情報処理振興事業協会

平成１２年度未踏ソフトウェア創造事業

ネットワーク RAID ファイルシステムの開発

成　果　報　告　書

平成１３年２月

松本　尚

情報処理振興事業協会
平成１２年度未踏ソフトウェア創造事業
ネットワーク RAID ファイルシステムの開発
要旨

a. プロジェクト名

『ネットワーク RAID ファイルシステムの開発』

b. 開発目的

ローカルエリアネットワーク（LAN）環境を構成する計算機のローカルディスクのうち遊休資源となっているものを有効活用した低コストで高信頼分散共有ファイルシステムを可能とするネットワーク RAID ファイルシステム（NRFS）を開発する。

c. 開発期間

平成 12年10月25日～平成13年2月28日

d. 実施体制

担当プロジェクト・マネージャー	長谷川　正治
開発者	松本　尚
プロジェクト実施管理組織	三菱マテリアル株式会社

e. 開発内容及び成果

(1) 開発内容

(a) NFS 実装調査

(b) NRFS 基本部の設計

(c) NRFS プロトタイプの作成

(2) 成果

(a) NFS 実装調査

NRFS と同様に LAN 環境上でファイル共有を実現する機構としては、Network File System（NFS）が広く普及している。

NRFS についても NFS の機構が活用できると考えられることから、NFS の実装の詳細を解析した。

(b) NRFS 基本部の設計

NRFS は NFS と同様に、クライアントとサーバにより構成される。ただし、NRFS では、サーバ計算機自体の障害に耐え得るよう、サーバを多重化して取り扱うとと

もに、障害検出訂正機能を実装する。

NFS の実装調査結果に基づいて、NFS を拡張する形で NRFS 基本部の設計を行うとともに、障害検出訂正機能の詳細についても検討を行った。

(c) NRFS プロトタイプの作成

NRFS 基本部の設計に基づいて NRFS プロトタイプを作成した。このプロトタイプでは、サーバ多重化に対応したクライアント並びにサーバを整備するとともに、特定の障害発生に対応した障害検出訂正機能を実装した。

作成したプロトタイプについて、実際にテストを行い、NRFS 基本部が設計通りに機能することを確認するとともに、そのパフォーマンスを測定し、実用上支障がないレベルで NRFS が実装可能であることを確認した。

また、実装した障害検出訂正機能について、障害発生を人為的に模擬したテストを行い、その健全性を確認した。

f. 今後の課題

実際の運用環境では、その発生頻度は低いと考えられるものの、様々な障害に遭遇し得る。また、LAN の障害や電源系統のトラブル等、外的な要因による障害に対しても、可能な限り安定した稼動が求められる。特に、今回は対応を見送ったサーバの停止やサーバとの通信経路のダウンといった障害に対しては、実用に向けて必ず対応する必要がある。性能に関しても、可能な限り NFS と同等の性能を達成したいと考えている。

このように、NRFS の実用化に向けては、クライアント並びにサーバの品質向上に加え、遭遇し得る障害を綿密に分析し、対応する障害検出訂正機能を設計・実装する等、機能面の充実並びに信頼性・安定性のより一層の向上が求められる。

したがって、本プロジェクト終了後も、引き続き開発作業を継続し、NRFS の実用化に向けて注力したい。

目　次

要旨
目次
図表一覧表

1. **プロジェクトの概要** …… 1

2. **開発計画** …… 2
2.1 **開発目的** …… 2
2.2 **開発内容** …… 2
2.2.1 NFS **実装調査** …… 2
2.2.2 NRFS **基本部の設計** …… 3
2.2.3 NRFS **プロトタイプの作成** …… 3
2.3 **開発期間** …… 3
2.4 **実施体制** …… 3

3. **成果報告** …… 4
3.1 NFS **実装調査** …… 4
3.1.1 **マウント** …… 6
3.1.2 KRPC …… 17
3.1.3 XDR …… 25
3.1.4 NFS …… 30
3.1.5 VFS …… 38
3.2 NRFS **基本部の設計** …… 42
3.2.1 NRFS **のデータ構造** …… 42
3.2.2 **サーバ多重マウント方式** …… 49
3.2.3 KRPC **プロトコルの拡張** …… 52
3.2.4 **障害訂正方式** …… 60
3.2.5 NRFS **の動作の概要** …… 65
3.3 NRFS **プロトタイプの作成** …… 67
3.3.1 NRFS **プロトタイプの実装** …… 67
3.3.2 NRFS **プロトタイプの機能検証** …… 70
3.3.3 NRFS **プロトタイプに関するまとめ** …… 119

4. **今期の成果と今後の課題** …… 120

添付資料

A. **ハードウェアチェックサム機構の流用についての調査** …… A-1

1. プロジェクトの概要

パーソナルコンピュータ（PC）およびワークステーションの価格性能比の改善と低価格化は著しく、オフィスや研究所はもちろん家庭においても複数台の計算機が設置され、ローカルエリアネットワーク（LAN）が構築されるようになってきている。LAN 環境ではファイルシステムを共有することで大きな利便性と管理コストの低減効果が得られるため分散共有ファイルシステムが広く採用されている。その代表例が Network File System (NFS) である。NFS が採用された LAN 環境においては、多くのファイルが共用されるため、個々の計算機のローカルディスクに対する容量の要求が低くなる。この使用率が低く遊休資源となっているディスクを有効活用し、低コストかつ高信頼な分散共有ファイルシステムを可能とするネットワーク RAID ファイルシステム（NRFS）を開発する。

従来の分散共有ファイルシステムでは、その信頼性が求められる場合、ハードディスク装置として専用ハードウェアによって構成された高価な RAID 装置を導入する方式が一般的である。これに対し、NRFS では、サーバ計算機自体の障害に耐え得るよう、複数の計算機を多重化したサーバとして取り扱うとともに、ミラーリング方式を拡張した多数決方式による障害検出訂正機能を実装する。これらの対応により、安価な PC をファイルサーバとして使用した場合でも信頼性の高い分散共有ファイルシステムを実現可能である。

なお、ネットワークインタフェースカード（NIC）のチェックサム演算機能やサーバ側によるチェックサム演算を活用することにより、障害検出訂正機能におけるデータ認証オーバヘッドコストおよびネットワークにおけるデータ転送量の軽減を図る。

PC のディスク容量が数十 Gbyte のオーダーに達している現状においては、ディスククラッシュやディスク障害によるトラブルは致命的であり、復旧に多大なコストを要する。NRFS は高価な RAID 装置や高価なサーバ用マシンを導入することなしに、低コストで高信頼な分散共有ファイルシステムを提供するシステムソフトウェアである。

今期は、NRFS の基本メカニズムの設計と開発に重点を置き、開発者が研究開発中である SPARC マシンの SSS-CORE 並びに PC の Linux を対象とし、NRFS のプロトタイプを Virtual File System（VFS）上に構築する。

2. 開発計画

2.1 開発目的

本プロジェクトは、ローカルエリアネットワーク（LAN）環境を構成する計算機のローカルディスクのうち遊休資源となっているものを有効活用した高信頼分散共有ファイルシステムを可能とするネットワーク RAID ファイルシステム（NRFS）を開発することを目的とする。

従来の分散共有ファイルシステムでは、その信頼性が求められる場合、ハードディスク装置として専用ハードウェアによって構成された高価な RAID 装置を導入することが一般的である。これに対し、NRFS では、サーバ計算機自体の障害に耐え得るよう、複数の計算機を多重化したサーバとして取り扱うとともに、障害検出訂正機能を実装する。

NRFS では、安価な PC をサーバとして活用可能であることから、低コストでより信頼性の高い分散共有ファイルシステムを実現可能である。

2.2 開発内容

NRFS と同様に LAN 環境上でファイル共有を実現する機構としては、Network File System（NFS）が広く普及している。NRFS についても NFS の機構が活用できると考えられることから、まず NFS 実装の詳細を解析することとした。

次に、NFS の実装調査結果に基づいて、NFS を拡張する形で NRFS 基本部の設計を行うとともに障害検出訂正機能の詳細について検討を行う。

続いて、NRFS の実用性を実証するために、NRFS プロトタイプの作成を行う。このプロトタイプでは、サーバ多重化に対応したクライアント並びにサーバを整備するとともに、特定の障害発生に対応した障害検出訂正機能を実装することとする。

2.2.1 NFS 実装調査

NRFS の着想の原点は、その名の通り、NFS と Redundant Arrays of Independent Disks（RAID）であり、NFS に RAID 装置のような冗長性を導入することで高信頼性を実現するものである。

NFS は、Sun Microsystems 社が公開しているプロトコル仕様（RFC1094 及び RFC1813）に従って開発されており、多くのプラットフォーム上でサポートされている。NRFS では NFS の機構が活用できると考えられることから、その開発にあたっては、NFS を拡張する形で進めることとした。NFS は多くのプラットフォーム上でサポートされており、将来的に NRFS の実用化を図る上でも都合が良い。

したがって、まず、NFS 実装の詳細を解析することとした。

2.2.2　NRFS 基本部の設計

NRFS は NFS と同様に、クライアントとサーバにより構成される。ただし、NRFS では、サーバ計算機自体の障害に耐え得るよう、サーバを多重化して取り扱うとともに、障害検出訂正機能を実装することとした。

ここでは、前節で実施する NFS 実装の調査結果をもとに、NRFS における各種データ構造を検討するとともに、サーバ多重化への対応並びに関連する RPC プロトコルの追加・拡張といった作業を中心に NRFS 基本部の設計を行う。

また、NRFS の最大の特長である障害検出訂正機能については、ミラーリング方式を拡張した多数決方式を採用することとした。具体的には、NRFS は３台以上のサーバにより構成されることとし、データ利用時にはこれらのサーバが保持するデータの一致性を確認し、不整合がある場合は多数決により正しいデータを判定するとともに不整合を訂正する機能を整備することとするが、当該機構について詳細を検討する。

2.2.3　NRFS プロトタイプの作成

前節で実施する NRFS 基本部の設計に基づいてプロトタイプを作成する。このプロトタイプでは、サーバ多重化に対応したクライアント並びにサーバを整備するとともに、特定の障害発生に対応した障害検出訂正機能を実装することとする。

次に、作成したプロトタイプについて、実際にテストを行い、NRFS の実用性を検証する。NRFS 基本部が設計通りに機能することを確認するとともに、そのパフォーマンスを測定し、実用上支障がないレベルで NRFS が実装可能であることを確認する。また、実装した障害検出訂正機能について、障害発生を人為的に模擬したテストを行い、その健全性を確認することとする。

2.3　開発期間

平成１２年１０月２５日～平成１３年２月２８日

2.4　実施体制

担当プロジェクト・マネージャー	長谷川　正治
開発者	松本　尚
プロジェクト実施管理組織	三菱マテリアル株式会社

3. 成果報告

本プロジェクトで開発するネットワーク RAID ファイルシステム（NRFS）は、ローカルエリアネットワーク（LAN）環境を構成する複数の計算機を多重化したサーバとして取り扱うとともに、サーバ計算機自体の障害に耐え得るよう障害検出訂正機能を実装した、高信頼分散共有ファイルシステムである。

NRFS と同様に LAN 環境上でファイル共有を実現する機構としては、Network File System（NFS）が広く普及している。NRFS についても NFS の機構が活用できると考えられることから、まず NFS 実装の詳細を解析することとした。

次に、NFS の実装調査結果に基づいて、NFS を拡張する形で NRFS 基本部の設計を行うとともに障害検出訂正機能の詳細について検討を行った。

続いて、NRFS 基本部の設計に基づいてプロトタイプを作成した。このプロトタイプでは、サーバ多重化に対応したクライアント並びにサーバを整備するとともに、特定の障害発生に対応した障害検出訂正機能を実装した。作成したプロトタイプについて、実際にテストを行い、NRFS の実用性を検証した。

以下に、その概要を報告する。

3.1 NFS 実装調査

NFS は、Sun Microsystems 社が公開しているプロトコル仕様（RFC1094 及び RFC1813）に従って開発されており、多くのプラットフォーム上でサポートされている。NRFS では NFS の機構が活用できると考えられることから、その開発にあたっては、NFS を拡張する形で進めることとした。NFS は多くのプラットフォーム上でサポートされており、将来的に NRFS の実用化を図る上でも都合が良い。

したがって、まず、NFS 実装の詳細を解析することとした。

Linux における NFS の実装は図 3.1-1 に示すように、統一的なファイルシステムインターフェースである Virtual File System（VFS）の管理下におかれ、他のファイルシステムと同じような位置付けになっている。

さらに NFS プロトコルは Remote Procedure Call（RPC）を用いて構成されるクライアント・サーバアプリケーションである。NFS クライアントは、RPC 要求をサーバに送ることで、サーバ上のファイルにアクセスする。External Data Representation（XDR、外部データ表現）は RPC メッセージ内の NFS プロトコルデータを送信側と受信側の両方のコンピュータで理解出来るようにフォーマットする。

このように、NFS の実装並びに NFS と密接に関連する VFS、RPC、XDR の実装を中心に調査を行うこととした。なお、クライアントがサーバ上の共有ディレクトリのファイルハンドルを最初に取得する際に使用する NFS マウントプロトコルは、NFS プロトコ

ルとは独立して実装されているが、NFS マウントプロトコルも NRFS の開発において重要であるため併せて解析した。

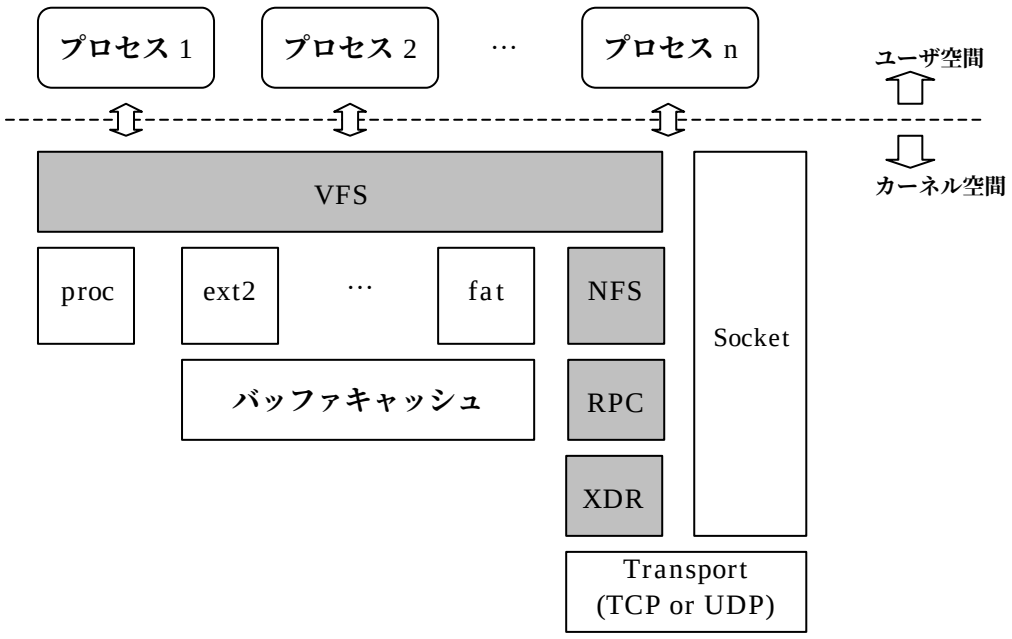


図 3.1-1　カーネル内における NFS 関連モジュール

3.1.1 マウント

Linux では、ユーザプロセスとして動作する NFS マウント用ツール *mount* は *util-linux* にまとめられており、Andries Brouwer (aeb@cwi.nl) によってメンテナンスされている。最新版の util-linux は、ftp://ftp.win.tue.nl/pub/linux/utils/util-linux からダウンロード可能である。今回の調査においては、バージョン 2.9z を使用した。

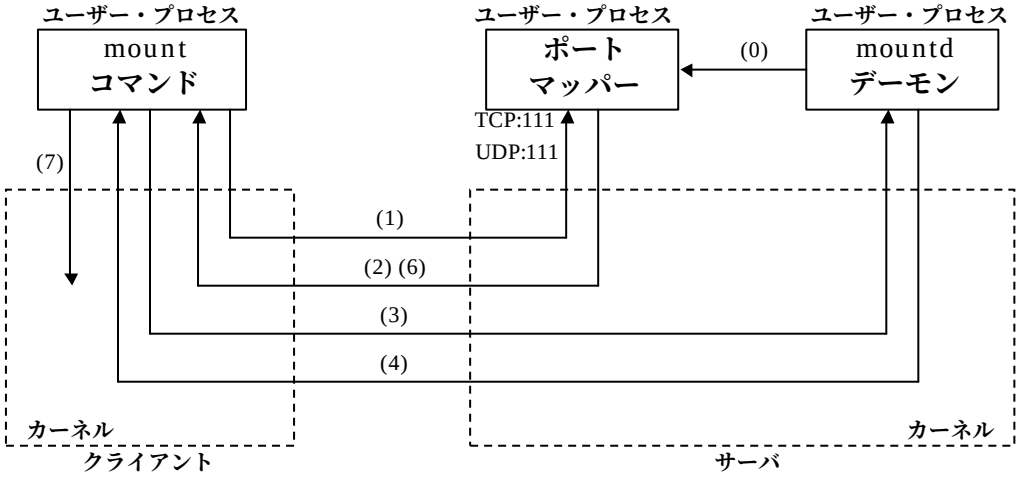
NFS クライアントは、NFS サーバ上の共有ファイルシステムにアクセスするために、最初に NFS マウントプロトコルを用いて当該ファイルシステムをマウントする必要がある。NFS マウント処理の手順を図 3.1.1-1 に示す。

マウントコマンドの処理について、クライアント側を中心に Linux ソースファイルレベルで調査した結果を表 3.1.1-1 に示す。一連の処理では様々な構造体が利用されているが、主なものを表 3.1.1-2～表 3.1.1-9 に列挙する。

また、クライアント側の関数呼び出し手続きについてその処理フローを整理した。ユーザ空間内の処理フローを表 3.1.1-10 に、カーネル空間内の処理フローを表 3.1.1-11 にそれぞれ示す。

参考までに NFS クライアントのマウント処理に関連する各種ソースファイルの一覧を表 3.1.1-12 に示す。

なお、サーバ側の処理に関しては、今回の NRFS 開発において NFS サーバの実装をそのまま活用できることが判明したため、ここでは割愛する。



(0) *mountd* デーモンプロセスは自プロセスのエフェメラルポート番号をポートマッパーに登録する（PMAPPROC_SET）。

(1) マウント開始時に、クライアントの *mount* ユーザプロセスはポートマッパーのウェルノウンポートに対して RPC コールを発行して、mountd デーモンのポート番号を問い合わせる（PMAPPROC_GETPORT）。

(2) ポートマッパーがマウントポートを返す。

(3) *mount* コマンドは *mountd* デーモンに対して TCP または UDP プロトコルを用いて、サーバ上のファイルシステムをマウントするためにパス名を引数として RPC コールを発行する（MOUNTPROC_MNT）。

(4) RPC に成功すると、*mountd* デーモンは当該ファイルシステムのファイルハンドルを返す。このファイルハンドルは以後、クライアントがサーバ上の共有ファイルシステムにアクセスするために必要なオブジェクトである。ただし、クライアントはファイルハンドルの中身に関しては一切関知せず、サーバのみがその内容（inode 番号などが格納されている）を把握する。

(5) ポートマッパーに対して、NFS ポート番号を問い合わせる。

(6) NFS ポート番号（2049 等）を返す。

(7) mount コマンドはファイルハンドル、NFS ポート番号などをとりまとめて、カーネルに対して *mount* システムコールを発行する。

図 3.1.1-1　NFS マウント処理の手順

表 3.1.1-1　マウントコマンドの処理概要

マウントコマンド（ユーザ空間内）	システムコール（カーネル空間内）
① コマンドラインオプションの解析およびパラメータの調整する。 ② TCP/UDP ベースの RPC を用いて、NFS サーバから指定したディレクトリのファイルハンドルを取る。 注： ヘッダファイルとクライアントの手続き、ＸＤＲ関連手続きの作成に rpcgen が使われている。 ③ ソケットなどを初期化する。 ④ ファイルハンドルを他のパラメータと一緒に nfs_mount_data 構造体(表 3.1.1-2)にまとめる。 ⑤ マウント元、マウント先、ファイルシステムタイプ、nfs_mount_data 構造体を引数にマウントシステムコールを呼び出す。 ⑥ マウントテーブルファイル(/etc/mtab)を更新する。	マウントエントリの管理変数 i）ファイルシステムタイプリスト ii）マウント済みファイルシステムリスト ---------- VFS mount ---------- ① nfs_mount_data 構造体をカーネルメモリ空間にコピーする。 ② マウント済みかをチェックし、マウント済みであれば、作業を中断して-EBUSY を返す。 ③ 空いているデバイス番号（0，x）を取得する。 ④ 管理変数 i) から NFS の file_system_type 構造体（表 3.1.1-4）へのポインターを取得する。 ⑤ 新しい super_block 構造体（表 3.1.1-3）を確保する。 ⑥ nfs_mount_data と super_block 構造体を引数に NFS の read_super()を呼び出す。 ---------- NFS read_super ---------- ・ NFS の UDP クライアントを作成する。 ・ カーネルスレッド rpciod を立上げる。 ・ NFS ルートファイルハンドルの設定値(attribute)を取得する。 ・ NFS の super_block 構造体の各メンバを設定する。 ---------- NFS read_super ---------- ⑦ マウントするデバイスをチェックする。 ⑧ 管理変数 ii)にエントリを追加する。 ---------- VFS mount ----------

表 3.1.1-2　NFS マウントデータ構造体

構造体	備考

include/linux/nfs_mount.h
 	
struct nfs_mount_data {
　int　　version;　　/* 1 */
　int　　fd;　　/* 1 */
　struct nfs_fh　root;　　/* 1 */
　int　　flags;　　/* 1 */
　int　　rsize;　　/* 1 */
　int　　wsize;　　/* 1 */
　int　　timeo;　　/* 1 */
　int　　retrans;　　/* 1 */
　int　　acregmin;　　/* 1 */
　int　　acregmax;　　/* 1 */
　int　　acdirmin;　　/* 1 */
　int　　acdirmax;　　/* 1 */
　struct sockaddr_in addr; /* 1 */
　char　　hostname[256];/* 1 */
　int　　namlen;　　/* 2 */
　unsigned int　bsize;　　/* 3 */
};
 	
注：この構造体の各フィールドを削除したり順番を変更してはならない。新しいフィールドが必要な時は最後に追加しなければならない。（マウントコマンドと整合性を取るため？）
	マウントツール（util-linux-2.xx/mount/*）によって作成される。
	
	by socket()
	by RPC call → MOUNTPROC_MNT
	
	7 when udp, 70 when tcp
	3
	3
	60
	30
	60
	port (by pmap_getport() or NFS_PORT=2049)
	NFS Server name

表 3.1.1-3　スーパーブロック構造体

構造体	備考
include/linux/fs.h	fs/nfs/inode.c
struct super_block {	
struct list_head s_list;	VFS
kdev_t s_dev;	
unsigned long s_blocksize;	nfs, call by nfs_block_size()
unsigned char s_blocksize_bits;	nfs, call by nfs_block_size()
unsigned char s_lock;	VFS
unsigned char s_rd_only;	VFS, (0)
unsigned char s_dirt;	VFS
struct file_system_type *s_type;	VFS (表3.1.1-4)
struct super_operations *s_op;	nfs, nfs_sops (表3.1.1-5)
struct dquot_operations *dq_op;	
unsigned long s_flags;	VFS, (0)
unsigned long s_magic;	nfs, NFS_SUPER_MAGIC
unsigned long s_time;	
struct dentry *s_root;	nfs, ←struct nfs_fh、nfs_dentry_operations(表3.1.1-6)
struct wait_queue *s_wait;	
struct inode *s_ibasket;	
short int s_ibasket_count;	
short int s_ibasket_max;	
struct list_head s_dirty;	
union {	
struct minix_sb_info minix_sb;	
struct ext2_sb_info ext2_sb;	
・・・	
struct nfs_sb_info nfs_sb;	nfs, ←struct nfs_server←struct rpc_clnt (表3.1.1-7)
・・・	
void *generic_sbp;	
} u;	
/* for VFS *only*. */	
struct semaphore s_vfs_rename_sem;	
};	

表 3.1.1-4　ファイルシステム構造体

構造体	備考

include/linux/fs.h
 	
struct file_system_type {
　const char *name;
　int fs_flags;
　struct super_block　*(*read_super)
　　(struct super_block *, void *, int);
　struct file_system_type * next;
};
	fs/nfs/inode.c で初期化
	
	nfs_fs_type = {
	　"nfs",
	　0,
	　nfs_read_super,
	
	　NULL
	}

表 3.1.1-5　スーパーブロック操作構造体

構造体	備考

include/linux/fs.h
 	
struct super_operations {
　void (*read_inode) (struct inode *);
　void (*write_inode) (struct inode *);
　void (*put_inode) (struct inode *);
　void (*delete_inode) (struct inode *);
　int (*notify_change) (struct dentry *, struct iattr *);
　void (*put_super) (struct super_block *);
　void (*write_super) (struct super_block *);
　int (*statfs) (struct super_block *, struct statfs *, int);
　int (*remount_fs) (struct super_block *, int *, char *);
　void (*clear_inode) (struct inode *);
　void (*umount_begin) (struct super_block *);
};
	fs/nfs/inode.c で初期化
	
	nfs_sops = {
	　nfs_read_inode,
	　NULL,
	　nfs_put_inode,
	　nfs_delete_inode,
	　nfs_notify_change,
	　nfs_put_super,
	　NULL,
	　nfs_statfs,
	　NULL,
	　NULL,
	　nfs_umount_begin
	};

表 3.1.1-6　ディレクトリキャッシュ及び操作定義構造体

構造体	備考
include/linux/dcache.h	fs/nfs/inode.c, dir.c

struct dentry {
int d_count;
unsigned int d_flags;
struct inode * d_inode; /* Where the name belongs to
- NULL is negative */
struct dentry * d_parent; /* parent directory */
struct dentry * d_mounts; /* mount information */
struct dentry * d_covers;
struct list_head d_hash; /* lookup hash list */
struct list_head d_lru; /* d_count = 0 LRU list */
struct list_head d_child; /* child of parent list */
struct list_head d_subdirs;/* our children */
struct list_head d_alias; /* inode alias list */
struct qstr d_name;
unsigned long d_time; /* used by d_revalidate */
struct dentry_operations *d_op;
struct super_block * d_sb; /* The root of dentry tree */
unsigned long d_reftime; /* last time referenced */
void * d_fsdata; /* fs-specific data */
unsigned char d_iname[DNAME_INLINE_LEN]; /* small names */
};
	itself
	
	nfs, ←nfs_dentry_operations
	itself's super_block
	
	nfs, ←struct nfs_fh

struct dentry_operations {
int (*d_revalidate)(struct dentry *, int);
int (*d_hash) (struct dentry *, struct qstr *);
int (*d_compare) (struct dentry *,
struct qstr *,
struct qstr *);
void (*d_delete)(struct dentry *);
void (*d_release)(struct dentry *);
void (*d_iput)(struct dentry *, struct inode *);
};
	nfs_dentry_operations = {
	nfs_lookup_revalidate,
	NULL,
	NULL,
	
	nfs_dentry_delete,
	nfs_dentry_release,
	NULL
	};

表 3.1.1-7　NFS スーパーブロック情報構造体

構造体	備考
include/linux/nfs_fs_sb.h	fs/nfs/inode.c

struct nfs_sb_info {
　struct nfs_server　s_server;
　struct nfs_fh　　s_root;
};
	
	nfs, struct rpc_clnt
	nfs, root file handle for NFS
	　　↑ﾏｳﾝﾄﾃﾞｰﾀ、
	　　｜ﾏｳﾝﾄｺﾏﾝﾄﾞ引数より

表 3.1.1-8　NFS サーバ情報構造体

構造体	備考
include/linux/nfs_fs_sb.h	fs/nfs/inode.c

struct nfs_server {
　struct rpc_clnt *client;　/* RPC client handle */
　int　　flags;　/* various flags */
　int　　rsize;　/* read size */
　int　　wsize;　/* write size */
　unsigned int　bsize;　/* server block size */
　unsigned int　acregmin;　/* attr cache timeouts */
　unsigned int　acregmax;
　unsigned int　acdirmin;
　unsigned int　acdirmax;
　char *　　hostname;　/* remote hostname */
};
	
	nfs, by rpc_create_client()
	nfs,
	nfs, by nfs_block_size()
	nfs, by nfs_block_size()
	nfs, by nfs_block_size()
	nfs, from ﾏｳﾝﾄﾃﾞｰﾀ, for file
	nfs, from ﾏｳﾝﾄﾃﾞｰﾀ, for file
	nfs, from ﾏｳﾝﾄﾃﾞｰﾀ, for dir
	nfs, from ﾏｳﾝﾄﾃﾞｰﾀ, for dir
	nfs, from ﾏｳﾝﾄﾃﾞｰﾀ
	　　↑ﾏｳﾝﾄｺﾏﾝﾄﾞ引数より

