Very Simple Network File System (VSNFS)

KarthikBalaji Gandhi, Prabakar Radhakrishnan, Praveen Krishnamoorthy
Department of Computer Science, Stony Brook University,
Stony Brook, NY – 11794

{findkb, prabakarcse, kpraveen85}@gmail.com

1. Abstract:

The need for accessing data stored in a remote system in the network has increased manifolds over the last two decades. The success story scripted by SUN Microsystem's Network File System(NFS) stands testimony to the very fact stated above. The sheer complexity involved in designing and implementing a filesystem that too a network based filesystem fascinated and motivated us to implement a bare minimal Network File System by name Very Simple Network Filesystem(VSNFS)

2. Keywords:

Very Simple Network Filesystem, Network Filesystem, Remote Procedure Call, External Data Representation

3. Introduction:

Network filesystem enables file access over a network through the usage of Remote Procedure Calls. Notable network filesystems include Sun's Network File System(NFS), Andrew FileSystem, Common Internet FileSystem(CIFS). The modern day Network Filesystems are very complex and hard to comprehend. We have implemented a bare minimal network filesystem that caters to the user request for the remote access of files. It is always useful to have a design template which can be extended and tuned for specific requirements.

Our design is loosely based on the design of NFSv2 specification(RFC#1094). VSNFS is implemented as two loadable kernel modules one for the server and one for the client for the Linux Kernel 2.6.31

4. Background:

Though the goal of both the Network Filesystem and disk based filesystem is to provide the user with the easy management of files and data, the way in which it is achieved is completely different. The Network Filesystem has a client and a server part wherein the client requests for data, and the server in turn contacts the Virtual Filesystem Switch(VFS) in the server machine which in turn calls the specific disk based filesystem(eg, ext3, ext2..) and the requested data is propagated back to the client. The client interface tries its best to hide the very fact that the data requested by the user has to be fetched from a remote system in the network. The Network filesystem acts as a intermediate agent between the client program and the actual disk based filesystem in the server side by locating the files and transports the data

5. Design:

There are three important parts in the design of the Very Simple Network FileSystem namely,

- VSNFS Protocol
- VSNFS Server
- VSNFS Client

we are using the Sun Remote Procedure Call (RPC) mechanism to invoke the procedure in the server from the client side. Though the remote procedure call framework is transport protocol independent we are using only TCP/IP as our transport mechanism. We have chosen TCP over UDP, because of the high reliability offered by TCP even when the network is highly congested.

The protocol is defined in terms of set of procedures, their arguments and results. The two categories of VSNFS procedures that we have implemented are Filesystem level operations(eg,mount) and File level operations(eg,VSNFS_LOOKUP, VSNFS_READDIR).

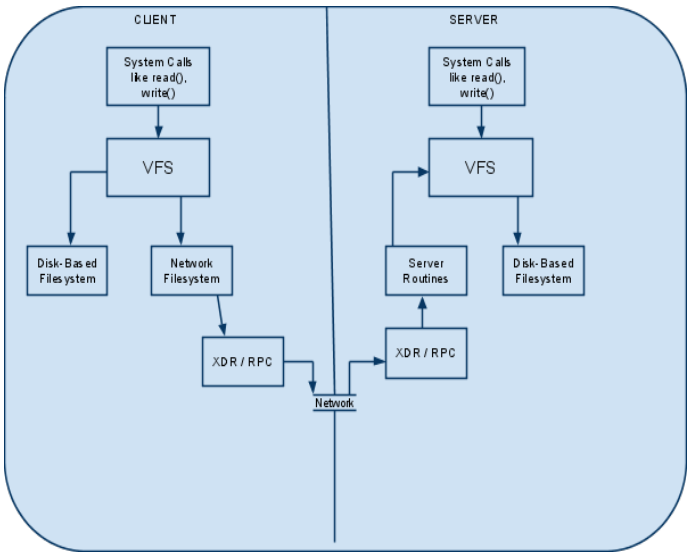

Figure 1: VSNFS Design Architecture

An ideal control flow, when a read() or any other system call is issued in the client system is picturised by the above figure. The VSNFS client requests for the data through the Remote Procedure Call (RPC) and the data transmitted uses External Data Representation(XDR). One of the goals of a network filesystem is to cater the needs of the client irrespective of the Operating System it runs on and the architecture. The XDR ensures that the data transmitted is correctly interpreted irrespective of the "Endianness" on the other side.

we support very few features in the initial version of VSNFS with the following assumptions.

- We limit the number of files in root directory.
- There is no permission checks for clients.
- Only one mount point.
- Can serve only one client.

6. Implementation:

The Implementation of VSNFS could be reached at http://code.google.com/p/vsnfs/

The file handle is the basic identification structure which uniquely identifies a file/directory in the server. The client cannot interpret the file handle but just saves it and passes to the server when some operation has to be done at later point in time. The server interprets the filehandle and does the requested operation on the file/directory

```
struct vsnfs_fh {
        unsigned char data[VSNFS_FHSIZE];
        int type;
        };
```

The variable *data* holds the inode number of the file/directory and *type* specifies if it corresponds to a file or a directory.

6.1 VSNFS Server:

The VSNFS server has been implemented as a loadable kernel module. When the module is loaded, a RPC service (Kernel Daemon) is started. The module is unloaded when the service is stopped. The service runs in a single thread. This is so because the server is designed to cater to just one client. The RPC procedure runs in the context of this service thread. The number of service threads can be increased easily.

