Unified File Locking (UFL) Specification

Revision History

Date	Modified by:	Description
01/17/00	J Gerthe	Initial Draft
01/28/00	J Gerthe	Added specifics on Linux versions and performance
02/09/00	J Gerthe	Incorporated comments from A Tridgell and added spec for FILE_SHARE_DELETE flag operation

1 General

1.1 Overview

The following is the proposed operational specification for a Unified File Locking mechanism to be added to Samba 2.06 or later and the core Linux OS (kernel 2.2.13 or later). It covers interaction of SMB/CIFS locking paradigms with that of NFS for all pertinent combinations. Note that:

- Win32/CIFS clients indicate a Windows system using the SMB/CIFS protocol to access file information.
- Win32/PC-NFS indicates a Windows system using a third party NFS product using the NFS protocol to access file information, and
- UNIX/NFS indicates a Unix system using NFS protocol to access file information.

Also note that the term primary client refers to a client that is first to open a file, and secondary client indicates a client attempting to perform a subsequent operation on that file.

1.2 Performance

In no instance of locking outlined below, will the addition of UFL code to Samba or core Linux cause file read or write performance to be substantially compromised. This applies to cases where the file locking mechanism results in a grant of access to a secondary client, over the existing non-UFL performance levels of Linux.

2 UFL operation for primary SMB/CIFS client with specific open mode

This section defines the operation of the UFL mechanism when a Win32/CIFS client has opened a file on the NAS device using Win32 CreateFile() API. The two parameters passed to this call that are important here are the dwDesiredAccess and dwShareMode. The dwDesiredAccess can have several states, either: 0 indicating query access, GENERIC_READ and/or GENERIC_WRITE (meanings are obvious). The dwShareMode can have a value of 0, or a combination of one or more of the following values: FILE_SHARE_DELETE, FILE_SHARE_READ, or FILE_SHARE_WRITE.
For simplification:

