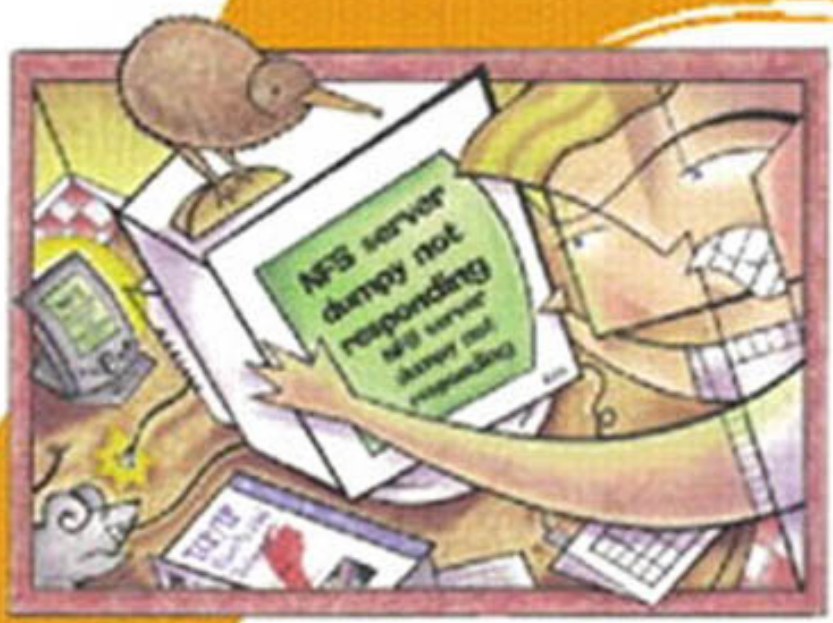
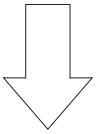
NFS Illustrated

Brent Callaghan



ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



NFS ILLUSTRATED

TABLE 7.1 NFS Version 3 Errors

Error name	*Value*	*Meaning*
NFS3_OK	0	Indicates that the call completed successfully.
NFS3ERR_PERM	1	Not owner. The operation was not allowed because the caller is either not a privileged user (root) or not the owner of the target of the operation.
NFS3ERR_NOENT	2	No such file or directory. The file or directory name specified does not exist.
NFS3ERR_IO	5	I/O error. A hard error (for example, a disk error) occurred while processing the requested operation.
NFS3ERR_NXIO	6	I/O error. No such device or address.
NFS3ERR_ACCES	13	Permission denied. The caller does not have the correct permission to perform the requested operation. Contrast this with NFS3ERR_PERM, which restricts itself to owner or privileged user permission failures.
NFS3ERR_EXIST	17	File exists. The file specified already exists.
NFS3ERR_XDEV	18	Attempt to do a cross-device hard link.
NFS3ERR_NODEV	19	No such device.
NFS3ERR_NOTDIR	20	Not a directory. The caller specified a nondirectory in a directory operation.
NFS3ERR_ISDIR	21	Is a directory. The caller specified a directory in a nondirectory operation.
NFS3ERR_INVAL	22	Invalid argument or unsupported argument for an operation. Two examples are attempting a READLINK on an object other than a symbolic link and attempting to SETATTR a time field on a server that does not support this operation.
NFS3ERR_FBIG	27	File too large. The operation would have caused a file to grow beyond the server's limit.
NFS3ERR_NOSPC	28	No space left on device. The operation would have caused the server's filesystem to exceed its limit.
NFS3ERR_ROFS	30	Read-only filesystem. A modifying operation was attempted on a read-only filesystem.
NFS3ERR_MLINK	31	Too many hard links.
NFS3ERR_NAMETOOLONG	63	The filename in an operation was too long.