注：NFS サーバにアクセスするためのパラメータを格納する構造体である。

表 3.1.1-9　NFS ルートファイルハンドル構造体

構造体	備考
include/linux/nfs_fs_sb.h	fs/nfs/inode.c

struct nfs_fh {
　char　　data[NFS_FHSIZE];
};
	
	NFS Ver.2 では 32 バイト、Ver.3 では 64 バイトに拡張される。

表 3.1.1-10　マウントコマンドのユーザ空間内処理フロー

関数の呼び出しフロー	備考（引数、その他）	ファイル名

main() {
  Call mount_one
}
 	
mount_one() {
  Call try_mount_one
}
 	
try_mount_one() {
  Call nfsmount
  Call try_mount5
 	
  Call my_addmntent
}
 	
try_mount5 {
  Call mount5
}
 	
mount5 {
  Call mount ← SYSTEM CALL
  count up successful sys-call
}
 	
カーネル空間へ
	マウント元、マウント先、解析済みオプション
	
	マウント元、マウント先、ファイルシステムタイプ、メインから受け取ったオプションの元に新たに作成したその他のオプション
	
	マウント元、マウント先、その他のオプション
	マウント元、マウント先、タイプ、nfsmount が作成したマウントデータ、その他のオプション
	
	マウント元、マウント先、タイプ、nfsmount が作成したマウントデータ、その他のオプション
	
	マウント元、マウント先、タイプ、nfsmount が作成したマウントデータ、その他のオプション
		mount.c

nfsmount() {
  Call clnttcp_create()
  or clntudp_create()
  Call authunix_create_default
  Call clnt_call
  (get root file handle)
  Call socket
  Call bindresvport
  Call pmap_getport
  Call auth_destroy
  Call clnt_destroy
  Call close
  (for closing clnt sock)
}
	Port=0,sock=RPC_ANYSOCK,IP addr
	
	Program=MOUNTPROC_MNT
		nfsmount.c

表 3.1.1-11　マウントコマンドのカーネル空間内処理フロー

ユーザ空間より

関数の呼び出しフロー	備考（引数、その他）	ファイル名

sys_mount() {
  Call get_unnamed_dev
 	
  Call do_mount
}
	
	メジャー番号が０の空いているマイナー番号
	
	デバイス番号、マウント元、マウント先、ファイルシステム名、 マウントデータ（呼出す前、にマウント済みかのチェックあり）
		super.c

do_mount() {
  Call read_super
  Call fs_may_mount
  Call add_vfsmnt
  Call dmount
}
	
	デバイス番号、マウント元、マウント先、ファイルシステム名、 マウントデータ	

read_super() {
  Call get_fs_type
  Call get_empty_super
  Call nfs_read_super
}
	
	取得したスーパーブロック構造体、マウントデータ	

nfs_read_super() {
  Call xprt_create_proto
  Call rpc_create_client
  Call rpciod_up
  Call nfs_proc_getattr
}
	
	Proto=IPPROTO_UDP,IP addr, …
	
	RPC, NFS ルート ファイル情報の取得ため
		inode.c

表 3.1.1-12　NFS クライアントのマウント関連ソースファイル一覧

ファイル名	実装内容	備　考
fstab.c	fatab ファイル操作関連	mount.c の中から call
getusername.c	パスワードファイルからユーザ名を取得する手続きの実装	
lomount.c	ループデバイスの操作関連	mount.c の中から call
losetup.c	ループデバイスの操作関連。	mount.c の中から call
mntent.c	マウントエントリ操作用ヘルパー手続き	mount.c の中から call
mount.c	マウント操作	
mount_by_label.c	ラベルオプション指定された時使用する手続き	mount.c の中から call
nfsmount.x	NFS マウントための RPC 定義	
nfsmount.c	NFS マウント操作	mount.c の中から call
nfsmount_clnt.c	NFS クライアント用 RPC 手続き	nfsmount.x より生成
nfsmount_xdr.c	NFS クライアント用 XDR 手続き	nfsmount.x より生成
realpath.c	正規化された絶対パス名を作成する手続き	glibc の realpath に相当
sundries.c	文字列操作用ヘルプ手続き	
umount.c	アンマウント操作	
version.c	バージョンの設定	

3.1.2　KRPC

NFS は RPC 型のプロトコルであり、共有ファイルシステムを分配するサーバと当該ファイルシステムに対し RPC によりアクセスし、利用するクライアントからなるクライアント・サーバシステムである。

前項に示した NFS マウントされたファイルシステムに対するクライアントからのアクセス（システムコール）は、後述する Virtual File System（VFS）により NFS に渡され、該当する NFS サーバに対して NFS クライアントの RPC プロシージャが実行される。

サーバ側では NFS サーバデーモン nfsd がクライアントからの RPC を受け取り、各処理に該当する RPC プロシージャがファイルあるいはファイルシステムオブジェクトの操作を行う（実際の処理は関連する VFS プロシージャが行う）とともに必要に応じてクライアントへ情報を返す。

サーバより返された情報は、呼び出し元のクライアント RPC プロシージャより VFS を経由して当該処理の起源であるユーザプロセスへ戻される。

NFS のアクセスについてその概略フローを図 3.1.2-1 に示す。また、RPC 通信モデルの概略を図 3.1.2-2 に、RPC タスクのライフサイクル略図を図 3.1.2-3 にそれぞれ示す。

NFS の RPC プロトコルには 18 種のプロシージャがある。クライアント、サーバそれぞれについてプロシージャの一覧を表 3.1.2-1、2 に示す。なお、ここでは、各プロシージャに関連付けられる External Data Representation（XDR、外部データ表現）のエンコード／デコード手続きについても便宜上、併記した。XDR については後述する（3.1.3 項）。

NFS サーバにおける RPC プロシージャの概要を表 3.1.2-3 にまとめる。なお、ここでは、各サーバプロシージャに対し、実際の処理を担う VFS プロシージャを併記した。

これらを整理した結果を図 3.1.2-4 に示す。

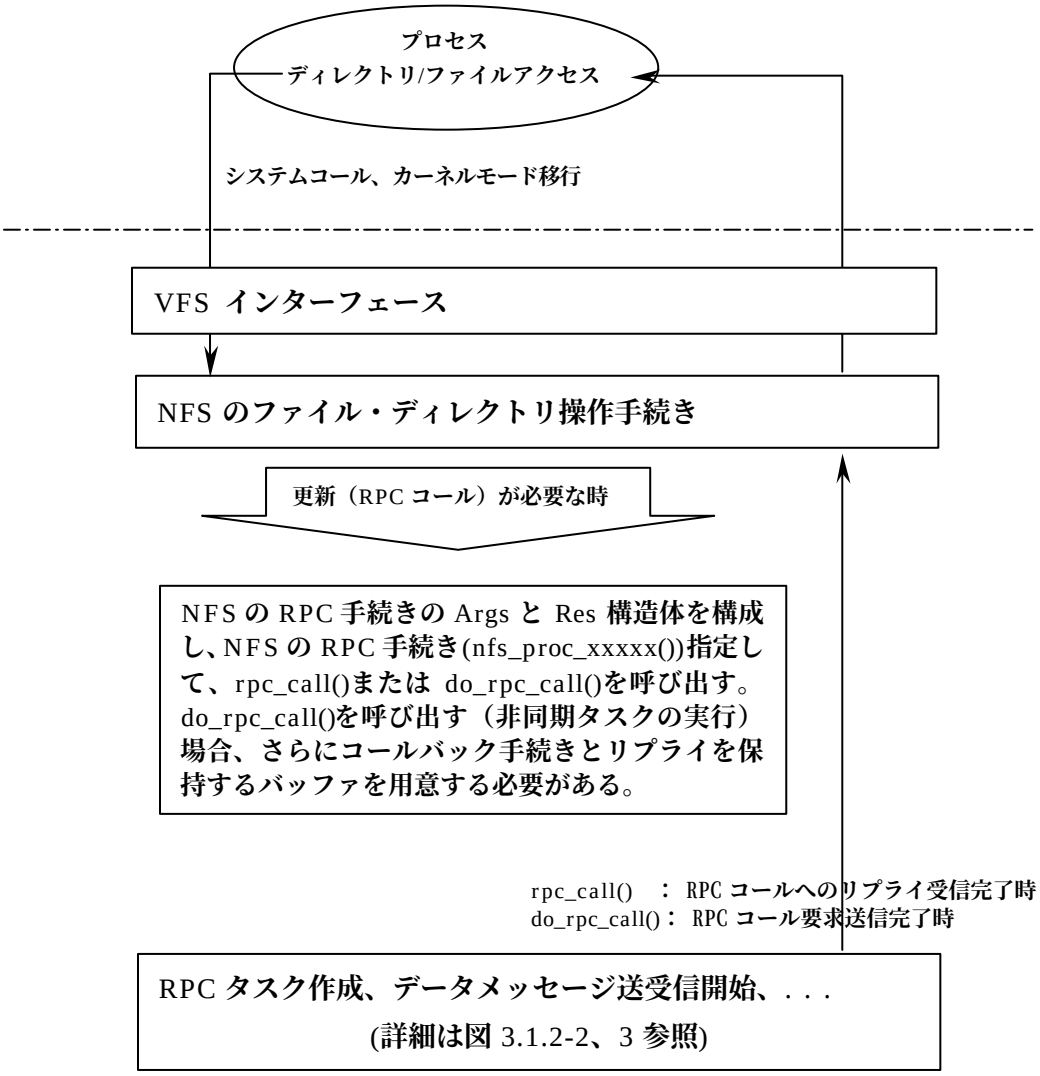


図 3.1.2-1　NFS ファイルシステムのアクセス略図（同期タスク）

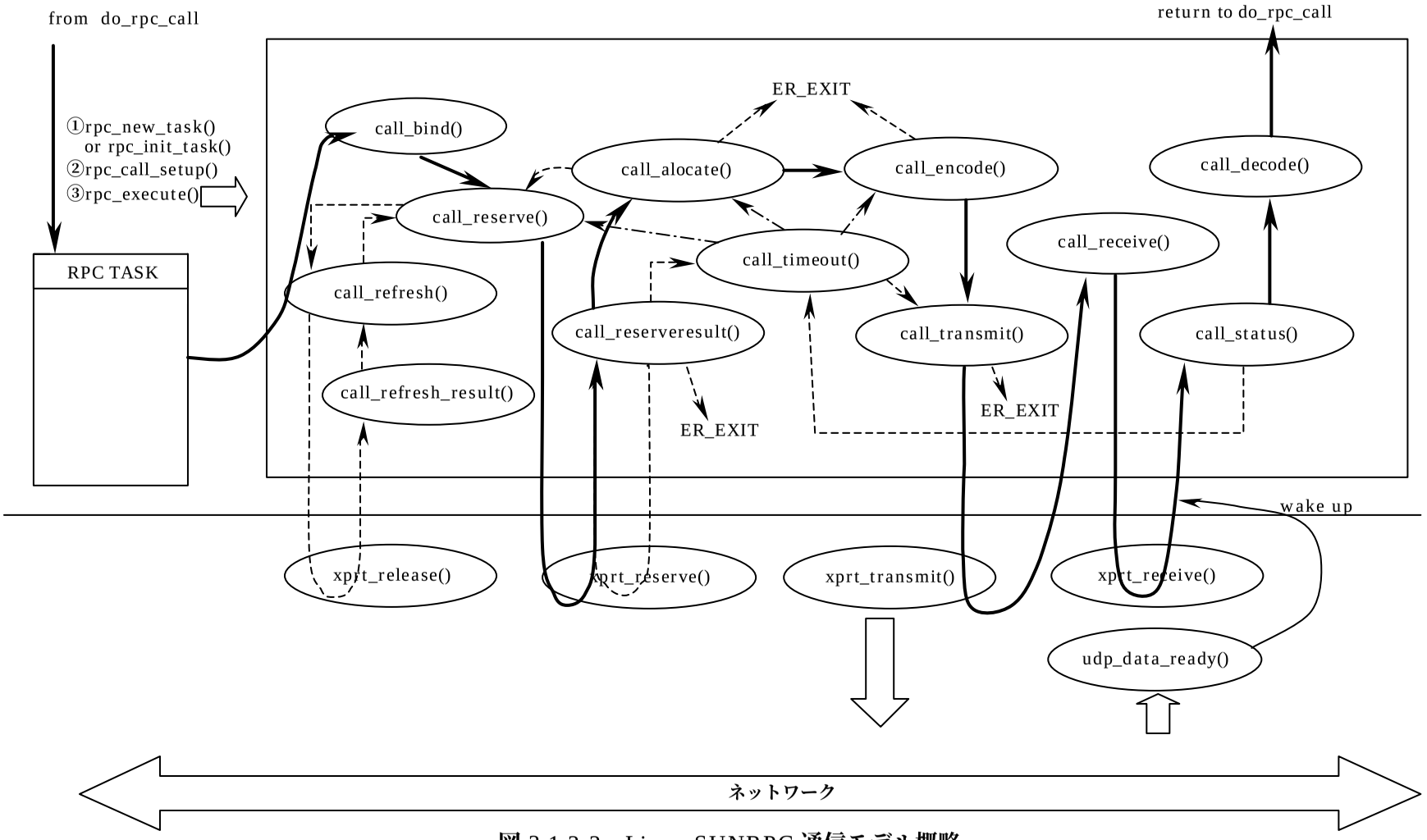


図 3.1.2-2　Linux SUNRPC 通信モデル概略

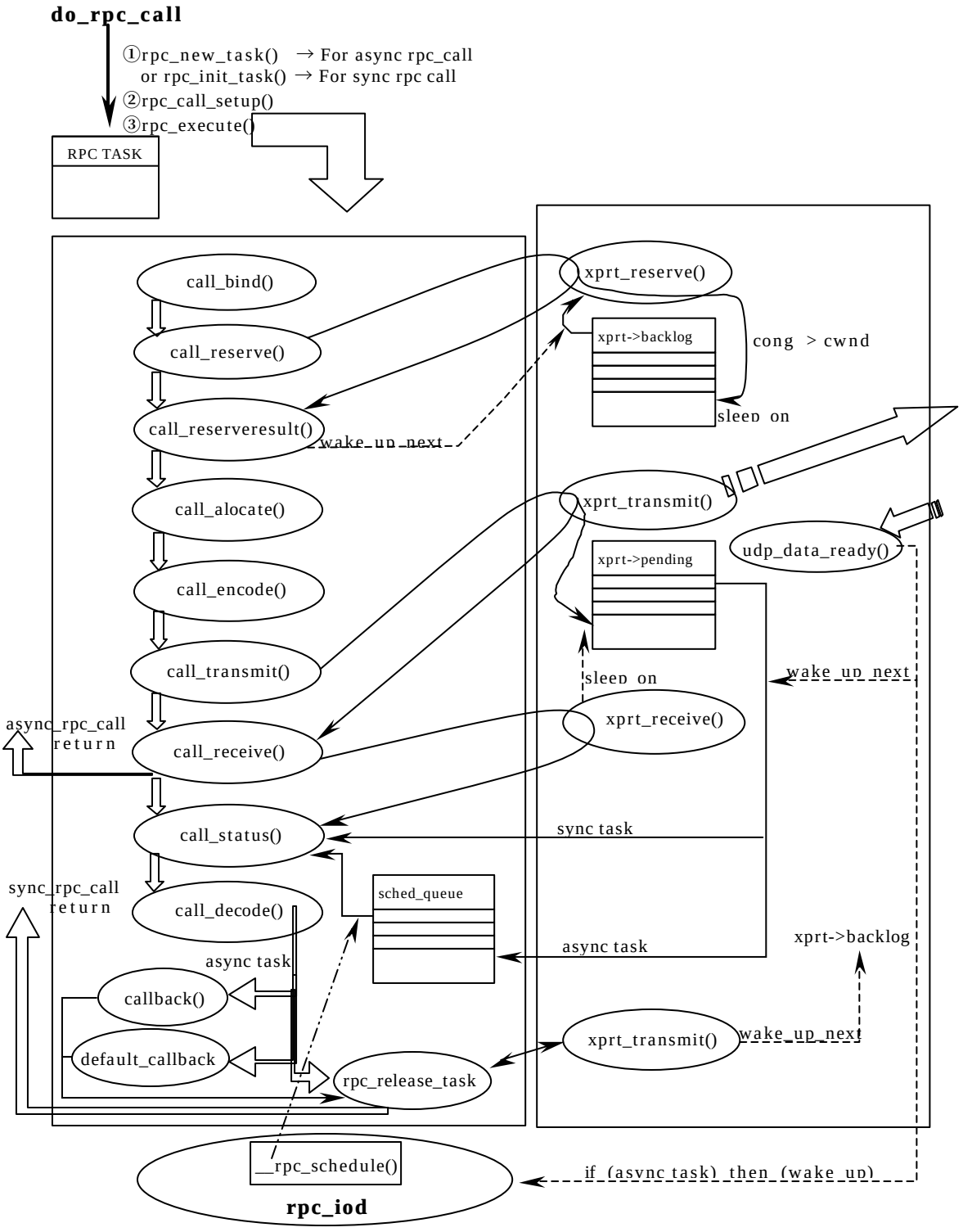


図 3.1.2-3　RPC **タスクのライフサイクル略図**

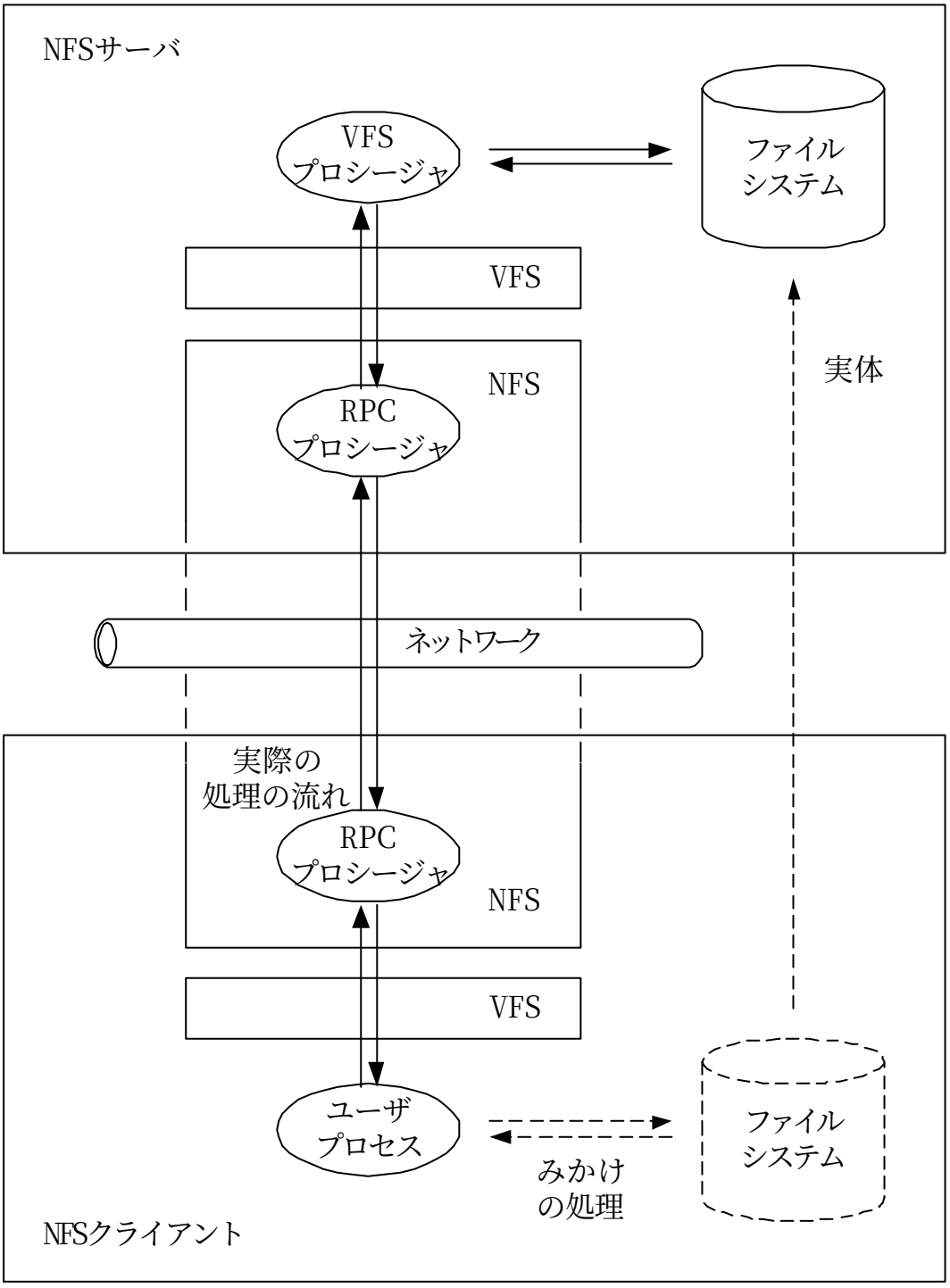


図3.1.2-4　NFSのRPCによるアクセス概略

表 3.1.2-1　NFS クライアントの RPC プロシージャ一覧

```
#define QUADLEN(len)            (((len) + 3) >> 2)

#define PROC(proc, argtype, restype)    \
    { "nfs_" #proc,                                     \
      (kxdrproc_t) nfs_xdr_##argtype,                   \
      (kxdrproc_t) nfs_xdr_##restype,                   \
      MAX(NFS_##argtype##_sz,NFS_##restype##_sz) << 2   \
    }
```

RPC プロシージャ	XDR エンコーダ	XDR デコーダ

```
static struct rpc_procinfo      nfs_procedures[18] = {
    PROC(null,          enc_void,       dec_void),
    PROC(getattr,       fhandle,        attrstat),
    PROC(setattr,       sattrargs,      attrstat),
    PROC(root,          enc_void,       dec_void),
    PROC(lookup,        diropargs,      diropres),
    PROC(readlink,      fhandle,        readlinkres),
    PROC(read,          readargs,       readres),
    PROC(writecache,    enc_void,       dec_void),
    PROC(write,         writeargs,      attrstat),
    PROC(create,        createargs,     diropres),
    PROC(remove,        diropargs,      stat),
    PROC(rename,        renameargs,     stat),
    PROC(link,          linkargs,       stat),
    PROC(symlink,       symlinkargs,    stat),
    PROC(mkdir,         createargs,     diropres),
    PROC(rmdir,         diropargs,      stat),
    PROC(readdir,       readdirargs,    readdirres),
    PROC(statfs,        fhandle,        statfsres),
};
```

表 3.1.2-2　NFS サーバの RPC プロシージャ一覧

```
#define nfsd_proc_none          NULL
#define nfssvc_release_none     NULL
struct nfsd_void { int dummy; };

#define PROC(name, argt, rest, relt, cache)     \
 { (svc_procfunc) nfsd_proc_##name,             \
   (kxdrproc_t) nfssvc_decode_##argt,           \
   (kxdrproc_t) nfssvc_encode_##rest,           \
   (kxdrproc_t) nfssvc_release_##relt,          \
   sizeof(struct nfsd_##argt),                  \
   sizeof(struct nfsd_##rest),                  \
   0,                                           \
   cache                                        \
 }
```

RPC プロシージャ	XDR デコーダ	XDR エンコーダ		
struct svc_procedure		nfsd_procedures2[18] = {		
PROC(null,	void,	void,	none,	RC_NOCACHE),
PROC(getattr,	fhandle,	attrstat,	fhandle,	RC_NOCACHE),
PROC(setattr,	sattrargs,	attrstat,	fhandle,	RC_REPLBUFF),
PROC(none,	void,	void,	none,	RC_NOCACHE),
PROC(lookup,	diropargs,	diropres,	fhandle,	RC_NOCACHE),
PROC(readlink,	fhandle,	readlinkres,	none,	RC_NOCACHE),
PROC(read,	readargs,	readres,	fhandle,	RC_NOCACHE),
PROC(none,	void,	void,	none,	RC_NOCACHE),
PROC(write,	writeargs,	attrstat,	fhandle,	RC_REPLBUFF),
PROC(create,	createargs,	diropres,	fhandle,	RC_REPLBUFF),
PROC(remove,	diropargs,	void,	none,	RC_REPLSTAT),
PROC(rename,	renameargs,	void,	none,	RC_REPLSTAT),
PROC(link,	linkargs,	void,	none,	RC_REPLSTAT),
PROC(symlink,	symlinkargs,	void,	none,	RC_REPLSTAT),
PROC(mkdir,	createargs,	diropres,	fhandle,	RC_REPLBUFF),
PROC(rmdir,	diropargs,	void,	none,	RC_REPLSTAT),
PROC(readdir,	readdirargs,	readdirres,	none,	RC_REPLBUFF),
PROC(statfs,	fhandle,	statfsres,	none,	RC_NOCACHE),
};				

表 3.1.2-3　NFS サーバにおける RPC プロシージャの概要

RPCプロシージャ	概要	主に使用するVFSプロシージャ
nfsd_proc_null	実質的に何も行わないプロシージャで、NFSにおけるPINGの役割を果たす。　サーバでは、受け取った引数のRPC、NFS引数の整合性がとれているかのみをチェックしRPCレスポンスで結果を返す。	なし
nfsd_proc_getattr	ファイルハンドルで示されるファイル、ディレクトリ属性を返す。	fh_verify
nfsd_proc_setattr	ファイルハンドルで示されるファイル、ディレクトリ属性を設定する。	nfsd_setattr
nfsd_proc_none	NFSプロトコル仕様上は、nfsd_proc_rootであり、NFSで公開されているｒｏｏｔパスのファイルハンドルを返すものである。 ただし、Linux KNFSでは、当該プロシージャはマウントプロトコルで置き換えられため、実装されていない。	なし
nfsd_proc_lookup	ファイルハンドルで示されるディレクトリ下の、ファイル／ディレクトリのファイルハンドルを返す。	nfsd_lookup
nfsd_proc_readlink	ファイルハンドルで示されるリンク情報を返す。	nfsd_readlink
nfsd_proc_read	ファイルハンドルで示されるファイルのオフセット位置からレングス長のデータを読み込む。	nfsd_read
nfsd_proc_none	NFSプロトコル仕様上は、nfsd_proc_writecacheであり、NFS書き込み時に、メモリキャッシュされているデータをディスクに書き込むもので、syncの役割を果たす。 ただし、Linux KNFSでは、当該プロシージャは未実装である。	なし
nfsd_proc_write	ファイルハンドルで示されるファイルのオフセット位置からレングス長のデータを書き込む。	nfsd_write
nfsd_proc_create	ファイルハンドルで示されるディレクトリ下に、指定ファイル名、属性を持つファイルを作成する。	fh_verify nfsd_permission nfsd_lookup fh_lock_parent fh_update nfsd_create nfsd_setattr fh_unlock
nfsd_proc_remove	ファイルハンドルで示されるファイルを削除する。	nfsd_unlink
nfsd_proc_rename	ファイルハンドルで示されるディレクトリ下のファイル／ディレクトリ名を指定されたファイルハンドルで示されるディレクトリ下のファイル／ディレクトリ名に変更する。	nfsd_rename
nfsd_proc_link	２つのファイルハンドルで示されるファイル／ディレクトリを指定された名称でハードリンクする。	nfsd_link
nfsd_proc_symlink	ファイルハンドルで示されるディレクトリ下のファイル／ディレクトリを指定された名称、属性でシンボリックリンクする。	nfsd_symlink nfsd_setattr
nfsd_proc_mkdir	ファイルハンドルで示されるディレクトリ下に、指定された名称、属性でディレクトリを作成する。	nfsd_create
nfsd_proc_rmdir	ファイルハンドルで示されるディレクトリ下の指定された名称のディレクトリを削除する。	nfsd_unlink
nfsd_proc_readdir	ファイルハンドルで示されるディレクトリのディレクトリ情報を返す。	nfsd_readdir
nfsd_proc_statfs	ファイルハンドルで示されるファイル／ディレクトリのステータス情報を返す。	nfsd_statfs

