Gfarm カーネルドライバ 設計と
メタデータサーバ通信基本機能実装作業

プログラム設計書

数理技研

2012 年 3 月 28 日

目 次

はじめに

本ドキュメントは、ユーザーランドで動作する Gfarm ファイルシステムをカーネル内 FS から直接呼び出すことによって性能向上を図るためのカーネルドライバの仕様を記述するものである。

1　システム概要

以下に本システムのモジュール関連図を示す。

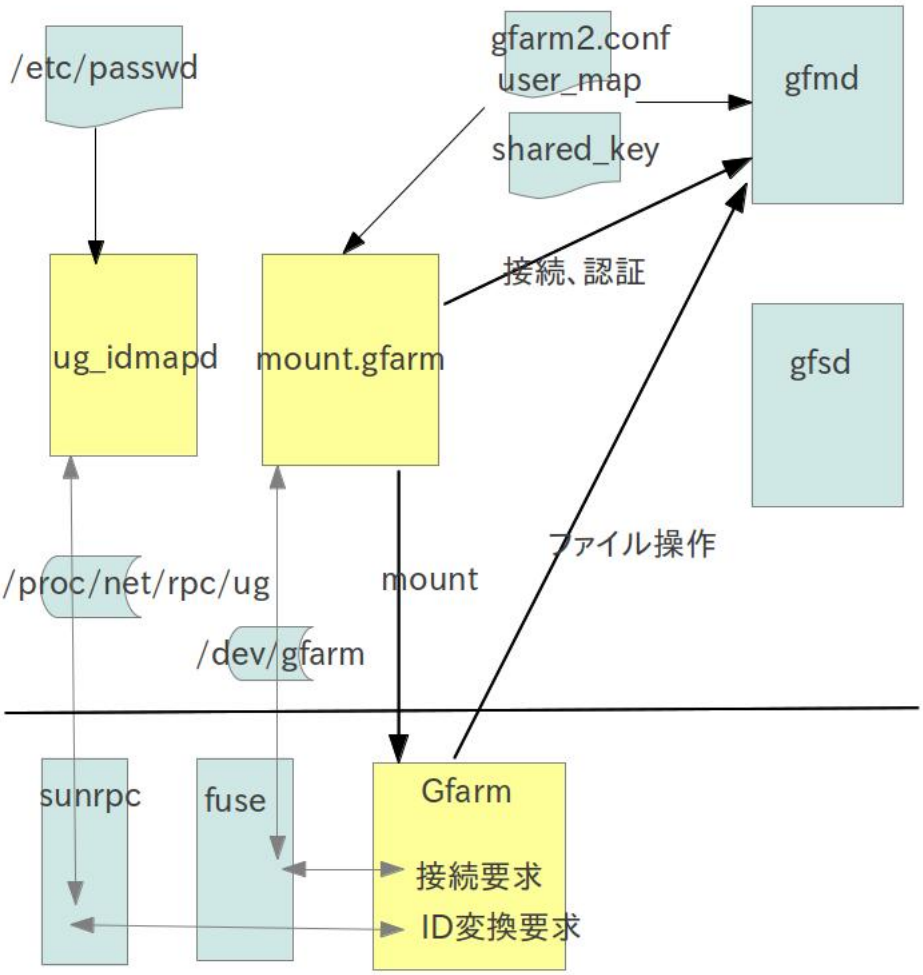

図 1: モジュール関連図

Gfarm では各種認証機構をサポートしているが、カーネル内で認証ライブラリを利用するのが難しいことから、今回開発では認証はユーザーランドで行う。将来は、簡単な認証機構を導入するなどして、カーネル内に閉じて接続を行うことも考えられる。但し、その場合も名前解決などはユーザーランドで行うことになる。

認証はユーザー毎に行うので、サーバーとの接続もユーザー毎に行う。同一ユーザーで複数の接続を行うことも考えられるが、ポート数の問題などもあり、今回はユーザ毎に一つの接続とする。

