pNFS Layer (SAL)

Design Document

Author : Philippe Deniel (philippe.deniel@cea.fr)

Version 0.1

Abstract : this document presents the design proposal for the functions required for managing pNFS implementation in nfs-ganesha

1 Why such an API

The pNFS part of the NFSv4.x protocol can be implemented with various “flavors” (using NFS as storage, or devices accessed in a “block” way, or via OSD2). Each flavor is very different from the other but roughly, with a high-level approach, it is possible to make a common and generic API that could be mapped on top of every flavor. Such an API is required since part of the pNFS management is to be made in Cache_Inode (allocation / destruction of storage resources at files's creation and deletion) and in the NFS layer (implementation of pNFS specific NFSv4.x operations).

This API is made of three part:

- what you need to get pNFS specific configuration element from the configuration items
- what you need to manage pNFS related storage resources (creation, deletion, truncation)
- what you need to fill in the structure inside the NFSv4.x/pNFS operations

2 Content of the API

This section describes the structure and calls specific to pNFS.

2.1 Structures

- pnfs_parameter_t: contains the parameter required to configure the pNFS engine
- pnfs_fileloc_t: contains all information required to locate “pNFS storage” related to a file envolved in pNFS access.
- pnfs_file_t: represents a file managed by pNFS
- pnfs_client_t: represent the information a thread need to use pNFS. One such structure should exist per thread.

All of these structure are union for now. Each field of the union is designed to be dedicated to a specific pNFS flavor. For later use with dlopen (the pNFS flavor is associated with a specific pNFS

shared object), a 'char opaque[constant]' will be added (methodology to be followed is the same as for FSALs provided as shared objects).

2.2 API functions

All functions return a 32 bits integer. This value is to be mapped on the NFSv4.x error (to avoid later conversion).

2.2.1 Configuration specific functions

- int nfs_read_pnfs_conf(config_file_t in_config,
  pnfs_parameter_t * pparam);

  This function reads the pNFS configuration from the configuration file for the given pNFS flavour.

2.2.2 pNFS module management functions

- int pnfs_init(pnfs_client_t * pnfsclient,
  pnfs_layoutfile_parameter_t * pnfs_layout_param) ;

  This function does the initialization step for a given pNFS library

- void pnfs_terminate();

  This function shuts down the pNFS library.

2.2.3 pNFS storage resources functions

- int pnfs_create_file(pnfs_client_t * pnfsclient,
  pnfs_fileloc_t * pnfs_fileloc, /* IN */
  pnfs_file_t * pnfs_file) ; /* OUT */

  This function creates a file. The pnfs_fileloc_t contains hints to be used during resources allocation.

- int pnfs_remove_file(pnfs_client_t * pnfsclient,
  pnfs_file_t * pfile) ;

  Remove a pNFS file (pnfs_file_t has been build with pnfs_create_file or pnfs_lookup_file).

- int pnfs_lookup_file(pnfs_client_t * pnfsclient,
  pnfs_fileloc_t * pnfs_fileloc,
  pnfs_file_t * pnfs_file) ;

  Gets pnfs_file_t from a pnfs_file_loc_t. This is useful as a server restarts and gets storage

allocated by a former instance of the server.

- int pnfs_truncate_file(pnfs_client_t * pnfsclient,
  size_t newsize,
  pnfs_file_t * pnfs_file) ;

  Truncates the storage associated with a pnfs_file_t.

2.2.4 pNFS protocol functions

- void pnfs_encode_getdeviceinfo(pnfs_client_t * pnfsclient ,
  char *buff,
  unsigned int *plen) ;

  Encodes the reply for OP4_GETDEVICEINFO.

- void pnfs_encode_layoutget(pnfs_client_t * pnfsclient ,
  pnfs_file_t * pnfs_file,
  char *buff,
  unsigned int *plen) ;

  Encodes the result for OP4_LAYOUTGET.