TABLE 7.1 NFS Version 3 Errors

Error name	*Value*	*Meaning*
NFS3ERR_NOTEMPTY	66	An attempt was made to remove a directory that was not empty.
NFS3ERR_DQUOT	69	Resource (quota) hard limit exceeded. The user's resource limit on the server has been exceeded.
NFS3ERR_STALE	70	Invalid filehandle. The filehandle given in the arguments was invalid. The file referred to by that filehandle no longer exists, or access to it has been revoked.
NFS3ERR_REMOTE	71	Too many levels of remote in path. The file handle given in the arguments referred to a file on a nonlocal file system on the server.
NFS3ERR_BADHANDLE	10001	Illegal NFS filehandle. The filehandle failed internal consistency checks.
NFS3ERR_NOT_SYNC	10002	Update synchronization mismatch was detected during a SETATTR operation.
NFS3ERR_BAD_COOKIE	10003	READDIR or READDIRPLUS cookie is stale.
NFS3ERR_NOTSUPP	10004	Operation is not supported.
NFS3ERR_TOOSMALL	10005	Buffer or request is too small.
NFS3ERR_SERVERFAULT	10006	An error occurred on the server that does not map to any legal NFS version 3 protocol error values. Client should translate this into an appropriate error. UNIX clients may choose to translate this to EIO.
NFS3ERR_BADTYPE	10007	An attempt was made to create an object of a type not supported by the server.
NFS3ERR_JUKEBOX	10008	Server initiated the request but was not able to complete it in a timely fashion. Client should wait and then try the request with a new RPC transaction ID. For example, this error should be returned from a server that supports hierarchical storage and receives a request to process a file that has been migrated. In this case, server should start the immigration process and respond to client with this error.

nfs_fh3

```
struct nfs_fh3 {
    opaque<NFS3_FHSIZE>data;
}
```

The `nfs_fh3` filehandle is the variable-length opaque object that is returned by the server on LOOKUP, CREATE, MKDIR, SYMLINK, MKNOD, LINK, or READDIRPLUS operations, which is used by the client on subsequent operations to reference a filesystem object such as a file or directory. The filehandle contains all the information the server needs to distinguish an individual file. To the client, the filehandle is opaque. The client stores filehandles for use in a later request and can compare two filehandles from the same server for equality by doing a byte-by-byte comparison but cannot otherwise interpret the contents of filehandles. If two filehandles from the same server are equal, they must refer to the same file, but if they are not equal, no conclusions can be drawn. Servers should try to maintain a one-to-one correspondence between filehandles and files, but this is not required. Clients should use filehandle comparisons only to improve performance, not for correct behavior.

Servers can revoke the access provided by a filehandle at any time. If the filehandle passed in a call refers to a filesystem object that no longer exists on the server or access for that filehandle has been revoked, the error NFS3ERR_STALE should be returned.

WebNFS clients and servers recognize a public filehandle as one that has zero length (see section 16.4).

nfstime3

The `nfstime3` structure gives the number of seconds and nanoseconds since midnight, Greenwich Mean Time, January 1, 1970. It is used to pass time and date information.

```
struct nfstime3 {
    uint32  seconds;    /* Time in sec since midnight 1/1/70 */
    uint32  nseconds;   /* Fractional nanoseconds */
}
```

Since negative values are not permitted, it cannot be used to refer to files older than 1970. The times associated with files are all server times except in the case of a SETATTR operation where the client can explicitly set the file time. A server converts to and from local time when processing time values, preserving as much accuracy as possible.

fattr3

The file attribute structure defines the attributes of a filesystem object (Table 7.2). It is returned by most operations on an object; in the case of operations that affect two objects (for example, a MKDIR that modifies the target directory attributes and defines new attributes for the newly created directory), the attributes for both are returned.

```
struct fattr3 {
```

```
        ftype3      type;   /* The file type */
        uint32      mode;   /* File permission bits */
        uint32      nlink;  /* Number of hard links */
        uint32      uid;    /* File user ID (owner) */
        uint32      gid;    /* File group ID */
        uint64      size;   /* File size in bytes */
        uint64      used;   /* Disk space used */
        specdata3   rdev;   /* File device information */
        uint64      fsid;   /* Filesystem identifier */
        uint64      fileid; /* File number within filesystem */
        nfstime3    atime;  /* Last access time */
        nfstime3    mtime;  /* Last modify time */
        nfstime3    ctime;  /* Last attribute change time */
    }
```

