オペレーティングシステム2006
ファイル管理 (3)
演習へのヒント他

2006年11月17日

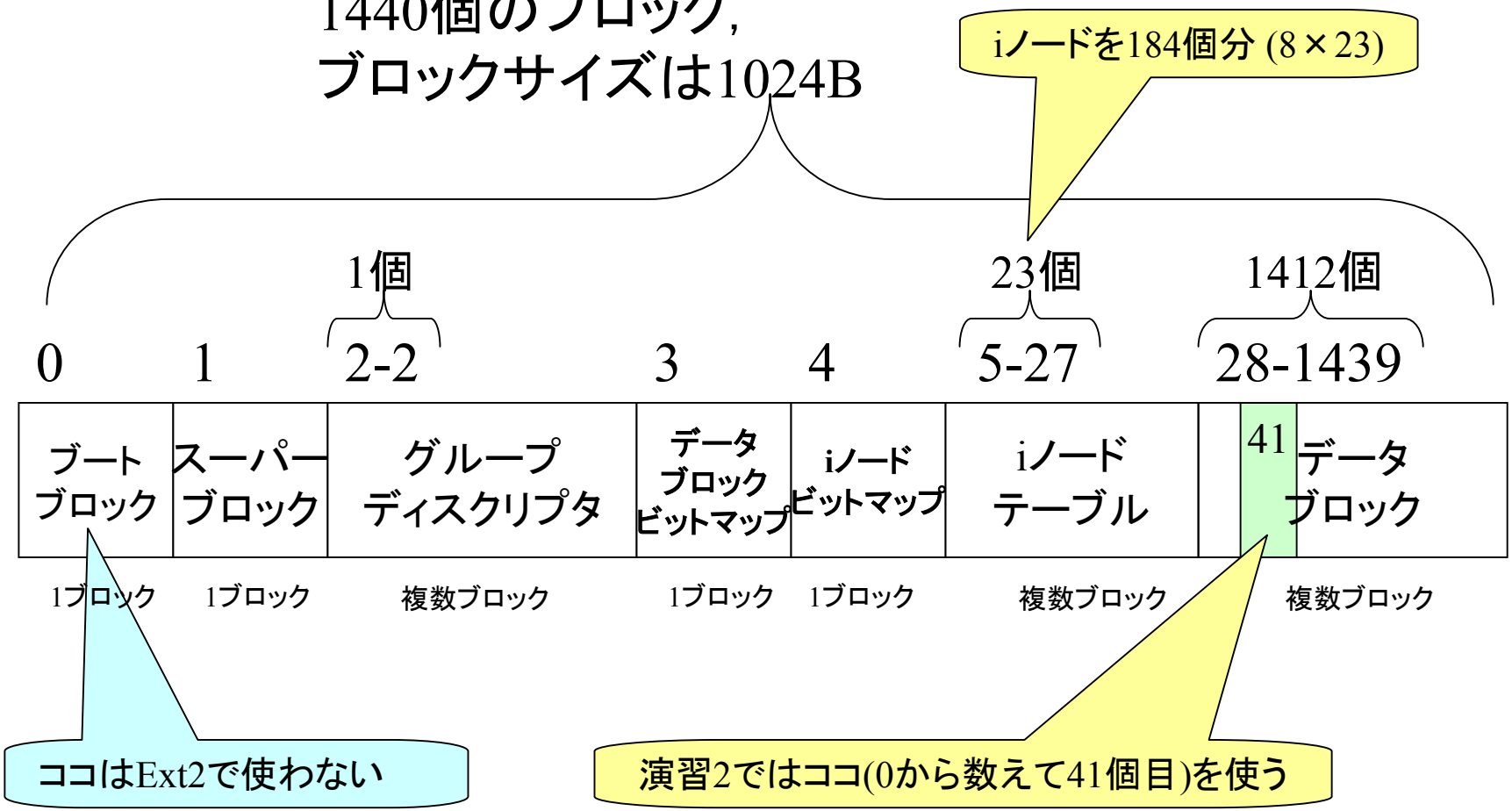
海谷 治彦

目次

- Ext2の構造再び
- 実際のExt2データ(ディスクの中身)を解析
  - ディレクトリを保持するデータブロックについて
  - iノードの探索
- バイトオーダーについて
  - i386とPPCの違い
  - Little Endian vs Big Endian

演習で使うFSの構造 (ブロックグループが1個)

