システムプログラム概論

ファイルシステム

第7講: 平成20年10月27日 (月) 1限 S1教室

中村 嘉隆(なかむら よしたか)
奈良先端科学技術大学院大学 助教
y-nakamr@is.naist.jp
http://narayama.naist.jp/~y-nakamr/

今日の講義概要

- ファイルとディレクトリ
- ファイルシステムの実現
- ファイルシステムの実例

ファイルシステムとは

- **ファイル**の必要性
  - プロセスは実行中に情報をアドレス空間に保存
    → 仮想アドレス空間サイズより大きなサイズの情報を扱いたい
  - プロセス終了時保存した情報は消失
    → 長期間保存したい
  - 情報を複数のプロセスからアクセス可能にしたい
- **ファイルシステム**
  - OS においてファイルを扱う部分
  - 構造，名前付け，アクセス方法など

ファイルの名前付け

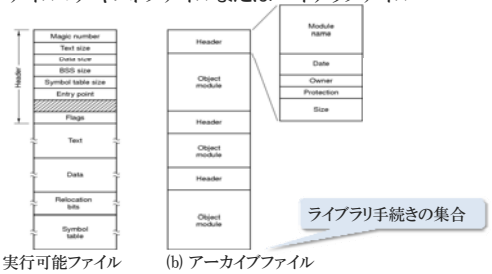
- ファイルは生成時に文字列による名前がつけられる
- OS により文字列の制限が異なる(UNIX: 255 文字まで)

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPLG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

- ファイル名は 2 つの部分で構成されることが多い
  - **名前.拡張子**
- 拡張子でファイルの種類をあらわす
  → 対応するアプリケーションソフトとの関連付け
- 拡張子部分が 2 つ以上あることも
  - `file.c.Z`
  - `src.tar.gz`

ファイルの種類

- 標準ファイル，ディレクトリ，デバイスファイル(キャラクタ型/ブロック型)
  - 標準ファイル:テキストファイルまたはバイナリファイル



(a) 実行可能ファイル (b) アーカイブファイル

ファイルへのアクセス

- **シーケンシャルアクセス**
  - ファイル中の全ての文字またはレコードをファイルの先頭から順番に読込み
  - スキップしたり，順序を無視しての読込みはできないが，巻き戻しは可能
  - 媒体が磁気テープであればこの方式は便利
- **ランダムアクセス**
  - ファイル中の文字またはレコードは任意の順序で読込み可能
  - 多くのアプリケーションで必要(特にデータベースでは必須)
  - 読込み開始位置の指定方法
    1. 各読込み操作終了後にファイル中の現在位置を返す
    2. "シーク"操作により，読込み開始位置を設定後，読込みを行う

ファイルの属性

- 各ファイルは名前，データ以外に属性情報を持つ
  - どんな属性を持つかは OS によりまちまち

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

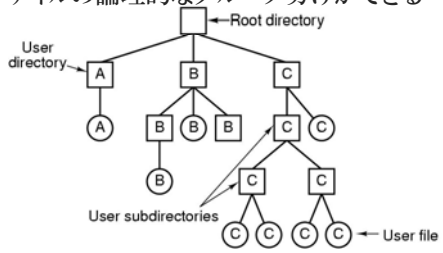
ディレクトリ

- ファイルを収納する“箱”（別名“フォルダ”）
  - ディレクトリ自体も特殊なファイル
- ディレクトリシステムの種類
  - 単一レベルディレクトリ
  - 二階層ディレクトリ
  - 多階層ディレクトリ

} 省略

多階層ディレクトリ

- 木構造からなる階層ディレクトリシステム
  - ユーザがサブディレクトリを作ることが可能
  - ファイルの論理的なグループ分けができる

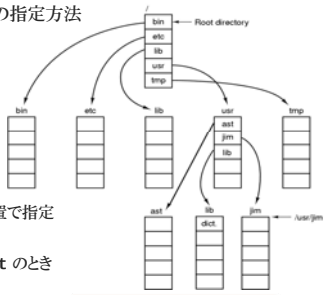


パス名（Path names）

- 木構造ディレクトリ上でのファイルの指定方法
  - 絶対パス名と相対パス名
- 絶対パス名
  - UNIX の場合
    `/usr/ast/mailbox`
  - Windows の場合
    `¥usr¥ast¥mailbox`
- 相対パス名
  - カレントディレクトリからの相対位置で指定
    - 「.」はカレントディレクトリ
    - 「..」は親ディレクトリ

  例）カレントディレクトリが `/usr/ast` のとき

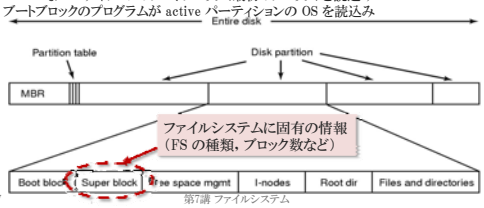
```
cp mailbox mailbox.bak
cp ../lib/dictionary
```