TABLE 7.2 `fattr3` File Attributes

Attribute	*Description*
`type`	Type of the file
`mode`	Protection mode bits (see Table 7.3)
`nlink`	Number of hard links to file—that is, number of different names for same file
`uid`	User ID of the owner of the file
`gid`	Group ID of the group of the file
`size`	Size of the file in bytes
`used`	Number of bytes of disk space that the file actually uses (which can be smaller than the `size` because the file may have holes or it may be larger due to fragmentation)
`rdev`	Identifies the device file if the file type is NF3CHR or NF3BLK (see `specdata3` on page 141)
`fsid`	Filesystem identifier for the filesystem
`fileid`	Number that uniquely identifies the file within its filesystem (on UNIX this would be the inode number)
`atime`	The time when the file data were last accessed
`mtime`	The time when the file data were last modified
`ctime`	The time when the attributes of the file were last changed. Writing to the file changes the `ctime` in addition to the `mtime`.

In some cases, the attributes are returned in the structure `wcc_data`, which is defined later; in other cases the attributes are returned alone. The main changes from the NFS version 2 protocol are that many of the fields have been widened and the major/minor device information is now presented in a distinct structure rather than being packed into a 32-bit field. All servers should

support this set of attributes even if they have to simulate some of the fields. The mode bits are defined in Table 7.3.

TABLE 7.3 `fattr3` Structure Access Modes

Mode Bit (octal)	*Description*
0004000	Set user ID on execution.
0002000	Set group ID on execution.
0001000	On directories, restricted deletion flag. On regular files, do-not-cache flag (not defined in POSIX).
0000400	Read permission for owner.
0000200	Write permission for owner.
0000100	Execute permission for owner on a file. Or lookup (search) permission for owner in directory.
0000040	Read permission for group.
0000020	Write permission for group.
0000010	Execute permission for group on a file. Or lookup (search) permission for group in directory.
0000004	Read permission for others.
0000002	Write permission for others.
0000001	Execute permission for others on a file. Or lookup (search) permission for others in directory.

post_op_attr

This structure is used for returning attributes in operations that are not directly involved with manipulating attributes. One of the principles of the NFS version 3 protocol is to return the real value from the indicated operation and not an error from an incidental operation. The `post_op_attr` structure was designed to allow the server to recover from errors encountered while getting attributes.

```
union post_op_attr switch (boolean) {
    case TRUE:
        fattr3  attributes;
    case FALSE:
        void;
}
```

This structure appears to make returning attributes optional. However, server implementors are strongly encouraged to make their best effort to return

attributes whenever possible, even when returning an error. In returning attributes in the results of all version 3 procedures, the protocol provides clients with a frequent supply of fresh attributes, precluding the need for additional GETATTR requests to restore stale, cached attributes.

pre_op_attr

This is the subset of preoperation attributes needed to better support the weak cache consistency semantics.

```
union pre_op_attr switch (boolean) {
    case TRUE:
        uint64      size;  /* File size in bytes */
        nfstime3    mtime; /* File modification time */
        nfstime3    ctime; /* Attribute modification time */
    case FALSE:
        void;
}
```

The `size` is the file size in bytes of the object before the operation; `mtime` is the time of last modification of the object before the operation; `ctime` is the time of last change to the attributes of the object before the operation.

The use of `mtime` by clients to detect changes to filesystem objects residing on a server is dependent on the granularity of the time base on the server.

wcc_data

When a client performs an operation that modifies the state of a file or directory on the server, it cannot immediately determine from the postoperation attributes whether the operation just performed was the only operation on the object since the last time the client received the attributes for the object. This is important, since if an intervening operation has changed the object, the client will need to invalidate any cached data for the object (except for the data that it just wrote).

To deal with this, the notion of weak cache consistency data or `wcc_data` is introduced. A `wcc_data` structure consists of certain key fields from the object attributes before the operation, together with the object attributes after the operation. This information allows the client to manage its cache more accurately than in NFS version 2 protocol implementations. The term *weak cache consistency* emphasizes the fact that this mechanism does not provide the strict server-client consistency that a strong cache consistency protocol would provide.

```
struct wcc_data {
    pre_op_attr     before;
    post_op_attr    after;
}
```