3.1.3 XDR

前項にも示したように、NFS は RPC 型のプロトコルであり、NFS マウントされたファイルシステムに対するクライアントからのアクセスについては、NFS クライアント並びに NFS サーバにおける RPC プロシージャを用いて実装されている。

通常、RPC を使用するアプリケーションが前提としているのは、バイトストリームよりも複雑な構造体やデータ型を利用するシステムコール型のインターフェースであり、プレゼンテーション層が、引数バッファのエンコーディング及びデコーディングを行う。その結果が RPC に渡され、クライアントまたはサーバへ転送される。

NFS のプレゼンテーション層で使用されるプロトコルは、External Data Representation（XDR、外部データ表現）プロトコルである。XDR は Sun Microsystems 社で開発されたプロトコルで、canonical form（カノニカル形式）と呼ばれる固定ネットワーク・バイトオーダーリングという考え方に基づいて構築されている。

XDR のルールでは、『送信元はローカル形式からカノニカル形式に変換（エンコーディング）を行い、ディスティネーションがカノニカル形式からローカル形式への変換（デコーディング）を行う』ため、個々のマシンの特異性とは無関係に、構造体データの受け渡しがホスト間で容易にできるようになる。

NFS クライアント並びにサーバの各 RPC プロシージャに関連付けられた XDR エンコード／デコード手続きについては、RPC プロシージャとともに前項の表 3.1.2-1、2 に併記した。

各エンコーダ／デコーダの概要を表 3.1.3-1～4 にまとめる。

表 3.1.3-1　NFS クライアントの RPC プロシージャに関連付けられた XDR エンコーダの概要

XDRエンコーダ	適用される RPCプロシージャ	関連する構造体	主要な処理
nfs_xdr_enc_void	nfs_proc_root nfs_proc_write_cache	なし	xdr_adjust_iovec()
nfs_xdr_fhandle	nfs_proc_getattr nfs_proc_readlink nfs_proc_statfs	struct nfs_fh	xdr_encode_fhandle() xdr_adjust_iovec()
nfs_xdr_sattrargs	nfs_proc_setattr	struct nfs_sattrargs	xdr_encode_fhandle() xdr_encode_sattr() xdr_adjust_iovec()
nfs_xdr_diropargs	nfs_proc_lookup nfs_proc_remove nfs_proc_rmdir	struct nfs_diropargs	xdr_encode_fhandle() xdr_encode_string() xdr_adjust_iovec()
nfs_xdr_readargs	nfs_proc_read	struct nfs_readargs	xdr_encode_fhandle() htonl() xdr_adjust_iovec()
nfs_xdr_writeargs	nfs_proc_write	struct nfs_writeargs	xdr_encode_fhandle() htonl() xdr_adjust_iovec()
nfs_xdr_createargs	nfs_proc_create nfs_proc_mkdir	struct nfs_createargs	xdr_encode_fhandle() xdr_encode_string() xdr_encode_sattr() xdr_adjust_iovec()
nfs_xdr_renameargs	nfs_proc_rename	struct nfs_renameargs	xdr_encode_fhandle() xdr_encode_string() xdr_adjust_iovec()
nfs_xdr_linkargs	nfs_proc_link	struct nfs_linkargs	xdr_encode_fhandle() xdr_encode_string() xdr_adjust_iovec()
nfs_xdr_symlinkargs	nfs_proc_symlink	struct nfs_symlinkargs	xdr_encode_fhandle() xdr_encode_string() xdr_encode_sattr() xdr_adjust_iovec()
nfs_xdr_readdirargs	nfs_proc_readdir	struct nfs_readdirargs	xdr_encode_fhandle() htonl() xdr_adjust_iovec()

表 3.1.3-2　NFS クライアントの RPC プロシージャに関連付けられた XDR デコーダの概要

XDRデコーダ	適用される RPCプロシージャ	関連する構造体	主要な処理
nfs_xdr_dec_void	nfs_proc_root nfs_proc_write_cache	なし	
nfs_xdr_attrstat	nfs_proc_getattr nfs_proc_setattr nfs_proc_write	struct nfs_fattr	ntohl() xdr_decode_fattr()
nfs_xdr_diropres	nfs_proc_lookup nfs_proc_create nfs_proc_mkdir	struct nfs_diropok	ntohl() xdr_decode_fhandle() xdr_decode_fattr()
nfs_xdr_readlinkres	nfs_proc_readlink	struct nfs_readlinkres	ntohl() xdr_decode_string2()
nfs_xdr_readres	nfs_proc_read	struct nfs_readres	ntohl() xdr_decode_fattr()
nfs_xdr_stat	nfs_proc_remove nfs_proc_rename nfs_proc_link nfs_proc_symlink nfs_proc_rmdir	なし	
nfs_xdr_readdirres	nfs_proc_readdir	struct nfs_readdirres	ntohl()
nfs_xdr_statfsres	nfs_proc_statfs	struct nfs_fsinfo	ntohl()

表 3.1.3-3　NFS サーバの RPC プロシージャに関連付けられた
XDR デコーダの概要

XDRデコーダ	適用される RPCプロシージャ	関連する構造体	主要な処理
nfssvc_decode_void	nfsd_proc_none	なし	xdr_argsize_check()
nfssvc_decode_fhandle	nfsd_proc_getattr nfsd_proc_readlink nfsd_proc_statfs	struct svc_fh	decode_fh() xdr_argsize_check()
nfssvc_decode_sattrargs	nfsd_proc_setattr	struct nfsd_sattrargs	decode_fh() decode_sattr() xdr_argsize_check()
nfssvc_decode_diropargs	nfsd_proc_lookup nfsd_proc_remove nfsd_proc_rmdir	struct nfsd_diropargs	decode_fh() decode_filename() xdr_argsize_check()
nfssvc_decode_readargs	nfsd_proc_read	struct nfsd_readargs	decode_fh() ntohl() xdr_argsize_check()
nfssvc_decode_writeargs	nfsd_proc_write	struct nfsd_writeargs	decode_fh() ntohl() xdr_argsize_check()
nfssvc_decode_createargs	nfsd_proc_create nfsd_proc_mkdir	struct nfsd_createargs	decode_fh() decode_filename() xdr_argsize_check()
nfssvc_decode_renameargs	nfsd_proc_rename	struct nfsd_linkargs	decode_fh() decode_filename() xdr_argsize_check()
nfssvc_decode_linkargs	nfsd_proc_link	struct nfsd_linkargs	decode_fh() decode_filename() xdr_argsize_check()
nfssvc_decode_symlinkargs	nfsd_proc_symlink	struct nfsd_symlinkargs	decode_fh() decode_filename() decode_pathname() decode_sattr() xdr_argsize_check()
nfssvc_decode_readdirargs	nfsd_proc_readdir	struct nfsd_readdirargs	decode_fh() ntohl() xdr_argsize_check()

表 3.1.3-4　NFS サーバの RPC プロシージャに関連付けられた
XDR エンコーダの概要

XDRエンコーダ	適用される RPCプロシージャ	関連する構造体	主要な処理
nfssvc_encode_void	nfsd_proc_none nfsd_proc_remove nfsd_proc_rename nfsd_proc_link nfsd_proc_symlink nfsd_proc_rmdir	なし	xdr_ressize_check()
nfssvc_encode_attrstat	nfsd_proc_getattr nfsd_proc_setattr nfsd_proc_write	struct nfsd_attrstat	encode_fattr() xdr_ressize_check()
nfssvc_encode_diropres	nfsd_proc_lookup nfsd_proc_create nfsd_proc_mkdir	struct nfsd_diropres	encode_fh() encode_fattr() xdr_ressize_check()
nfssvc_encode_readlinkres	nfsd_proc_readlink	struct nfsd_readlinkres	htonl() xdr_ressize_check()
nfssvc_encode_readres	nfsd_proc_read	struct nfsd_readres	encode_fattr() htonl() xdr_ressize_check()
nfssvc_encode_readdirres	nfsd_proc_readdir	struct nfsd_readdirres	xdr_ressize_check()
nfssvc_encode_statfsres	nfsd_proc_statfs	struct nfsd_statfsres	htonl() xdr_ressize_check()

3.1.4 NFS

NFS クライアントが NFS ファイルシステムへアクセスする場合、大まかに次の 2 段階の手順が踏まれる。

(1) ステップ１

与えられたパス名からそのファイルのファイルハンドル及び属性を取得する。

① パス名をチェックし、絶対パスでなければ、絶対パスになるように補完する。相対パス（/で始まらないパス名）は、ファイルにアクセスするプロセスのカレントディレクトリを探し当てて補完に使用する。

② パス名をたどって、ファイルハンドルと属性を取得する。

- キャッシュ（VFS:dentry 構造体及び NFS:nfs_dentry 構造体配列を使って管理する）の中から検索し、キャッシュにない場合または期限が切れたパスの場合、Lookup の RPC コールを発行して、NFS サーバから取得する。
- 与えられたパス名がシンボリックリンクを含んだ場合、readlink の RPC コールを発行して、NFS サーバからリンク元のパス名を取得して、再び②の検索を行なう。

③ 得られたファイルハンドルと属性を用いて、与えられたパス名用の inode が作成される。inode 構造体は表 3.1.4-1 を参照のこと。

- ディレクトリキャッシュが更新される。
- ファイルハンドルはディレクトリキャッシュエントリの d_fsdata フィールドに格納される。（表 3.1.1-6 参照）
- 属性によって、inode 構造体の i_op フィールドにディレクトリ、ファイル、シンボリックリンク操作用構造体へのポインタがセットされる。

(2) ステップ２

取得した inode ポインタにセットされた i_op とディレクトリキャッシュエントリに格納されているファイルハンドルを用いて、NFS クライアントがサポート操作を行なう。

NFS ファイルシステムがサポートするディレクトリ操作は表 3.1.4-5 に示すように、以下の４つがある。

nfs_dir_read	エラーを返す空エントリ
nfs_readdir	ディレクトリの読み出し、NFS のディレクトリキャッシュ使用
nfs_open	ディレクトリオープン、何もしない
nfs_release	ディレクトリ解放、何もしない

NFS のディレクトリキャッシュの使用目的は、RPC コールの通信量を抑えるためで

あり、次の構造体の配列（サイズ：NFS_MAX_DIRCACHE=16）によって管理される。

```
struct nfs_dirent {
        dev_t           dev;            /* device number */
        ino_t           ino;            /* inode number */
        u32             cookie;         /* cookie of first entry */
        unsigned short  valid : 1,      /* data is valid */
                        locked : 1;     /* entry locked */
        unsigned int    size;           /* # of entries */
        unsigned long   age;            /* last used */
        unsigned long   mtime;          /* last attr stamp */
        struct wait_queue * wait;
        __u32 *         entry;          /* three __u32's per entry */
};
```

NFS のディレクトリを読み出しする際、まずディレクトリキャッシュの中から該当エントリがあるかを検索する。その結果により、以下の場合分けがされる。

(1) エントリが見つからない場合

① *readdir* RPC を発行して、NFS サーバーからディレクトリの中身を取得する。

② 使用されていないまたは最も古いキャッシュエントリに入れる。

(2) エントリが見つかった場合

① エントリの有効性をチェックする。（使用するタイムスタンプ：age, mtime, 及び inode->i_mtime, inode->u.nfs_i.attritimeo）

② 無効になったエントリがあった場合、*readdir* RPC を発行して、NFS サーバからディレクトリの中身を取得し、更新する。

Linux の VFS に提供する NFS のディレクトリキャッシュ操作手続きは、dentry_operations 構造体（表 3.1.1-6 参照）として以下の３つが定義されている。

nfs_lookup_revalidate()	キャッシュの有効性チェック及び更新
nfs_dentry_delete()	使わなくなったキャッシュエントリの削除（dput( )使用）
nfs_dentry_release()	キャッシュエントリのプライベートメモリの解放

ファイルへの読み書きは Linux の一般的な手続き generic_file_read() と generic_file_write()を使っている。

上記２つの手続きで使用するファイルシステム依存の操作は nfs_file_operations(表 3.1.4-5 参照）と nfs_file_inode_operations（表 3.1.4-4 参照）で定義している。参考までに NFS クライアント関連のソースファイルの一覧を表 3.1.4-6 に示しておく。

表 3.1.4-1　inode 構造体

構造体	備考
include/linux/fs.h	fs/nfs/inode.c after lookup every filed be setted
struct inode {	
struct list_head i_hash;	
struct list_head i_list;	
struct list_head i_dentry;	
unsigned long i_ino;	inode 番号
unsigned int i_count;	参照カウント
kdev_t i_dev;	ファイルデバイス番号
umode_t i_mode;	ファイルタイプとアクセス権限
nlink_t i_nlink;	ハードリンク数
uid_t i_uid;	所有者
gid_t i_gid;	所有者
kdev_t i_rdev;	デバイス（デバイスファイルの場合）
off_t i_size;	サイズ
time_t i_atime;	最終アクセス時刻
time_t i_mtime;	最終変更時刻
time_t i_ctime;	作成時刻
unsigned long i_blksize;	ブロックサイズ
unsigned long i_blocks;	ブロック数
unsigned long i_version;	dchache バージョン管理
unsigned long i_nrpages;	
struct semaphore i_sem;	アクセス制御
struct semaphore i_atomic_write;	
struct inode_operations *i_op;	inode 操作(表 3.1.4-3)
struct super_block *i_sb;	スーパーブロック(表 3.1.1-3)
struct wait_queue *i_wait;	wait キュー
struct file_lock *i_flock;	ファイルロック
struct vm_area_struct *i_mmap;	メモリ領域
struct page *i_pages;	
struct dquot *i_dquot[MAXQUOTAS];	
unsigned long i_state;	
unsigned int i_flags;	フラグ(i_sb->s_flags)
unsigned char i_pipe;	inode がパイプを表現する
unsigned char i_sock;	inode がソケットを表現する
int i_writecount;	書き込み許可された数
unsigned int i_attr_flags;	
__u32 i_generation;	
union {	ファイルシステム固有の情報
・・・	
struct nfs_inode_info nfs_i;	NFS inode 情報(表 3.1.4-2)
・・・	
} u;	
};	

表 3.1.4-2　NFS inode 情報構造体

構造体	備考

include/linux/fs.h
 	
struct nfs_inode_info {
　struct pipe_inode_info　pipeinfo;
 	
　/* Various flags */
　unsigned short　　flags;
 	
　unsigned long　　read_cache_jiffies;
　unsigned long　　read_cache_mtime;
　unsigned long　　attrtimeo;
 	
　struct nfs_wreq *　　writeback;
};
	fs/nfs/inode.c
	after lookup every filed be setted
	
	NFS ファイルシステム上でパイプを使う時に使用する。データは inod->u.pipe_i によってアクセスされるので、このフィールドは最初になくてはならない。
	
	inode がキャッシュされた時間
	inode に変更があった時間
	設定したキャッシュ情報が無効になる時間
	キャッシュ更新条件
	　jiffies - read_cache_jiffies > attrtimeo
	キャッシュ無効になり、廃棄される条件
	　mtime != read_cache_mtime
	　　↑ inode のメンバーフィールド
	
	NFSv3 用書込みキュー

表 3.1.4-3　inode 操作構造体

構造体	備考

include/linux/fs.h
 	
struct inode_operations {
　　struct file_operations * default_file_ops;
　　int (*create) (struct inode *,struct dentry *,int);
　　struct dentry * (*lookup) (struct inode *,struct dentry *);
　　int (*link) (struct dentry *,struct inode *,struct dentry *);
　　int (*unlink) (struct inode *,struct dentry *);
　　int (*symlink) (struct inode *,struct dentry *,const char *);
　　int (*mkdir) (struct inode *,struct dentry *,int);
　　int (*rmdir) (struct inode *,struct dentry *);
　　int (*mknod) (struct inode *,struct dentry *,int,int);
　　int (*rename) (struct inode *,struct dentry *,struct inode *,
　　　　struct dentry *);
　　int (*readlink) (struct dentry *, char *,int);
　　struct dentry * (*follow_link) (struct dentry *, struct dentry *,
　　　　unsigned int);
　　int (*readpage) (struct file *, struct page *);
　　int (*writepage) (struct file *, struct page *);
　　int (*bmap) (struct inode *,int);
　　void (*truncate) (struct inode *);
　　int (*permission) (struct inode *, int);
　　int (*smap) (struct inode *,int);
　　int (*updatepage) (struct file *,struct page *, unsigned long,
　　　　unsigned int, int);
　　int (*revalidate) (struct dentry *);
};
	NFS における inode 操作の詳細は表 3.1.4-4 を参照のこと。

表 3.1.4-4　NFS における inode 操作構造体

ディレクトリ inode 操作定義	ファイル inode 操作定義	ｼﾝﾎﾞﾘｯｸﾘﾝｸ inode 操作定義
fs/nfs/dir.c	**fs/nfs/file.c**	**fs/nfs/symlink.c**

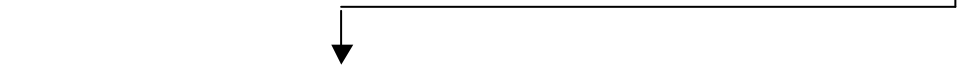
nfs_dir_inode_operations={
&nfs_dir_operations,
nfs_create,　↑(表 3.1.4-5)
nfs_lookup,
nfs_link,
nfs_unlink,
nfs_symlink,
nfs_mkdir,
nfs_rmdir,
nfs_mknod,
 	
nfs_rename,
NULL,
 	
NULL,
NULL,
NULL,
NULL,
NULL,
NULL,
NULL,
NULL,
 	
nfs_revalidate,
};
	nfs_file_inode_operations = {
	&nfs_file_operations,
	(表 3.1.4-5)
	NULL, /* create */
	NULL, /* lookup */
	NULL, /* link */
	NULL, /* unlink */
	NULL, /* symlink */
	NULL, /* mkdir */
	NULL, /* rmdir */
	NULL, /* mknod */
	
	NULL, /* rename */
	NULL, /* readlink */
	
	NULL, /* follow_link*/
	nfs_readpage, /* readpage */
	nfs_writepage,/* writepage */
	NULL, /* bmap */
	NULL, /* truncate */
	NULL, /* permission*/
	NULL, /* smap */
	nfs_updatepage,/* updatepage*/
	
	nfs_revalidate,/* revalidate*/
	};
		nfs_symlink_inode_operations={
		NULL,
		NULL,
		NULL,
		NULL,
		NULL,
		NULL,
		NULL,
		NULL,
		NULL,
		
		NULL,
		nfs_readlink,
		
		nfs_follow_link,
		NULL,
		NULL,
		NULL,
		NULL,
		};

表 3.1.4-5　ファイル操作構造体

構造体

include/linux/fs.h

```
struct file_operations {
   loff_t (*llseek) (struct file *, loff_t, int);
   ssize_t (*read) (struct file *, char *, size_t, loff_t *);
   ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
   int (*readdir) (struct file *, void *, filldir_t);
   unsigned int (*poll) (struct file *, struct poll_table_struct *);
   int (*ioctl) (struct inode *, struct file *, unsigned int,
                                                unsigned long);
   int (*mmap) (struct file *, struct vm_area_struct *);
   int (*open) (struct inode *, struct file *);
   int (*flush) (struct file *);
   int (*release) (struct inode *, struct file *);
   int (*fsync) (struct file *, struct dentry *);
   int (*fasync) (int, struct file *, int);
   int (*check_media_change) (kdev_t dev);
   int (*revalidate) (kdev_t dev);
   int (*lock) (struct file *, int, struct file_lock *);
};
```



ディレクトリ操作定義	ファイル操作定義

fs/nfs/dir.c

```
nfs_dir_operations = {
   NULL,
   nfs_dir_read,
   NULL,
   nfs_readdir,
   NULL,

   NULL,
   NULL,
   nfs_open,
   NULL,
   nfs_release,
   NULL
};
```

fs/nfs/file.c

```
nfs_file_operations = {
   NULL,              /* lseek - default */
   nfs_file_read,     /* read */
   nfs_file_write,    /* write */
   NULL,              /* readdir - bad */
   NULL,              /* select - default */

   NULL,              /* ioctl - default */
   nfs_file_mmap,     /* mmap */
   nfs_open,          /* open */
   nfs_file_flush,    /* flush */
   nfs_release,       /* release */
   nfs_fsync,         /* fsync */
   NULL,              /* fasync */
   NULL,              /* check_media_change */
   NULL,              /* revalidate */
   nfs_lock,          /* lock */
};
```

表 3.1.4-6　NFS クライアント関連ソースファイル一覧

ファイル名	実装内容	備　考
dir.c	ディレクトリ操作関連手続き	自前ディレクトリキャッシュ使用
file.c	ファイル操作関連手続き	
inode.c	inode(ファイル、ディレクトリ)関連及び NFS ファイルシステム登録等の手続き	
mount_clnt.c	クライアントのマウント関連手続き。	nfsroot.c 中の手続きが使用
nfs2xdr.c	NFS Ver.2 がサポートする RPC/XDR 手続き	
nfs3xdr.c	NFS Ver. 3 がサポートする RPC/XDR 手続き	RPC は V2 と同じ
nfsroot.c	NFS をルートファイルシステムとして使用する時、NFS ルートサポート関連手続き	カーネル初期化の為に用意した手続き
proc.c	RPC プログラム関連ヘルパー手続き	
read.c	NFS 読出し手続き	file.c 中の手続きが使用 (through generic_file_read)
write.c	NFS 書出し手続き	file.c 中の手続きが使用 (through generic_file_write)

3.1.5 VFS

NFS クライアントは、あたかもローカルファイルシステムにアクセスするように NFS サーバの共有ファイルシステムにアクセスできる。また、NFS サーバが異なるアーキテクチャを使用していたり、ファイルシステム構造が異なる全く別のオペレーティングシステムで動作している場合でも、NFS は、リモートファイルシステムの実際の構造を隠蔽し、ローカルファイルシステムと同じ構造に見えるようにしている。

前者（ネットワーク上での位置を隠蔽）は、Virtual File System（VFS、仮想ファイルシステム）を利用することで実現している。NFS では、VFS 上でのファイルシステム操作（VFS オペレーション）を一貫性のあるインターフェースを利用して定義することにより、実装上の相違点を隠すことができる。

後者（ファイルシステムの実際の構造を隠蔽）は、v-node（仮想ノード）のインターフェース定義によって実現される。v-node は UNIX のファイルシステムの i-node 構造に似ているが、その下の物理的なファイルシステム構造を隠している。

NFS クライアントプロセスから実行されるシステムコールは、クライアント側の VFS により v-node オペレーション（ファイルやディレクトリに対する操作）に変換される。サーバ側では、この v-node オペレーションが対応するファイルシステムへの操作に変換され、実行される。

ファイルシステムを管理する VFS は、ファイルシステムに特有の関数を呼び出し、各種の実装を通じて構造体の内容を埋める。これらの関数はファイルシステムの実装によって提供され、以下のシステムコールを使って VFS に知らされる。

```
int register_filesystem(struct file_system_type *);
```

プロトタイプは source-root/include/linux/fs.h で定義されている。該当ファイルシステムの初期化は source-root/fs/filesystems.c の中で以下のように call される。

```
int init_file-system-name_fs(void)
{
        return  register_filesystem(file-system-name_fs_type);
}
```

file-system-name: 該当するファイルシステム名に置き換える。
NRFS 場合は nrfs とすれば良い。

ファイルシステムタイプ構造体は source-root/include/linux/fs.h で次のように定義されている。

```
struct file_system_type {
        const char *name;
        int fs_flags;
        struct super_block *(*read_super) (struct super_block *, void *, int);
        struct file_system_type * next;
}
```

NFS のファイルシステムタイプ変数は次のように宣言されている

```
static struct file_system_type nfs_fs_type = {
        "nfs",
        0               /* FS_NO_DCACHE - this doesn't work right now*/,
        nfs_read_super,
        NULL
}
```

これによって、VFS にはファイルシステムの名前"nfs"、 フラグ（ここでは 0)、実装用の関数が与えられる。関数 nfs_read_super はマウントのインターフェースになる。

VFS 関連のソースファイルの一覧を表 3.1.5-1 に示す。

表 3.1.5-1　VFS 関連ソースファイル一覧

ファイル名	実装内容	備　考
attr.c	i ノード属性の設定手続き	
bad_inode.c	I ノード操作時のエラー処理手続き	
Binfmt_aout.c	旧 a.out フォーマットバイナリの識別・実行	
Binfmt_elf.c	ELF フォーマットバイナリの識別・実行	
Binfmt_em86.c	EM86 フォーマットバイナリの識別・実行	
Binfmt_java.c	JAVA フォーマットバイナリの識別・実行	
Binfmt_misc.c	バイナリフォーマットの検出、実行	将来的には binfmt_java, binfmt_em86 等に代わるもの。
Binfmt_script.c	スクリプトファイルの識別・実行	
block_dev.c	ブロックデバイス用、一般的な操作手続き	Read(), write(), fsync 等
Buffer.c	ブロックデバイス用キャッシュバッファ	
Dcache.c	ディレクトリ名検索用キャッシュ	
Devices.c	ファイルシステム含むデバイスの操作手続き	
dquot.c	クォータサポートうする手続き	
exec.c	バイナリ操作サポート	
fcntl.c	fcntl.c 操作手続き	
fifo.c	FIFO 操作手続き	
file.c	files_struct 用動的 fd ファイル識別子配列	
file_table.c	使用中ファイル管理用テーブル	
filesystems.c	カーネル組み込みファイルシステム初期化	
inode.c	使用中 i ノード管理テーブル	
ioctl.c	ioctl 操作用ヘルパー手続き（ioctl コールの受け渡し）	
locks.c	ロック関連操作手続き	
namei.c	ファイル名関連のシステムコール	
noquot.c	クォータサポートなし時の処理手続き	
open.c	open()に関連するシステムコール	
pipe.c	pipe 操作用手続き	

表 3.1.5-1　VFS 関連ソースファイル一覧（続き）

ファイル名	実装内容	備　考
read_write.c	VFS 用読み書き関連手続き	
readdir.c	ディレクトリ読み出し用手続き	
select.c	Select システムコール手続き	
stat.c	ファイルシステム状態取得用手続き	
super.c	ファイルシステムの登録、マウント/アンマウンドサービス用スーパーブロック操作手続き	do_mount(), do_umount() read_super() struct super block

3.2 NRFS 基本部の設計

VFS 並びに NFS 実装の詳細を解析した結果、NRFS サーバ側では、NFS サーバをそのまま流用して基本部を構成し、障害検出用にチェックサム機能を付加する程度の拡張で実現可能であることが判明した。

NRFS クライアント側では、サーバ多重化への対応が必要となるが、VFS 層並びにライブラリは変更せず、mount 操作の拡張並びにサーバ情報に係るデータ構造の拡張により対応することとした。

以下に、NRFS 基本部の設計についてまとめる。

3.2.1 NRFS のデータ構造

(1) NRFS ファイルシステムの定義

VFS に対して NRFS ファイルシステムを登録する際に必要となるファイルシステムタイプを表 3.2.1-1 のように定義する。

これにより、VFS にはファイルシステム名"nrfs"、マウントのインターフェースとして nrfs_read_super( )関数が与えられる。

(2) NRFS スーパーブロック情報の定義

NRFS クライアントが保持するスーパーブロック用のサーバ情報を表 3.2.1-2 のように定義する。対象とする NRFS サーバ台数分、この構造体を配列で定義する（ここでは#define 文でサーバを 3 台に設定している）。

スーパーブロック構造体の中で定義される NRFS ファイルシステム固有の情報については表 3.2.1-3 のように定義する。

(3) NRFS inode 情報の定義

NFS の struct nfs_inode_info の定義を流用して、NRFS サーバから取得した inode 番号を保持できるフィールドを追加して NRFS の inode 情報構造体を定義した。当該構造体を表 3.2.1-4 に示す。

(4) ファイルシステム固有情報の定義

ファイルシステム固有データとして、表 3.2.1-5 の構造体を定義する。

これはディレクトリの操作が成功する度に、VFS のディレクトリエントリ構造体（include/linux/dcache.h/struct dentry）の d_fsdata フィールドにセットされる。