The server registers the "vsnfsd" program with the RPC framework by specifying the version, procedures it handles and various other information related to the program.Each RPC procedure has a corresponding

- Decode function, which decodes the XDR arguments
- Processing function, which carries out the procedure

- Encode function, which encodes the XDR result
- Release function, which release the allocated resources (if any)

Apart from this the size of the argument received, and the response argument is also specified. Regarding the memory for arguments received/response the server uses xdr_buff for the received and response arguments. It also provides sufficient number of pages (in our case it's always 1) for returning data in VSNFS_READ and VSNFS_READDIR procedures. As we limit the number of clients to one and since the RPC calls are blocking there is no need of any special synchronization mechanism(ie, no need to use any semaphore/mutex to protect the global datastructure)

6.1.1 Client Server Communication :

The communication between client and server is via RPC. We have used Sun's ONC RPC implementation.

To keep it simple we haven't used Portmapper facility. Instead the client has to know the server's port number and initiates the communication directly with the server.· Marshalling of arguments and results are done via XDR (Extended Data Representation). The maximum size of the return value of any RPC procedure is fixed to be the page size. XDR requires the data to be in Big Endian format. So appropriate conversion need to be done at both the client and server side

The first byte of the RPC response contains the result of the RPC operation. The client should honor the result before processing the response data.

6.1.2 Operation Procedures:

6.1.2.1 VSNFS_NULL

This is a procedure which could be used to check if the RPC program is up and running. We used this procedure to understand how to return values, hence we receive a integer and return by incrementing it

6.1.2.2 VSNFS_GETROOT

In NFS, a separate RPC program(mountd) provides the same feature as this. The VSNFS_GETROOT is invoked by the client only once while it is mounting the filesystem. The client passes the path to be mounted and the length of the path in a structure. In return it receives the file handle generated by the server for that path. The VSNFS_GETROOT doesn't do any permission checks for the exported path but generates the file handle after lookup.

6.1.2.3 VSNFS_LOOKUP

Whenever the client needs to operate on a file/directory, it needs the filehandle of that. The lookup procedure checks if the filename specified is present and then provides the file handle for that. The parameters passed are the parent filehandle and a filename. The procedures returns a new filehandle of the file/directory. In VSNFS, because we track the file/directory based only on the inode number provided in the file handle, we construct a table which provides the mapping between the inode number and the directory it corresponds to, whenever a successful lookup is done. Next time when the lower level directory name/filename is passed along with parent file handle, the whole path is constructed in the server side by using the table before calling the vfs procedure for lookup.

6.1.2.4 VSNFS_READDIR

The parameters the client passes is the parent filehandle and in return it receives a page containing multiple directory entries each of which contains information of the inode, filename, length of filename and a filehandle. The procedure gives the list of files and directories under the directory.

6.2 VSNFS Client:

The client side is designed to provide transparent interface to the VSNFS server just like normal NFS client.Since we do not support file attributes we have values fixed for fields in the inode. For example, inode->i_size is set to 4096 by default. We just support 2 types of files namely regular and directory.

The main data structure used by the client is vsnfs_server. This contains all the details of the server and other RPC related information.

```
struct vsnfs_server {
        struct rpc_clnt *cl_rpcclient;
        const struct vsnfs_rpc_ops *cl_rpc_ops;
        struct sockaddr_in cl_addr;
        size_t cl_addrlen;
        int flags;
        int timeout;
        char ip_addr[16];
        int server_port;
        char *mnt_path;
        struct vsnfs_fh root_fh;
};
```

The client parses the mount options to get the ip port remote directory and local directory.It then initialize the vsnfs server structure described above. Part of it is creating rpc client and initializing it to RPC procedures we defined. It then initialize the super block values and sets the vsnfs server to superblock private field. It then registers a unnamed block device for the filesystem. There is no private inode for vsnfs. We just use the i_private field of inode to store the file handle returned by the server. Since we do not need any more data this is easy and convenient. As already specified the file handle contains inode number and file type. So we could easily lock it up with iget_locked().

6.3 Mounting VSNFS:

After loading both the server and client module, issue the following command.

$ sudo mount -t vsnfs 127.0.0.1:/tmp /mnt

Visit the following link for detailed information on the installation and usage instruction of VSNFS.
http://code.google.com/p/vsnfs/source/browse/trunk/README

7. Conclusion:

VSNFS successfully implements a very simple network file system. The current implementation is very basic and has its own limitations. But the design and implementation is done in a way that VSNFS can be extended to support full file system features.

8. Future Work:

- Implementing the other important procedures like mkdir, create file, etc
- Extending the server to cater multiple clients and multiple mounts.
- Adding permission checks for security

9. Acknowledgements:

We would like to acknowledge Prof. Erez Zadok, for his guidance and we thank Vasily Tarasov for patiently clearing all our doubts and mentoring us throughout.

10. References:

- RFC1094: "NFS: Network File System Protocol Specification"
- RFC5531: "RPC: Remote Procedure Call Protocol Specification Version 2"
- XDR/RPC : http://ou800doc.caldera.com/en/SDK_netapi/CTOC-xdrN.intro.html
- Linux Test Project (LTP): http://ltp.sourceforge.net/