SNFS: Simple Network File System
CS 416: Operating System Design
Due: 11:55 PM, 05/06/2013

TAs: Ying Zhan (Sections 1 and 2), Bill Katsak (Section 3)

1. Announcements

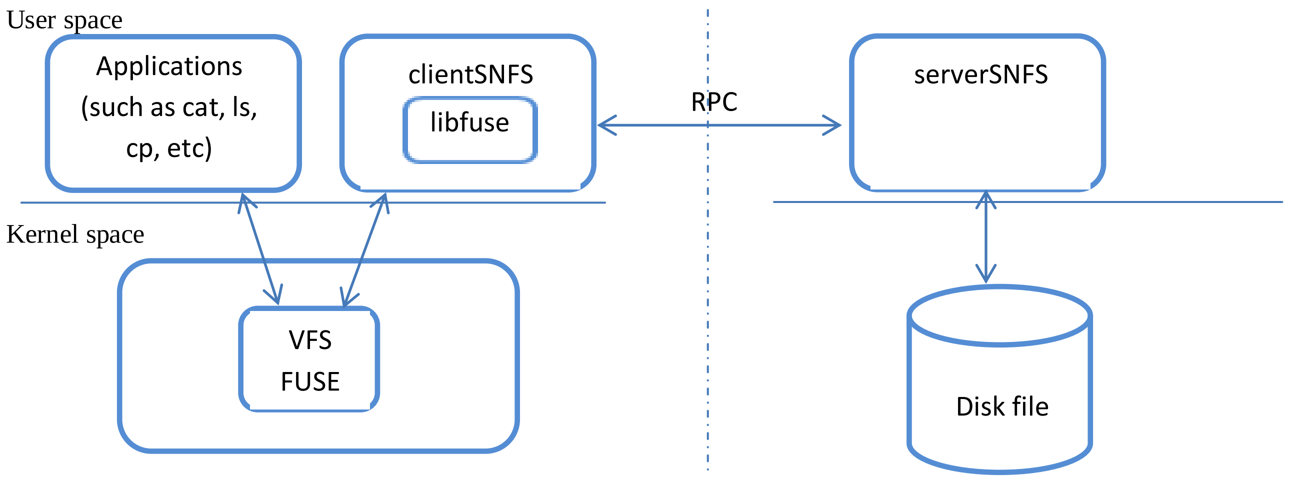
Assignment open date: April 3, 2013
Group size: 4 people per group
Due date: May 6, 2013 (11:55 pm)
Submission: Electronically, via Sakai
Logistics: The assignment should be carried out in iLab machines
Language of implementation: C or C++

2. Background

The purpose of this assignment is to implement client and server prototypes for a simple network file system (SNFS). The following is a summary of the basic architecture.

* A server-side executable, called serverSNFS, that is responsible for file management and request handling. The server is multi-threaded and can serve multiple requests concurrently. More details about the server are described below.

* A client-side executable, called clientSNFS, that acts as the proxy between applications and server. It should intercept file-related system calls (such as open file, read file, etc.) by using FUSE (details below), propagate those requests to the server through the network, and then retrieve the reply from the server to implement client applications such as vim, cat, cp, your own app, etc.



3. Implementation details

3.1 serverSNFS

* The serverSNFS implements the functionality corresponding to the client request, using its local Linux file system.

* The serverSNFS is multi-threaded so that client requests can be served concurrently. You can dynamically spawn new threads or use thread pooling.

* Your serverSNFS executable must accept at least these parameters:

-port port: TCP port to use for RPC communication.

-mount directory_path : Place on server machine where files are to be stored by the server.

-For example: serverSNFS -port 12345 -mount /tmp/OS416_NFS

* Do not remove the mount directory on server termination! For example, if the mount point is /tmp/OS416_NFS and a client creates a new file "file1.txt" by using fopen("file1.txt"), the server should create the file /tmp/OS416_NFS/file1.txt, and this file should persist after the server process ends.

3.2 clientSNFS, FUSE

* The clientSNFS is essentially a proxy between local applications and the remote file server. It should intercept file-related system calls by using FUSE and communicate with serverSNFS through RPCs (Remote Procedure Calls) in order to propagate requests and retrieve replies.

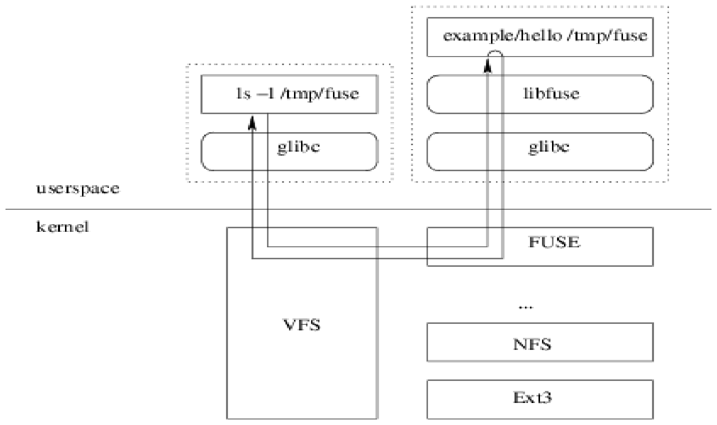
* The clientSNFS must accept at least these parameters.