今回開発するモジュールとしては以下のものがある。

- gfarm

  カーネル内ファイルシステム。

- mount.gfarm

  ユーザーランドのヘルパーデーモンで、mount コマンドから起動される。mount 後はカーネルからの接続要求を待ち受け、メタデータサーバに接続し、認証を行う。

- ug_idmapd

  ユーザーランドのヘルパーデーモンで、ユーザー ID、グループ ID の変換を行う。

ユーザーランドとの通信方法については、以下の方式とする。

- 接続要求

  カーネルドライバが、メタデータサーバとの接続を依頼するインタフェースには、fuse モジュールがエクスポートしているインタフェースを利用する。

  これは、アプリケーションが /dev/gfarm をオープンし、このファイルを読み込むことによって、カーネルモジュールの要求を得、ユーザー空間でこれを解決して、当該ファイルに応答を書き込む仕組みである。

  ファイルとファイルシステムを結びつける方法は、アプリケーションがオープンファイルデスクリプタを mount 時に通知することで行う。

  この方法は、mount と結びついて安全であるが、他方、アプリケーションが異常終了した際の救済手段を別途考える必要がある。

  接続要求は将来開発でなくなるので、救済手段は講じない。

- ID 変換要求

  カーネルドライバが、uid, gid と名前の変換を依頼するインタフェースには sunrpc モジュールがエクスポートしているインタフェースを利用する。

  これは、アプリケーションが /proc/net/rpc/配下のファイルをオープンし、このファイルを読み込むことによって、カーネルモジュールの要求を得、ユーザー空間でこれを解決して、当該ファイルに応答を書き込むとともに、sunrpc モジュールがキャッシュ機構を提供し、要求応答を一定期間キャッシュし、この検索再利用を可能とさせる仕組みである。

  本システムでは、以下のインタフェースを作成する。

/proc

```
/proc/net/rpc/ug.idtoname:
        channel    要求チャネル
        content    キャッシュ情報
        flush    キャッシュフラッシュ指示

/proc/net/rpc/ug.nametoid:
        channel
        content
        flush
```

2 mount

mount は メタデータサーバ毎に行う。同一のメタデータサーバーへの複数の mount は特に禁じない。

mount は ユーザーランドでメタデータサーバーとの接続を確認した上で、mount システムコールを発行するので、接続するユーザーを指定した mount となる。

2.1 mount.gfarm

mount.gfarm は mount コマンドから mount -t gfarm を指定された時に呼び出される。
オプションは以下のものがある。

- luser=name

  メタデータサーバとの接続を行うユーザーのローカルユーザ名。指定されなければローカルユーザ ID から得る。

- uid=uid

  メタデータサーバとの接続を行うユーザーのローカルユーザ ID。指定されなければローカルユーザ名から得る。得られれなければ実行ユーザー ID から得る

- key_path=path

  共通鍵方式の鍵ファイルのパスを指定する。指定されなければ luser のホームディレクトリの鍵ファイルを参照する。

- conf_path=path

  コンフィグレーションファイルのパスを指定する。指定されなければ luser のホームディレクトリのファイルを参照する。

mount の一般的なオプションも受け入れるが、動作はメタデータサーバに依存する。

mount.gfarm の mount 動作概要

1. コンフィグレーションファイルを読み込む。
2. 指定されたユーザーでメタデータサーバに接続する。
3. 認証を済ませる。
4. /dev/gfarm をオープンする。
5. 接続デスクリプタとデバイスデスクリプタを mount 引数に加える。
6. コンフィグレーションファイルとグローバル名変換ファイルを読み込み、mount 引数に加える。
7. mount システムコールを発行する。
8. /etc/mtab に登録する。
9. 接続要求待ちループに入る。
10. /dev/gfarm から接続要求があれば fork する。
11. fork した子
    (a) 指定ユーザーのための接続を行う。
    (b) 認証を行う。
    (c) ファイルデスクリプタを /dev/gfarm に書き込む
    (d) 終了する。
12. /dev/gfarm から終了を読み込んだら終了する。

