
ファイル2	4	6
ファイル3	6	10

ファイル名	数	開始
ファイル1	5	1
ファイル3	6	10

ファイル名	数	開始
ファイル1	5	1
ファイル3	6	10
~~ファイル4~~	~~5~~	?

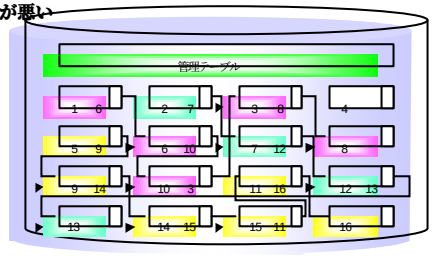


ファイルの割り当て 連結リスト方式

ファイルを構成するブロックを先頭から順にリンクで連結する
○ファイルサイズ（必要ブロック数）を変更が容易
○断片化による無駄な領域が発生しない
×リンク不良によりファイルアクセスが不可能になる
×直接アクセスの効率が悪い

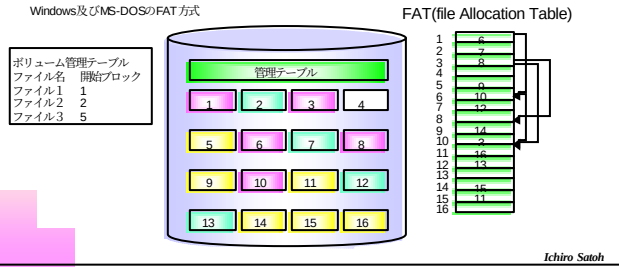
ボリューム管理テーブル

ファイル名	開始ブロック
ファイル1	1
ファイル2	2
ファイル3	5



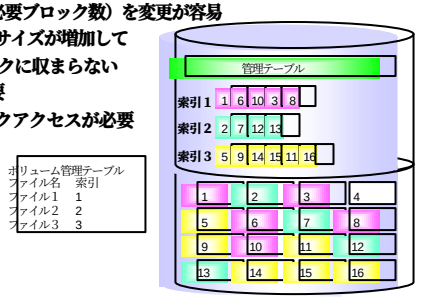
ファイルの割り当て：リスト検索表方式

ファイルを構成するブロックを先頭から順にリンクで連結する
リンク情報を取り出して、検索（FAT）としてメモリ上におく方法
○直接アクセスも高速化
×検索情報（リンク表）が大量のメモリを消費



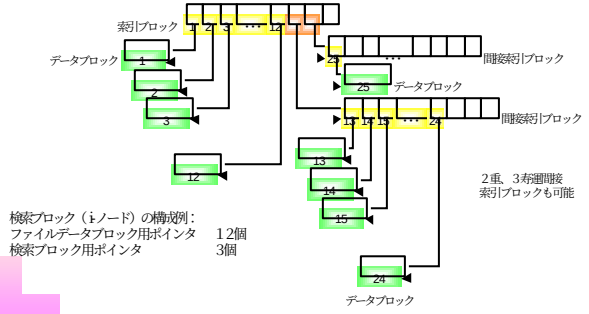
ファイルの割り当て：索引方式

ファイル構成するブロック番号を表形式で管理（索引ブロック内に格納）
○断片化による無駄な領域が発生しない
△ファイルサイズ（必要ブロック数）を変更が容易
ただし、ファイルサイズが増加して
一つの索引ブロックに収まらない
ときは拡張が必要
△最少２回のディスクアクセスが必要



ファイルの割り当て：UNIX（iノード）

索引方式では大容量用ファイルは一つの索引ブロックでは収まらない
索引ブロックの索引ブロック（間接ブロック）を導入



ディレクトリの実装

ディレクトリ：
ファイル名からファイルの格納されているブロックを知る方法を提供

ディレクトリエントリ：
各ファイルの名前、属性（日付、アクセス権等）、ブロック番号を保持