オペレーティングシステム演習

第13回（2009.07.09）

ファイルシステムと i ノード

i-node（アイノード）

- Index Node
  - （UNIX系OS）のファイル管理テーブル
  - 教科書P78

  - 特徴
    - ファイルを作成するときは、必要最小限の大きさ
      - 無駄なファイルスペースを作らない。
    - ファイル数の制約がない
    - など

i-node 構造体

```
EXTERN struct inode {
  mode_t i_mode;                /* file type, protection, etc. */
  nlink_t i_nlinks;             /* how many links to this file */
  uid_t i_uid;                  /* user id of the file's owner */
  gid_t i_gid;                  /* group number */
  off_t i_size;                 /* current file size in bytes */
  time_t i_atime;               /* time of last access (V2 only) */
  time_t i_mtime;               /* when was file data last changed */
  time_t i_ctime;               /* when was inode itself changed (V2 only)*/
  zone_t i_zone[V2_NR_TZONES];  /* zone numbers for direct, ind, and dbl ind */

  /* The following items are not present on the disk. */
  dev_t i_dev;                  /* which device is the inode on */
  ino_t i_num;                  /* inode number on its (minor) device */
  int i_count;                  /* # times inode used; 0 means slot is free */
  int i_ndzones;                /* # direct zones (Vx_NR_DZONES) */
  int i_nindirs;                /* # indirect zones per indirect block */
  struct super_block *i_sp;     /* pointer to super block for inode's device */
  char i_dirt;                  /* CLEAN or DIRTY */
  char i_pipe;                  /* set to I_PIPE if pipe */
  char i_mount;                 /* this bit is set if file mounted on */
  char i_seek;                  /* set on LSEEK, cleared on READ/WRITE */
  char i_update;                /* the ATIME, CTIME, and MTIME bits are here */
} inode[NR_INODES];
```

i-node構造体(2)

- i_num -- i-node番号
- i_uid -- ユーザID : ファイルの所有者
- i_dev -- どんなデバイスにファイルが作られているか。
- i_sp -- super_blockへのポインタ
- i_zone -- zone_number : データが書き込まれる本体部分

- それ以外のパラメータが何を意味しているか
  - コメントを読んでみる。

i_mode の使い方

```
mount.c:  bits = rip->i_mode & I_TYPE;
mount.c:  if (root_ip != NIL_INODE && root_ip->i_mode == 0) r = EINVAL;
mount.c:        mdir = ((rip->i_mode & I_TYPE) == I_DIRECTORY);  /* TRUE iff dir
 */
mount.c:        rdir = ((root_ip->i_mode & I_TYPE) == I_DIRECTORY);
mount.c:  if ( (rip->i_mode & I_TYPE) != I_BLOCK_SPECIAL) {
open.c:         switch (rip->i_mode & I_TYPE) {
open.c:  rip->i_mode = bits;    /* set mode */
open.c: mode_word = rip->i_mode & I_TYPE;
path.c:         if ( (rip->i_mode & I_TYPE) == I_DIRECTORY)
path.c:  if ( (ldir_ptr->i_mode & I_TYPE) != I_DIRECTORY) return(ENOTDIR);
pipe.c:  rip->i_mode &= ~I_REGULAR;
pipe.c:  rip->i_mode |= I_NAMED_PIPE;   /* pipes and FIFOs have this bit set */
protect.c:  rip->i_mode = (rip->i_mode & ~ALL_MODES) | (mode & ALL_MODES);
protect.c:  if (!super_user && rip->i_gid != fp->fp_effgid)rip->i_mode &= ~I_SET
_GID_BIT;
protect.c:      rip->i_mode &= ~(I_SET_UID_BIT | I_SET_GID_BIT);
protect.c:  bits = rip->i_mode;
read.c:  mode_word = rip->i_mode & I_TYPE;
read.c:  block_spec = (rip->i_mode & I_TYPE) == I_BLOCK_SPECIAL;
read.c:  block_spec = (rip->i_mode & I_TYPE) == I_BLOCK_SPECIAL;
stadir.c:  if ( (rip->i_mode & I_TYPE) != I_DIRECTORY)
stadir.c:  mo = rip->i_mode & I_TYPE;
stadir.c:  statbuf.st_mode = rip->i_mode;
#
```

i_modeに設定される値

```
/* Flag bits for i_mode in the inode. */
#define I_TYPE          0170000 /* this field gives inode type */
#define I_REGULAR       0100000 /* regular file, not dir or special */
#define I_BLOCK_SPECIAL 0060000 /* block special file */
#define I_DIRECTORY     0040000 /* file is a directory */
#define I_CHAR_SPECIAL  0020000 /* character special file */
#define I_NAMED_PIPE    0010000 /* named pipe (FIFO) */
#define I_SET_UID_BIT   0004000 /* set effective uid_t on exec */
#define I_SET_GID_BIT   0002000 /* set effective gid_t on exec */
#define ALL_MODES       0006777 /* all bits for user, group and others */
#define RWX_MODES       0000777 /* mode bits for RWX only */
#define R_BIT           0000004 /* Rwx protection bit */
#define W_BIT           0000002 /* rWx protection bit */
#define X_BIT           0000001 /* rwX protection bit */
#define I_NOT_ALLOC     0000000 /* this inode is free */
```