2.2 gfsk_mount_data

mount のためのオプションはバイナリデータとする。

gfsk_mount_data

```
struct gfsk_strdata {
        int     d_len;
        char    *d_buf;
};
struct gfsk_fbuf {
        struct gfsk_strdata f_name;          /* file name */
        struct gfsk_strdata f_buf;           /* file content */
};

#define GFSK_VER1       0x30313031
#define GFSK_VER        GFSK_VER1
struct gfsk_mount_data {
        int     m_version;
        char    m_fsid[8];              /* out: file system id */
        struct gfsk_fbuf m_fbuf[GFSK_FBUF_MAX];
        int     m_dfd;                  /* dev fd */
        uid_t   m_uid;
        char    m_uidname[GFSK_MAX_USERNAME_LEN];
        int     m_mfd;                  /* meta sever fd */
        char    m_host[MAXHOSTNAMELEN];    /* connected host by m_fd */
        int     m_optlen;
        char    m_opt[1];               /* option string */
};
#define GFSK_OPTLEN_MAX (PAGE_SIZE - sizeof(struct gfsk_mount_data))
```

- m_fbuf はコンフィグレーションファイル、グローバル名変換ファイルなどを mmap してカーネルに内容を通知するための構造である。

  カーネル空間を圧迫する恐れがあるので望ましくないが、今後の検討課題とする。

- m_dfd は /dev/gfarm を開いたファイルデスクリプタで、カーネルからの接続要求を受け付けるためのものである。

- m_mfd は メタデータサーバに接続したファイルデスクリプタでカーネルに引き渡すものである。

- m_uid,m_uidname はメタデータサーバに接続したユーザー情報である。

- m_host は接続したメタデータサーバである。

- m_opt は一般的な mount オプション文字列である。

2.3 gfarmfs

Gfarm カーネルドライバ版 の mount 動作概要

1. module ロード時

   (a) modprobe のオプションで設定パラメタを渡される。設定パラメタは以下である。

- ug_timeout_sec=N
  ug_idmapd からの応答待ち時間を設定する。デフォルトは 1 秒である。
- log_level=N
  ログレベルを指定する。0 はログが少なく 7 は多い。

(b) register_filesystem(file_system_type) で get_sb, kill_sb 関数を登録する。

(c) fuse 利用 のための登録を行う。

(d) ug_idmapd のための登録を行う。

(e) 将来、dns が必要になった場合は、そのインタフェースを/proc につくる。

2. mount 時

(a) mount.gfarm から mount システムコールが発行される。

(b) マウントオプションをチェックする。
mount は複数可能とする。ただし、今期試験は単数のみとする。
mount の単位は本来 メタサーバ (グループ) 毎かもしれないが、nfs 同様、特にチェックはしない。

(c) ファイルシステム固有データ構造を初期化する。

(d) コンフィグレーションファイルに従い初期化する。

(e) 渡された接続 fd でサーバーから root ディレクトリ情報を得る。

(f) fill_super で fs 情報を得る。

2.4 umount

MNT_DETACH はサポートしない。MNT_FORCE が指定されたら通信中のプロセスは起こし EIO で戻す。
umount 時は gfskd に通知した上、gfskd の file struct の private メンバ (gfarm_fsctx を指している) をクリアし、以降の read を失敗させる。

3 データ構造

3.1 外部変数閉じ込め

既存の Gfarm で外部変数となっていて、mount 毎に保持する必要のあるものは struct gfarm_context に閉じ込める。現在ローカルに定義されている構造体については、ポインターメンバーとして、各初期化時にアロケートし、gfarm_context からポイントする。
gfarm_context を関数引数として連れ回すのは大変なので、task コンテクストからとれるようにする。struct task の journal_info がファイルシステムでテンポラリに利用可能なので、これを利用する。

gfarm_context

```
struct gfarm_context {
    /* global variables in config.c */
    char *metadb_server_name;
    int metadb_server_port;
    char *metadb_admin_user;
    char *metadb_admin_user_gsi_dn;
.....
};
#ifdef __KERNEL__
#define gfarm_ctxp (gfsk_task_ctxp→gk_gfarm_ctxp)
#define errno       gfarm_ctxp→gc_errno
#else
extern struct gfarm_context  *gfarm_ctxp;
#endif
```