UNIX のディレクトリツリー

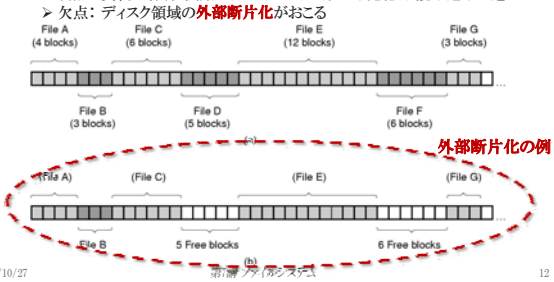
ファイルシステムの実現

- ディスクは**パーティション**に分割される
  - パーティション毎に異なるファイルシステムを保存可能
- MBR (Master Boot Record)
  - ディスクの最初のセクタ(セクタ 0)に保存
  - 各パーティションの開始・終了アドレスのテーブルを保持
  - 一つのパーティションだけが active
- OS のブート
  - BIOS が MBR を読み込み
  - Active なパーティションのブートブロック(最初のブロック)を読込み
  - ブートブロックのプログラムが active パーティションの OS を読込み



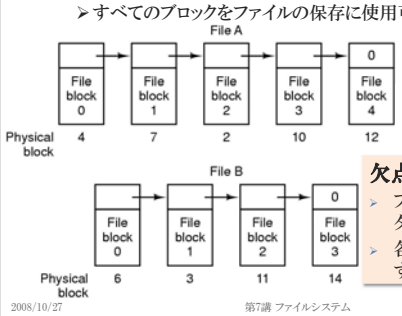
ファイルの実現 (1)

- ファイルへのディスクブロックの割付：**連続割付**と**リンク割付**
  - **連続割付(Contiguous Allocation)**
    - 50KB のファイルなら，50 個の連続した 1KB ブロックを割り付ける
    - 利点：実装が容易(開始アドレス，ブロック数のみ記憶)，読み込みが速い
    - 欠点：ディスク領域の**外部断片化**がおこる



ファイルの実現 (2)

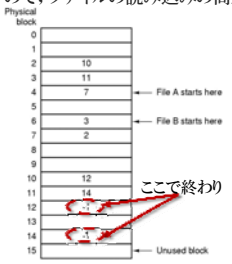
- リンク割付(Linked List Allocation)
  - 各ブロックの最初のワードを次のブロックへのポインタに使用
  - すべてのブロックをファイルの保存に使用可能



欠点:
- ブロックへのアクセスはランダムアクセスとなるため遅い
- 各ブロックでポインタを記憶する必要がある

リンク割付における改善

- FAT(File Allocation Table)をメモリ上に用意
  - 各エントリは次のブロックの番号を記憶
  - どのブロックがどういう順番で連結されているかがメモリの参照で分かるので，ファイルの読み込みの高速化が可能

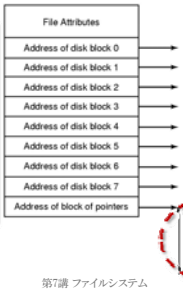


欠点:
- FAT はディスクサイズに比例してメモリ容量を消費
- 20GB のディスクで FAT のサイズは 60～80MB になる

ファイルの実現 (3)

- i-node(index-node)
  - ファイルの属性＋使用ブロックへのアドレスのリスト

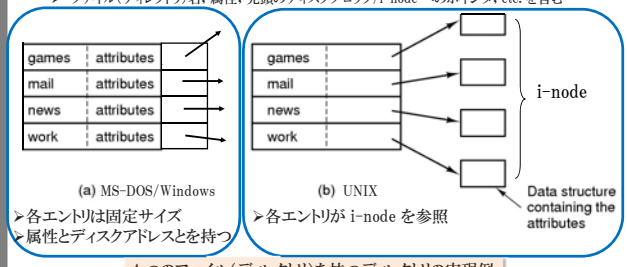
利点:
- i-node はファイルをオープンしている時だけ，メモリに読込まれる
- メモリ消費量は，ディスクの容量に関係なく，kn バイト(i-node のサイズが k バイト，同時オープンファイル数が n のとき)



ファイルサイズが大きくなり i-node サイズを超えた場合

ディレクトリの実現

- ディレクトリ
  - ディレクトリエントリのリストからなるファイル
- ディレクトリエントリ
  - ファイル(ディレクトリ)名，属性，先頭のディスクブロック/i-node へのポインタ，etc. を含む

