Data Storage Systems

University of Minho

Guide 6
Remote File System

2025

The main goal of this guide is to design and implement a remote file system with FUSE. Files' data should be kept at the remote server while files' metadata should be stored locally at the client.

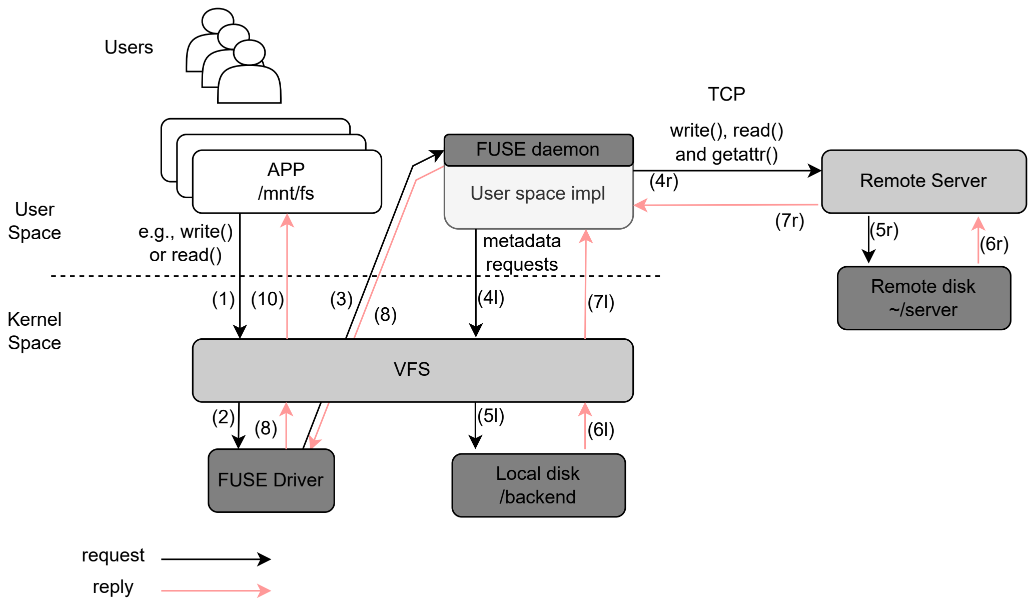

Figure 1: Simplified FUSE architecture and flow of events for a remote file system implementation

1 Setup

Use the VM and FUSE environment from the previous guides. Both client and server programs will run on the same VM and communicate through the *localhost* interface. Ensure that all the necessary directories (*e.g.*, /backend, /home/vagrant/server) are created. Use and adapt the code provided along with this guide to solve the exercises described next.

2 Exercises

For the next exercises consider a flat file system namespace (*i.e.*, all files are stored at the root directory).

1. Implement a first version of your remote file system that *writes* data from files both locally (at the client) and to the remote storage server. *Read* requests should be served from the locally stored data.

   - Inspect and update where needed the *storserver.c*, *passthrough.c*, *remote.c* and *remote.h* files containing sample code for the client and server.
   - When a *write* to a new file is received by the server, the corresponding file should be created.

   Try out your program with small files created by you and use *printf()* or a log file to debug your programs.

2. Update your code so that file *read* requests are served instead from the remote server.

3. Update your code so that file *write* and *read* requests are only handled by the remote server (*i.e.*, files' data should only be kept at the server). *Metadata* requests should be handled locally at the client.

   - Inspect the *xmp_getattr* function at the *passthrough.c* file. Note that the file size should be retrieved from the server. Any idea on why?

3 Extras

1. Test your remote file system with two VMs. Adapt the VagrantFile provided with the previous guides.
2. Add support for file deletion at the remote server.
3. Avoid opening and closing files at the server for each *read/write* operation.
   Hint: Consider *open* and *close* operations issued by the client and maintain a map of opened file descriptors at the remote server.
   Note: we still want the remote protocol to be stateless (*i.e.*, we do not want shared state between the client and server).
4. Consider a non-flat file system name space (*i.e.*, support a hieararchy of directories and files).
5. Update the server to handle requests concurrently (*e.g.*, with a thread pool). Be careful with concurrent updates to files/directories.

Learning outcomes

Implement a remote file system with FUSE. Understand the design implications of serving data and metadata locally and from a remote server.