(5) ファイル/ディレクトリ属性情報の定義

NRFS ファイルシステムにおけるファイル及びディレクトリの属性情報を表

3.2.1-6 のように定義する。

(6) データアクセス要求用構造体の定義

データ読み出し要求用構造体並びにデータ書き込み要求用構造体の定義を表 3.2.1-7、8 にそれぞれ示す。

(7) ディレクトリキャッシュエントリ構造体の定義

基本的に NFS と同じ構造体を流用するが、inode 番号（ino）のみを多重化する。

表 3.2.1-1　NRFS ファイルシステム定義

構造体定義	NRFS ファイルシステム

include/linux/fs.h
struct file_system_type {
const char *name;
int fs_flags;
struct super_block *(*read_super) (struct super_block *, void *, int);
struct file_system_type * next;
};
	fs/nrfs/inode.c
	struct file_system_type nrfs_fs_type = {
	"nrfs",
	0,
	nrfs_read_super,
	NULL
	}

表 3.2.1-2　NRFS サーバ情報構造体定義

構造体	備考

include/linux/nrfs_fs_sb.h
#defined NRFS_MAXSERVERS 3
 	
struct nrfs_server {
struct rpc_clnt * client;
char * hostname; /* remote hostname */
char * dirname; /* server exported path */
unsigned int bsize; /* server block size */
};
	
	RPC クライアントハンドル
	サーバーのホスト名
	マウント元のパス名（エラー訂正時使用）
	サーバーブロックサイズ

表 3.2.1-3　NRFS スーパーブロック情報構造体定義

構造体	備考

include/linux/nrfs_fs_sb.h
struct nrfs_sb_info {
struct nrfs_server s_server[NRFS_SUPER_MAX];
struct nrfs_fh s_root[NRFS_SUPER_MAX];
int s_stat[NRFS_SUPER_MAX];
int s_cnt;
int flags;
int rsize;
int wsize;
unsigned int acregmin;
unsigned int acregmax;
unsigned int acdirmin;
unsigned int acdirmax;
 	
};
	
	表 3.2.1-2 参照
	NFS と同じの構造体を使用して多重化
	サーバーの状態
	ﾏｳﾝﾄした NRFS ｻｰﾊﾞｰ数
	マウンドオプションの保存用
	ブロック読込みサイズ
	ブロック書込みサイズ
	属性キャッシュのタイムアウト

表 3.2.1-4　NRFS inode 情報構造体定義

構造体

```
include/linux/nrfs_fs_i.h

struct nrfs_inode_info {
  /*
   * This is a place holder so named pipes on NFS filesystems
   * work (more or less correctly). This must be first in the
   * struct because the data is really accessed via inode->u.pipe_i.
   */
  struct pipe_inode_info  pipeinfo;

  /*
   * Various flags
   */
  unsigned short          flags;

  /*
   * read_cache_jiffies is when we started read-caching this inode,
   * and read_cache_mtime is the mtime of the inode at that time.
   * attrtimeo is for how long the cached information is assumed
   * to be valid. A successful attribute revalidation doubles
   * attrtimeo (up to acregmax/acdirmax), a failure resets it to
   * acregmin/acdirmin.
   *
   * We need to revalidate the cached attrs for this inode if
   *
   *      jiffies - read_cache_jiffies > attrtimeo
   *
   * and invalidate any cached data/flush out any dirty pages if
   * we find that
   *
   *      mtime != read_cache_mtime
   */
   unsigned long     read_cache_jiffies;
   unsigned long     read_cache_mtime;
   unsigned long     attrtimeo;
   ino_t               ino[NRFS_MAXSERVERS]
  /*
   * This is the list of dirty unwritten pages.
   * NFSv3 will want to add a list for written but uncommitted
   * pages.
   */
   struct nrfs_wreq *       writeback;

   /* for nrfs repair protocol */
    int repair_count[NRFS_MAXSERVERS];
};
```

構造体	備考
`ino_t ino[NRFS_MAXSERVERS]`	**サーバー情報多重化ための追加** NRFS_MAXSERVERS は include/linux/nrfs.h の中で定義
`int repair_count[NRFS_MAXSERVERS];`	**エラー回数**

表 3.2.1-5　NRFS ファイルシステム固有情報構造体定義

構造体	備考

include/linux/nrfs.h
 	
struct nrfs_fsdata {
　struct nrfs_fh　　fhs[NRFS_MAXSERVERS];
};	

表 3.2.1-6　ファイル／ディレクトリ属性情報の定義

構造体	備考

include/linux/nrfs.h
 	
struct nrfs_fattr {
　enum nrfs_ftype type;
　__u32　　mode;
　__u32　　nlink;
　__u32　　uid;
　__u32　　gid;
　__u32　　size;
　__u32　　blocksize;
　__u32　　rdev;
　__u32　　blocks;
　__u32　　fsid;
　__u32　　fileid;
　struct nrfs_timeatime;
　struct nrfs_timemtime;
　struct nrfs_timectime;
　unsigned int　　rvstat; /* to determine v_ino　*/
　　__u32　v_ino;　/* pass to VFS's iget() */
};
	NFS と同じ構造体を流用する
	
	最もレスポンスの早いサーバの ID
	サーバ ID で拡張した fileid、仮想ディレクトリ／ファイルの inode 番号として使用する

表 3.2.1-7　ファイルシステム固有情報構造体定義（読み出し用）

構造体	備考

fs/nrfs/read.c
 	
struct nrfs_rreqmgn {
  struct dentry * mr_dentry; /* inode from which to read */
  struct page * mr_page; /* page to be read */
  struct nrfs_fattr mr_fattr; /* for nrfs_refresh_inode */
  void * mr_buf[NRFS_MAXSERVERS];
  int mr_stat[NRFS_MAXSERVERS];
  int mr_offset; /* read offset for re-read */
  int mr_rsize; /* read size for re-read */
  int mr_wsize; /* write size for recover */
  int mr_rqid; /* read request ID */
  int mr_scnt; /* number of server alive */
  int mr_txcnt; /* sent read request counter */
  int mr_rvchk; /* sent read request counter */
  int mr_idxdat; /* server index that called for data */
  int mr_idxok; /* server index that reply good data */
  int mr_idxng; /* server index that reply bad data */
  int mr_lock; /* lock flag for update page data */
#ifdef NRFS_CHECKSUM_USE
  u32 mr_sumdat[NRFS_MAXSERVERS];
#endif
}
 	
struct nrfs_rreq {
#ifdef NRFS_CHECKSUM_USE
  struct nrfs_csumreadargs ra_args; /* XDR argument struct */
  struct nrfs_csumreadres ra_res; /* … and result struct */
#else
  struct nrfs_readargs ra_args; /* XDR argument struct */
  struct nrfs_readres ra_res; /* … and result struct */
#endif
  struct nrfs_fattr ra_fattr; /* fattr storage */
  struct nrfs_rreqmgn * ra_rqmgn;
  u32 ra_proc; /* rpc program number */
  int ra_idx; /* point to which buf is used */
	1 回の読み出しリクエストに対し複数読み出しコール管理用
	
	読み出しコール管理用

表 3.2.1-8　ファイルシステム固有情報構造体定義（書き込み用）

構造体	備考

include/linux/nrfs_fs.h
 	
struct nrfs_wreqmgn {
　struct file *　　mw_file;　　/* dentry referenced */
　struct page *　　mw_page;　　/* page to be written */
　unsigned int　　mw_offset;　/* offset within page */
　unsigned int　　mw_bytes;　　/* dirty range */
　pid_t　　mw_pid;　　/* owner process */
　int　　mw_rqid;
　int　　mw_scnt;
　int　　mw_rcnt;
　int　　mw_stat[NRFS_MAXSERVERS];
　struct nrfs_wreq * mw_wreq[NRFS_MAXSERVERS];
　struct nrfs_fattr　mw_fattr[NRFS_MAXSERVERS];
　unsigned short　　mw_flags;
}
 	
struct nrfs_wreq {
　struct rpc_listitem　　wb_list;　/* linked list of req's */
　struct rpc_task　　wb_task;　/* RPC task */
　struct wait_queue *　　wb_wait;　/* wait for completion */
　unsigned int　　wb_count; /* user count */
　unsigned short　　wb_flags; /* status flags */
　struct nrfs_writeargs　　wb_args;　/* NRFS RPC stuff */
　struct nrfs_fattr　　wb_fattr; /* file attributers */
　struct nrfs_wreqmgn *　　wb_reqmgn;
　int　　wb_idx;
}
	1 回の書き込みリクエストに対し複数書き込みコール管理用
	
	書き込みコール管理用

表 3.2.1-9　ディレクトリキャッシュエントリ構造体定義

構造体	備考

include/linux/nrfs.h
 	
struct nrfs_dirent {
　dev_t　　dev;　　/* device number */
　ino_t　　ino[NRFS_MAXSERVERS]; /* inode number */
　u32　　cookie;　　/* cookie of first entry */
　unsigned short　　valid　: 1, /* data is valid */
　　locked : 1; /* entry locked */
　unsigned int　　size;　　/* # of entries */
　unsigned long　　age;　　/* last used */
　unsigned long　　mtime;　　/* last attr stamp */
　struct wait_queue *　wait;
　__u32 *　　entry;　　/* three __u32's per entry */
};
	inode 番号の多重化

3.2.2 サーバ多重マウント方式

今期の開発環境である Red Hat Linux 6.2J Second Edition が使用している mount-2.10f-1.src.rpm パッケージをベースに NRFS ファイルシステムのマウントが使えるようにする。mount-2.10f-1.src.rpm パッケージの中身は下記の通りである。

① util-linux-2.10f.tar.bz2
② util-linux-2.10f-realpath.patch
③ util-linux-2.9o-mount-nfsv3.patch
④ util-linux-2.9w-mount-nfsv3try.patch

NRFS 用に変更するポイントとして、次の点をまず実装する。

(1) NFS のマウント手続きをそのまま利用する
(2) コマンドライン書式を NFS と区別するため、ダブルコロンを使用する。

＜例＞

```
# mount  server::export_dir  mount_point
```

(3) 同じマウントポイントに対して異なるサーバのマウントの実行を許可する。
(4) 同じサーバに対して、多重マウントのチェックを追加し、禁止する。

NRFS 用に変更するために編集するファイル(util-linux-2.10f/mount の下にある)を表 3.2.2-1 に示す。

また、1つのマウントポイントに複数の NRFS サーバの共有ファイルシステムをマウント出来るように、Linux カーネル内のファイルシステムマウントフローを表 3.2.2-2 のように変更する。

表 3.2.2-1　マウントコマンド変更ファイル一覧

ファイル名	説明（変更点）	備考
nrfsmount.c	① 引数処理の変更 ② nfsmount 実装コードとの競合の回避	追加
mount.c	① コマンドラインオプション処理変更 ② nrfsmount エントリ追加	既存ファイル編集
sundries.h	nrfsmount 手続きのプロトタイプ定義エントリ追加	既存ファイル編集
Makefile	nrfsmount.c エントリ追加	既存ファイル編集

表 3.2.2-2　カーネル内のマウントフロー及び NRFS 用の変更点

通常のマウントフロー	備考

sys_mount(dev_name, dir_name, type, new_flag, mount_data) {
　if リマウント {
　　マウントオプションをカーネル空間にコピー(to page);
　　do_remount(dir_name, flags, page);
　　後始末処理、do_remount から貰った結果をマウントコマンドに返す；
　}
 	
　マウントオプションをカーネル空間にコピー(to page);
　サポートしているファイルシステムであるかをチェック;
　ローカルであれば、物理デバイス関連チェック;
　未使用のデバイス番号を取得（to dev）;
 	
　do_mount(dev, dev_name, dir_name, flag, page) {
　　マウント先がディレクトリかをチェック、マウント先の dentry を取得(ERROR: -ENOTDIR);
　　ERROR = -EBUSY;
　　if マウントポイントがマウント済み {
　　　if（マウント NRFS）{
　　　取得したデバイス番号を返す；
　　　ERROR = do_mount_nrfs_n(dentry, fstype->name, flag, page){
　　　同一サーバが既にマウントしているかをチェックする
　　　dentry よりスーパーブロック(sb)を取得する
　　　dentry のファイルシステムタイプは NRFS かをチェックする
　　　shrink_dcache_sb(sb);
　　　fsync_dev(sb->dev);
　　　fstype->read_super();
　　　}
　　}
　　後始末処理、ERROR を sys_mount に返す
　　}
　　read_super(dev, type, flags, data, 0) {（スーパーブロック取得、ERROR: -EINVAL);
　　デバイス番号のチェック；
　　if 指定したデバイス番号はマウント済み　{
　　そのデバイス番号にマウントしているスーパーブロックを do_mount()に返す；
　　}
　　ファイルシステムタイプ名（type）から指定するファイルシステムを探し出す；
　　新しいスーパーブロック用メモリ確保；
　　指定するファイルシステムの read_super()手続きを呼び出す；
　　指定するファイルシステムのスーパーブロックを do_mount()に返す
　　}
　　スーパーブロックが指定したデバイス番号にマウントできるかをチェック（ERROR: -EBUSY）
　　VFS のマウント済みリストに追加する（ERROR: －ENOMEM）；
　　マウントポイント（dentry）に取得したスーパーブロックを関連付ける;
　　後始末処理、マウンド結果を sys_mount()に返す
　}
　後始末処理、do_mount から貰った結果をマウントコマンドに返す；
}
	1)1 回目の NRFS マウントは従来通り(NFS)のマウントフローで行う。
	
	2) 2 回目以降の NRFS マウントは追加した分岐に入り、
	①マウント済みのマウントポイントのスーパーブロックに新しい NRFS サーバ情報を追加する
	②マウント済みのマウントポイントの inode に新しい NRFS サーバに関する情報（inode 番号等）を追加する
	
	変更点

3.2.3　KRPC プロトコルの拡張

NRFS では、サーバを多重化し、各サーバ上のファイルシステムにおいてデータを同一の構成に保持することで、高信頼性の原点となる冗長性を確保している。各サーバ上に保持されたデータは、サーバやディスクに障害がない限り同等のものとなるよう取り扱われる。

さらに NRFS では、万一の障害に備え、その検出機能並びに障害訂正機能を整備することとした。具体的には、NRFS は３台以上のサーバにより構成されることとし、データ利用時にはこれらのサーバが保持するデータの一致性を当該データのチェックサム情報により確認することとし、不整合がある場合は多数決により正しいデータを判定するとともに不整合を訂正する機能を整備することとした。

なお、ファイル読み込み時に、多重化された全てのサーバから保持するデータを取得するとした場合、各サーバとクライアント間で同等のデータが通信されることとなり、システム全体でのネットワーク通信負荷が無用に増大することとなる。正常なデータであれば１系統のみの通信で良く、またデータの認証についてはチェックサム情報のみの通信で対応可能である。したがって、実際にデータを送出（チェックサム情報も付加）するのは任意のサーバ１台のみとし、他のサーバはチェックサム情報のみを返すことで、通信負荷の軽減を図ることとした。

このように、NRFS では、NFS と同様の RPC プロシージャに加え、チェックサム情報を利用するためのプロシージャ並びに障害訂正機能を利用するためのプロシージャを新たに整備した。NRFS サーバに実装した RPC プロシージャの一覧を表 3.2.3-1 に、その概要を表 3.2.3-2 にそれぞれ示すとともに、新たに追加した NRFS プロシージャのインターフェースを表 3.2.3-3 に示す。

また、追加した NRFS プロシージャそれぞれに適用する XDR デコーダ／エンコーダを整備した。その概要を表 3.2.3-4、5 に示す。

表 3.2.3-1　NRFS サーバの RPC プロシージャ一覧

```
#define nfsd_proc_none          NULL
#define nfssvc_release_none     NULL
struct nfsd_void { int dummy; };

#define PROC(name, argt, rest, relt, cache)     \
 { (svc_procfunc) nfsd_proc_##name,             \
   (kxdrproc_t) nfssvc_decode_##argt,           \
   (kxdrproc_t) nfssvc_encode_##rest,           \
   (kxdrproc_t) nfssvc_release_##relt,          \
   sizeof(struct nfsd_##argt),                  \
   sizeof(struct nfsd_##rest),                  \
   0,                                           \
   cache                                        \
 }
```

RPC プロシージャ	XDR デコーダ	XDR エンコーダ		

```
struct svc_procedure            nfsd_procedures2[18] = {
  PROC(null,     void,        void,         none,      RC_NOCACHE),
  PROC(getattr,  fhandle,     attrstat,     fhandle,   RC_NOCACHE),
  PROC(setattr,  sattrargs,   attrstat,     fhandle,   RC_REPLBUFF),
  PROC(none,     void,        void,         none,      RC_NOCACHE),
  PROC(lookup,   diropargs,   diropres,     fhandle,   RC_NOCACHE),
  PROC(readlink, fhandle,     readlinkres, none,       RC_NOCACHE),
  PROC(read,     readargs,    readres,      fhandle,   RC_NOCACHE),
  PROC(none,     void,        void,         none,      RC_NOCACHE),
  PROC(write,    writeargs,   attrstat,     fhandle,   RC_REPLBUFF),
  PROC(create,   createargs,  diropres,     fhandle,   RC_REPLBUFF),
  PROC(remove,   diropargs,   void,         none,      RC_REPLSTAT),
  PROC(rename,   renameargs,  void,         none,      RC_REPLSTAT),
  PROC(link,     linkargs,    void,         none,      RC_REPLSTAT),
  PROC(symlink,  symlinkargs, void,         none,      RC_REPLSTAT),
  PROC(mkdir,    createargs,  diropres,     fhandle,   RC_REPLBUFF),
  PROC(rmdir,    diropargs,   void,         none,      RC_REPLSTAT),
  PROC(readdir,  readdirargs, readdirres,  none,       RC_REPLBUFF),
  PROC(statfs,   fhandle,     statfsres,    none,      RC_NOCACHE),
  PROC(checksum,      checksumargs,     checksumres,     , ),
  PROC(checksumread,  checksumreadargs, checksumreadres, , ),
  PROC(repair,        repairargs,       repairres,       , ),
};
```

表 3.2.3-2　NRFS サーバにおける RPC プロシージャの概要（１／２）

RPCプロシージャ	概要	主に使用するVFSプロシージャ
nfsd_proc_null	実質的に何も行わないプロシージャで、NFSにおけるPINGの役割を果たす。　サーバでは、受け取った引数のRPC、NFS引数の整合性がとれているかのみをチェックしRPCレスポンスで結果を返す。	なし
nfsd_proc_getattr	ファイルハンドルで示されるファイル、ディレクトリ属性を返す。	fh_verify
nfsd_proc_setattr	ファイルハンドルで示されるファイル、ディレクトリ属性を設定する。	nfsd_setattr
nfsd_proc_none	NFSプロトコル仕様上は、nfsd_proc_rootであり、NFSで公開されているｒｏｏｔパスのファイルハンドルを返すものである。 ただし、Linux KNFSでは、当該プロシージャはマウントプロトコルで置き換えられため、実装されていない。	なし
nfsd_proc_lookup	ファイルハンドルで示されるディレクトリ下の、ファイル／ディレクトリのファイルハンドルを返す。	nfsd_lookup
nfsd_proc_readlink	ファイルハンドルで示されるリンク情報を返す。	nfsd_readlink
nfsd_proc_read	ファイルハンドルで示されるファイルのオフセット位置からレングス長のデータを読み込む。	nfsd_read
nfsd_proc_none	NFSプロトコル仕様上は、nfsd_proc_writecacheであり、NFS書き込み時に、メモリキャッシュされているデータをディスクに書き込むもので、syncの役割を果たす。 ただし、Linux KNFSでは、当該プロシージャは未実装である。	なし
nfsd_proc_write	ファイルハンドルで示されるファイルのオフセット位置からレングス長のデータを書き込む。	nfsd_write
nfsd_proc_create	ファイルハンドルで示されるディレクトリ下に、指定ファイル名、属性を持つファイルを作成する。	fh_verify nfsd_permission nfsd_lookup fh_lock_parent fh_update nfsd_create nfsd_setattr fh_unlock
nfsd_proc_remove	ファイルハンドルで示されるファイルを削除する。	nfsd_unlink
nfsd_proc_rename	ファイルハンドルで示されるディレクトリ下のファイル／ディレクトリ名を指定されたファイルハンドルで示されるディレクトリ下のファイル／ディレクトリ名に変更する。	nfsd_rename
nfsd_proc_link	２つのファイルハンドルで示されるファイル／ディレクトリを指定された名称でハードリンクする。	nfsd_link
nfsd_proc_symlink	ファイルハンドルで示されるディレクトリ下のファイル／ディレクトリを指定された名称、属性でシンボリックリンクする。	nfsd_symlink nfsd_setattr
nfsd_proc_mkdir	ファイルハンドルで示されるディレクトリ下に、指定された名称、属性でディレクトリを作成する。	nfsd_create
nfsd_proc_rmdir	ファイルハンドルで示されるディレクトリ下の指定された名称のディレクトリを削除する。	nfsd_unlink
nfsd_proc_readdir	ファイルハンドルで示されるディレクトリのディレクトリ情報を返す。	nfsd_readdir
nfsd_proc_statfs	ファイルハンドルで示されるファイル／ディレクトリのステータス情報を返す。	nfsd_statfs

表 3.2.3-2　NRFS サーバにおける RPC プロシージャの概要（２／２）

RPCプロシージャ	概要	主に使用するVFSプロシージャ
nfsd_proc_checksum	ファイルハンドルで示されるファイルのオフセット位置からレングス長のチェックサムを計算し結果を返す。	nfsd_checksum_read
nfsd_proc_checksumread	ファイルハンドルで示されるファイルのオフセット位置からレングス長のデータの読み込みとチェックサム計算を行い結果を返す。	nfsd_checksum_read
nfsd_proc_repair	ファイルハンドルで示されるファイルまたはディレクトリの復旧をリカバリデーモンに依頼する。	なし

表 3.2.3-3　NRFS プロシージャインターフェース（１／３）

NRFS checksum	
NRFS checksum は、ファイルハンドルで示されるファイルのｏｆｆｓｅｔ位置からｃｏｕｎｔバイトの間のチェックサムを計算し結果を返す。 ｓｕｍｔｙｐｅ、ｓｕｍｓｉｚｅは、将来チェックサム計算を拡張するための予約情報で、現在のバージョンでは、それぞれ、０、４の固定値を使用する。	
リクエスト番号	１８
プロシージャ	nfsd_proc_checksum
引数	u32[8] fh; ファイルハンドル u32　offset; ファイルハンドル先頭からのオフセットバイト u32　count; チェックサム計算を行うバイト数 u32　totalsize; 予約（ｃｏｕｎｔ＝＝ｃｏｕｎｔ） u32　sumtype; 計算タイプ　予約（0固定） u32　sumsize; 計算結果長　予約（４固定）
デコーダ	nfssvc_decode_checksumargs
戻り値	u32　status; 計算結果　０：正常、0以外は異常 u32　sumtype; 計算タイプ　予約（0固定） u32　sumsize; 計算結果長　予約（４固定） u32　sumdata[(sumsize+3)>>2 : == 1]; 計算結果 ファイル属性情報： u32　i_ftype; u32　i_i_mode; u32　i_i_nlink; u32　i_uid; u32　i_gid; u32　i_size; u32　i_blksize; u32　i_rdev; u32　i_blocks; u32　i_dev; u32　i_ino; u32　i_atime; u32　i_amsec(== 0); u32　i_mtime; u32　i_mmsec(== 0); u32　i_ctime; u32　i_cmsec(== 0);
エンコーダ	Nfssvc_encode_checksumres

表 3.2.3-3　NRFS プロシージャインターフェース（２／３）

NRFS checksum read	
NRFS checksum は、ファイルハンドルで示されるファイルのｏｆｆｓｅｔ位置からｃｏｕｎｔバイトの間のデータの読み込みとチェックサム計算を行い結果を返す。 ｓｕｍｔｙｐｅ、ｓｕｍｓｉｚｅは、将来チェックサム計算を拡張するための予約情報で、現在のバージョンでは、それぞれ、０、４の固定値を使用する。	
リクエスト番号	１９
プロシージャ	nfsd_proc_checksumread
引数	u32[8] fh; ファイルハンドル u32　offset; ファイルハンドル先頭からのオフセットバイト u32　count; チェックサム計算を行うバイト数 u32　totalsize; 予約（ｃｏｕｎｔ＝＝ｃｏｕｎｔ） u32　sumtype; 計算タイプ　予約（０固定） u32　sumsize; 計算結果長　予約（４固定）
デコーダ	nfssvc_decode_checksumreadargs
戻り値	u32　status; 計算結果　０：正常、０以外は異常 u32　sumtype; 計算タイプ　予約（０固定） u32　sumsize; 計算結果長　予約（４固定） u32　sumdata[(sumsize+3)>>2 : == 1]; 計算結果 ファイル属性情報： u32　i_ftype; u32　i_i_mode; u32　i_i_nlink; u32　i_uid; u32　i_gid; u32　i_size; u32　i_blksize; u32　i_rdev; u32　i_blocks; u32　i_dev; u32　i_ino; u32　i_atime; u32　i_amsec(== 0); u32　i_mtime; u32　i_mmsec(== 0); u32　i_ctime; u32　i_cmsec(== 0); u32　count;　読み込んだデータのバイト数 u32　data[[(count+3)>>2]　;読み込んだデータ
エンコーダ	Nfssvc_encode_checksumreadres

表 3.2.3-3　NRFS プロシージャインターフェース（３／３）

NRFS reparir	
NRFS reparir は、クライアントからの要求により、ファイルハンドルで示されるファイルまたはディレクトリの復旧をリカバリデーモンに依頼する。	
リクエスト番号	２０
プロシージャ	nfsd_proc_ reparir
引数	u32[8] fh; ファイルハンドル u32　e_type; 復旧モード　０：ファイル復旧／１：ディレクトリ復旧 u32　i_mode; ファイルモード属性 u32　filn; ファイル／ディレクトリ名の長さ u32　file[(filn+3)>>2];; ファイル／ディレクトリ名 復旧対象サーバの公開情報： u32　IPaddress; サーバの IP アドレス u32　expn; 公開パス名長 u32　export[(expn+3)>>2]; 公開パス名 u32　ncnt; 復旧に使用可能なサーバの公開情報数 復旧に使用可能なサーバの公開情報：[ncnt] u32　IPaddress; サーバの IP アドレス u32　expn; 公開パス名長 u32　export[(expn+3)>>2]; 公開パス名
デコーダ	nfssvc_decode_ reparir args
戻り値	u32　status; リカバリデーモンへのリクエスト結果 ０：正常、０以外は異常
エンコーダ	nfssvc_encode_ reparir res

表 3.2.3-4　NRFS サーバに新たに追加した RPC プロシージャに関連付けられた XDR デコーダの概要

XDRデコーダ	適用される RPCプロシージャ	関連する構造体	主要な処理
nfssvc_decode_checksumargs	nfsd_proc_checksum	struct nfsd_checksumargs	decode_fh() ntohl() xdr_argsize_check()
nfssvc_decode_checksumreadargs	nfsd_proc_checksumread	struct nfsd_checksumreadargs	decode_fh() ntohl() xdr_argsize_check()
nfssvc_decode_repairargs	nfsd_proc_repair	struct nfsd_repairargs	decode_fh() decode_filename() ntohl() xdr_argsize_check()

表 3.2.3-5　NRFS サーバに新たに追加した RPC プロシージャに関連付けられた XDR エンコーダの概要

XDRエンコーダ	適用される RPCプロシージャ	関連する構造体	主要な処理
nfssvc_encode_checksumres	nfsd_proc_checksum	struct nfsd_checksumres	htonl() encode_fattr() xdr_ressize_check()
nfssvc_encode_checksumreadres	nfsd_proc_checksumread	struct nfsd_checksumreadres	htonl() encode_fattr() xdr_ressize_check()
nfssvc_encode_repairres	nfsd_proc_repair	struct nfsd_repairres	xdr_ressize_check()

3.2.4 障害訂正方式

NRFS は NFS と同様に、クライアントとサーバにより構成される。ただし、NRFS では、サーバ計算機自体の障害に耐え得るよう、サーバを多重化して取り扱うとともに、障害検出訂正機能を実装することとした。

具体的には、NRFS は３台以上のサーバにより構成されることとし、データ利用時にはこれらのサーバが保持するデータの一致性を当該データのチェックサム情報により確認することとし、不整合がある場合は多数決により正しいデータを判定するとともに不整合を訂正する機能を整備することとした。

なお、ファイル読み込み時に、多重化された全てのサーバから保持するデータを取得するとした場合、各サーバとクライアント間で同等のデータが通信されることとなり、システム全体でのネットワーク通信負荷が無用に増大することとなる。正常なデータであれば１系統のみの通信で良く、またデータの認証についてはチェックサム情報のみの通信で対応可能である。したがって、実際にデータを送出（チェックサム情報も付加）するのは任意のサーバ１台のみとし、他のサーバはチェックサム情報のみを返すことで、通信負荷の軽減を図ることとした。