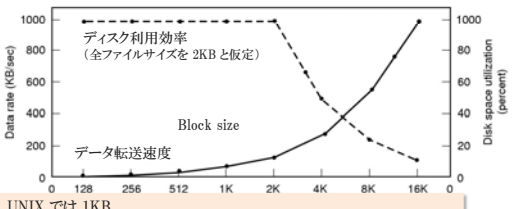


- 各エントリは固定サイズ
- 属性とディスクアドレスとを持つ
- 各エントリが i-node を参照

4 つのファイル(ディレクトリ)を持つディレクトリの実現例

ディスクの領域管理

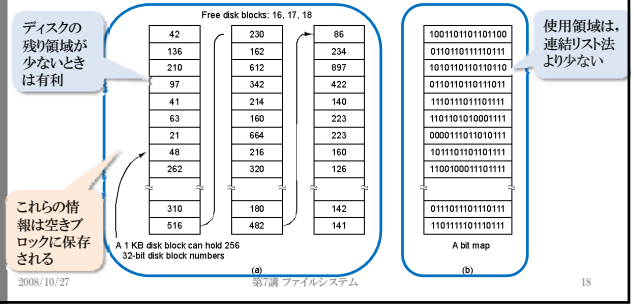
- ブロックサイズ(読み書きの単位)
  - 大きいほどファイルへのアクセス時間が高速
    - 読み書きするブロック数が減り，シークの回数が少なくなる
  - 小さいほどディスク領域の利用効率が高い
    - 32KBブロックで，1KB のファイルを保存する場合，利用効率は 1/32≒3%



- UNIX では 1KB
- Windows(FAT)では，512B～32KB(アドレス長，ディスク容量に依存)

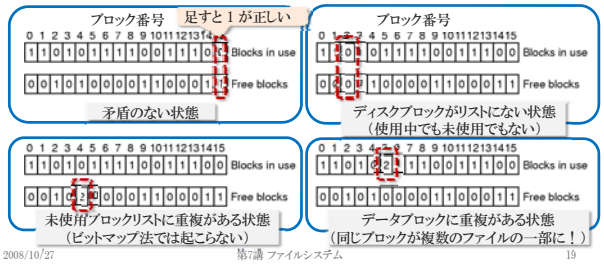
未使用ブロックの管理

- 連結リスト法
  - 未使用ブロックのアドレスリスト(固定数)の連結リストで表す
- ビットマップ法
  - 各ブロックの使用状況を 1 ビット(空きを 0)で表す



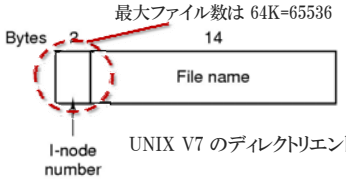
ファイルシステムの整合性

- ファイルシステムの整合性が狂う要因
  - ディスクブロック(ディレクトリエントリ, i-node など)の書き込み完了前にシステムがクラッシュ(あるいは電源オフなど)
- ファイルシステムの不整合の検出・修正が必要(fsck, scandisk 等を利用)
  - 各ディスクブロックについて, ファイル中に何回出てくるか, 未使用ブロックリストに何回出てくるかをカウント



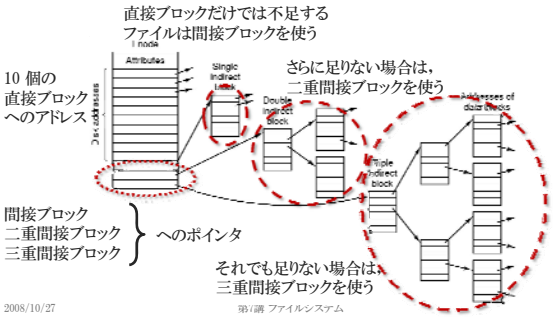
UNIX V7 ファイルシステム (1)

- DEC PDP-11(1970年)で使用されたファイルシステム
  - 多階層ディレクトリ(リンクも可), 14 文字のファイル名(任意の ASCII 文字が使用可)
  - ディスクブロックの割付は i-node 方式
  - i-node は, 属性情報(ファイルサイズ, 3 つの時間, 所有者 ID, グループ ID, アクセス権, リンクカウント)を含む



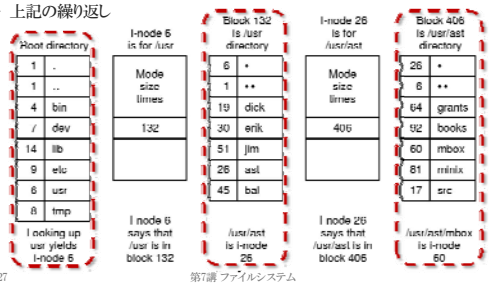
UNIX V7 ファイルシステム (2)

- 大容量ファイルの保存方法



UNIX V7 ファイルシステム (3)

- パス `/usr/ast/mbox` の探索手順
  - ルートディレクトリからはじめる
  - ディレクトリ(ファイル)の中身を見て, 該当するディレクトリ(ファイル)の i-node を 読み込む
  - 上記の繰り返し



ファイルシステムのまとめ

- ファイルシステム
  - ファイルとディレクトリの実現方法
- ディスクの領域管理
- ファイルシステムの実例
- UNIX V7

テスト

- 10月29日(水) 2 限
- 参考書, 講義資料持込み可

- 試験内容
  - 各概念が理解できているか
  - プロセス, スレッド, コンテキストスイッチ, スケジューリング, 競合回避, スワッピング, 仮想記憶, ページング, セグメンテーション, 入出力制御, 割込み制御, ファイルとディレクトリ, 外部断片化, 内部断片化, など
  - 各アルゴリズムが正確に再現できるか
    - スケジューリング, ページング, ディスクブロック割付, 利用効率, など