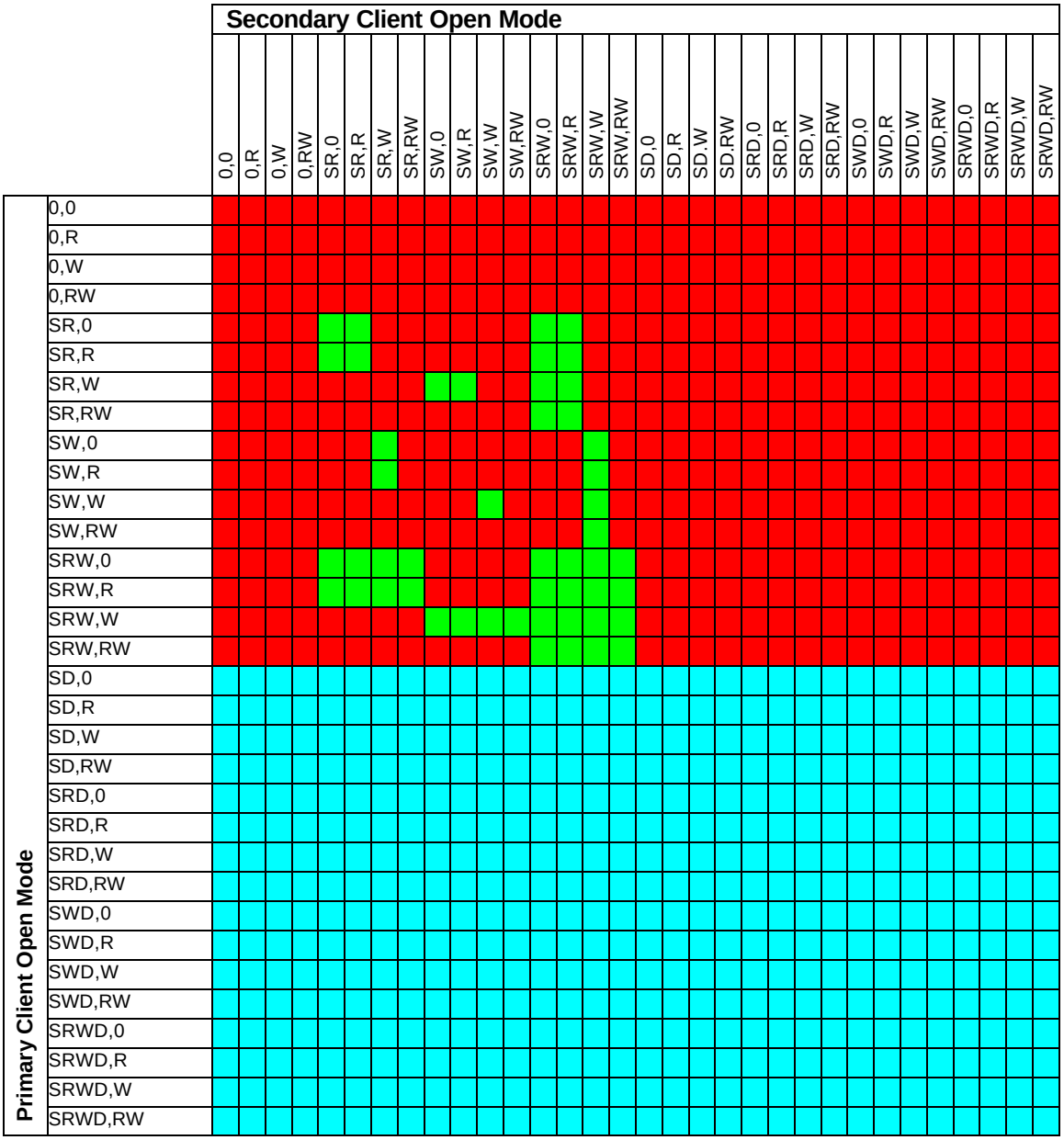
- the dwDesiredAccess values will hereafter be referred to as 0, R, W, or RW,
- the dwShareMode values will be referred to as 0, SR, SW, SRW, SD, SRD, …,
- an executable file are those denoted by suffixes of *.exe, *.com, *.dll, or *.sym

2.1 Secondary access from a Win32/CIFS client

The following karnaugh maps define the result when a secondary client attempts to open the same file using the same API w/associated combinations of parameters. The operation of these calls appears to be dependent upon the OS of the client, as well as the co-location of the primary and secondary process. Windows 98 operates differently than Windows NT (ie. FILE_SHARE_DELETE is only specified for NT). And the operation is dependent upon whether the secondary open attempt is from the same connection (two processes on one client) or from two separate connections (clients), and also whether the file being opened is an executable or not.

