Chapter 49: The Network File System (NFS)

1. Introduction to NFS

- The **Network File System (NFS)** is a distributed file system protocol that allows a client computer to access files over a network as if they were on its local storage.
- **Goals:**
  - Enable data sharing among multiple clients.
  - Centralize administration (e.g., backups, security).

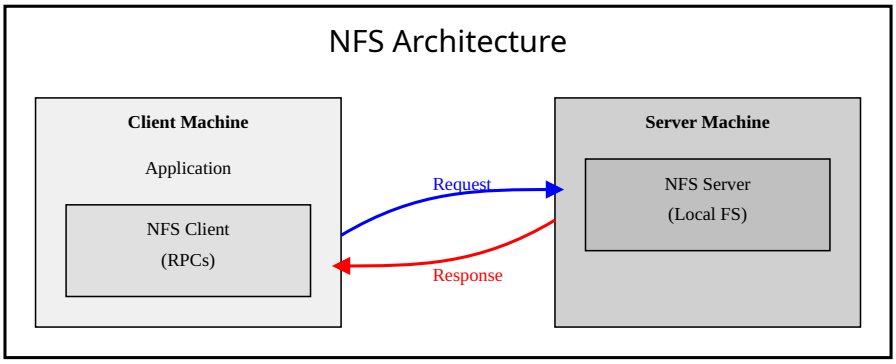


Figure 1: NFS Architecture

2. NFS Design Principles

- **Stateless Protocol:** The NFS server does not maintain any state about its clients. Each request must contain all the information necessary for the server to complete it. This simplifies server crash recovery.
- **Idempotent Operations:** Most NFS operations are idempotent, meaning that performing them multiple times has the same effect as performing them once. This allows clients to safely retry requests after a timeout.

3. The NFS Protocol

- NFS uses RPCs to perform file operations.
- **File Handle:** Instead of file descriptors, NFS uses a **file handle** to uniquely identify a file or directory. The file handle contains a volume ID, an inode number, and a generation number.
- **Key Operations:**
  - `LOOKUP(dir, name)`: Looks up a name in a directory and returns a file handle.
  - `READ(fh, offset, count)`: Reads data from a file.
  - `WRITE(fh, offset, count, data)`: Writes data to a file.

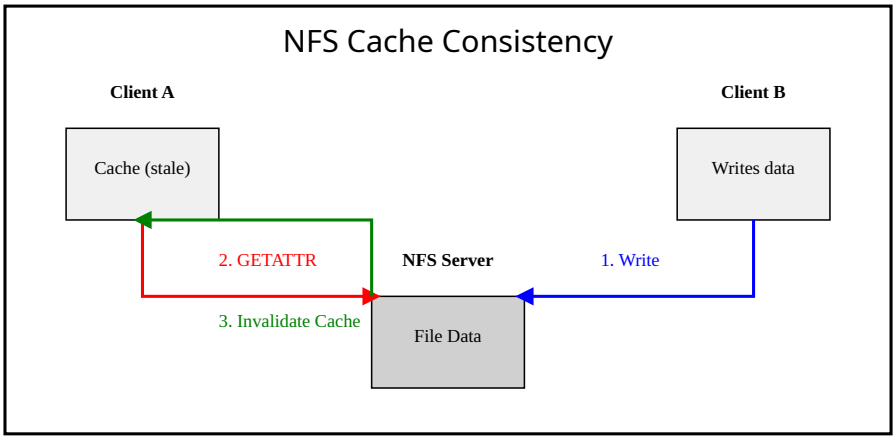
- `CREATE(dir, name, attr)`: Creates a new file.
- `REMOVE(dir, name)`: Deletes a file.
- `GETATTR(fh)`: Gets the attributes of a file.

NFS Protocol Code Example

```c
// Simplified NFS client-side read operation
int nfs_read(file_handle_t fh, char *buffer, int offset, int count) {
    // Create an RPC message
    rpc_msg_t msg;
    msg.op = NFS_OP_READ;
    msg.fh = fh;
    msg.offset = offset;
    msg.count = count;

    // Send the RPC and wait for the response
    rpc_response_t *resp = send_rpc(server, &msg);

    // Copy the data from the response to the user's buffer
    memcpy(buffer, resp->data, resp->count);

    return resp->count;
}
```

4. Caching and Consistency

- NFS clients cache data and metadata to improve performance.
- This introduces the **cache consistency problem**.
- **NFS Solution:**
  - **Flush-on-close:** When a client closes a file it has written to, it flushes all changes to the server.
  - **Attribute Caching and Validation:** Before using a cached block, the client sends a `GETATTR` request to the server to check the file's last modification time. If the server's copy is newer, the client invalidates its cache.



5. Server-Side Operations

- **Write Buffering:** The NFS server must not acknowledge a `WRITE` request until the data has been written to stable storage. This prevents data loss if the server crashes.
- **VFS/vnode Interface:** The Virtual File System (VFS) and vnode interface allow different file system types (local or remote) to coexist within the same OS.

6. Summary

- NFS is a classic example of a distributed file system.
- Its stateless design makes it simple and robust.
- Caching is essential for performance, but it introduces the challenge of cache consistency.
- NFS has evolved over the years, but the core principles of the original design are still relevant today.