データ読み込みの流れに沿って、チェックサム情報の比較による障害検出並びにデータ訂正要求の発行の一例を以下に示す。

(1) データの読み込み

NRFS クライアントにおいて､NRFS ファイルシステムの管理下のデータに対し、データ読み込み要求を行う場合は、以下のような手順で処理を行うこととなる。

① 当該クライアントがマウントした全ての NRFS サーバのうち、任意のサーバ１台に対して、読み込みデータとそのチェックサム情報の転送を要求する。その他のサーバに対しては、相当するデータのチェックサム情報の転送のみを要求する。

② サーバから返されたチェックサム情報を比較する。一致した場合、取得したデータは正当なものと見做す。

③ チェックサム情報の不整合が検出された場合、多数決原理に基づき、取得したデータの正当性を確認する。取得したデータが正当と判定された場合、不整合の生じたサーバに対し、障害訂正に有用な『障害情報』とともに『訂正要求』を通知する。

④ 取得したデータに誤りがあると判定された場合、正当なデータを保持していると判定されたサーバのうちいずれか１台に対して、当該データの転送を要求する。

⑤ 正当なデータを取得したら、あらためて、不整合の生じたサーバに対し、障害訂正に有用な『障害情報』とともに『訂正要求』を通知する。

より詳細な処理ステップは、以下のようになる。

① マウントした NRFS サーバの台数分のデータ格納領域を確保する（図 3.2.4-1 参照、read_request、複数の read_call の管理も兼ねる）

② 各サーバにデータ転送を要求するが、この際、データの格納先については、①で確保した領域を指定する。(図 3.2.4-1 参照、read_call_1、read_call_2、…、read_call_n)

③ データ転送を要求した全てのサーバからの応答がそろった時点で、チェックサム情報を比較する。(将来的には、全てのサーバからの応答を待つのではなく、多数決による障害検出に必要なだけの情報がそろった時点で処理を開始するよう改良することとしたい)

④ チェックサム情報が全て一致している場合、VFS が読み込み要求する時に指定したキャッシュに読み込んだデータをコピーし、キャッシュページのロックを解除する。

⑤ チェックサム情報の不整合が検出された場合、多数決により正当なデータを保持しているサーバ並びに不整合の生じたサーバを判定する。

a. 取得したデータが正当なもの（データの取得元のサーバが正しいデータを保持している）と判定された場合、不整合の生じたサーバに対し、障害訂正に有用な『障害情報』とともに『訂正要求』を通知する。

b. 取得したデータに誤りがある（データの取得元のサーバに不整合が生じている）と判定された場合、正当なデータを保持していると判定されたサーバのうちいずれか１台に対して、当該データの転送を要求し、その応答に基づいて上記 a.の処理を実施する。

⑥ 特定のサーバに対する『訂正要求』には、その発行回数に上限（N）を、設けることとし、この上限を超えた場合、訂正要求の発行を停止するものとする。これは、マウント処理時の誤操作等により、保持する内容が本質的に異なるファイルシステム領域を同一マウントポイントにマウントしてしまった場合等に、致命的なダメージを避けるための対応である。なお、この上限 N は次式を用いてデータファイルサイズ(SIZE)より求める。

$$(SIZE >> 13) >> ((N - a) \times 3) \qquad 0$$

但し、a = 0: if SIZE < 4096, a = 1: if SIZE > 4096

⑦ ⑥に該当するサーバについては、以降の処理においてデータ転送要求の対象から除外するものとする。

参考までに、データ書き込みの流れについても以下に紹介する。

(2) データの書き込み

NRFS クライアントにおいて、NRFS ファイルシステムの管理下のデータに対し、データ書き込み要求を行う場合は、以下のような手順で処理を行うこととなる（図 3.2.4-2 参照)。

① 書き込み情報並びに書き込みコール管理情報を保持する一時的メモリ領域（write_request）を確保する。

② 全てのサーバに対して、同じ書き込み内容は①で保持している書き込み情報を利用するように書き込みコールタスク(write_call_x)を作成する。

③ 作成した書き込みコールタスクのスケジュリング：

a. RPC データ転送用スロットが確保できる場合、実行遅延（3 秒）を設定して、書き込み待ち行列（write_queue）に入れる。書き込み待ち行列の最大待ち数（WRITEBACK_MAX）は 128 である。

b. RPC データ転送用スロットが確保できない場合、タスクが同期実行を行う。

c. 待ち中のタスク数が最大待ち数の 3/4 になったら、待ち行列先頭にあるタスクが KRPC 実行の待ち行列に移され、書き込み内容を転送する実行状態に入る。

④ 下記の場合、その書き込み実行タスクは速やかに KRPC 実行の待ち行列に移され、書き込み内容を転送する実行状態に入る。

a. １つの書き込み要求内容サイズがキャッシュページサイズに達した。

b. 他のプロセスが書き込み内容に対して読み込み要求した。

⑤ 同じ書き込みに対しての書き込みタスクの実行がすべて終了した時点で、書き込み要求があるキャッシュページの更新済みフラグをセットして VFS に知らせる。

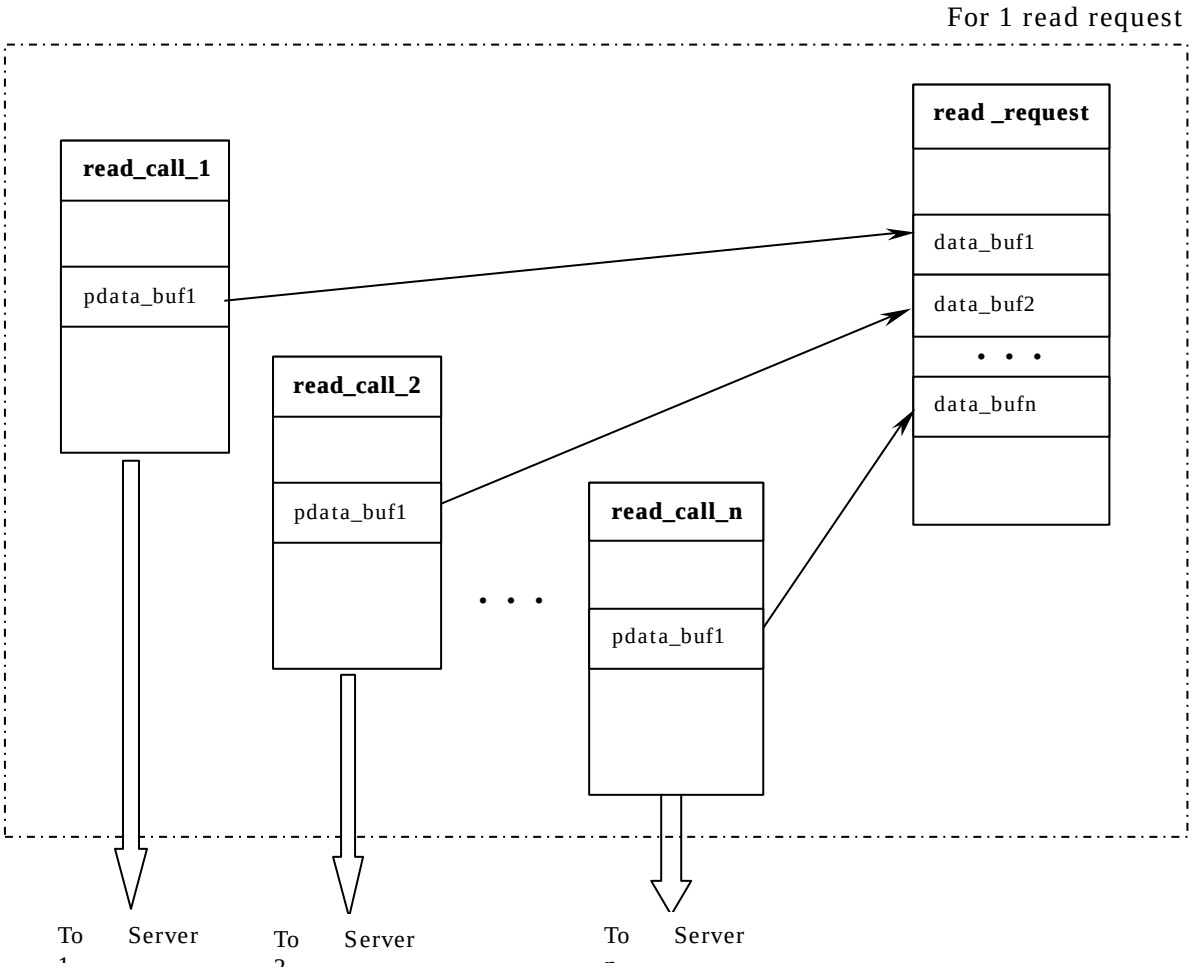


図 3.2.4-1　**読み込み処理使用データ間の関係**

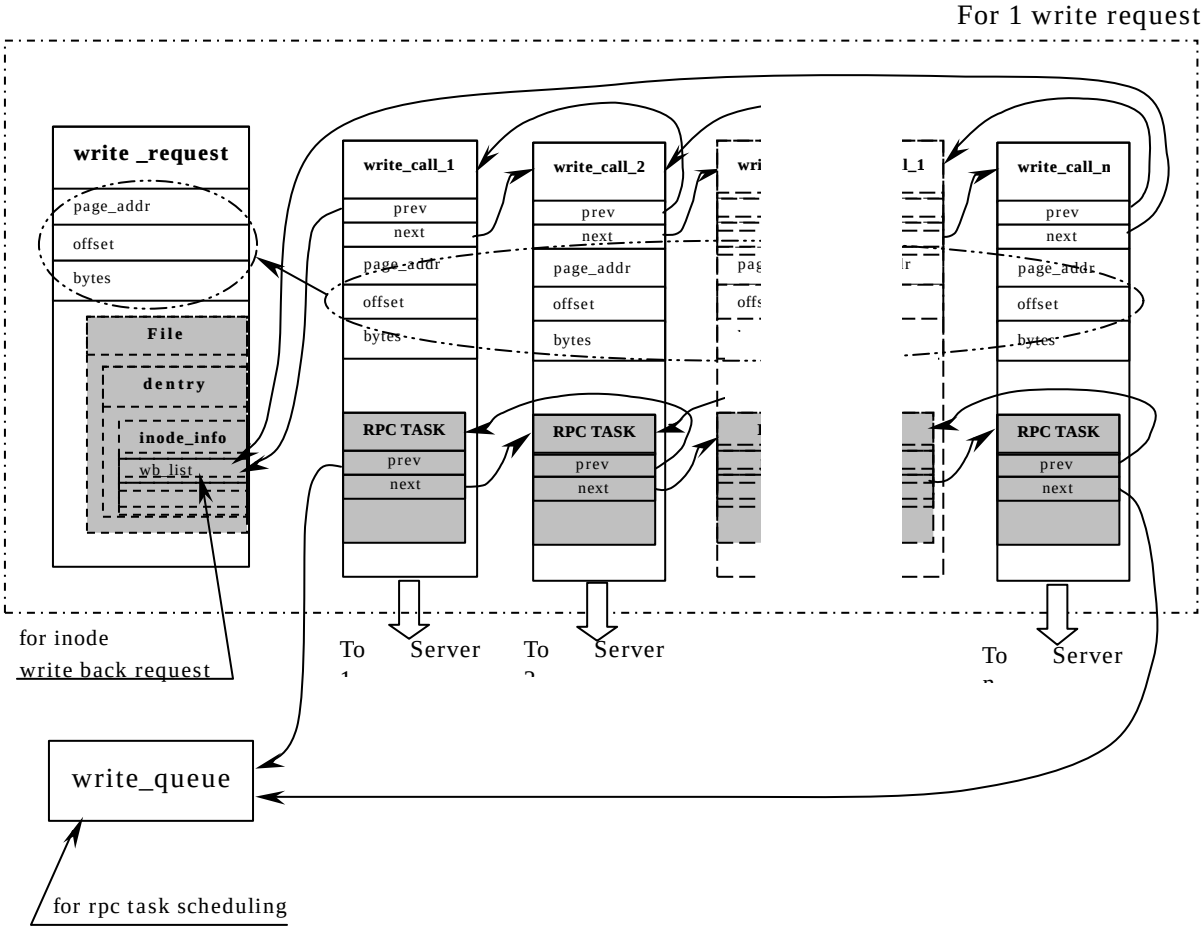


図 3.2.4-2　書き込み処理使用データ間の関係

3.2.5　NRFS の動作の概要

NRFS の動作の概要を図 3.2.5-1 に示す。

動作の流れは、以下のようになる。

①マウント、NFS、NRFS で使用するポート番号の問い合わせ

ポートマッパーは、クライアントからの問い合わせに対し、当該サービスで使用するポート番号を返す（Ⅰ）。

②マウント情報の問い合わせ、マウント要求

マウント・デーモンは、クライアントからの問い合わせに対し、現在マウントを行っているクライアント（Ⅱ）、公開されているディレクトリ情報（Ⅲ）を返す。

クライアントからのマウント要求時には、対象ディレクトリのファイルハンドル（Ⅲ）を返す。

③NFS リクエスト

NRFS サーバは、クライアントからの NFS 操作リクエストに対し、該当する NFS プロシージャを呼出し、結果（Ⅳ）をクライアントに返す。

④NRFS リクエスト

NRFS サーバは、クライアントからの NRFS 操作リクエストにより該当する NRFS プロシージャを呼出し、結果（Ⅴ）をクライアントに返す。

NRFS サーバでは、主に④に対して追加・拡張を行い NRFS 機能を実装した。

なお、（A）は NRFS サーバ初期化時のエクスポート情報並びにファイルハンドル情報の初期化を示し、（B）は NFS、NRFS プロシージャ動作時に発生する VFS オペレーション並びに v-node オペレーションを示す。

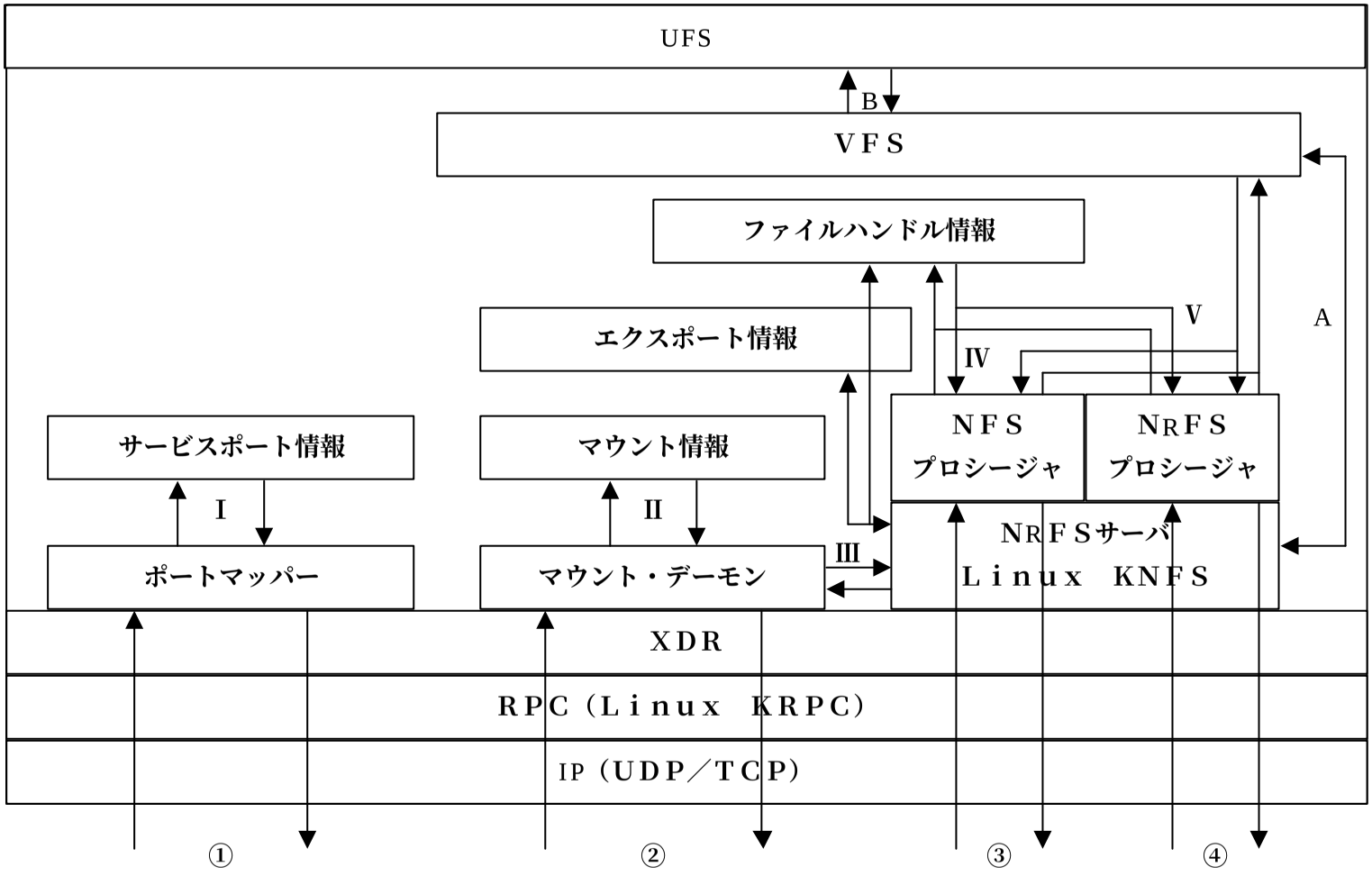


図 3.2.5-1　NRFS 動作の概観図

3.3 NRFS プロトタイプの作成

前節で実施した NRFS 基本部の設計に基づいてプロトタイプを作成する。このプロトタイプでは、サーバ多重化に対応したクライアント並びにサーバを整備するとともに、特定の障害発生に対応した障害検出訂正機能を実装する。

次に、作成したプロトタイプについて、実際にテストを行い、NRFS の実用性を検証する。NRFS 基本部が設計通りに機能することを確認するとともに、そのパフォーマンスを測定し、実用上支障がないレベルで NRFS が実装可能であることを確認する。また、実装した障害検出訂正機能について、障害発生を人為的に模擬したテストを行い、その健全性を確認する。

3.3.1 NRFS プロトタイプの実装

(1) NRFS プロトタイプの実装環境

NRFS プロトタイプの実装にあたっては、図 3.3.1-1 に示す環境で作業を行った。この NRFS サーバ３台と NRFS クライアント１台の組合せは、多数決方式による障害検出が可能な最小構成である。

実運用では、必要に応じて NRFS サーバを増設することでより信頼性を高めることが可能であり、また複数の NRFS クライアントに対してサービスを提供することとなる。

なお、NRFS サーバ、クライアントともに、以下のプラットフォームを採用した。

- Red Hat Linux 6.2J Second Edition / Kernel 2.2.16-3

(2) NRFS プロトタイプで対応する障害

今回実装する NRFS プロトタイプでは、次の２種の障害発生を想定し、その検出機能並びに訂正機能を整備することとした。

① ファイル内容の不整合

② ディレクトリあるいはファイルの欠損

(3) NRFS プロトタイプの実装手順

前節に示した NRFS 基本部の設計にしたがって、以下の手順で作業を進めた。

① NFS クライアントの NFS 層のデータ構造を多重サーバ対応に拡張

NRFS では、NFS と異なり、複数のサーバを取り扱うこととなる。

NRFS スーパーブロック情報の構造体定義を配列化し、サーバ情報を複数取り扱えるよう拡張した。

② NFS クライアントの NFS 層を多重サーバ対応に拡張

NRFS クライアントについて、NRFS inode 情報を配列化し、多重サーバに

対応するよう拡張した。

③　NFS クライアントの mount 機構を多重サーバ対応に拡張

NRFS クライアントについて、一つのマウントポイントに異なる NRFS サーバのファイルシステムをマウント可能なようマウント機構の実装を変更した。

④　NRFS サーバの RPC としてチェックサム対応プロシージャを追加

NRFS サーバについて、障害検出訂正機能に用いるチェックサム機能関連の RPC プロシージャを追加した。

⑤　NRFS クライアントの障害検出ルーチンの作成

NRFS クライアントについて、各サーバより受け取ったチェックサム情報を比較し、障害を検出する機能を整備した。

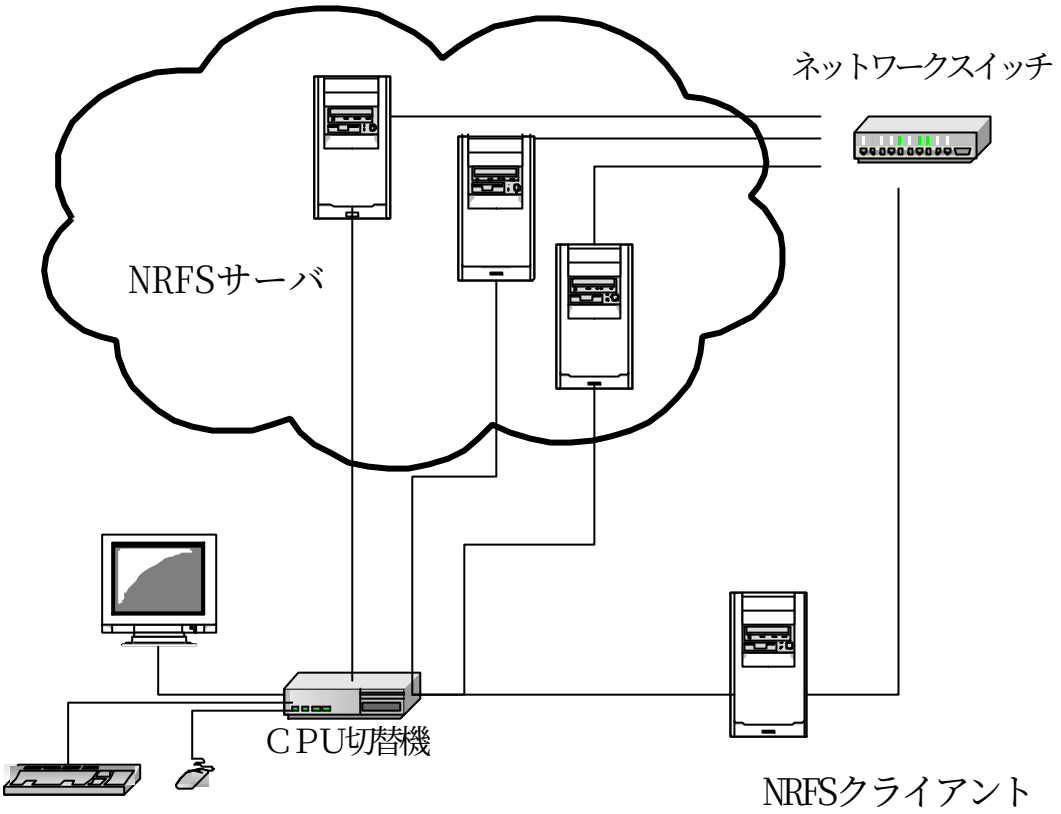
⑥　NRFS クライアントによるデータ訂正機能の作成

データ訂正機能の初期バージョンとして、障害を検出したクライアントを主体としたデータ訂正処理を整備した。

⑦　NRFS サーバによるデータ訂正機能の作成

次のステップとして、障害を検出したクライアントより障害に関する情報を受け取り、NRFS サーバ間で協調しながら障害発生箇所を特定するとともに、データ訂正処理を行う機能を整備した。

具体的には、任意のディレクトリあるいはファイルの欠損について訂正可能な機能を整備した。



・PC本体× 4台
　　サーバ 3台
　　クライアント1台
・CPU切替機 1台
・ネットワークスイッチ1台

図3.3.1-1　NRFSプロトタイプの実装環境

3.3.2 NRFS プロトタイプの機能検証

作成した NRFS プロトタイプに対して実施した機能検証の概要を以下にまとめる。

なお、ここでは、NRFS クライアント並びにサーバの挙動について順を追って確認できるよう、適切なログを出力させた。ログを出力する場合、その処理が律速となり、NRFS 本来の機能は格段に遅くなる。このため、通常のバージョンではこうしたログの出力は行わないものとする。

NRFS プロトタイプの機能検証にあたっては、プロトタイプの実装環境（図 3.3.1-1）と同等の環境で作業を行った。

(1) NRFS クライアントの多重サーバ対応の検証

NRFS クライアントにおける多重サーバ対応の健全性を検証するため、以下に示す一連の処理に対する挙動を確認した。

① mount コマンドによる多重マウント操作
② mkdir コマンドによるディレクトリ作成
③ rmdir コマンドによるディレクトリ削除
④ cp コマンドによるファイル複製
⑤ cat コマンドによるファイル参照
⑥ ln コマンドによるシンボリックリンク操作
⑦ rm コマンドによるファイル削除
⑧ umount コマンドによるマウント解除操作

NRFS クライアント上で入力したコマンドとその応答について、主要部分を抽出し、図 3.3.2-1 に示す。

この一連の処理に対する NRFS クライアントの応答ログを図 3.3.2-2 に、各 NRFS サーバの応答ログを図 3.3.2-3～5 にそれぞれ示す。

入力コマンドに対する応答並びに各ログに示すように、NRFS クライアントにおいて多重サーバのマウントが正常に行われ、当該ファイルシステムに対する各種操作が各 NRFS サーバに対して正常に反映され、またマウント解除操作についても正常に行われたことが確認された。

このように、３台の NRFS サーバにより構成されるネットワーク RAID ファイルシステムが、NRFS クライアントにより一般の NFS と同様に取扱い可能であり、NRFS における多重サーバの取扱いの健全性が確認された。

(2) NRFS 障害検出機能並びに訂正機能の検証

今回実装する NRFS プロトタイプでは、次の２種の障害発生を想定し、その検出機能並びに訂正機能を整備することとした。

① ファイル内容の不整合
② ディレクトリあるいはファイルの欠損

これらの障害発生を人為的に模擬し、その障害検出機能並びに訂正機能の健全性を検証した。

A. 障害『ファイル内容の不整合』について

人為的に、NRFS サーバ上のファイルに不整合を発生させた後、当該ファイルに対し、NRFS クライアントからアクセスすることで、障害検出機能並びに訂正機能の健全性を確認した。

ここでは、一例として NRFS が提供するファイルシステム（/home/nrfs）のファイル（root/left/README.kernel-sources）について、一つのサーバ上でのみ内容を書き換えることで不整合を発生させることとし、不整合の生じたファイルに対し『cat コマンドによるファイル参照』を行い、障害検出からファイルが訂正されるまでの一連の動作を確認した。

NRFS クライアントの応答ログを図 3.3.2-6 に、NRFS サーバの応答ログを図 3.3.2-7 にそれぞれ示す。また、NRFS サーバ間で協調して動作する訂正機能のログを図 3.3.2-8 に示す。

ファイル内容の不整合について、その検出と訂正処理は、以下の流れで動作する。

① ファイルの読み込み時に各サーバが返すチェックサム情報を確認
② チェックサム情報の不一致によりファイル内容の不整合を検出
③ 『多数決』により障害が発生したサーバと正常なサーバを識別
④ 障害が発生したサーバに対し、『障害情報』とともに『訂正要求』を発行
⑤ 障害が発生したサーバより、正常なサーバに対し『訂正処理』を要求
⑥ 『訂正処理要求』を受け取ったサーバと障害が発生したサーバとの間で強調して訂正処理を実行

各ログを確認し、期待通りの処理が行われ、また不整合の生じたファイルが正しく訂正されたことを確認した。

B. 障害『ディレクトリあるいはファイルの欠損』について

人為的に、NRFS サーバ上において特定のディレクトリ並びにファイルを消去した後、当該ディレクトリとファイルに対し、NRFS クライアントからアクセスすることで、障害検出機能並びに訂正機能の健全性を確認した。

ここでは、一例として NRFS が提供するファイルシステム（/home/nrfs）のディレクトリ（root/right）並びにファイル（root/left/README.kernel-sources）について、一つのサーバ上でのみ削除することで不整合を発生させることとし、当該ディレクトリについては『ls コマンドによるディレクトリ内容のリスト表示』を、ファイルに対しは『cat コマンドによるファイル参照』を行い、障害検出から

訂正までの一連の動作を確認した。

NRFS クライアントの応答ログを図 3.3.2-9 に、NRFS サーバの応答ログを図 3.3.2-10 にそれぞれ示す。また、NRFS サーバ間で協調して動作する訂正機能のログを図 3.3.2-11 に示す。