カーネル内では各システムコールの入り口となる関数で、gfsk_task_context をスタックに作成し、current task に設定し、戻りでクリアする。

3.2 ファイルシステムデータ構造

linux のファイルシステム関連のデータには以下のものがある。

1. struct super_block

   マウント毎のファイルシステム情報。

   fs 用の void *s_fs_info がある。

2. struct dentry

   ディレクトリキャッシュでネガティブキャッシュもある。lookup で 作成し、inode_operations.lookup に渡して、inode を結びつけさせる。

   fs 用の void *d_fsdata がある。

3. struct inode

   inode_operations.lookup で dentry に結びつける時に fs で作成する。dentry を解放する dentry_iput 時に inode があれば、d_iput が定義されていれば呼びだす、inode 解放は fs に任される。d_iput が定義されていなければ inode の参照数を落とし、0 なら drop_inode が定義されていれば削除を任せる。

   fs 用の void *i_private もあるが、fs 固有 inode に含ませる実装も多い。

4. struct file

   ファイルのオープンコンテクストで、ファイルオープン時、作成後 file_operations.open を呼び出す。最後のクローズ時、file_operations.release を呼び出す。

   fs 用の void *private_data がある。

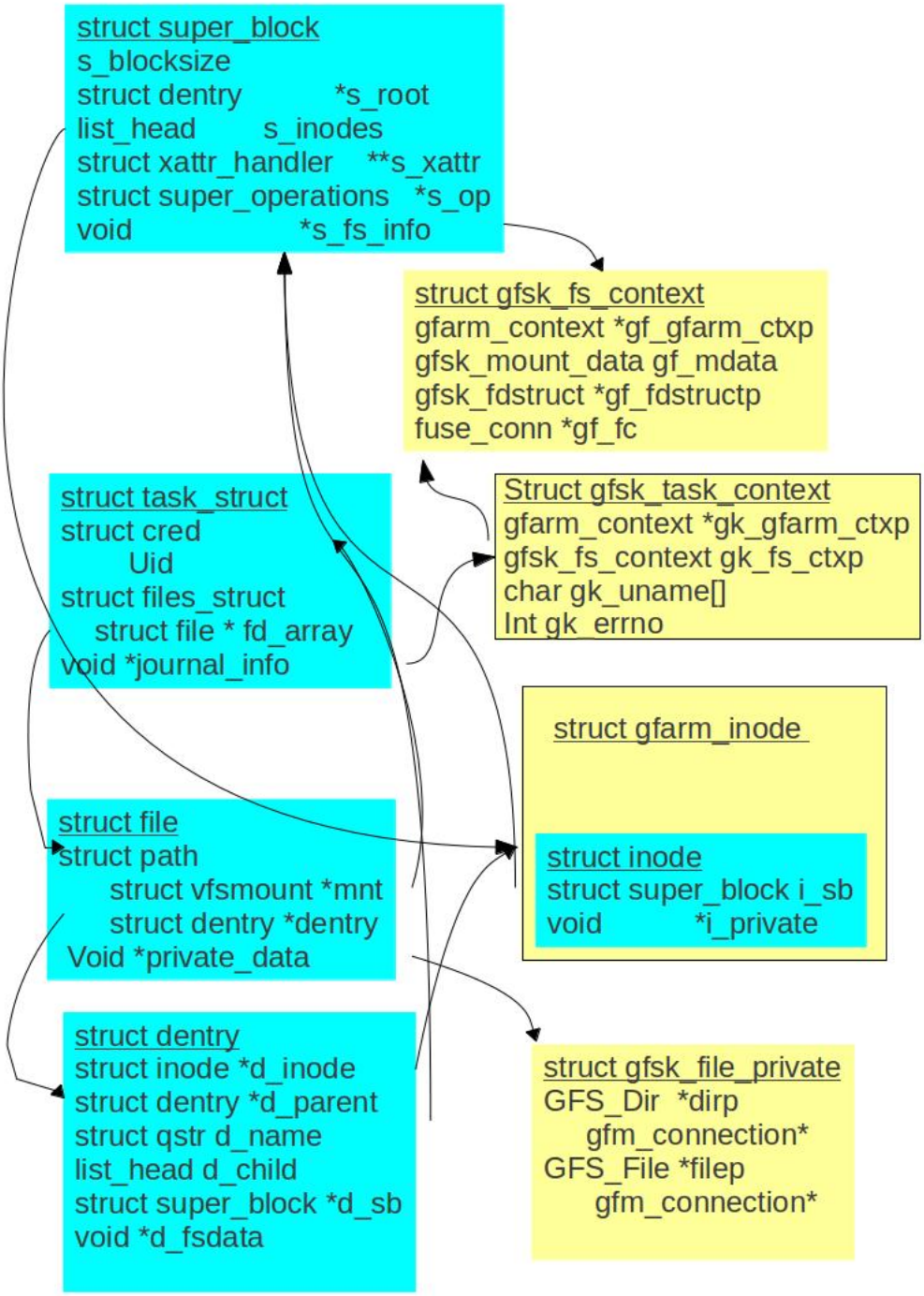

図 2: データ構造関連図

各 linux のデータ構造に対応して図 2 のように固有のデータ構造をつくる。

- gfsk_fs_context
  カーネルドライバ固有ファイルシステム情報
  - struct gfsk_mount_data gf_mdata
    mount 引数をカーネル内で保持する。コンフィグレーションファイルもここからマップされる。
  - struct gfarm_context *gf_gfarm_ctxp