1440個のブロック,
ブロックサイズは1024B



演習3の大まかな流れ

- ブロック41を取り出す.
  - サンプル getblock.c (任意のブロックを切り出すもの)
- その中身を構造体 ext2_dir_entry_2 に従い解析する.
  - この構造体は可変長なのでタチが悪い.
  - 個々のインスタンスの長さは読み込んで，メンバー rec_lenの値を読まないと解らない.
  - よって，まずは本構造体が最大長であると仮定して，ext2_dir_entry_2 構造体に読み込み，rec_len の長さを得て，次のインスタンスまでポインタを進めればよい.
    - 上記がヒントですが，さらに多少の工夫が必要.

ext2_dir_entry_2

型: 1 通常ファイル, 2 ディレクトリ, 7 シンボリックリンク 他

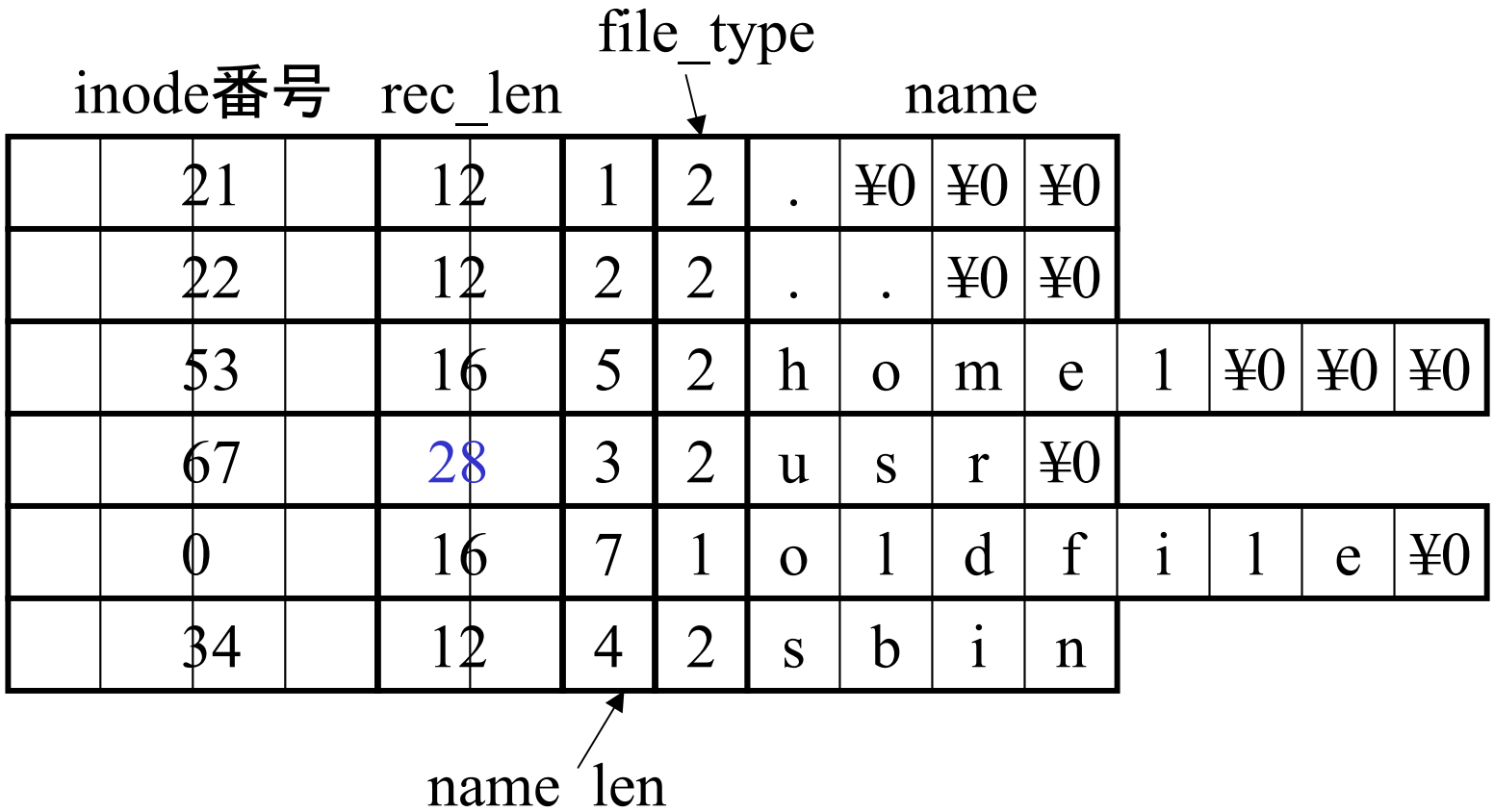
```
// include/linux/ext2_fs.h の501行目

struct ext2_dir_entry_2 {
    __u32   inode; // そのディレクトリのiノード番号
    __u16   rec_len; // 構造体のサイズ, name[]のため可変
    __u8    name_len; // ファイル名の長さ ¥0 は含まず
    __u8    file_type;              // ファイルの型番号
    char    name[EXT2_NAME_LEN]; // ファイル名
};
```