ビットのセット・リセット

- ビットを設定する時
  - A |= B;
  - OR演算を行うと、特定のビットだけ設定される。
- ビットをリセットする時
  - A &= ~C;
  - ビット反転の値と、算術AND演算を行うと、そのビットだけがリセットされる。
- ビットを判定する時
  - If(A & D) ・・・・・・
  - AND演算の結果は、そのビットが設定されていればTRUEになる。
- データ領域を圧縮するために、C言語でよく使われる

Super blockの情報

```
EXTERN struct super_block {
  ino_t s_ninodes;              /* # usable inodes on the minor device */
  zone1_t  s_nzones;            /* total device size, including bit maps etc */
  short s_imap_blocks;          /* # of blocks used by inode bit map */
  short s_zmap_blocks;          /* # of blocks used by zone bit map */
  zone1_t s_firstdatazone;      /* number of first data zone */
  short s_log_zone_size;        /* log2 of blocks/zone */
  off_t s_max_size;             /* maximum file size on this device */
  short s_magic;                /* magic number to recognize super-blocks */
  short s_pad;                  /* try to avoid compiler-dependent padding */
  zone_t s_zones;               /* number of zones (replaces s_nzones in V2) */

  /* The following items are only used when the super_block is in memory. */
  struct inode *s_isup;         /* inode for root dir of mounted file sys */
  struct inode *s_imount;       /* inode mounted on */
  unsigned s_inodes_per_block;  /* precalculated from magic number */
  dev_t s_dev;                  /* whose super block is this? */
  int s_rd_only;                /* set to 1 iff file sys mounted read only */
  int s_native;                 /* set to 1 iff not byte swapped file system */
  int s_version;                /* file system version, zero means bad magic */
  int s_ndzones;                /* # direct zones in an inode */
  int s_nindirs;                /* # indirect zones per indirect block */
  bit_t s_isearch;              /* inodes below this bit number are in use */
  bit_t s_zsearch;              /* all zones below this bit number are in use*/
} super_block[NR_SUPERS];
```

スーパーブロックの保持する情報

- 利用可能なi_node数
- デバイスの最大サイズ
- そのデバイスの最大ファイルサイズ
- i_nodeのブロック数
  - など

CLEAN/DIRTYって何？

- i_node構造体: 変数 i_dirt
  - 取る値は、CLEAN, DIRTY
  - ほとんどの所で、DIRTYに設定されている。
  - 唯一CLEANに設定されている場所は？
    - rw_inode()
  - メモリ上のi_node情報と、デバイス上のi_node情報が、一致していると
    - ⇒ CLEAN
  - メモリ上のi_node情報だけが更新されている
    - ⇒ DIRTY

試験の方式

- コマンドレファレンスほか、返却済みのレポートのみ参照可
- インターネット検索可、ラボのPCでのMINIX環境を使用して課題を解く。

- 問題1：OSの用語の解説を求める。（15点）
- 問題2：個別課題（学籍番号ごとに、指定された関数の本体定義の場所を探す。当日に画面に課題を掲示）（10点）
- ソースコードを検索し、ソースコードの内容を答える問題 10点×4（40点）
  - 課題は、10問設定し、そのうち4問を選択する。

問題3の例

- MINIXでの Super Blockの配列サイズはいくつか？
  - 問題ヒント: struct super_blockから調べる。
  - Super_block が、ファイルシステムに関係している、という知識から/usr/src/fsを調べる。
  - Grep でsuper_blockを検索しsuper.hを読む
  - NR__SUPER が、配列のサイズ
  - Grep で NR_SUPERを検索する。
  - /usr/src/fs/const.h でのdefine文を読む。
    - 答え:8

演習課題(1)

- 試験問題(問題3)と同様の課題を、ファイルシステムから出題します。試しに解いてみて下さい。
  - TAには操作方法を質問して下さい。中身は、ノートなどを参照して、自分で工夫してください。
- 例題1:
  - ユーザが関数fopen()をコールしてファイルを作成しようとした時に、OSに渡されるシステムコール番号はいくつか？

    また、fopenでファイルを作成する際に、システムコールが発行されるまでを、トレースせよ。
- 例題2:
  - FLOPPYディスク関係の割り込みが入った時に、発生する割り込み処理番号は何か。
  - 調べ方から説明せよ。

演習課題（2）

- 例題3：
  - ユーザとして、kusanagi, katori, kimuraの3人を、このユーザ名で登録したい。
  - kusanagiが作成したrecipeというファイルは、katoriが閲覧することもあるが、kimuraには見せたくない。
  - どのように登録し、設定したら良いか。その登録操作（または結果）と、ファイルrecipeへの操作を説明せよ。
    - ユーザIDなどは任意とする。（ただし、説明は必要）
- 例題4：
  - MINIXの画面の背景色を灰色、文字色を濃い青（紺色）に設定したい。
  - どのファイルの何行目を、どのように修正したらよいか。説明せよ。