ディレクトリあるいはファイルの欠損について、その検出と訂正処理は、以下の流れで動作する。

① 欠損ディレクトリあるいはファイルに対するアクセス時（『LOOKUP 処理』等）にその欠損を検出
② 『多数決』により欠損が『障害』であると判断された場合、障害が発生したサーバと正常なサーバを識別
③ 障害が発生したサーバに対し、『障害情報』とともに『訂正要求』を発行
④ 障害が発生したサーバより、正常なサーバに対し『訂正処理』を要求
⑤ 『訂正処理要求』を受け取ったサーバと障害が発生したサーバとの間で強調して訂正処理を実行

各ログを確認し、期待通りの処理が行われ、また欠損したディレクトリ並びにファイルについて正しく生成されたことを確認した。

(3) NRFS プロトタイプのパフォーマンス測定

これまでの検討で、NRFS プロトタイプについて設計通り機能することが確認された。しかしながら、NRFS はファイルシステムとしての機能を提供するものであることから、そのパフォーマンスについても十分実用に耐え得るものである必要がある。

ここでは、以下のケースについて、NFS 並びに NRFS のパフォーマンスをそれぞれ測定し、比較することとした。

① NRFS 上からローカルファイルシステムへのファイルコピー
② ローカルファイルシステム上から NRFS へのファイルコピー
③ NFS 上からローカルファイルシステムへのファイルコピー
④ ローカルファイルシステム上から NFS へのファイルコピー

測定結果を表 3.3.2-1 にまとめる。

この結果から、NRFS のパフォーマンスはサーバ多重化により NFS よりも若干劣るものの、実用上は支障がないレベルであることを確認した。

今後は、より詳細な分析を行い、処理機構の見直しあるいはチューニング等により、一層のパフォーマンス向上を図り、NFS と同等レベルにまで改善させたいと考えている。

①mountコマンドによる
多重マウント操作

```
[root@nrpc04 tests]# mount nrpc01::/home/nrfs /mnt/nrfs
[root@nrpc04 tests]# mount nrpc02::/home/nrfs /mnt/nrfs
[root@nrpc04 tests]# mount nrpc03::/home/nrfs /mnt/nrfs
[root@nrpc04 tests]# mount
    /dev/hda3 on / type ext2 (rw)
    none on /proc type proc (rw)
    /dev/hda1 on /boot type ext2 (rw)
    /dev/hda8 on /home type ext2 (rw)
    /dev/hda7 on /tmp type ext2 (rw)
    /dev/hda5 on /usr type ext2 (rw)
    /dev/hda6 on /var type ext2 (rw)
    none on /dev/pts type devpts (rw,gid=5,mode=620)
    automount(pid389) on /misc type autofs (rw,fd=5,pgrp=389,minproto=2,maxproto=3)
    nrpc01::/home/nrfs on /mnt/nrfs type nrfs (rw,addr=172.20.37.140)
    nrpc02::/home/nrfs on /mnt/nrfs type nrfs (rw,addr=172.20.37.141)
    nrpc03::/home/nrfs on /mnt/nrfs type nrfs (rw,addr=172.20.37.142)
```

②mkdirコマンドによる
ディレクトリ作成

```
[nrfs@nrpc04 ~/tests]$ cd /mnt/nrfs
[nrfs@nrpc04 nrfs]$ ls
    testdir

[nrfs@nrpc04 nrfs]$ mkdir root
[nrfs@nrpc04 nrfs]$ mkdir root/right
[nrfs@nrpc04 nrfs]$ mkdir root/left
[nrfs@nrpc04 nrfs]$ mkdir root/right/right
[nrfs@nrpc04 nrfs]$ mkdir root/right/left
[nrfs@nrpc04 nrfs]$ ls -lR root
    root:
    合計 8
    drwxrwxr-x    2 nrfs     nrfs         4096 Mar  6 13:13 left
    drwxrwxr-x    4 nrfs     nrfs         4096 Mar  6 13:13 right

    root/left:
    合計 0

    root/right:
    合計 8
    drwxrwxr-x    2 nrfs     nrfs         4096 Mar  6 13:13 left
    drwxrwxr-x    2 nrfs     nrfs         4096 Mar  6 13:13 righet

    root/right/left:
    合計 0

    root/right/righet:
    合計 0
```

③rmdirコマンドによる
ディレクトリ削除

```
[nrfs@nrpc04 nrfs]$ rmdir root/right/right
[nrfs@nrpc04 nrfs]$ rmdir root/right/left
[nrfs@nrpc04 nrfs]$ ls -lR root
    root:
    合計 8
    drwxrwxr-x    2 nrfs     nrfs         4096 Mar  6 13:13 left
    drwxrwxr-x    2 nrfs     nrfs         4096 Mar  6 13:14 right

    root/left:
    合計 0

    root/right:
    合計 0
```

④cpコマンドによる
ファイル複製

```
[nrfs@nrpc04 nrfs]$ ls -l /var/log/dmesg
    -rw-r--r--    1 root     root         2864 Mar  6 11:03 /var/log/dmesg

[nrfs@nrpc04 nrfs]$ cp /var/log/dmesg root/left
[nrfs@nrpc04 nrfs]$ ls -lR root
    root:
    合計 8
    drwxrwxr-x    2 nrfs     nrfs         4096 Mar  6 13:14 left
    drwxrwxr-x    2 nrfs     nrfs         4096 Mar  6 13:14 right

    root/left:
    合計 4
    -rw-r--r--    1 nrfs     nrfs         2864 Mar  6 13:13 dmesg

    root/right:
    合計 0
```

⑤catコマンドによる
ファイル参照

```
[nrfs@nrpc04 nrfs]$ ls -l testdir/cat
    合計 16
    -rw-r--r--    1 nrfs     nrfs        13561 Mar  6 12:39 README

[nrfs@nrpc04 nrfs]$ cat testdir/cat/README
    Linux kernel release 2.2.xx


    These are the release notes for Linux version 2.2.  Read them carefully,
    as they tell you what this is all about, explain how to install the
```

```
    kernel, and what to do if something goes wrong.

    However, please make sure you don't ask questions which are already answered
    in various files in the Documentation directory.  See DOCUMENTATION below.

    WHAT IS LINUX?

    ...

       If you for some reason cannot do the above (you have a pre-compiled
       kernel image or similar), telling me as much about your setup as
       possible will help.

     - Alternately, you can use gdb on a running kernel. (read-only; i.e. you
       cannot change values or set break points.) To do this, first compile the
       kernel with -g; edit arch/i386/Makefile appropriately, then do a "make
       clean". You'll also need to enable CONFIG_PROC_FS (via "make config").

       After you've rebooted with the new kernel, do "gdb vmlinux /proc/kcore".
       You can now use all the usual gdb commands. The command to look up the
       point where your system crashed is "l *0xXXXXXXXX". (Replace the XXXes
       with the EIP value.)

       gdb'ing a non-running kernel currently fails because gdb (wrongly)
       disregards the starting offset for which the kernel is compiled.
```

⑥lnコマンドによる
シンボリックリンク操作

```
[nrfs@nrpc04 nrfs]$ ln -s ~/tests tests
[nrfs@nrpc04 nrfs]$ ls -l tests
    lrwxrwxrwx    1 nrfs     nrfs           16 Mar  6 13:15 tests -> /home/nrfs/tests
```

```
[nrfs@nrpc04 nrfs]$ ls -lR root
    root:
    合計 8
    drwxrwxr-x    2 nrfs     nrfs         4096 Mar  6 13:14 left
    drwxrwxr-x    2 nrfs     nrfs         4096 Mar  6 13:14 right

    root/left:
    合計 4
    -rw-r--r--    1 nrfs     nrfs         2864 Mar  6 13:13 dmesg

    root/right:
    合計 0

[nrfs@nrpc04 nrfs]$ rm root/left/dmesg
[nrfs@nrpc04 nrfs]$ ls -lR root
    root:
    合計 8
    drwxrwxr-x    2 nrfs     nrfs         4096 Mar  6 13:15 left
    drwxrwxr-x    2 nrfs     nrfs         4096 Mar  6 13:14 right

    root/left:
    合計 0

    root/right:
    合計 0

[nrfs@nrpc04 nrfs]$ cd ~/tests
```

⑦rmコマンドによる
ファイル削除

⑧umountコマンドによる
マウント解除操作

```
[root@nrpc04 tests]# umount /mnt/nrfs
[root@nrpc04 tests]# mount
    /dev/hda3 on / type ext2 (rw)
    none on /proc type proc (rw)
    /dev/hda1 on /boot type ext2 (rw)
    /dev/hda8 on /home type ext2 (rw)
    /dev/hda7 on /tmp type ext2 (rw)
    /dev/hda5 on /usr type ext2 (rw)
    /dev/hda6 on /var type ext2 (rw)
    none on /dev/pts type devpts (rw,gid=5,mode=620)
    automount(pid389) on /misc type autofs (rw,fd=5,pgrp=389,minproto=2,maxproto=3)
```

図3.3.2-1　NRFSクライアントの多重サーバ対応の検証
（NRFSクライアント上で入力したコマンドとその応答）

```
<4>NRFS: call  getattr from SERVER#0
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 624f4
<4>NRFS: SERVER#0 reply getattr

<4>NRFS: refresh_inode(4/402676 ct=1)
<4>NRFS: __nrfs_fhget(4/402676 ct=1)

<4>NRFS: call  getattr from SERVER#1
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 5a70d
<4>NRFS: SERVER#1 reply getattr

<4>NRFS: call  getattr from SERVER#2
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 75e43
<4>NRFS: SERVER#2 reply getattr

<4>NRFS: call  getattr from SERVER#0
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 624f4
<4>NRFS: SERVER#0 reply getattr

<4>NRFS: call  getattr from SERVER#1
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 5a70d
<4>NRFS: SERVER#1 reply getattr

<4>NRFS: call  getattr from SERVER#2
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 75e43
<4>NRFS: SERVER#2 reply getattr

<4>NRFS: refresh_inode(4/402676 ct=1)


<4>NRFS: nrfs_readdir(///)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 3

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 3

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 3

<4>NRFS: nrfs_readdir(///)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS: call  getattr from SERVER#0
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 624f4
<4>NRFS: SERVER#0 reply getattr

<4>NRFS: call  getattr from SERVER#1
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 5a70d
<4>NRFS: SERVER#1 reply getattr

<4>NRFS: call  getattr from SERVER#2
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 75e43
<4>NRFS: SERVER#2 reply getattr

<4>NRFS: refresh_inode(4/402676 ct=1)


<4>NRFS: lookup(//root)

<4>NRFS: call SERVER#0 to lookup root
<4>RPC:      diropres status 2
<4>NRFS: SERVER#0 reply lookup: -2
<1>NRFS: Can't find root on SERVER#0 !!!

<4>NRFS: call SERVER#1 to lookup root
<4>RPC:      diropres status 2
<4>NRFS: SERVER#1 reply lookup: -2
<1>NRFS: Can't find root on SERVER#1 !!!

<4>NRFS: call SERVER#2 to lookup root
<4>RPC:      diropres status 2
<4>NRFS: SERVER#2 reply lookup: -2
<1>NRFS: Can't find root on SERVER#2 !!!

<4>NRFS: mkdir(4/402676, root

<4>NRFS: call SERVER#0 mkdir root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa9
<4>NRFS: SERVER#0 reply mkdir: 0

<4>NRFS: call SERVER#1 mkdir root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ad
```

mkdir root

```
<4>NRFS: SERVER#1 reply mkdir: 0

<4>NRFS: call SERVER#2 mkdir root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 75e41
<4>NRFS: SERVER#2 reply mkdir: 0

<4>NRFS dentry_delete(//root, d_flags=0 means do nothing)

<4>NRFS: lookup(//root)

<4>NRFS: call SERVER#0 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa9
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ad
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 75e41
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(//root, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(//root, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(//root, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(//root set fileid=2:1074224705)
<4>NRFS: refresh_inode(4/1074224705 ct=1)
<4>NRFS: __nrfs_fhget(4/1074224705 ct=1)
```

```
<4>NRFS: lookup(root/right)

<4>NRFS: call SERVER#0 to lookup right
<4>RPC:      diropres status 2
<4>NRFS: SERVER#0 reply lookup: -2
<1>NRFS: Can't find right on SERVER#0 !!!

<4>NRFS: call SERVER#1 to lookup right
<4>RPC:      diropres status 2
<4>NRFS: SERVER#1 reply lookup: -2
<1>NRFS: Can't find right on SERVER#1 !!!

<4>NRFS: call SERVER#2 to lookup right
<4>RPC:      diropres status 2
<4>NRFS: SERVER#2 reply lookup: -2
<1>NRFS: Can't find right on SERVER#2 !!!

<4>NRFS: mkdir(4/1074224705, right
```

mkdir root/right

```
<4>NRFS: call SERVER#0 mkdir right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aaa
<4>NRFS: SERVER#0 reply mkdir: 0

<4>NRFS: call SERVER#1 mkdir right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ae
<4>NRFS: SERVER#1 reply mkdir: 0

<4>NRFS: call SERVER#2 mkdir right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 2b3a2
<4>NRFS: SERVER#2 reply mkdir: 0

<4>NRFS dentry_delete(root/right, d_flags=0 means do nothing)
<4>NRFS dentry_delete(//root, d_flags=0 means do nothing)
```

```
<4>NRFS: lookup(root/left)

<4>NRFS: call SERVER#0 to lookup left
<4>RPC:      diropres status 2
<4>NRFS: SERVER#0 reply lookup: -2
<1>NRFS: Can't find left on SERVER#0 !!!

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 2
<4>NRFS: SERVER#1 reply lookup: -2
<1>NRFS: Can't find left on SERVER#1 !!!

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:      diropres status 2
<4>NRFS: SERVER#2 reply lookup: -2
<1>NRFS: Can't find left on SERVER#2 !!!

<4>NRFS: mkdir(4/1074224705, left
```

mkdir root/left

```
<4>NRFS: call SERVER#0 mkdir left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aab
<4>NRFS: SERVER#0 reply mkdir: 0

<4>NRFS: call SERVER#1 mkdir left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625af
<4>NRFS: SERVER#1 reply mkdir: 0

<4>NRFS: call SERVER#2 mkdir left
```

```
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 5a623
<4>NRFS: SERVER#2 reply mkdir: 0

<4>NRFS dentry_delete(root/left, d_flags=0 means do nothing)
<4>NRFS dentry_delete(//root, d_flags=0 means do nothing)


<4>NRFS: lookup(root/right)

<4>NRFS: call SERVER#0 to lookup right
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 66aaa
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup right
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 625ae
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup right
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 2b3a2
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(root/right, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(root/right, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(root/right, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(root/right set fileid=2:1073918882)
<4>NRFS: refresh_inode(4/1073918882 ct=1)
<4>NRFS: __nrfs_fhget(4/1073918882 ct=1)


<4>NRFS: lookup(right/righet)

<4>NRFS: call SERVER#0 to lookup righet
<4>RPC:       diropres status 2
<4>NRFS: SERVER#0 reply lookup: -2
<1>NRFS: Can't find righet on SERVER#0 !!!

<4>NRFS: call SERVER#1 to lookup righet
<4>RPC:       diropres status 2
<4>NRFS: SERVER#1 reply lookup: -2
<1>NRFS: Can't find righet on SERVER#1 !!!

<4>NRFS: call SERVER#2 to lookup righet
<4>RPC:       diropres status 2
<4>NRFS: SERVER#2 reply lookup: -2
<1>NRFS: Can't find righet on SERVER#2 !!!

<4>NRFS: mkdir(4/1073918882, righet

<4>NRFS: call SERVER#0 mkdir righet
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 66aac
<4>NRFS: SERVER#0 reply mkdir: 0

<4>NRFS: call SERVER#1 mkdir righet
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 625b0
<4>NRFS: SERVER#1 reply mkdir: 0

<4>NRFS: call SERVER#2 mkdir righet
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 623e1
<4>NRFS: SERVER#2 reply mkdir: 0

<4>NRFS dentry_delete(right/righet, d_flags=0 means do nothing)
<4>NRFS dentry_delete(root/right, d_flags=0 means do nothing)


<4>NRFS: lookup(right/left)

<4>NRFS: call SERVER#0 to lookup left
<4>RPC:       diropres status 2
<4>NRFS: SERVER#0 reply lookup: -2
<1>NRFS: Can't find left on SERVER#0 !!!

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:       diropres status 2
<4>NRFS: SERVER#1 reply lookup: -2
<1>NRFS: Can't find left on SERVER#1 !!!

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:       diropres status 2
<4>NRFS: SERVER#2 reply lookup: -2
<1>NRFS: Can't find left on SERVER#2 !!!

<4>NRFS: mkdir(4/1073918882, left

<4>NRFS: call SERVER#0 mkdir left
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 66aad
<4>NRFS: SERVER#0 reply mkdir: 0

<4>NRFS: call SERVER#1 mkdir left
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 625b1
<4>NRFS: SERVER#1 reply mkdir: 0

<4>NRFS: call SERVER#2 mkdir left
<4>RPC:       diropres status 0
```

mkdir root/right/righet

mkdir root/right/left

```
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 662c3
<4>NRFS: SERVER#2 reply mkdir: 0

<4>NRFS dentry_delete(right/left, d_flags=0 means do nothing)
<4>NRFS dentry_delete(root/right, d_flags=0 means do nothing)


<4>NRFS: call SERVER#0 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa9
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ad
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 75e41
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074224705 ct=1)

<4>NRFS: nrfs_readdir(//root)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 4

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 4

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 4

<4>NRFS: call SERVER#0 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aaa
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ae
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 2b3a2
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1073918882 ct=1)

<4>NRFS dentry_delete(root/right, d_flags=0 means do nothing)

<4>NRFS: lookup(root/left)

<4>NRFS: call SERVER#0 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aab
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625af
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 5a623
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(root/left, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(root/left, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(root/left, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(root/left set fileid=2:1074112035)
<4>NRFS: refresh_inode(4/1074112035 ct=1)
<4>NRFS: __nrfs_fhget(4/1074112035 ct=1)

<4>NRFS dentry_delete(root/left, d_flags=0 means do nothing)

<4>NRFS: nrfs_readdir(//root)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS: call SERVER#0 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aab
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625af
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup left
```

```
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 5a623
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074112035 ct=1)

<4>NRFS: nrfs_readdir(root/left)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 2

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 2

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 2

<4>NRFS: nrfs_readdir(root/left)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS dentry_delete(root/left, d_flags=0 means do nothing)

<4>NRFS: call SERVER#0 to lookup right
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 66aaa
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup right
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 625ae
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup right
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 2b3a2
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1073918882 ct=1)

<4>NRFS: nrfs_readdir(root/right)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 4

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 4

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 4

<4>NRFS: lookup(right/righet)

<4>NRFS: call SERVER#0 to lookup righet
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 66aac
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup righet
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 625b0
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup righet
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 623e1
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(right/righet, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(right/righet, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(right/righet, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(right/righet set fileid=2:1074144225)
<4>NRFS: refresh_inode(4/1074144225 ct=1)
<4>NRFS: __nrfs_fhget(4/1074144225 ct=1)

<4>NRFS dentry_delete(right/righet, d_flags=0 means do nothing)

<4>NRFS: lookup(right/left)

<4>NRFS: call SERVER#0 to lookup left
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 66aad
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 625b1
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:       diropres status 0
<4>RPC:       diropres OK type 2 mode 40775 dev 308 ino 662c3
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(right/left, intall fh from SERVER#0, 0)
```

```
<4>NRFS: nrfs_fhget(right/left, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(right/left, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(right/left set fileid=2:1074160323)
<4>NRFS: refresh_inode(4/1074160323 ct=1)
<4>NRFS: __nrfs_fhget(4/1074160323 ct=1)

<4>NRFS dentry_delete(right/left, d_flags=0 means do nothing)

<4>NRFS: nrfs_readdir(root/right)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS: call SERVER#0 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aad
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625b1
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 662c3
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074160323 ct=1)

<4>NRFS: nrfs_readdir(right/left)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 2

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 2

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 2

<4>NRFS: nrfs_readdir(right/left)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS dentry_delete(right/left, d_flags=0 means do nothing)

<4>NRFS: call SERVER#0 to lookup righet
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aac
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup righet
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625b0
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup righet
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 623e1
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074144225 ct=1)

<4>NRFS: nrfs_readdir(right/righet)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 2

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 2

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 2

<4>NRFS: nrfs_readdir(right/righet)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS dentry_delete(right/righet, d_flags=0 means do nothing)
```

```
<4>NRFS: call SERVER#0 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa9
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ad
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 75e41
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074224705 ct=1)

<4>NRFS: call SERVER#0 to lookup righet
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aac
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup righet
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625b0
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup righet
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 623e1
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074144225 ct=1)

<4>NRFS: rmdir(4/1073918882, righet

<4>NRFS: call SERVER#0 to rmdir righet
<4>NRFS: SERVER#0 reply rmdir: 0

<4>NRFS: call SERVER#1 to rmdir righet
<4>NRFS: SERVER#1 reply rmdir: 0

<4>NRFS: call SERVER#2 to rmdir righet
<4>NRFS: SERVER#2 reply rmdir: 0

<4>NRFS dentry_delete(right/righet, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/2:1074144225)
<4>NRFS: delete_inode(4/2:1074144225)
```

rmdir root/right/righet

```
<4>NRFS: call SERVER#0 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aad
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625b1
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 662c3
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074160323 ct=1)

<4>NRFS: rmdir(4/1073918882, left

<4>NRFS: call SERVER#0 to rmdir left
<4>NRFS: SERVER#0 reply rmdir: 0

<4>NRFS: call SERVER#1 to rmdir left
<4>NRFS: SERVER#1 reply rmdir: 0

<4>NRFS: call SERVER#2 to rmdir left
<4>NRFS: SERVER#2 reply rmdir: 0

<4>NRFS dentry_delete(right/left, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/2:1074160323)
<4>NRFS: delete_inode(4/2:1074160323)


<4>NRFS dentry_delete(root/right, d_flags=0 means do nothing)

<4>NRFS: call SERVER#0 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa9
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ad
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 75e41
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074224705 ct=1)
```

rmdir root/right/left

```
<4>NRFS: nrfs_readdir(//root)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 4

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 4

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 4

<4>NRFS: call SERVER#0 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aaa
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ae
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 2b3a2
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1073918882 ct=1)

<4>NRFS: call  getattr from SERVER#0
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40775 dev 308 ino 66aaa
<4>NRFS: SERVER#0 reply getattr

<4>NRFS: call  getattr from SERVER#1
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40775 dev 308 ino 625ae
<4>NRFS: SERVER#1 reply getattr

<4>NRFS: call  getattr from SERVER#2
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40775 dev 308 ino 2b3a2
<4>NRFS: SERVER#2 reply getattr

<4>NRFS: refresh_inode(4/1073918882 ct=1)

<4>NRFS dentry_delete(root/right, d_flags=0 means do nothing)

<4>NRFS: call SERVER#0 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aab
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625af
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 5a623
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074112035 ct=1)

<4>NRFS dentry_delete(root/left, d_flags=0 means do nothing)

<4>NRFS: nrfs_readdir(//root)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS: call SERVER#0 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aab
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625af
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 5a623
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074112035 ct=1)

<4>NRFS: nrfs_readdir(root/left)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 2

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 2
```

```
<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 2

<4>NRFS: nrfs_readdir(root/left)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS dentry_delete(root/left, d_flags=0 means do nothing)

<4>NRFS: call SERVER#0 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aaa
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ae
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 2b3a2
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1073918882 ct=1)

<4>NRFS: nrfs_readdir(root/right)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 2

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 2

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 2

<4>NRFS: nrfs_readdir(root/right)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS dentry_delete(root/right, d_flags=0 means do nothing)

<4>NRFS: call  getattr from SERVER#0
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 624f4
<4>NRFS: SERVER#0 reply getattr

<4>NRFS: call  getattr from SERVER#1
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 5a70d
<4>NRFS: SERVER#1 reply getattr

<4>NRFS: call  getattr from SERVER#2
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 75e43
<4>NRFS: SERVER#2 reply getattr

<4>NRFS: refresh_inode(4/402676 ct=1)

<4>NRFS: call  getattr from SERVER#0
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 624f4
<4>NRFS: SERVER#0 reply getattr

<4>NRFS: call  getattr from SERVER#1
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 5a70d
<4>NRFS: SERVER#1 reply getattr

<4>NRFS: call  getattr from SERVER#2
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 75e43
<4>NRFS: SERVER#2 reply getattr

<4>NRFS: refresh_inode(4/402676 ct=1)


<4>NRFS: lookup(//testdir)

<4>NRFS: call SERVER#0 to lookup testdir
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa4
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup testdir
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625a8
<4>NRFS: SERVER#1 reply lookup: 0
```

```
<4>NRFS: call SERVER#2 to lookup testdir
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 71f61
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(//testdir, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(//testdir, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(//testdir, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(//testdir set fileid=2:1074208609)
<4>NRFS: refresh_inode(4/1074208609 ct=1)
<4>NRFS: __nrfs_fhget(4/1074208609 ct=1)

<4>NRFS: lookup(testdir/cp)

<4>NRFS: call SERVER#0 to lookup cp
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa6
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup cp
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625aa
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup cp
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 71f62
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(testdir/cp, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(testdir/cp, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(testdir/cp, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(testdir/cp set fileid=2:1074208610)
<4>NRFS: refresh_inode(4/1074208610 ct=1)
<4>NRFS: __nrfs_fhget(4/1074208610 ct=1)

<4>NRFS dentry_delete(testdir/cp, d_flags=0 means do nothing)

<4>NRFS: nrfs_readdir(testdir/cp)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 3

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 3

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 3

<4>NRFS: lookup(cp/MAINTAINERS)

<4>NRFS: call SERVER#0 to lookup MAINTAINERS
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 66aa8
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup MAINTAINERS
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 625ac
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup MAINTAINERS
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 71f63
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(cp/MAINTAINERS, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(cp/MAINTAINERS, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(cp/MAINTAINERS, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(cp/MAINTAINERS set fileid=2:1074208611)
<4>NRFS: refresh_inode(4/1074208611 ct=1)
<4>NRFS: __nrfs_fhget(4/1074208611 ct=1)

<4>NRFS dentry_delete(cp/MAINTAINERS, d_flags=0 means do nothing)

<4>NRFS: nrfs_readdir(testdir/cp)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS dentry_delete(cp/MAINTAINERS, d_flags=0 means do nothing)


<4>nrfs: read(cp/MAINTAINERS, 4096@0)

<4>NRFS: nrfs_readpage (c034d0e8, cp/MAINTAINERS, 4096@0)
<4>NRFS: nrfs_readpage_async(c034d0e8, cp/MAINTAINERS, 4096@0)

<4>NRFS: 11 read buffer(c52e8000) for SERVER#0 allocated
<4>NRFS: 11 read buffer(c5331000) for SERVER#1 allocated
<4>NRFS: 11 read buffer(c5330000) for SERVER#2 allocated

<4>NRFS: 11 executing async read call for SERVER#0(nrpc01)
```