通常, 255

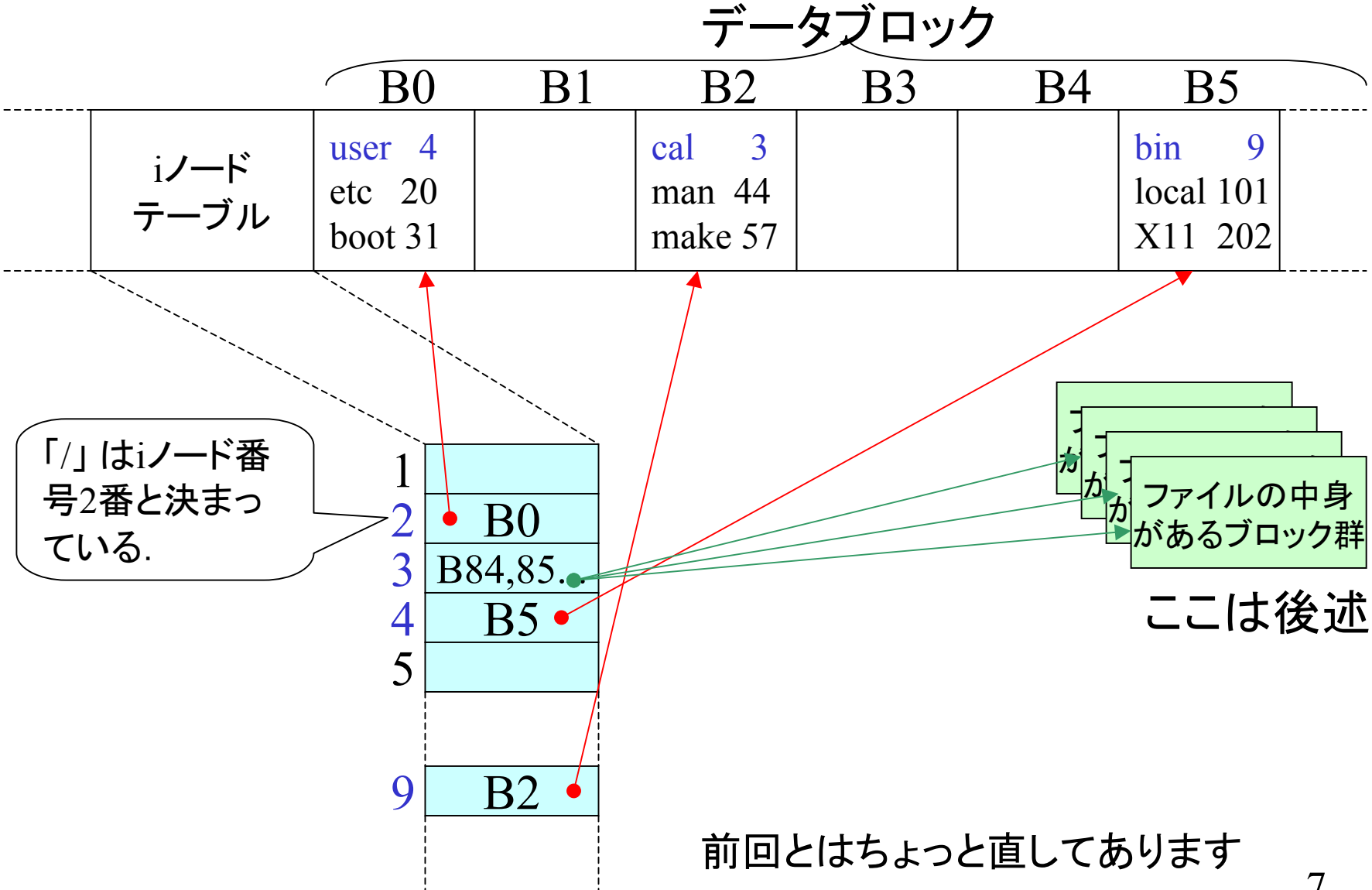
効率化のため4の倍数長になっている. 不要な部分には ¥0 文字が詰めてある.