-port port: TCP port that serverSNFS is listening on.

-address address: IP address or hostname where serverSNFS is located.

-mount directory_path : Place on client machine where remote file system should be exported.

* FUSE is a powerful tool which allows you to implement a user-level file system. After you mount the file system exported by serverSNFS, external applications (ls, cat, cp, etc.) can access this remote file system through standard calls such as open(), read(), write(), etc. VFS in the Linux kernel will vector these system calls into the FUSE kernel module, which then translates them into up-calls to the clientSNFS.



* Here is a workflow example:

- Let /tmp/fuse be the directory on the client side where the remote file system would be exported.

- At this point in time, there are two files already stored in serverSNFS, namely file01 and file02, under the directory /home/OS416_NFS on the remote server side.

- We create a new file under /tmp/fuse using the command "cat /etc/hosts > /tmp/fuse/file03"

- During the execution of this command, related system calls (open, write, close) would be vectored into clientSNFS which would propagate these requests to serverSNFS.

- Accordingly, serverSNFS would have created a new file at /home/OS416_NFS/file03 with same content as that in /etc/hosts on the client.

- To check the correctness, you can verify the content manually or by using diff command

* At least the following subset of functionalities should be implemented so that the file system can work smoothly: {create,open,write,close,truncate,opendir,readdir,releasedir,mkdir,getattr}

* To install FUSE on your own computer (it is already available on the iLab Machines):

- Download FUSE version > 2.8
  - In our example. 2.8.6 (http://sourceforge.net/projects/fuse/files/fuse-2.X/2.8.6/).
- Extract source.
- Read README (README contains comprehensive instructions)
- Run configure, make and make install,
- Add '/usr/local/lib' to '/etc/ld.so.conf' and run 'ldconfig'.

3.3 RPC between clientSNFS and serverSNFS

* RPCs (or Remote Procedure Calls) are a kind of inter-process communication mechanism: RPCs are initiated by the client, which sends a request message to a known remote server in order to execute a specified procedure with any supplied parameters. The remote server sends a response to the client, and the application continues its processing.

* You must implement a simple API that uses the RPC design pattern. You can do it by manually sending/receiving commands and parameters over raw network sockets or you may use existing third-party RPC libraries (Thrift, XML-RPC, etc.). **You do not need to implement a general RPC mechanism with sockets**, just enough to send/receive the required commands/parameters.

* Tips about socket programming:

- Byte ordering conventions ("big endian" versus "little endian") on different machines. x86 uses little endian, however, TCP/IP networks use big endian. Use htonl() and ntohl() functions to ensure that all partners interpret the byte orderings in the same way

- Make sure to check for an error return from all system calls

```
#include <errno.h>

if ( (sockid = socket(AF_INET, SOCK_STREAM, 0)) < 0) {

        printf("error creating client socket, error%d\n",errno);

}
```

- Make sure to close the socket when the connection is no longer needed, otherwise the socket can stay open, occupying the port and possibly leading to unexpected problems the next time you want to use the port. In addition, port numbers are a system-wide value, so make sure the one that you choose is only used by your program, and is larger than 1024. My suggestion is (10000 + the last four digits of your RUID). This will avoid conflicts with other users if you happen to be sharing an iLab machine.

- There are many things that can go wrong at multiple stages in many ways. Move forward with small, careful steps. Simply getting your sockets up and running correctly can be difficult if you are not used to C's networking primitives.

* More info about socket programming:

  - Book: Unix Network Programming

  - Code snippet (for reference only):

    http://www.pythonprasanna.com/Papers%20and%20Articles/Sockets/tcpclient.c

    http://www.pythonprasanna.com/Papers%20and%20Articles/Sockets/tcpserver.c

4. FUSE Tips

- The fuse homepage (http://fuse.sourceforge.net) has a lot of good info about FUSE, including examples.
- Please take a look at /ilab/users/yz280/cs416/project03 for another example of a trivial FS implementation with client/server interaction via sockets.
- Check out this presentation for some more examples: http://www.slideserve.com/arlo/introduction-to-fuse-file-system-in-user-space.

5. Submission

* One tar file named **exactly** "project3.tar" via Sakai. No binary executables, just source files and report.pdf

* Upon extraction, at least there should one "report.pdf" file and two directories, clientSNFS and serverSNFS, each should contain source files and a makefile. Your report should at least describe the work you've done, the difficulties you faced, and a couple of examples of use of your file system. Please also include the responsibilities of the members of the group.

* Auxiliary files (optional) such as use-cases may be considered, if you include them into another directory "misc"

* Please exactly follow the layout as stated here, since your TA may use scripts to grade your project. Any violation might unnecessarily affect your score.

6. Grading

* 20% Working basic RPC implementation

* 20% Working operations create, open, close, truncate, getattr

* 30% Working operations read, write, opendir, readdir, releasedir, mkdir

* 20% Report

The grading will be done on iLab machines, please test your code there before submission. Good luck.