    Gfarm fs の外部変数を閉じ込めた fs 毎のデータ。
  - struct gfsk_fdstruct *gf_fdstructp
    ファイルデスクリプタを struct file ではなく、int で受け渡すための管理データ。

  - struct fuse_conn *gf_fc
    mount.gfarm に依頼するためのインタフェースコンテクスト。

  ユーザー ID からサーバー接続情報を探すためのリストは、gfm_server_cache にユーザー名があるので、 これを利用する。

- struct gfsk_task_context

  current task に結びつけるコンテクスト情報で、ファイルシステム IF に入ったときに設定し、関数から出るときにクリアする。

  - struct gfsk_fs_context *gk_fs_ctxp
    操作中のファイルシステム。
  - char gk_uname[GFSK_USERNAME_MAX]
    当該タスクのユーザー名キャッシュ。
  - int gk_errno
    Gfarm ライブラリ内で参照される errno データ。

- struct gfarm_inode

  - struct inode inode
    inode 管理は vfs 層の inode 管理を利用する。
    linux では inode の作成は ファイルシステムが用意していればそれを、そうでなければ inode_cache から作成する。
    さらに、ino をファイル識別に用いるファイルシステムのために ino による inode 管理も提供しており、iget_locked で探索、作成を行える。
    Gfarm では inum, igen が用意されているので、inum を利用する。 igen はファイルシステム内でチェックし、更新されていれば古いファイルを捨てる処理を行う。
    コンパウンド要求の途中で得られたのみの情報でも inode を作成する。
  - uint64_t i_gen
    vfs inode に持てない i_gen 情報。
  - loff_t i_direntsize
    ページキャッシュに保持しているディレクトリファイルのサイズ。