cp testdir/cp/MAINTAINERS
root/left/dmesg

```
<4>NRFS: 11 executing async read call for SERVER#1(nrpc02)
<4>NRFS: 11 executing async read call for SERVER#2(nrpc03)

<4>NRFS: nrfs_readpage (c034d0c0, cp/MAINTAINERS, 4096@4096)
<4>NRFS: nrfs_readpage_async(c034d0c0, cp/MAINTAINERS, 4096@4096)

<4>NRFS: 12 read buffer(c62dc000) for SERVER#0 allocated
<4>NRFS: 12 read buffer(c62dd000) for SERVER#1 allocated
<4>NRFS: 12 read buffer(c62ec000) for SERVER#2 allocated

<4>NRFS: 12 executing async read call for SERVER#0(nrpc01)
<4>NRFS: 12 executing async read call for SERVER#1(nrpc02)
<4>NRFS: 12 executing async read call for SERVER#2(nrpc03)

<4>NRFS: nrfs_readpage (c034e588, cp/MAINTAINERS, 4096@8192)
<4>NRFS: nrfs_readpage_async(c034e588, cp/MAINTAINERS, 4096@8192)

<4>NRFS: deferring async READ request.

<4>NRFS: nrfs_readpage_sync(c034e588)

<4>NRFS: 13 read buffer(c62ed000) for SERVER#0 allocated
<4>NRFS: 13 read buffer(c5f2a000) for SERVER#1 allocated
<4>NRFS: 13 read buffer(c5f2b000) for SERVER#2 allocated

<4>NRFS: 13 nrfs_readpage_sync(cp/MAINTAINERS on nrpc01, 4096@8192, c53bd000)

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:2748)
<4>NRFS: 11 callback(xid=1423) for page c5339000 from server#1, result 0

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:2748)
<4>NRFS: 11 callback(xid=1424) for page c5339000 from server#2, result 0

<4>RPC:      checksum and data read reply OK status 0
<4>RPC:      checksum and data readres OK, csum(0:4:2748), count 4096
<4>NRFS: 11 callback(xid=1422) for page c5339000 from server#0, result 4096

<4>NRFS: 11 all read call for page c5339000 completed
<4>NRFS: 11 validate data replied from each server
<4>NRFS: 11 update page c5339000 with data reply from SERVER#0

<4>NRFS: refresh_inode(4/1074208611 ct=1)

<4>NRFS: 11 readpage_async done, 10 successful, 0 failures

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:1f38)
<4>NRFS: 12 callback(xid=1426) for page c5338000 from server#1, result 0

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:1f38)
<4>NRFS: 12 callback(xid=1427) for page c5338000 from server#2, result 0

<4>RPC:      checksum and data read reply OK status 0
<4>RPC:      checksum and data readres OK, csum(0:4:1f38), count 4096
<4>NRFS: 12 callback(xid=1425) for page c5338000 from server#0, result 4096

<4>NRFS: 12 all read call for page c5338000 completed
<4>NRFS: 12 validate data replied from each server
<4>NRFS: 12 update page c5338000 with data reply from SERVER#0

<4>NRFS: refresh_inode(4/1074208611 ct=1)

<4>NRFS: 12 readpage_async done, 11 successful, 0 failures

<4>RPC:      checksum and data read reply OK status 0
<4>RPC:      checksum and data readres OK, csum(0:4:f1f), count 4096
<4>NRFS: 13 nrfs_readpage_sync(cp/MAINTAINERS on nrpc02, 4096@8192, c53bd000)

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:f1f)
<4>NRFS: 13 nrfs_readpage_sync(cp/MAINTAINERS on nrpc03, 4096@8192, c53bd000)

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:f1f)
<4>NRFS: 13 validate data replied from each server
<4>NRFS: 13 update page c53bd000 with data reply from SERVER#0

<4>NRFS: refresh_inode(4/1074208611 ct=1)

<4>NRFS: nrfs_readpage (c034e560, cp/MAINTAINERS, 4096@12288)
<4>NRFS: nrfs_readpage_async(c034e560, cp/MAINTAINERS, 4096@12288)

<4>NRFS: 14 read buffer(c52e8000) for SERVER#0 allocated
<4>NRFS: 14 read buffer(c5330000) for SERVER#1 allocated
<4>NRFS: 14 read buffer(c5331000) for SERVER#2 allocated

<4>NRFS: 14 executing async read call for SERVER#1(nrpc02)
<4>NRFS: 14 executing async read call for SERVER#0(nrpc01)
<4>NRFS: 14 executing async read call for SERVER#2(nrpc03)

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:79e7)
<4>NRFS: 14 callback(xid=1432) for page c53bc000 from server#0, result 0

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:79e7)
<4>NRFS: 14 callback(xid=1433) for page c53bc000 from server#2, result 0

<4>RPC:      checksum and data read reply OK status 0
<4>RPC:      checksum and data readres OK, csum(0:4:79e7), count 4096
```

```
<4>NRFS: 14 callback(xid=1431) for page c53bc000 from server#1, result 4096

<4>NRFS: 14 all read call for page c53bc000 completed
<4>NRFS: 14 validate data replied from each server
<4>NRFS: 14 update page c53bc000 with data reply from SERVER#1

<4>NRFS: refresh_inode(4/1074208611 ct=1)

<4>NRFS: 14 readpage_async done, 12 successful, 0 failures

<4>nrfs: read(cp/MAINTAINERS, 4096@4096)

<4>NRFS: nrfs_readpage (c034eda8, cp/MAINTAINERS, 4096@16384)
<4>NRFS: nrfs_readpage_async(c034eda8, cp/MAINTAINERS, 4096@16384)

<4>NRFS: 15 read buffer(c52e8000) for SERVER#0 allocated
<4>NRFS: 15 read buffer(c62dd000) for SERVER#1 allocated
<4>NRFS: 15 read buffer(c62dc000) for SERVER#2 allocated

<4>NRFS: 15 executing async read call for SERVER#0(nrpc01)
<4>NRFS: 15 executing async read call for SERVER#1(nrpc02)
<4>NRFS: 15 executing async read call for SERVER#2(nrpc03)

<4>NRFS: nrfs_readpage (c034ed80, cp/MAINTAINERS, 4096@20480)
<4>NRFS: nrfs_readpage_async(c034ed80, cp/MAINTAINERS, 4096@20480)

<4>NRFS: 16 read buffer(c62ed000) for SERVER#0 allocated
<4>NRFS: 16 read buffer(c62ec000) for SERVER#1 allocated
<4>NRFS: 16 read buffer(c5f2b000) for SERVER#2 allocated

<4>NRFS: 16 executing async read call for SERVER#0(nrpc01)
<4>NRFS: 16 executing async read call for SERVER#1(nrpc02)
<4>NRFS: 16 executing async read call for SERVER#2(nrpc03)

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:1f64)
<4>NRFS: 15 callback(xid=1435) for page c53f1000 from server#1, result 0

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:1f64)
<4>NRFS: 15 callback(xid=1436) for page c53f1000 from server#2, result 0

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:5746)
<4>NRFS: 16 callback(xid=1438) for page c53f0000 from server#1, result 0

<4>RPC:      checksum and data read reply OK status 0
<4>RPC:      checksum and data readres OK, csum(0:4:1f64), count 4096
<4>NRFS: 15 callback(xid=1434) for page c53f1000 from server#0, result 4096

<4>NRFS: 15 all read call for page c53f1000 completed
<4>NRFS: 15 validate data replied from each server
<4>NRFS: 15 update page c53f1000 with data reply from SERVER#0

<4>NRFS: refresh_inode(4/1074208611 ct=1)

<4>NRFS: 15 readpage_async done, 13 successful, 0 failures

<4>RPC:      checksum and data read reply OK status 0
<4>RPC:      checksum and data readres OK, csum(0:4:5746), count 1108
<4>NRFS: 16 callback(xid=1437) for page c53f0000 from server#0, result 1108

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:5746)
<4>NRFS: 16 callback(xid=1439) for page c53f0000 from server#2, result 0

<4>NRFS: 16 all read call for page c53f0000 completed
<4>NRFS: 16 validate data replied from each server
<4>NRFS: 16 update page c53f0000 with data reply from SERVER#0

<4>NRFS: refresh_inode(4/1074208611 ct=1)

<4>NRFS: 16 readpage_async done, 14 successful, 0 failures

<4>nrfs: read(cp/MAINTAINERS, 4096@8192)
<4>nrfs: read(cp/MAINTAINERS, 4096@12288)
<4>nrfs: read(cp/MAINTAINERS, 4096@16384)
<4>nrfs: read(cp/MAINTAINERS, 4096@20480)
<4>nrfs: read(cp/MAINTAINERS, 4096@21588)

<4>nrfs: flush(4/2:1074208611)

<4>NRFS dentry_delete(cp/MAINTAINERS, d_flags=0 means do nothing)


<4>NRFS: call SERVER#0 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aab
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625af
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 5a623
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074112035 ct=1)

<4>NRFS dentry_delete(root/left, d_flags=0 means do nothing)
```

```
<4>NRFS dentry_delete(root/left, d_flags=0 means do nothing)

<4>NRFS: lookup(left/dmesg)

<4>NRFS: call SERVER#0 to lookup dmesg
<4>RPC:      diropres status 2
<4>NRFS: SERVER#0 reply lookup: -2
<1>NRFS: Can't find dmesg on SERVER#0 !!!

<4>NRFS: call SERVER#1 to lookup dmesg
<4>RPC:      diropres status 2
<4>NRFS: SERVER#1 reply lookup: -2
<1>NRFS: Can't find dmesg on SERVER#1 !!!

<4>NRFS: call SERVER#2 to lookup dmesg
<4>RPC:      diropres status 2
<4>NRFS: SERVER#2 reply lookup: -2
<1>NRFS: Can't find dmesg on SERVER#2 !!!

<4>NRFS dentry_delete(left/dmesg, d_flags=0 means do nothing)

<4>NRFS: create(4/1074112035, dmesg)

<4>NRFS: call SERVER#0 create dmesg
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100600 dev 308 ino 66aac
<4>NRFS: SERVER#0 reply create: 0

<4>NRFS: call SERVER#1 create dmesg
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100600 dev 308 ino 625b0
<4>NRFS: SERVER#1 reply create: 0

<4>NRFS: call SERVER#2 create dmesg
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100600 dev 308 ino 5a624
<4>NRFS: SERVER#2 reply create: 0

<4>NRFS: nrfs_fhget(left/dmesg, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(left/dmesg, intall fh from SERVER#1, 1)
<4>NRFS: nrfs_fhget(left/dmesg, intall fh from SERVER#2, 0)

<4>NRFS: nrfs_fhget(left/dmesg set fileid=1:537273776)
<4>NRFS: refresh_inode(4/537273776 ct=1)
<4>NRFS: __nrfs_fhget(4/537273776 ct=1)
<4>nrfs: write(left/dmesg(537273776), 2864@0)

<4>NRFS:  nrfs_updatepage(left/dmesg 2864@0, sync=0)

<4>NRFS:   find write request(4/537273776, c0369e00)

<4>NRFS:   create_write_request(left/dmesg, 0+2864)

<4>NRFS:   append_write_request(c62e2958, c6b30ea0)
<4>NRFS:   append_write_request(c62e2958, c6b30d80)
<4>NRFS:   append_write_request(c62e2958, c6b30900)

<4>NRFS:   21->1449 schedule_write_request(SERVER#0, async)
<4>NRFS:   21->1450 schedule_write_request(SERVER#1, async)
<4>NRFS:   21->1451 schedule_write_request(SERVER#2, async)

<4>nrfs: flush(4/1:537273776)

<4>NRFS:   21->1449 nrfs_wback_begin (left/dmesg, status=0 flags=0)
<4>NRFS:   21->1450 nrfs_wback_begin (left/dmesg, status=0 flags=0)
<4>NRFS:   21->1451 nrfs_wback_begin (left/dmesg, status=0 flags=0)

<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 1 mode 100600 dev 308 ino 66aac
<4>NRFS:   21->1449 nrfs_wback_result (left/dmesg, status=0, flags=100)
<4>NRFS:   remove_write_request(c62e2958, c6b30ea0)

<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 1 mode 100600 dev 308 ino 625b0
<4>NRFS:   21->1450 nrfs_wback_result (left/dmesg, status=0, flags=100)
<4>NRFS:   remove_write_request(c62e2958, c6b30d80)

<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 1 mode 100600 dev 308 ino 5a624
<4>NRFS:   21->1451 nrfs_wback_result (left/dmesg, status=0, flags=100)
<4>NRFS:   remove_write_request(c62e2958, c6b30900)

<4>NRFS:   21 all write back call for left/dmesg completed.

<4>NRFS: refresh_inode(4/537273776 ct=1)

<4>NRFS dentry_delete(left/dmesg, d_flags=0 means do nothing)

<4>NRFS: call SERVER#0 to lookup dmesg
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100600 dev 308 ino 66aac
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup dmesg
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100600 dev 308 ino 625b0
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup dmesg
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100600 dev 308 ino 5a624
<4>NRFS: SERVER#2 reply lookup: 0
```

```
<4>NRFS: refresh_inode(4/537273776 ct=1)

<4>NRFS: nrfs_notify_change(left/dmesg))

<4>NRFS: call SERVER#0 to setattr
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 1 mode 100644 dev 308 ino 66aac
<4>NRFS: SERVER#0 reply setattr

<4>NRFS: call SERVER#1 to setattr
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 1 mode 100644 dev 308 ino 625b0
<4>NRFS: SERVER#1 reply setattr

<4>NRFS: call SERVER#2 to setattr
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 1 mode 100644 dev 308 ino 5a624
<4>NRFS: SERVER#2 reply setattr

<4>NRFS: refresh_inode(4/537273776 ct=1)

<4>NRFS dentry_delete(left/dmesg, d_flags=0 means do nothing)


<4>NRFS: call SERVER#0 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa9
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ad
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 75e41
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074224705 ct=1)

<4>NRFS: nrfs_readdir(//root)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 4

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 4

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 4

<4>NRFS: call SERVER#0 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aaa
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ae
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 2b3a2
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1073918882 ct=1)

<4>NRFS dentry_delete(root/right, d_flags=0 means do nothing)

<4>NRFS: call SERVER#0 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aab
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625af
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 5a623
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074112035 ct=1)

<4>NRFS: call  getattr from SERVER#0
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40775 dev 308 ino 66aab
<4>NRFS: SERVER#0 reply getattr

<4>NRFS: call  getattr from SERVER#1
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40775 dev 308 ino 625af
<4>NRFS: SERVER#1 reply getattr

<4>NRFS: call  getattr from SERVER#2
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40775 dev 308 ino 5a623
<4>NRFS: SERVER#2 reply getattr
```

```
<4>NRFS: refresh_inode(4/1074112035 ct=1)

<4>NRFS: nrfs_readdir(//root)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS: call SERVER#0 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aab
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625af
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 5a623
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074112035 ct=1)

<4>NRFS: nrfs_readdir(root/left)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 3

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 3

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 3

<4>NRFS: call SERVER#0 to lookup dmesg
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 66aac
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup dmesg
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 625b0
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup dmesg
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 5a624
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/537273776 ct=1)

<4>NRFS dentry_delete(left/dmesg, d_flags=0 means do nothing)

<4>NRFS: nrfs_readdir(root/left)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS: call SERVER#0 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aaa
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ae
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 2b3a2
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1073918882 ct=1)

<4>NRFS: nrfs_readdir(root/right)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 2

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 2

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 2

<4>NRFS: nrfs_readdir(root/right)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0
```

```
<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS dentry_delete(root/right, d_flags=0 means do nothing)


<4>NRFS: call SERVER#0 to lookup testdir
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa4
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup testdir
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625a8
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup testdir
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 71f61
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074208609 ct=1)

<4>NRFS: lookup(testdir/cat)

<4>NRFS: call SERVER#0 to lookup cat
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa5
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup cat
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625a9
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup cat
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 274c2
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(testdir/cat, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(testdir/cat, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(testdir/cat, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(testdir/cat set fileid=2:1073902786)
<4>NRFS: refresh_inode(4/1073902786 ct=1)
<4>NRFS: __nrfs_fhget(4/1073902786 ct=1)

<4>NRFS dentry_delete(testdir/cat, d_flags=0 means do nothing)

<4>NRFS dentry_delete(testdir/cat, d_flags=0 means do nothing)

<4>NRFS: nrfs_readdir(testdir/cat)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 3

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 3

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 3

<4>NRFS: lookup(cat/README)

<4>NRFS: call SERVER#0 to lookup README
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 66aa7
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup README
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 625ab
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup README
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 274c3
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(cat/README, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(cat/README, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(cat/README, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(cat/README set fileid=2:1073902787)
<4>NRFS: refresh_inode(4/1073902787 ct=1)
<4>NRFS: __nrfs_fhget(4/1073902787 ct=1)

<4>NRFS dentry_delete(cat/README, d_flags=0 means do nothing)

<4>NRFS: nrfs_readdir(testdir/cat)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0
```

cat testdir/cat/README

```
<4>nrfs: read(cat/README, 512@0)

<4>NRFS: nrfs_readpage (c0373fb8, cat/README, 4096@0)
<4>NRFS: nrfs_readpage_async(c0373fb8, cat/README, 4096@0)

<4>NRFS: 17 read buffer(c52e8000) for SERVER#0 allocated
<4>NRFS: 17 read buffer(c5f2b000) for SERVER#1 allocated
<4>NRFS: 17 read buffer(c5f2a000) for SERVER#2 allocated

<4>NRFS: 17 executing async read call for SERVER#1(nrpc02)
<4>NRFS: 17 executing async read call for SERVER#0(nrpc01)
<4>NRFS: 17 executing async read call for SERVER#2(nrpc03)

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:5adc)
<4>NRFS: 17 callback(xid=1513) for page c62cb000 from server#0, result 0

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:5adc)
<4>NRFS: 17 callback(xid=1514) for page c62cb000 from server#2, result 0

<4>RPC:      checksum and data read reply OK status 0
<4>RPC:      checksum and data readres OK, csum(0:4:5adc), count 4096
<4>NRFS: 17 callback(xid=1512) for page c62cb000 from server#1, result 4096

<4>NRFS: 17 all read call for page c62cb000 completed
<4>NRFS: 17 validate data replied from each server
<4>NRFS: 17 update page c62cb000 with data reply from SERVER#1

<4>NRFS: refresh_inode(4/1073902787 ct=1)

<4>NRFS: 17 readpage_async done, 15 successful, 0 failures

<4>nrfs: read(cat/README, 512@512)
<4>nrfs: read(cat/README, 512@1024)
<4>nrfs: read(cat/README, 512@1536)
<4>nrfs: read(cat/README, 512@2048)

<4>NRFS: nrfs_readpage (c036c948, cat/README, 4096@4096)
<4>NRFS: nrfs_readpage_async(c036c948, cat/README, 4096@4096)

<4>NRFS: 18 read buffer(c52e8000) for SERVER#0 allocated
<4>NRFS: 18 read buffer(c5331000) for SERVER#1 allocated
<4>NRFS: 18 read buffer(c5330000) for SERVER#2 allocated

<4>NRFS: 18 executing async read call for SERVER#0(nrpc01)
<4>NRFS: 18 executing async read call for SERVER#1(nrpc02)
<4>NRFS: 18 executing async read call for SERVER#2(nrpc03)

<4>NRFS: nrfs_readpage (c036c920, cat/README, 4096@8192)
<4>NRFS: nrfs_readpage_async(c036c920, cat/README, 4096@8192)

<4>NRFS: 19 read buffer(c62dc000) for SERVER#0 allocated
<4>NRFS: 19 read buffer(c62dd000) for SERVER#1 allocated
<4>NRFS: 19 read buffer(c62ec000) for SERVER#2 allocated

<4>NRFS: 19 executing async read call for SERVER#0(nrpc01)
<4>NRFS: 19 executing async read call for SERVER#1(nrpc02)
<4>NRFS: 19 executing async read call for SERVER#2(nrpc03)

<4>NRFS: nrfs_readpage (c034d458, cat/README, 4096@12288)
<4>NRFS: nrfs_readpage_async(c034d458, cat/README, 4096@12288)

<4>NRFS: 20 read buffer(c62ed000) for SERVER#0 allocated
<4>NRFS: 20 read buffer(c5f2a000) for SERVER#1 allocated
<4>NRFS: 20 read buffer(c5f2b000) for SERVER#2 allocated

<4>NRFS: 20 executing async read call for SERVER#0(nrpc01)
<4>NRFS: 20 executing async read call for SERVER#1(nrpc02)
<4>NRFS: 20 executing async read call for SERVER#2(nrpc03)

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:e1a2)
<4>NRFS: 18 callback(xid=1516) for page c5fd5000 from server#1, result 0

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:e1a2)
<4>NRFS: 18 callback(xid=1517) for page c5fd5000 from server#2, result 0

<4>RPC:      checksum and data read reply OK status 0
<4>RPC:      checksum and data readres OK, csum(0:4:e1a2), count 4096
<4>NRFS: 18 callback(xid=1515) for page c5fd5000 from server#0, result 4096

<4>NRFS: 18 all read call for page c5fd5000 completed
<4>NRFS: 18 validate data replied from each server
<4>NRFS: 18 update page c5fd5000 with data reply from SERVER#0

<4>NRFS: refresh_inode(4/1073902787 ct=1)

<4>NRFS: 18 readpage_async done, 16 successful, 0 failures

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:8dfb)
<4>NRFS: 19 callback(xid=1519) for page c5fd4000 from server#1, result 0

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:8dfb)
<4>NRFS: 19 callback(xid=1520) for page c5fd4000 from server#2, result 0

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:12ad)
<4>NRFS: 20 callback(xid=1522) for page c534f000 from server#1, result 0
```

```
<4>RPC:       checksum read reply OK status 0
<4>RPC:       checksum readres OK, csum(0:4:12ad)
<4>NRFS: 20 callback(xid=1523) for page c534f000 from server#2, result 0

<4>RPC:       checksum and data read reply OK status 0
<4>RPC:       checksum and data readres OK, csum(0:4:8dfb), count 4096
<4>NRFS: 19 callback(xid=1518) for page c5fd4000 from server#0, result 4096

<4>NRFS: 19 all read call for page c5fd4000 completed
<4>NRFS: 19 validate data replied from each server
<4>NRFS: 19 update page c5fd4000 with data reply from SERVER#0

<4>NRFS: refresh_inode(4/1073902787 ct=1)

<4>NRFS: 19 readpage_async done, 17 successful, 0 failures

<4>RPC:       checksum and data read reply OK status 0
<4>RPC:       checksum and data readres OK, csum(0:4:12ad), count 1273
<4>NRFS: 20 callback(xid=1521) for page c534f000 from server#0, result 1273

<4>NRFS: 20 all read call for page c534f000 completed
<4>NRFS: 20 validate data replied from each server
<4>NRFS: 20 update page c534f000 with data reply from SERVER#0

<4>NRFS: refresh_inode(4/1073902787 ct=1)

<4>NRFS: 20 readpage_async done, 18 successful, 0 failures

<4>nrfs: read(cat/README, 512@2560)
<4>nrfs: read(cat/README, 512@3072)
<4>nrfs: read(cat/README, 512@3584)
<4>nrfs: read(cat/README, 512@4096)
<4>nrfs: read(cat/README, 512@4608)
<4>nrfs: read(cat/README, 512@5120)
<4>nrfs: read(cat/README, 512@5632)
<4>nrfs: read(cat/README, 512@6144)
<4>nrfs: read(cat/README, 512@6656)
<4>nrfs: read(cat/README, 512@7168)
<4>nrfs: read(cat/README, 512@7680)
<4>nrfs: read(cat/README, 512@8192)
<4>nrfs: read(cat/README, 512@8704)
<4>nrfs: read(cat/README, 512@9216)
<4>nrfs: read(cat/README, 512@9728)
<4>nrfs: read(cat/README, 512@10240)
<4>nrfs: read(cat/README, 512@10752)
<4>nrfs: read(cat/README, 512@11264)
<4>nrfs: read(cat/README, 512@11776)
<4>nrfs: read(cat/README, 512@12288)
<4>nrfs: read(cat/README, 512@12800)
<4>nrfs: read(cat/README, 512@13312)
<4>nrfs: read(cat/README, 512@13561)

<4>nrfs: flush(4/2:1073902787)

<4>NRFS dentry_delete(cat/README, d_flags=0 means do nothing)
```

```
<4>NRFS: call  getattr from SERVER#0
<4>RPC:       attrstat status 0
<4>RPC:       attrstat OK type 2 mode 40777 dev 308 ino 624f4
<4>NRFS: SERVER#0 reply getattr

<4>NRFS: call  getattr from SERVER#1
<4>RPC:       attrstat status 0
<4>RPC:       attrstat OK type 2 mode 40777 dev 308 ino 5a70d
<4>NRFS: SERVER#1 reply getattr

<4>NRFS: call  getattr from SERVER#2
<4>RPC:       attrstat status 0
<4>RPC:       attrstat OK type 2 mode 40777 dev 308 ino 75e43
<4>NRFS: SERVER#2 reply getattr

<4>NRFS: refresh_inode(4/402676 ct=1)

<4>NRFS: lookup(//tests)

<4>NRFS: call SERVER#0 to lookup tests
<4>RPC:       diropres status 2
<4>NRFS: SERVER#0 reply lookup: -2
<1>NRFS: Can't find tests on SERVER#0 !!!

<4>NRFS: call SERVER#1 to lookup tests
<4>RPC:       diropres status 2
<4>NRFS: SERVER#1 reply lookup: -2
<1>NRFS: Can't find tests on SERVER#1 !!!

<4>NRFS: call SERVER#2 to lookup tests
<4>RPC:       diropres status 2
<4>NRFS: SERVER#2 reply lookup: -2
<1>NRFS: Can't find tests on SERVER#2 !!!

<4>NRFS dentry_delete(//tests, d_flags=0 means do nothing)
<4>NRFS dentry_delete(//tests, d_flags=0 means do nothing)
<4>NRFS dentry_delete(//tests, d_flags=0 means do nothing)

<4>NRFS: symlink(4/402676, tests, /home/nrfs/tests)

<4>NRFS: call SERVER#0 to symlink tests -> /home/nrfs/tests
<4>NRFS: SERVER#0 reply symlink: 0

<4>NRFS: call SERVER#1 to symlink tests -> /home/nrfs/tests
<4>NRFS: SERVER#1 reply symlink: 0
```

ln -s ~/tests tests
user : nrfs
HOMEDIR : /home/nrfs

```
<4>NRFS: call SERVER#2 to symlink tests -> /home/nrfs/tests
<4>NRFS: SERVER#2 reply symlink: 0

<4>NRFS dentry_delete(//tests, d_flags=0 means do nothing)


<4>NRFS: lookup(//tests)

<4>NRFS: call SERVER#0 to lookup tests
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 5 mode 120777 dev 308 ino 62c0b
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup tests
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 5 mode 120777 dev 308 ino 5a70e
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup tests
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 5 mode 120777 dev 308 ino 75e42
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(//tests, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(//tests, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(//tests, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(//tests set fileid=2:1074224706)
<4>NRFS: refresh_inode(4/1074224706 ct=1)
<4>NRFS: __nrfs_fhget(4/1074224706 ct=1)

<4>NRFS dentry_delete(//tests, d_flags=0 means do nothing)

<4>nrfs: readlink(//tests)

<4>NRFS: call SERVER#0 readlink
<4>NRFS: SERVER#0 reply readlink: 0

<4>NRFS: call SERVER#1 readlink
<4>NRFS: SERVER#1 reply readlink: 0

<4>NRFS: call SERVER#2 readlink
<4>NRFS: SERVER#2 reply readlink: 0

<4>NRFS dentry_delete(//tests, d_flags=0 means do nothing)


<4>NRFS: call SERVER#0 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa9
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ad
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 75e41
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074224705 ct=1)

<4>NRFS: nrfs_readdir(//root)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 4

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 4

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 4

<4>NRFS: call SERVER#0 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aaa
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ae
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 2b3a2
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1073918882 ct=1)

<4>NRFS dentry_delete(root/right, d_flags=0 means do nothing)

<4>NRFS: call SERVER#0 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aab
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625af
<4>NRFS: SERVER#1 reply lookup: 0
```

```
<4>NRFS: call SERVER#2 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 5a623
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074112035 ct=1)

<4>NRFS: nrfs_readdir(//root)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS: call SERVER#0 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aab
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625af
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 5a623
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074112035 ct=1)

<4>NRFS: nrfs_readdir(root/left)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 3

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 3

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 3

<4>NRFS: call SERVER#0 to lookup dmesg
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 66aac
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup dmesg
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 625b0
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup dmesg
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 5a624
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/537273776 ct=1)

<4>NRFS dentry_delete(left/dmesg, d_flags=0 means do nothing)

<4>NRFS: nrfs_readdir(root/left)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS: call SERVER#0 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aaa
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ae
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 2b3a2
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1073918882 ct=1)

<4>NRFS: nrfs_readdir(root/right)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 2

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 2

<4>NRFS: call SERVER#2 to readdir 0
```

```
<4>NRFS: SERVER#2 reply readdir: 2

<4>NRFS: nrfs_readdir(root/right)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS dentry_delete(root/right, d_flags=0 means do nothing)


<4>NRFS: call SERVER#0 to lookup dmesg
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 66aac
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup dmesg
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 625b0
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup dmesg
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 5a624
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/537273776 ct=1)

<4>NRFS dentry_delete(left/dmesg, d_flags=0 means do nothing)
<4>NRFS dentry_delete(left/dmesg, d_flags=0 means do nothing)

<4>NRFS: unlink(4/1074112035, dmesg)

<4>NRFS:    silly-rename(left/dmesg, ct=1)
<4>NRFS:    safe_remove(left/dmesg, 537273776)
<4>NRFS: put_inode(4/1:537273776)
<4>NRFS: delete_inode(4/1:537273776)

<4>NRFS: call SERVER#0 to remove dmesg
<4>NRFS: SERVER#0 reply remove: 0

<4>NRFS: call SERVER#1 to remove dmesg
<4>NRFS: SERVER#1 reply remove: 0

<4>NRFS: call SERVER#2 to remove dmesg
<4>NRFS: SERVER#2 reply remove: 0

<4>NRFS dentry_delete(left/dmesg, d_flags=0 means do nothing)
```

rm root/left/dmesg

```
<4>NRFS: call SERVER#0 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa9
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ad
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 75e41
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074224705 ct=1)

<4>NRFS: nrfs_readdir(//root)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 4

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 4

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 4

<4>NRFS: call SERVER#0 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aaa
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ae
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 2b3a2
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1073918882 ct=1)

<4>NRFS dentry_delete(root/right, d_flags=0 means do nothing)

<4>NRFS: call SERVER#0 to lookup left
```

```
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aab
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625af
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 5a623
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074112035 ct=1)

<4>NRFS: call  getattr from SERVER#0
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40775 dev 308 ino 66aab
<4>NRFS: SERVER#0 reply getattr

<4>NRFS: call  getattr from SERVER#1
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40775 dev 308 ino 625af
<4>NRFS: SERVER#1 reply getattr

<4>NRFS: call  getattr from SERVER#2
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40775 dev 308 ino 5a623
<4>NRFS: SERVER#2 reply getattr

<4>NRFS: refresh_inode(4/1074112035 ct=1)

<4>NRFS: nrfs_readdir(//root)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS: call SERVER#0 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aab
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625af
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 5a623
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074112035 ct=1)

<4>NRFS: nrfs_readdir(root/left)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 2

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 2

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 2

<4>NRFS: nrfs_readdir(root/left)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS: call SERVER#0 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aaa
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625ae
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 2b3a2
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1073918882 ct=1)

<4>NRFS: nrfs_readdir(root/right)

<4>NRFS: call SERVER#0 to readdir 0
```

```
<4>NRFS: SERVER#0 reply readdir: 2

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 2

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 2

<4>NRFS: nrfs_readdir(root/right)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS dentry_delete(root/right, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/2:1074208611)
<4>NRFS: delete_inode(4/2:1074208611)

<4>NRFS dentry_delete(testdir/cp, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/2:1074208610)
<4>NRFS: delete_inode(4/2:1074208610)

<4>NRFS: put_inode(4/2:1073902787)
<4>NRFS: delete_inode(4/2:1073902787)

<4>NRFS dentry_delete(testdir/cat, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/2:1073902786)
<4>NRFS: delete_inode(4/2:1073902786)

<4>NRFS dentry_delete(//testdir, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/2:1074208609)
<4>NRFS: delete_inode(4/2:1074208609)

<4>NRFS: put_inode(4/2:1074224706)
<4>NRFS: delete_inode(4/2:1074224706)

<4>NRFS dentry_delete(root/left, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/2:1074112035)
<4>NRFS: delete_inode(4/2:1074112035)

<4>NRFS: put_inode(4/2:1073918882)
<4>NRFS: delete_inode(4/2:1073918882)

<4>NRFS dentry_delete(//root, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/2:1074224705)
<4>NRFS: delete_inode(4/2:1074224705)

<4>NRFS dentry_delete(///, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/0:402676)
<4>NRFS: delete_inode(4/0:402676)
```