例: とあるデータブロックの中身



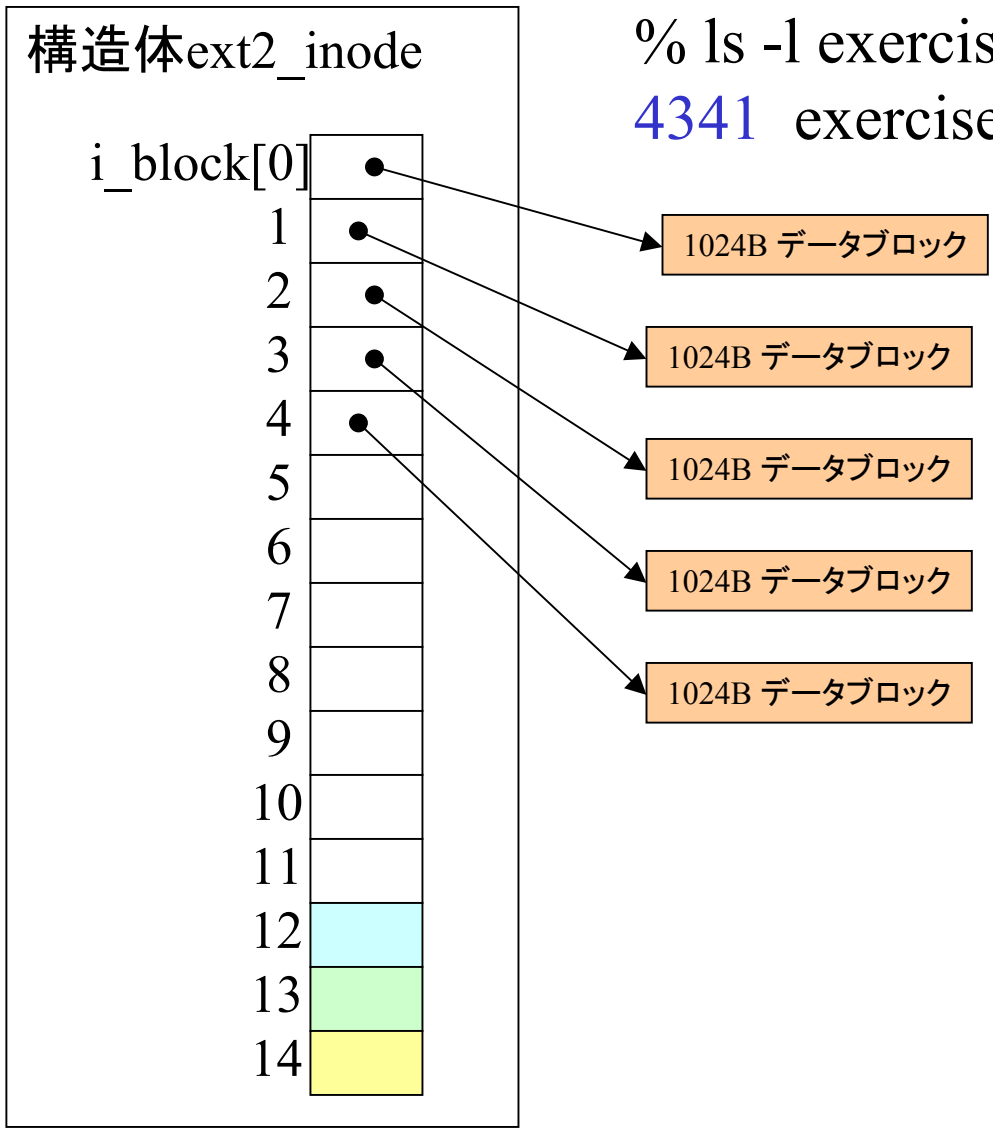
inode番号	rec_len	name_len	file_type	name
21	12	1	2	. ¥0 ¥0 ¥0
22	12	2	2	. . ¥0 ¥0
53	16	5	2	h o m e 1 ¥0 ¥0 ¥0
67	28	3	2	u s r ¥0
0	16	7	1	o l d f i l e ¥0
34	12	4	2	s b i n

例: /usr/bin/cal の番号を探す



前回とはちょっと直してあります

ファイルの格納例 (1)



% ls -l exercise01/index.html
4341 exercise01/index.html

直接ブロック5個利用.