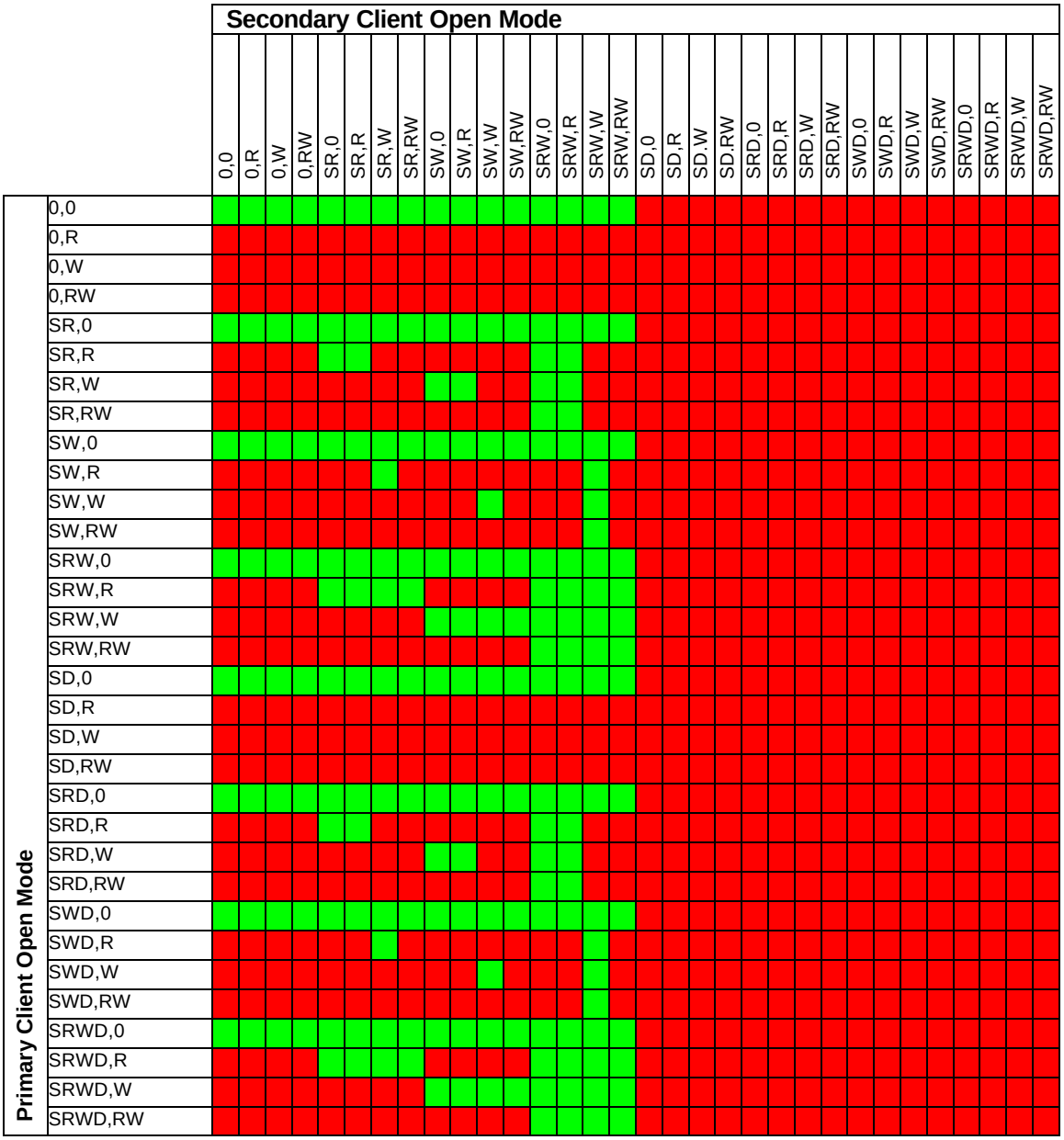
2.1.1 Primary Windows 98 client with secondary access from a Windows 98 client on either two separate or a single common CIFS connection

The matrix of the results the UFL will provide for a file being accessed by two Windows 98 processes (whether using the same or two separate CIFS connections) is as follows where red shading indicates a denial of the request for the secondary client and blue indicates an invalid condition (ie. Windows 98 will not open a file with this mode). Note also that there was no difference for an executable vs. non-executable file with this configuration:



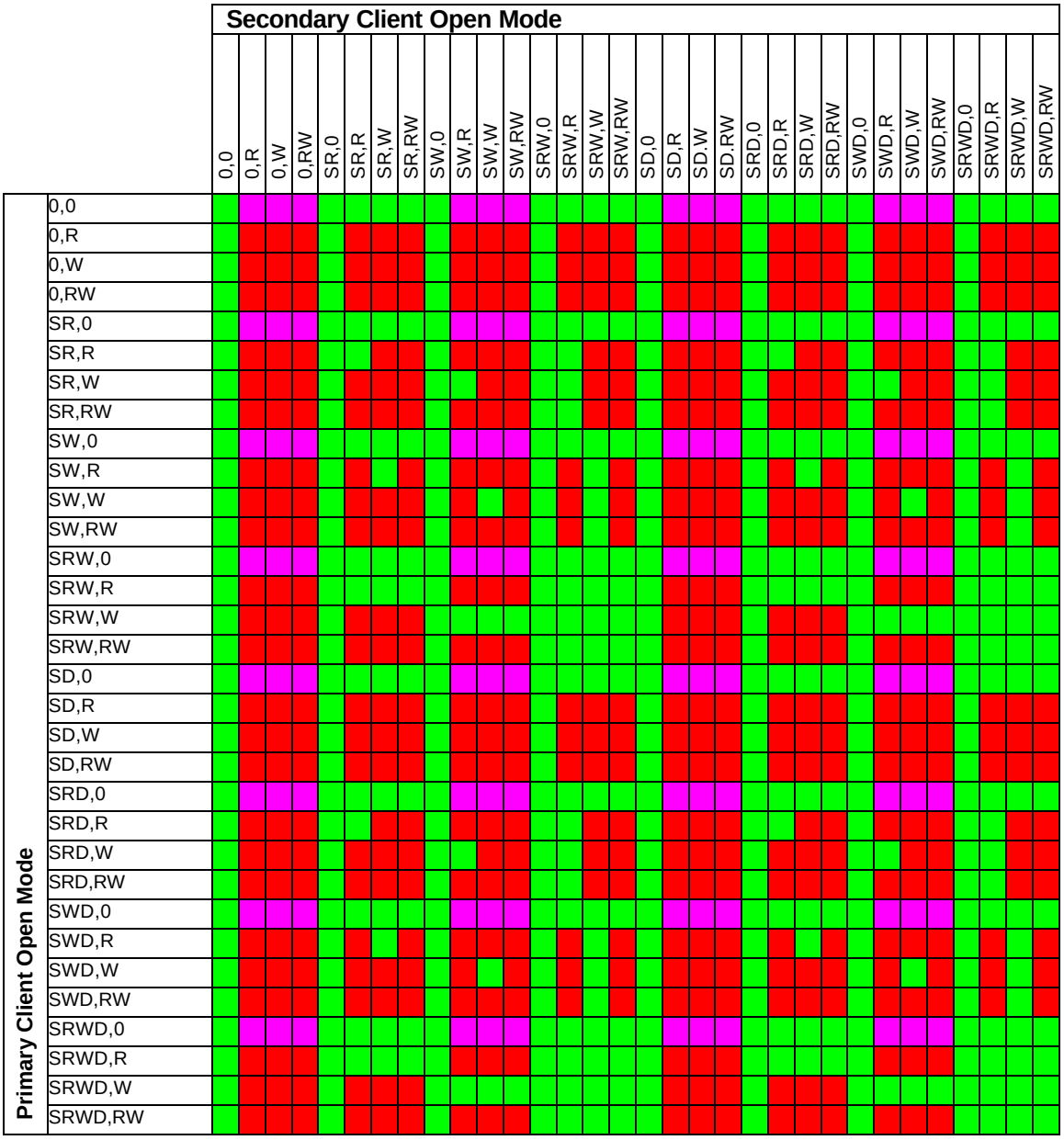
2.1.2 Primary Windows NT client with secondary access from either a Windows 98 or NT client on a separate connection

The matrix of the results the UFL will provide for a file being accessed by a primary Windows NT client on a different CIFS connection from a secondary client that is either Window 98 or NT is as follows where red shading indicates a denial of the request for the secondary client. Note that there was no difference for an executable vs. non-executable file with this configuration:



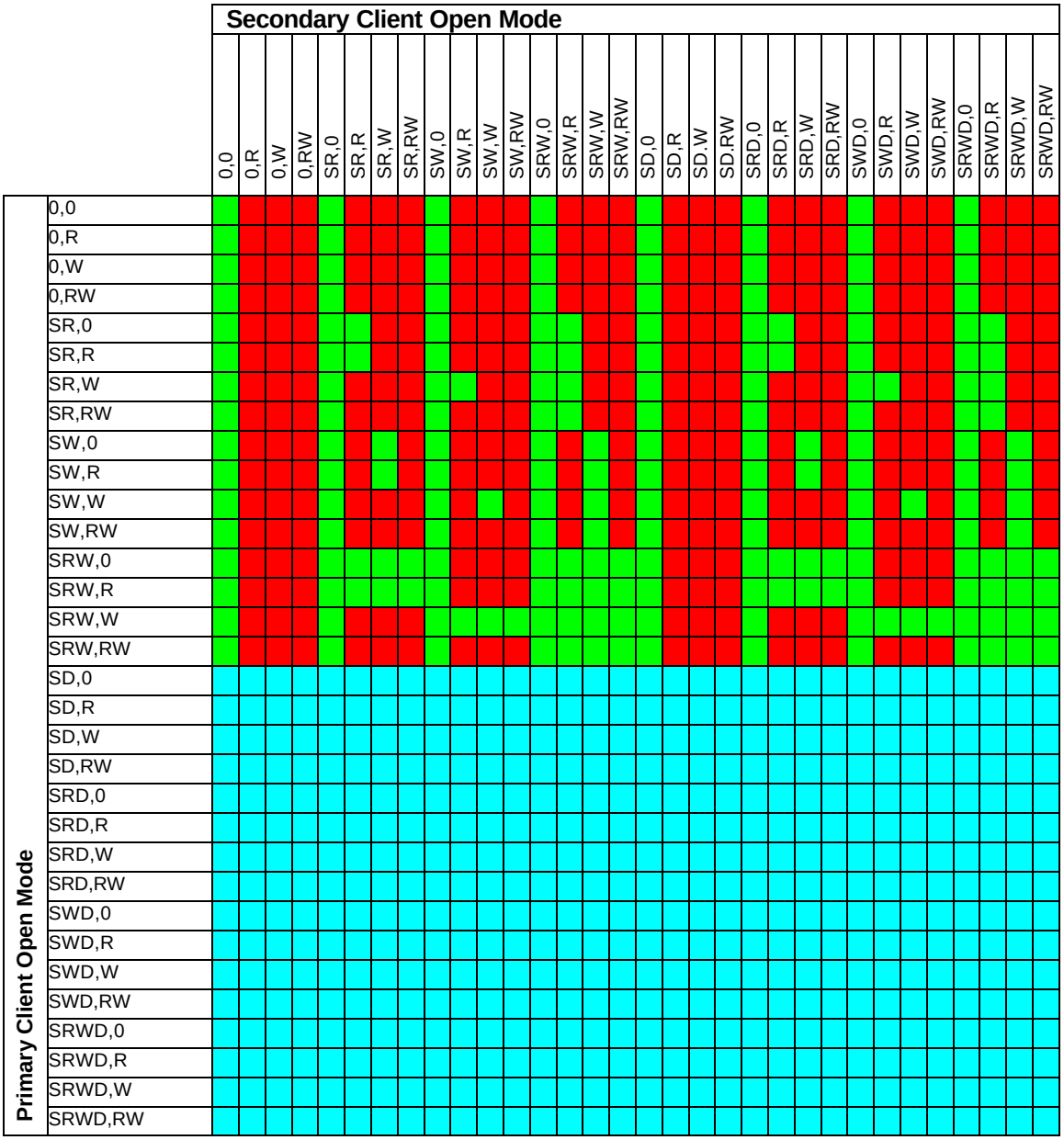
2.1.3 Primary Windows NT client with secondary access from a Windows NT client on the same common connection

The matrix of the results the UFL will provide for a file being accessed by two Windows NT processes using the same CIFS connections is as follows where red shading indicates a denial of the request for the secondary client when the file is a non-executable and pink indicating a denial only when the file is an executable:



2.1.4 Primary Windows 98 client with secondary access from a Windows NT client on a different CIFS connection

The matrix of the results the UFL will provide for a file being opened by a primary Windows 98 client and a secondary Windows NT client is as follows where red shading indicates a denial of the request for the secondary client and blue indicates an invalid condition (ie. Windows 98 will not open a file with this mode). Note also that there was no difference for an executable vs. non-executable file with this configuration:



2.2 Secondary access from a Win32/PC-NFS or UNIX/NFS client

If a secondary NFS client attempts to open this file, the UFL mechanism will allow it to succeed due to the fact that NFS is stateless, and there is no way of knowing what operation the secondary client will wish to perform in the future. However, subsequent requests from the secondary client will be allowed or denied by the UFL mechanism based on the operation being requested. The matrix of the results the UFL will provide are as follows where red shading indicates a denial of the request for the secondary client:

		Secondary Client Operation Requested			
		Read	Write	Delete	Rename/ Move
Primary Client Open Mode	Share=0 Access=0				
	Share=0 Access=R				
	Share=0 Access=W				
	Share=0 Access=RW				
	Share=SR Access=0				
	Share=SR Access=R				
	Share=SR Access=W				
	Share=SR Access=RW				
	Share=SW Access=0				
	Share=SW Access=R				
	Share=SW Access=W				
	Share=SW Access=RW				
	Share=SRW Access=0				
	Share=SRW Access=R				
	Share=SRW Access=W				
	Share=SRW Access=RW				

3 UFL mechanism for a primary SMB/CIFS client with a byte range lock

This section defines the operation of the UFL mechanism when a Win32/CIFS client has opened a file using the Win32 CreateFile() API followed by a call to the LockFile() API. There are no pertinent parameters in this second API call other than the byte-range being requested and whether it is in conflict with a secondary request. There also exists an extension API known as LockFileEx(). The dwFlags parameter in this call has the ability to allow for a "read-only" byte range lock request. Note, Type = SH refers to the NON-use of the LOCKFILE_EXCLUSIVE_LOCK flag in the LockFileEx() API, ie. shared mode. BRC = Byte Range Conflict

3.1 Secondary access from another Win32/CIFS client

When a Win32/CIFS client is holding a byte-range lock, and subsequent to this a secondary Win32/CIFS client requests a byte range lock, the byte range being requested will be tested and, if in conflict, the UFL mechanism shall cause the lock request to fail. The byte range considered will be the full 64 bits allowed by the lockfile() API.
Additionally, the UFL shall insure that operations on the file by the secondary client shall fail according to the following matrix of operations and byte conflict status.

		Secondary Client Operation					
		Read BRC = T	Read BRC = F	Write BRC = T	Write BRC = F	Delete	Rename/ Move
Primary Client Lock Mode	Type = 0						
	Type = SH						

3.2 Secondary access from a Win32/PC-NFS or UNIX/NFS client

Byte range locking on UNIX is accomplished via one of two API calls: fcntl() or lockf(). The fcntl() API allows for something similar to the LOCKFILE_EXCLUSIVE_LOCK flag on Win32 platforms.

When a Win32/CIFS client is holding a byte-range lock, and subsequent to this a secondary Win32/PC-NFS or UNIX/NFS client requests a byte range lock, the byte range being requested will be tested and, if in conflict, the UFL mechanism shall cause the lock request to fail. This test will only be valid for the first 31 bits of the byte range to allow for some clients which have aberrant behavior due to signed/unsigned issues. The remaining upper bits will be considered to be the same to allow for the safest mode of operation between two clients. Additionally, the UFL shall insure that operations on the file by the secondary client shall fail according to the following matrix of operations and byte conflict status.

		Secondary Client Operation					
		Read BRC = T	Read BRC = F	Write BRC = T	Write BC = F	Delete	Rename/ Move
Primary Client Lock Mode	Type = 0						
	Type = SH						

4 UFL operation for a primary SMB/CIFS client with an oplock

Although not explicitly exposed in the API, the SMB/CIFS protocol supports a performance oriented feature known as oplocks. This section defines the operation of the UFL mechanism when a Win32/CIFS client has opened a file and obtained an oplock (automatic if it is the first client opening that file). Oplocks also have two levels. The first, an exclusive oplock allows for a client to perform read or write operations, whereas the second, a level-II oplock allows client access with the understanding that it is for reading only.