- struct gfsk_file_private

  プロセス毎のオープンファイル情報

  - union GFS_Dir *kf_dir; GFS_File *kf_file; u;
    オープンファイル情報およびサーバーコンテクスト。

  これを dentry につなげて、プロセスが異なっても、ベースディレクトリ情報として用いる方法については、複雑になるので検討しない。

4 処理概要

4.1 構成ファイル

Gfarm クライアントは以下のような設定ファイルを持っている。

gfarm2.conf	Gfarm 設定ファイル
local_user_map	グローバル/ローカルアカウントのマップファイル
local_group_map	グローバル/ローカルグループのマップファイル
.gfarm_shared_key	ユーザー毎の認証鍵ファイル

各種ファイルの扱いは以下のようにする。

- gfarm2.conf

  マウント時にファイル内容を引数として渡し、カーネル内でキャッシュする。

- local_user_map local_group_map

  マウント時にファイル内容を引数として渡し、カーネル内でキャッシュする。複数ファイル、更新には当面対応しない。

- .gfarm_shared_key

  ヘルパーデーモンが読み出す。

4.2 接続管理

メタデータサーバとの接続は、認証が接続毎に行われること、状態がメタデータサーバで接続毎に管理されていることからカーネルドライバでもユーザー毎に接続を張るものとする。

1. 各 mount 毎にユーザー別の接続を張る。
2. 1 ユーザーで複数の接続については当面考えない。
3. 接続の使用は gfp_cached_connection にロックを設け、要求/応答でロックし、他の要求は待たせる。
4. 一般には、gfp_cached_connection_acquire() でロックし、gfp_cached_or_uncached_connection_free() でアンロックする。uncached_connection はロックは不要であるが、その場合、コストも低いので区別しない。
5. コンパウンド要求はひとかたまりのものとして占有するため、compound_fd_op, compound_file_op などをロックで括る。
6. オープンファイルに関しては、file-⟩private にユーザ接続情報を持たせる。
7. 新しいユーザーの場合、ヘルパーデーモンに、ユーザー ID を渡し、接続を依頼する。

   ユーザー ID での問い合わせを受けたヘルパーデーモンが fork して 当該ユーザーに setuid() する。

   ユーザー空間でメタデータサーバにつなぎ、必要な認証を済ませたのち、接続 fd をカーネルドライバに伝える。

   この時、接続したプロセスのコンテキストで fd を file に変換して fd 管理に登録する。

   カーネルドライバは socket の file 構造体を保持して送受信を行う。file 構造体から引き継ぐので、作成プロセスが終了しても構わない。
8. 接続の再利用管理は Gfarm に任せる。現在は個数管理なので、将来ユーザー数に合わせた管理が必要になる。
9. サーバーとの接続が切れた場合もユーザー空間に依頼し再接続を行う。認証鍵を問い合わせ、メタデータサーバリストからサーバを探し、接続する。

4.3 ファイル管理

Gfarm のファイルは mount 毎に個別の inode に対応させる。即ち、同一 mount で複数の接続があっても同じファイルには同じ inode を対応させる。

4.3.1 ファイルキャッシュ

ファイルの存在や属性のキャッシュに関して、今期は競合を考慮しない。ここでの競合とは、他の方法でのファイルシステムの利用、即ち、ユーザーランドでのメタデータサーバの利用や、FUSE でのアクセス、複数マウントによるアクセスである。

4.3.2 uid, gid

ファイル属性の uid, gid は getattr の延長で名前から id に変換する。
名前の 変換には sunrpc のキャッシュと問い合わせ機構を利用し、ユーザーランドのヘルパーデーモン ug_idmapd が変換する。キャッシュ時間はユーザーランドで指定する。

4.3.3 通常ファイル

Gfarm では、通常ファイルは struct gfs_file で管理されており、ほぼ linux の struct file に 対応する。
通常ファイルの buffer に関しては検討の余地があるが、いずれにしろ通常ファイルのサポートは来期の課題とする。

4.3.4 ディレクトリ

Gfarm では、ディレクトリは struct gfs_dir で管理されており、同時にディレクトリキャッシュも持っている。
本システムでは Gfarm のキャッシュ機構は用いず、linux のページキャッシュを利用する。

