

inode

# テーマ : inode

大事(そう)なところだけ抜粋します

- ※ 意見には個人差があります
- ※ 誤りが含まれている可能性もあります

要点的な

# 大事なこと

- ブロックデバイス内で管理されている inode は `ino.h` の構造体
- 図 9.4 inode 領域
  - ブロック番号は 0 から
  - inode 番号は 1 から
- 図 9.9 間接参照
  - でも主流になったのは V7 方式…。

# 似た関数

- for inode      for data
- 
- 同期      iupdat()      update()
- 領域取得      ialloc()      alloc()
- 領域解放      ifree()      free()
-

# 紛らわしい関数

- リリース
  - brelse() : B\_BUSY をクリアし av-list に追加
  - prele() : inode[] の ILOCK をクリア

## 9.4. inode の取得・解放

# inode[]

- inode をメモリ上で管理している構造体配列
- 要素数は 100 (NINODE)
  - 最近参照した 100 個が保持される



# inode 構造体

- i\_size0 と \*i\_size1
  - なんで char\* ?
- これはメモリ上の inode[] 用

# iget()

- inode[] エントリを取得する
  - 引数 1 : デバイス番号
  - 引数 2 : inode 番号
- ロックされてたらスリープして待つ
- IMOUNT なら mount[] を取得して  
対応デバイスルートの inode[] エントリを返す
- 子関数
  - bread()
  - brelse()
  - iput()

# input()

- inode[] エントリのカウンタをデクリメントする
- 子関数
  - itrunc()
  - iupdat()
  - prele()

# iupdat()

- inode[] エントリの内容をブロックデバイスに反映
- inode の sync 的なやつ
- L12 – 14 は 263 ページに解説あり
- ldiv(), lrem()
  - 除算結果(整数)と余り
-

# iupdat()

- 子関数
  - getfs()
  - bread()
  - bwrite()

## 9.5. inode からストレージ領域への マッピング

# マッピングの種類

	ILARG	b and i	Max file size
直接	0	$b \leq 7$	4KB
間接	1	$i = b / 256$ $i < 7$	896KB
二重間接	1	$i = b / 256$ $i == 7$	32MB (実質16MB)

# bmap()

- ブロック番号の変換：論理 → 物理
- 子関数
  - alloc()
  - bdwrite()



# itrunc()

- inode[] エントリが使用しているストレージ領域のブロック番号をフリーリストに戻す。
- 子関数
  - bread()
  - free()
  - brelse()

## 9.6. ブロックデバイスの ブロック割り当て

# inode 領域の取得／解放

- ialloc()
  - getfs()
  - iget(), iput()
  - bread()
- ifree()
  - getfs()

# ialloc()

- ストレージ領域の未使用 inode を割り当てる

# ifree()

- inode を解放する

# ストレージ領域の取得／解放

- alloc()
  - badblock()
- free()
  - badblock()

# alloc()

- ストレージ領域の未使用ブロックを割り当てる

# free()

- ストレージ領域のブロックを解放する



# getfs()

- デバイス番号に対応した filesystem 構造体を取得
- 子関数
  - なし

# badblock()

- ブロック番号の妥当性をチェックする
- 子関数
  - なし

## 9.7. パス名から inode への変換

# ディレクトリの内容

- ファイル情報のテーブルを  
各ディレクトリが保持している
  - 上位 2 Byte : inode 番号
  - 下位 14 Byte : ファイルパス

# namei()

- ファイルパスを検索し、該当する file or dir の inode[] エントリを取得する。
- パスの取得
  - schar : カーネル空間
  - uchar : ユーザー空間

# access()

- file permission に権限があるかを確認する
- 子関数
  - getfs()

## 9.8. 初期化と同期

# iinit()

- / のある disk の superblock を読み出して mount[] の最初のエントリにセットする。
- 子関数
  - break()
  - getblk()
  - bcopy()
  - brelse()



# update()

- データの同期を行う: メモリ <-> デバイス
- データの sync 的なやつ
- 子関数
  - bcopy()
  - bwrite()
  - iupdat()
  - prele()
  - bflush()

# 第 9 章 完

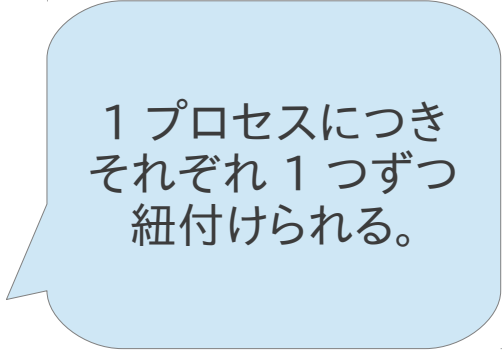
# 参考文献

- やる気のないはてだ
  - UNIX 6th code reading - ファイルシステム概要
    - <http://d.hatena.ne.jp/takahirox/20110815/1313398103>
-

EOF

# プロセス概要

- 基本的なルールは Linux と一緒
- 管理情報
  - proc 構造体
    - メモリ上の特定領域に常に存在
    - プロセス最大数は NPROC (= 50)
  - user 構造体
    - PPDA 領域内
    - グローバル変数 u でアクセス



1 プロセスにつき  
それぞれ 1 つずつ  
紐付けられる。

# つまりこう？

APR

UISD (0177600) : PDR[0] – [7] for Text

(UDSD) (0177620) : PDR[0] – [7] for Data

UISA (0177640) : PAR[0] – [7] for Text

UDSA (0177660) : PAR[0] – [7] for Data