4.1 Secondary access from another Win32/CIFS or Win32/PC-NFS client

If a secondary client attempts to open the file, the UFL mechanism shall cause the exclusive oplock of the primary client to be "broken" either completely or to a level II oplock depending upon the secondary clients desired access mode. (Details of this mode are as per the cifs6.txt document available on the samba.org site). This will result in the primary client closing or flushing all file and file lock changes to the NAS device. Subsequent to this, the UFL will allow the secondary client to open the file, if the mode is compatible with the type of file lock the primary client retains on the file as described in the preceding sections of this document.

4.2 Secondary access from a UNIX/NFS client

If a secondary NFS client attempts to perform an operation such as read, write, delete, or rename/move operation on the file, the UFL mechanism shall cause the oplock of the primary client to be "broken", resulting in the primary client closing or flushing all file and file lock changes to the NAS device. Subsequent to this, the UFL will allow the secondary client to perform the operation on the file, if the operation is compatible with the type of file lock the primary client retains on the file as described in the preceding sections of this document. The oplock held by the primary client will not be broken if the secondary client is merely doing a "stat" operation on the file. While this could result in inaccurate data being presented to the secondary client, it is felt that performance concerns here outweigh the likelihood of innaccurate information causing application failure.

5 UFL operation for a primary NFS client with a byte range lock

This section defines the operation of the UFL mechanism when a Unix/NFS client has opened a file and then used either fcntl() or lockf() APIs. Note, Type = SH refers to use of the non-exclusive lock flag in the fcntl() API, ie. shared mode. BRC = Byte Range Conflict

5.1 Secondary access from a Win32/CIFS client

When a UNIX/NFS client is holding a byte-range lock, and subsequent to this a secondary Win32/CIFS client attempts to open this file, the UFL mechanism will allow the open request to succeed. Subsequent to this, the secondary Win32/CIFS client may attempt an operation on the file. The byte range of the operation shall be checked with the first 31 bits considered only. The upper bits will be assumed to be the same as the primary client in order to fail in the safest mode. The UFL shall insure that operations on the file by the secondary client shall fail according to the following matrix of operations and byte conflict status.

		Secondary Client Operation					
		Read BRC = T	Read BRC = F	Write BRC = T	Write BRC = F	Delete	Rename/ Move
Primary Client Lock Mode	Type = 0						
	Type = SH						

5.2 Secondary access from a Win32/PC-NFS or UNIX/NFS client

Byte range locking on UNIX is accomplished via one of two API calls: fcntl() or lockf(). The fcntl() API allows for something similar to the LOCKFILE_EXCLUSIVE_LOCK flag on Win32 platforms.

When a NFS client is holding a byte-range lock, and subsequent to this a secondary Win32/PC-NFS or UNIX/NFS client requests a byte range lock, the byte range being requested will be tested using the lower 31 bits only and, if in conflict, the UFL mechanism shall cause the lock request to fail.

Additionally, the UFL shall insure that operations on the file by the secondary client shall fail according to the following matrix of operations and byte conflict status.

		Secondary Client Operation					
		Read BRC = T	Read BRC = F	Write BRC = T	Write BRC = F	Delete	Rename/ Move
Primary Client Lock Mode	Type = 0						
	Type = SH						

6 UFL operation for multi-protocol “Change Notify”

The proposed UFL mechanism shall extend the SMB/CIFS “Change Notify” capability to NFS clients. When NAS device file directories are altered by UNIX/NFS or Win32/PC-NFS clients, the UFL mechanism will provide the correct asynchronous SMB/CIFS change notification to Win32/CIFS clients in the same manner as if the change had been caused by a Win32/CIFS client within 30 seconds.