最後のブロックは245B
しか使ってないはず.
(4341-1024*4=245)

ext2_inodeの一部

```
struct ext2_inode {
        __u16   i_mode;         /* File mode */
        __u16   i_uid;          /* Owner Uid */
        __u32   i_size;         /* Size in bytes */
        __u32   i_atime;        /* Access time */
        __u32   i_ctime;        /* Creation time */
        __u32   i_mtime;        /* Modification time */
        __u32   i_dtime;        /* Deletion Time */
        __u16   i_gid;          /* Group Id */
        __u16   i_links_count;  /* Links count */
        __u32   i_blocks;       /* Blocks count */
        __u32   i_flags;        /* File flags */
        union {
                // ** 省略
        } osd1;                         /* OS dependent 1 */
        __u32   i_block[EXT2_N_BLOCKS];/* Pointers to blocks */
        __u32   i_version;      /* File version (for NFS) */
        __u32   i_file_acl;     /* File ACL */
        __u32   i_dir_acl;      /* Directory ACL */
        __u32   i_faddr;        /* Fragment address */
        union {
                // ** 省略
        } osd2;                         /* OS dependent 2 */
};
```

include/linux/ext2_fs.h
の217行目あたりから

全部で128バイト

演習3(かも)の指針

- 基本的に数ページ前のスライド「例: /usr/bin/calの番号を探す」と同じことをする.
- それによって，該当するファイルのiノード構造体が得られる.
- そこから，配列 i_block[] の中身を使って，該当するデータブロックを得る.

おまけサンプル

- readsuper.c
  - スーパーブロックの情報を読んで画面に出力するもの.
  - バイト単位のデータを扱う参考にしてください.
- 単なるおまけではなく, この手のバイト単位データを解析するプログラムの例題にもなっています.
  - ので, ちょっとだけ解説します.

いくつかの注意

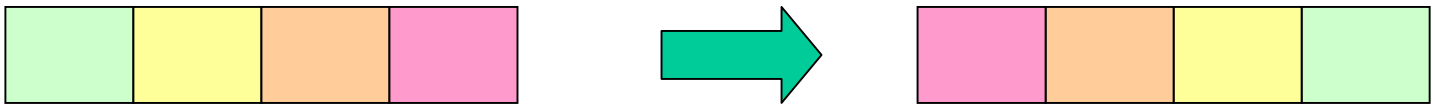
- iノードは1から順番に数えてください.
- ブロック番号は0から順番に数えてください.

バイトオーダー

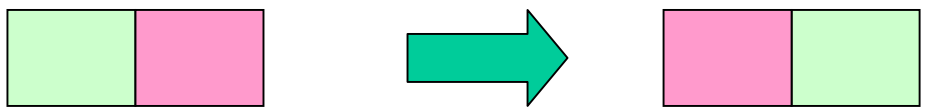
- 2バイト以上のデータを保存したり通信転送したりする場合，最上位のデータから送る場合と，最下位から送るかの違いがCPUによってある.
- 最上位: Big Endian と呼ぶ
  - PPC (Power PC, Macに搭載)のCPUはこっち
- 最下位: Little Endian と呼ぶ
  - i386 (WinPC)のCPUはこっち
- i386の場合は，データ型毎にバイト単位で順序がひっくり返っている！

マクロ rev32 rev16

rev32(x) 4byteのデータのバイトーダーをひっくり返す



rev16(x) 4byteのデータのバイトーダーをひっくり返す



ext2_os2006.h の最後のほうに入ってます.

オーダー確認プログラム

```
main(){
int x=1; // 0x00000001
  if (*(char*)&x) {
        /* little endian. memory image 01 00 00 00 */
        puts("little");
  }else{
        /* big endian. memory image 00 00 00 01 */
        puts("big");
  }
}
```

今回の演習での注意

- 今回の練習ではバイト単位のデータファイルを扱うので，バイトオーダーの問題がモロ関係する.
- マック所有が大多数なので，注意してください.
- Ext2は Little Endian (非マック型)で保存されています.