4.4 名前によるファイル操作

カーネル内の名前によるファイル操作は常に親ディレクトリと子ファイルという関係でファイルシステムに渡される。一般にカレントディレクトリからの相対パスがユーザーから渡され、これを vfs 層のディレクトリキャッシュを使いながら検索し、ここに存在しない時はファイルシステムに lookup で問い合わせながら、最後のセグメントに関してファイルシステムに操作が依頼される。
一方、Gfarm では一般にファイルパスへの操作として行う。コンパウンド要求で、ルートディレクトリから 1 セグメントずつファイルの存在を確かめながら、最後のセグメントの操作を依頼している。Gfarm には inode 番号 によるファイル操作はなく、ファイルハンドルの概念もサポートしていない。サーバー内では、Lookup 操作でも見つけたファイルをオープンファイルとして保持して、次のファイル操作のベースとしているが、この fd は要求しない限りクライアントには返されず、次のファイル操作で上書きされる時にクローズされる。
従って、名前によるファイル操作は次のような手順になる。

1. vfs 層で現在のベースディレクトリを得る。
2. セグメントのある間繰り返す (path_walk)。
   (a) vfs 層で dentry　キャッシュを参照する。
   (b) 存在しなければファイルシステムの lookup を呼び出す。
      i. マウントルートからのパスを生成し、コンパウンド要求を出す。
      ii. 得られた途中のディレクトリに関しては、inum, igen, mode で inode を作成し、dentry に結びつける。
   (c) ファイルシステムが revalidate を指定していれば呼び出す。
   (d) dentry に inode が存在しなければ ENOENT。
   (e) シンボリックリンクならリンクを追う。(path_walk)
3. ファイルシステムのファイル操作を呼び出す。
   (a) マウントルートからのパスを生成し、コンパウンド要求を出す。

カーネルドライバでは inode が重要なファイルオブジェクトとなるが、Gfarm での stat_cache との役割が重なる。inode 管理を行う場合、readdir の結果格納 (gfs_stat_cache_enter_internal0) や stat 取得関数が異なってくる。
カーネルドライバでは Gfarm のキャッシュ機構ではなく、inode によるキャッシュを行う。

4.5 readdir

readdir で得られるエントリー情報はページキャッシュに保存する。ユーザーの読み出しオフセットとページの変換を簡単にするために、名前を固定長としたエントリー情報をページに詰める。エントリー情報はページ境界を跨らずギャップを設ける。

Gfarm では複数回に分かれる readdir はサーバー側で管理され、クライアント側で読み込みオフセットを指定することができない。このため、同一ユーザーが複数のプロセスを走らせている場合は、readdir の連続性が損なわれる。また、異なるユーザーが同一ディレクトリを参照する場合も、readdir を継続することができない。

この制約のため、本システムでは、ディレクトリーの読み出しがあった場合には一気に全エントリーを読み出しキャッシュする。後にディレクトリの mtime が変わった場合は、キャッシュを捨てる。

また、readdir で得られたファイル属性は、dentry と inode に保存する。

5 修正方針

本システムは Gfarm ライブラリをカーネル内に持ち込み、カーネルドライバとして動作させるものである。修正に当たっては以下の方針と制約で臨んだ。

- ユーザー空間ライブラリはカーネル内に持ち込まない。

  このため、認証などでサポートできないものが生じた。また、DNS を使うための仕組みが必要などがあり、サーバ接続がユーザー空間にでてしまった。

- 浮動小数点はサポートしない。

  このため、スケジューリングなど今後に問題を残している。

- Gfarm 本体ソースに細かな ifdef を持ち込まない。

  このため、カーネルモジュールソースツリー内に /usr/include を模したヘッダファイルを置き、ユーザー空間ライブラリインタフェースを吸収した。

- 既存の実装、ツールに重なる開発は避ける。

  ユーザー空間とのインタフェースに fuse や sunrpc キャッシュ機構を利用した。