図3.3.2-2　NRFSクライアントの多重サーバ対応の検証（NRFSクライアントにおける応答ログの一例）

```
<4>nfsd: GETATTR  776/402676
<4>nfsd: GETATTR  776/402676
<4>nfsd: READDIR  776/402676 4096 bytes at 0
<4>nfsd: READDIR  776/402676 4096 bytes at 4096
<4>nfsd: GETATTR  776/402676


<4>nfsd: LOOKUP   776/402676 root
<4>nfsd: MKDIR    00000000 root


<4>nfsd: LOOKUP   776/402676 root
<4>nfsd: LOOKUP   776/420521 right
<4>nfsd: MKDIR    00000000 right


<4>nfsd: LOOKUP   776/420521 left
<4>nfsd: MKDIR    00000000 left


<4>nfsd: LOOKUP   776/420521 right
<4>nfsd: LOOKUP   776/420522 righet
<4>nfsd: MKDIR    00000000 righet


<4>nfsd: LOOKUP   776/420522 left
<4>nfsd: MKDIR    00000000 left


<4>nfsd: LOOKUP   776/402676 root
<4>nfsd: READDIR  776/420521 4096 bytes at 0

<4>nfsd: LOOKUP   776/420521 right
<4>nfsd: LOOKUP   776/420521 left
<4>nfsd: READDIR  776/420521 4096 bytes at 4096

<4>nfsd: LOOKUP   776/420521 left
<4>nfsd: READDIR  776/420523 4096 bytes at 0
<4>nfsd: READDIR  776/420523 4096 bytes at 4096

<4>nfsd: LOOKUP   776/420521 right
<4>nfsd: READDIR  776/420522 4096 bytes at 0
<4>nfsd: READDIR  776/420522 4096 bytes at 4096

<4>nfsd: LOOKUP   776/420522 righet
<4>nfsd: LOOKUP   776/420522 left
<4>nfsd: READDIR  776/420522 4096 bytes at 4096

<4>nfsd: LOOKUP   776/420522 left
<4>nfsd: READDIR  776/420525 4096 bytes at 0
<4>nfsd: READDIR  776/420525 4096 bytes at 4096

<4>nfsd: LOOKUP   776/420522 righet
<4>nfsd: READDIR  776/420524 4096 bytes at 0
<4>nfsd: READDIR  776/420524 4096 bytes at 4096


<4>nfsd: LOOKUP   776/402676 root
<4>nfsd: LOOKUP   776/420522 righet
<4>nfsd: RMDIR    00000000 righet


<4>nfsd: LOOKUP   776/420522 left
<4>nfsd: RMDIR    00000000 left


<4>nfsd: LOOKUP   776/402676 root
<4>nfsd: READDIR  776/420521 4096 bytes at 0

<4>nfsd: LOOKUP   776/420521 right
<4>nfsd: GETATTR  776/420522

<4>nfsd: LOOKUP   776/420521 left
<4>nfsd: READDIR  776/420521 4096 bytes at 4096

<4>nfsd: LOOKUP   776/420521 left
<4>nfsd: READDIR  776/420523 4096 bytes at 0
<4>nfsd: READDIR  776/420523 4096 bytes at 4096

<4>nfsd: LOOKUP   776/420521 right
<4>nfsd: READDIR  776/420522 4096 bytes at 0
<4>nfsd: READDIR  776/420522 4096 bytes at 4096

<4>nfsd: GETATTR  776/402676

<4>nfsd: GETATTR  776/402676


<4>nfsd: LOOKUP   776/402676 testdir
<4>nfsd: LOOKUP   776/420516 cp
<4>nfsd: READDIR  776/420518 4096 bytes at 0

<4>nfsd: LOOKUP   776/420518 MAINTAINERS
<4>nfsd: READDIR  776/420518 4096 bytes at 4096

<4>nfsd: CHECKSUM READ : 0-4096(4096)-4096 byte/SUM : 0-4-48270000

<4>nfsd: CHECKSUM READ : 4096-4096(4096)-4096 byte/SUM : 0-4-381f0000

<4>nfsd: CHECKSUM READ : 8192-4096(4096)-4096 byte/SUM : 0-4-1f0f0000

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 12288-4096(4096)-4096 byte/SUM : 0-4-e7790000
<4>nfsd: <<------------------<<
```

mkdir root

mkdir root/right

mkdir root/left

mkdir root/right/righet

mkdir root/right/left

rmdir root/right/righet

rmdir root/right/left

cp testdir/cp/MAINTAINERS root/left/dmesg

```
<4>nfsd: CHECKSUM READ : 16384-4096(4096)-4096 byte/SUM : 0-4-641f0000

<4>nfsd: CHECKSUM READ : 20480-1108(4096)-4096 byte/SUM : 0-4-46570000

<4>nfsd: LOOKUP   776/420521 left
<4>nfsd: LOOKUP   776/420523 dmesg
<4>nfsd: CREATE   776/420523 dmesg
<4>nfsd: WRITE    776/420524 2864 bytes at 0
<4>nfsd: LOOKUP   776/420523 dmesg
<4>nfsd: SETATTR  776/420524, valid=1, size=0


<4>nfsd: LOOKUP   776/402676 root
<4>nfsd: READDIR  776/420521 4096 bytes at 0

<4>nfsd: LOOKUP   776/420521 right
<4>nfsd: LOOKUP   776/420521 left
<4>nfsd: GETATTR  776/420523
<4>nfsd: READDIR  776/420521 4096 bytes at 4096

<4>nfsd: LOOKUP   776/420521 left
<4>nfsd: READDIR  776/420523 4096 bytes at 0

<4>nfsd: LOOKUP   776/420523 dmesg
<4>nfsd: READDIR  776/420523 4096 bytes at 4096

<4>nfsd: LOOKUP   776/420521 right
<4>nfsd: READDIR  776/420522 4096 bytes at 0
<4>nfsd: READDIR  776/420522 4096 bytes at 4096
```

cat testdir/cat/README

```
<4>nfsd: LOOKUP   776/402676 testdir
<4>nfsd: LOOKUP   776/420516 cat
<4>nfsd: READDIR  776/420517 4096 bytes at 0

<4>nfsd: LOOKUP   776/420517 README
<4>nfsd: READDIR  776/420517 4096 bytes at 4096

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 0-4096(4096)-4096 byte/SUM : 0-4-dc5a0000
<4>nfsd: <<-----------------<<

<4>nfsd: CHECKSUM READ : 4096-4096(4096)-4096 byte/SUM : 0-4-a2e10000

<4>nfsd: CHECKSUM READ : 8192-4096(4096)-4096 byte/SUM : 0-4-fb8d0000

<4>nfsd: CHECKSUM READ : 12288-1273(4096)-4096 byte/SUM : 0-4-ad120000
```

ln -s ~/tests tests
user : nrfs
HOMEDIR : /home/nrfs

```
<4>nfsd: GETATTR  776/402676
<4>nfsd: LOOKUP   776/402676 tests
<4>nfsd: SYMLINK  00000000 tests -> /home/nrfs/tests
```

```
<4>nfsd: LOOKUP   776/402676 tests
<4>nfsd: READLINK 00000000

<4>nfsd: LOOKUP   776/402676 root
<4>nfsd: READDIR  776/420521 4096 bytes at 0

<4>nfsd: LOOKUP   776/420521 right
<4>nfsd: LOOKUP   776/420521 left
<4>nfsd: READDIR  776/420521 4096 bytes at 4096

<4>nfsd: LOOKUP   776/420521 left
<4>nfsd: READDIR  776/420523 4096 bytes at 0

<4>nfsd: LOOKUP   776/420523 dmesg
<4>nfsd: READDIR  776/420523 4096 bytes at 4096

<4>nfsd: LOOKUP   776/420521 right
<4>nfsd: READDIR  776/420522 4096 bytes at 0
<4>nfsd: READDIR  776/420522 4096 bytes at 4096
```

rm root/left/dmesg

```
<4>nfsd: LOOKUP   776/420523 dmesg
<4>nfsd: REMOVE   00000000 dmesg


<4>nfsd: LOOKUP   776/402676 root
<4>nfsd: READDIR  776/420521 4096 bytes at 0

<4>nfsd: LOOKUP   776/420521 right
<4>nfsd: LOOKUP   776/420521 left
<4>nfsd: GETATTR  776/420523
<4>nfsd: READDIR  776/420521 4096 bytes at 4096

<4>nfsd: LOOKUP   776/420521 left
<4>nfsd: READDIR  776/420523 4096 bytes at 0
<4>nfsd: READDIR  776/420523 4096 bytes at 4096

<4>nfsd: LOOKUP   776/420521 right
<4>nfsd: READDIR  776/420522 4096 bytes at 0
<4>nfsd: READDIR  776/420522 4096 bytes at 4096
```

図3.3.2-3　NRFSクライアントの多重サーバ対応の検証
（NRFSサーバにおける応答ログの一例、NRFSサーバ１）

```
<4>nfsd: GETATTR  776/370445
<4>nfsd: GETATTR  776/370445
<4>nfsd: READDIR  776/370445 4096 bytes at 0
<4>nfsd: READDIR  776/370445 4096 bytes at 4096
<4>nfsd: GETATTR  776/370445
```

mkdir root

```
<4>nfsd: LOOKUP   776/370445 root
<4>nfsd: MKDIR    00000000 root
```

mkdir root/right

```
<4>nfsd: LOOKUP   776/370445 root
<4>nfsd: LOOKUP   776/402861 right
<4>nfsd: MKDIR    00000000 right
```

mkdir root/left

```
<4>nfsd: LOOKUP   776/402861 left
<4>nfsd: MKDIR    00000000 left
```

mkdir root/right/righet

```
<4>nfsd: LOOKUP   776/402861 right
<4>nfsd: LOOKUP   776/402862 righet
<4>nfsd: MKDIR    00000000 righet
```

mkdir root/right/left

```
<4>nfsd: LOOKUP   776/402862 left
<4>nfsd: MKDIR    00000000 left
```

```
<4>nfsd: LOOKUP   776/370445 root
<4>nfsd: READDIR  776/402861 4096 bytes at 0

<4>nfsd: LOOKUP   776/402861 right
<4>nfsd: LOOKUP   776/402861 left
<4>nfsd: READDIR  776/402861 4096 bytes at 4096

<4>nfsd: LOOKUP   776/402861 left
<4>nfsd: READDIR  776/402863 4096 bytes at 0
<4>nfsd: READDIR  776/402863 4096 bytes at 4096

<4>nfsd: LOOKUP   776/402861 right
<4>nfsd: READDIR  776/402862 4096 bytes at 0
<4>nfsd: READDIR  776/402862 4096 bytes at 4096

<4>nfsd: LOOKUP   776/402862 righet
<4>nfsd: LOOKUP   776/402862 left
<4>nfsd: READDIR  776/402862 4096 bytes at 4096

<4>nfsd: LOOKUP   776/402862 left
<4>nfsd: READDIR  776/402865 4096 bytes at 0
<4>nfsd: READDIR  776/402865 4096 bytes at 4096

<4>nfsd: LOOKUP   776/402862 righet
<4>nfsd: READDIR  776/402864 4096 bytes at 0
<4>nfsd: READDIR  776/402864 4096 bytes at 4096
```

rmdir root/right/righet

```
<4>nfsd: LOOKUP   776/370445 root
<4>nfsd: LOOKUP   776/402862 righet
<4>nfsd: RMDIR    00000000 righet
```

rmdir root/right/left

```
<4>nfsd: LOOKUP   776/402862 left
<4>nfsd: RMDIR    00000000 left
```

```
<4>nfsd: LOOKUP   776/370445 root
<4>nfsd: READDIR  776/402861 4096 bytes at 0

<4>nfsd: LOOKUP   776/402861 right
<4>nfsd: GETATTR  776/402862

<4>nfsd: LOOKUP   776/402861 left
<4>nfsd: READDIR  776/402861 4096 bytes at 4096

<4>nfsd: LOOKUP   776/402861 left
<4>nfsd: READDIR  776/402863 4096 bytes at 0
<4>nfsd: READDIR  776/402863 4096 bytes at 4096

<4>nfsd: LOOKUP   776/402861 right
<4>nfsd: READDIR  776/402862 4096 bytes at 0
<4>nfsd: READDIR  776/402862 4096 bytes at 4096

<4>nfsd: GETATTR  776/370445

<4>nfsd: GETATTR  776/370445
```

cp testdir/cp/MAINTAINERS root/left/dmesg

```
<4>nfsd: LOOKUP   776/370445 testdir
<4>nfsd: LOOKUP   776/402856 cp
<4>nfsd: READDIR  776/402858 4096 bytes at 0

<4>nfsd: LOOKUP   776/402858 MAINTAINERS
<4>nfsd: READDIR  776/402858 4096 bytes at 4096

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 0-4096(4096)-4096 byte/SUM : 0-4-48270000
<4>nfsd: <<-----------------<<

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 4096-4096(4096)-4096 byte/SUM : 0-4-381f0000
<4>nfsd: <<-----------------<<

<4>nfsd: >>- CHECKSUM ONLY ->>
```

```
<4>nfsd: CHECKSUM READ : 8192-4096(4096)-4096 byte/SUM : 0-4-1f0f0000
<4>nfsd: <<-----------------<<

<4>nfsd: CHECKSUM READ : 12288-4096(4096)-4096 byte/SUM : 0-4-e7790000

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 16384-4096(4096)-4096 byte/SUM : 0-4-641f0000
<4>nfsd: <<-----------------<<

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 20480-1108(4096)-4096 byte/SUM : 0-4-46570000
<4>nfsd: <<-----------------<<

<4>nfsd: LOOKUP   776/402861 left
<4>nfsd: LOOKUP   776/402863 dmesg
<4>nfsd: CREATE   776/402863 dmesg
<4>nfsd: WRITE    776/402864 2864 bytes at 0
<4>nfsd: LOOKUP   776/402863 dmesg
<4>nfsd: SETATTR  776/402864, valid=1, size=0


<4>nfsd: LOOKUP   776/370445 root
<4>nfsd: READDIR  776/402861 4096 bytes at 0

<4>nfsd: LOOKUP   776/402861 right
<4>nfsd: LOOKUP   776/402861 left
<4>nfsd: GETATTR  776/402863
<4>nfsd: READDIR  776/402861 4096 bytes at 4096

<4>nfsd: LOOKUP   776/402861 left
<4>nfsd: READDIR  776/402863 4096 bytes at 0

<4>nfsd: LOOKUP   776/402863 dmesg
<4>nfsd: READDIR  776/402863 4096 bytes at 4096

<4>nfsd: LOOKUP   776/402861 right
<4>nfsd: READDIR  776/402862 4096 bytes at 0
<4>nfsd: READDIR  776/402862 4096 bytes at 4096
```

cat testdir/cat/README

```
<4>nfsd: LOOKUP   776/370445 testdir
<4>nfsd: LOOKUP   776/402856 cat
<4>nfsd: READDIR  776/402857 4096 bytes at 0

<4>nfsd: LOOKUP   776/402857 README
<4>nfsd: READDIR  776/402857 4096 bytes at 4096

<4>nfsd: CHECKSUM READ : 0-4096(4096)-4096 byte/SUM : 0-4-dc5a0000

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 4096-4096(4096)-4096 byte/SUM : 0-4-a2e10000
<4>nfsd: <<-----------------<<

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 8192-4096(4096)-4096 byte/SUM : 0-4-fb8d0000
<4>nfsd: <<-----------------<<

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 12288-1273(4096)-4096 byte/SUM : 0-4-ad120000
<4>nfsd: <<-----------------<<
```

ln -s ~/tests tests
user : nrfs
HOMEDIR : /home/nrfs

```
<4>nfsd: GETATTR  776/370445
<4>nfsd: LOOKUP   776/370445 tests
<4>nfsd: SYMLINK  00000000 tests -> /home/nrfs/tests


<4>nfsd: LOOKUP   776/370445 tests
<4>nfsd: READLINK 00000000

<4>nfsd: LOOKUP   776/370445 root
<4>nfsd: READDIR  776/402861 4096 bytes at 0

<4>nfsd: LOOKUP   776/402861 right
<4>nfsd: LOOKUP   776/402861 left
<4>nfsd: READDIR  776/402861 4096 bytes at 4096

<4>nfsd: LOOKUP   776/402861 left
<4>nfsd: READDIR  776/402863 4096 bytes at 0

<4>nfsd: LOOKUP   776/402863 dmesg
<4>nfsd: READDIR  776/402863 4096 bytes at 4096

<4>nfsd: LOOKUP   776/402861 right
<4>nfsd: READDIR  776/402862 4096 bytes at 0
<4>nfsd: READDIR  776/402862 4096 bytes at 4096
```

rm root/left/dmesg

```
<4>nfsd: LOOKUP   776/402863 dmesg
<4>nfsd: REMOVE   00000000 dmesg


<4>nfsd: LOOKUP   776/370445 root
<4>nfsd: READDIR  776/402861 4096 bytes at 0

<4>nfsd: LOOKUP   776/402861 right
<4>nfsd: LOOKUP   776/402861 left
<4>nfsd: GETATTR  776/402863
<4>nfsd: READDIR  776/402861 4096 bytes at 4096

<4>nfsd: LOOKUP   776/402861 left
<4>nfsd: READDIR  776/402863 4096 bytes at 0
<4>nfsd: READDIR  776/402863 4096 bytes at 4096
```

```
<4>nfsd: LOOKUP   776/402861 right
<4>nfsd: READDIR  776/402862 4096 bytes at 0
<4>nfsd: READDIR  776/402862 4096 bytes at 4096
```

図3.3.2-4　NRFSクライアントの多重サーバ対応の検証
（NRFSサーバにおける応答ログの一例、NRFSサーバ２）

```
<4>nfsd: GETATTR  776/482883
<4>nfsd: GETATTR  776/482883
<4>nfsd: READDIR  776/482883 4096 bytes at 0
<4>nfsd: READDIR  776/482883 4096 bytes at 4096
<4>nfsd: GETATTR  776/482883


<4>nfsd: LOOKUP   776/482883 root
<4>nfsd: MKDIR    00000000 root
```

mkdir root

```
<4>nfsd: LOOKUP   776/482883 root
<4>nfsd: LOOKUP   776/482881 right
<4>nfsd: MKDIR    00000000 right
```

mkdir root/right

```
<4>nfsd: LOOKUP   776/482881 left
<4>nfsd: MKDIR    00000000 left
```

mkdir root/left

```
<4>nfsd: LOOKUP   776/482881 right
<4>nfsd: LOOKUP   776/177058 righet
<4>nfsd: MKDIR    00000000 righet
```

mkdir root/right/righet

```
<4>nfsd: LOOKUP   776/177058 left
<4>nfsd: MKDIR    00000000 left


<4>nfsd: LOOKUP   776/482883 root
<4>nfsd: READDIR  776/482881 4096 bytes at 0

<4>nfsd: LOOKUP   776/482881 right
<4>nfsd: LOOKUP   776/482881 left
<4>nfsd: READDIR  776/482881 4096 bytes at 4096

<4>nfsd: LOOKUP   776/482881 left
<4>nfsd: READDIR  776/370211 4096 bytes at 0
<4>nfsd: READDIR  776/370211 4096 bytes at 4096

<4>nfsd: LOOKUP   776/482881 right
<4>nfsd: READDIR  776/177058 4096 bytes at 0

<4>nfsd: LOOKUP   776/177058 righet
<4>nfsd: LOOKUP   776/177058 left
<4>nfsd: READDIR  776/177058 4096 bytes at 4096

<4>nfsd: LOOKUP   776/177058 left
<4>nfsd: READDIR  776/418499 4096 bytes at 0
<4>nfsd: READDIR  776/418499 4096 bytes at 4096

<4>nfsd: LOOKUP   776/177058 righet
<4>nfsd: READDIR  776/402401 4096 bytes at 0
<4>nfsd: READDIR  776/402401 4096 bytes at 4096


<4>nfsd: LOOKUP   776/482883 root
<4>nfsd: LOOKUP   776/177058 righet
<4>nfsd: RMDIR    00000000 righet
```

rmdir root/right/righet

```
<4>nfsd: LOOKUP   776/177058 left
<4>nfsd: RMDIR    00000000 left


<4>nfsd: LOOKUP   776/482883 root
<4>nfsd: READDIR  776/482881 4096 bytes at 0

<4>nfsd: LOOKUP   776/482881 right
<4>nfsd: GETATTR  776/177058

<4>nfsd: LOOKUP   776/482881 left
<4>nfsd: READDIR  776/482881 4096 bytes at 4096

<4>nfsd: LOOKUP   776/482881 left
<4>nfsd: READDIR  776/370211 4096 bytes at 0
<4>nfsd: READDIR  776/370211 4096 bytes at 4096

<4>nfsd: LOOKUP   776/482881 right
<4>nfsd: READDIR  776/177058 4096 bytes at 0
<4>nfsd: READDIR  776/177058 4096 bytes at 4096

<4>nfsd: GETATTR  776/482883

<4>nfsd: GETATTR  776/482883


<4>nfsd: LOOKUP   776/482883 testdir
<4>nfsd: LOOKUP   776/466785 cp
<4>nfsd: READDIR  776/466786 4096 bytes at 0

<4>nfsd: LOOKUP   776/466786 MAINTAINERS
<4>nfsd: READDIR  776/466786 4096 bytes at 4096

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 0-4096(4096)-4096 byte/SUM : 0-4-48270000
<4>nfsd: <<-----------------<<

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 4096-4096(4096)-4096 byte/SUM : 0-4-381f0000
<4>nfsd: <<-----------------<<

<4>nfsd: >>- CHECKSUM ONLY ->>
```

cp testdir/cp/MAINTAINERS
root/left/dmesg

```
<4>nfsd: CHECKSUM READ : 8192-4096(4096)-4096 byte/SUM : 0-4-1f0f0000
<4>nfsd: <<-----------------<<

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 12288-4096(4096)-4096 byte/SUM : 0-4-e7790000
<4>nfsd: <<-----------------<<

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 16384-4096(4096)-4096 byte/SUM : 0-4-641f0000
<4>nfsd: <<-----------------<<

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 20480-1108(4096)-4096 byte/SUM : 0-4-46570000
<4>nfsd: <<-----------------<<

<4>nfsd: LOOKUP   776/482881 left
<4>nfsd: LOOKUP   776/370211 dmesg
<4>nfsd: CREATE   776/370211 dmesg
<4>nfsd: WRITE    776/370212 2864 bytes at 0
<4>nfsd: LOOKUP   776/370211 dmesg
<4>nfsd: SETATTR  776/370212, valid=1, size=0


<4>nfsd: LOOKUP   776/482883 root
<4>nfsd: READDIR  776/482881 4096 bytes at 0

<4>nfsd: LOOKUP   776/482881 right
<4>nfsd: LOOKUP   776/482881 left
<4>nfsd: GETATTR  776/370211
<4>nfsd: READDIR  776/482881 4096 bytes at 4096

<4>nfsd: LOOKUP   776/482881 left
<4>nfsd: READDIR  776/370211 4096 bytes at 0

<4>nfsd: LOOKUP   776/370211 dmesg
<4>nfsd: READDIR  776/370211 4096 bytes at 4096

<4>nfsd: LOOKUP   776/482881 right
<4>nfsd: READDIR  776/177058 4096 bytes at 0
<4>nfsd: READDIR  776/177058 4096 bytes at 4096
```

cat testdir/cat/README

```
<4>nfsd: LOOKUP   776/482883 testdir
<4>nfsd: LOOKUP   776/466785 cat
<4>nfsd: READDIR  776/160962 4096 bytes at 0

<4>nfsd: LOOKUP   776/160962 README
<4>nfsd: READDIR  776/160962 4096 bytes at 4096

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 0-4096(4096)-4096 byte/SUM : 0-4-dc5a0000
<4>nfsd: <<-----------------<<

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 4096-4096(4096)-4096 byte/SUM : 0-4-a2e10000
<4>nfsd: <<-----------------<<

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 8192-4096(4096)-4096 byte/SUM : 0-4-fb8d0000
<4>nfsd: <<-----------------<<

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 12288-1273(4096)-4096 byte/SUM : 0-4-ad120000
<4>nfsd: <<-----------------<<
```

ln -s ~/tests tests
user : nrfs
HOMEDIR : /home/nrfs

```
<4>nfsd: GETATTR  776/482883
<4>nfsd: LOOKUP   776/482883 tests
<4>nfsd: SYMLINK  00000000 tests -> /home/nrfs/tests


<4>nfsd: LOOKUP   776/482883 tests
<4>nfsd: READLINK 00000000

<4>nfsd: LOOKUP   776/482883 root
<4>nfsd: READDIR  776/482881 4096 bytes at 0

<4>nfsd: LOOKUP   776/482881 right
<4>nfsd: LOOKUP   776/482881 left
<4>nfsd: READDIR  776/482881 4096 bytes at 4096

<4>nfsd: LOOKUP   776/482881 left
<4>nfsd: READDIR  776/370211 4096 bytes at 0

<4>nfsd: LOOKUP   776/370211 dmesg
<4>nfsd: READDIR  776/370211 4096 bytes at 4096

<4>nfsd: LOOKUP   776/482881 right
<4>nfsd: READDIR  776/177058 4096 bytes at 0
<4>nfsd: READDIR  776/177058 4096 bytes at 4096
```

rm root/left/dmesg

```
<4>nfsd: LOOKUP   776/370211 dmesg
<4>nfsd: REMOVE   00000000 dmesg


<4>nfsd: LOOKUP   776/482883 root
<4>nfsd: READDIR  776/482881 4096 bytes at 0

<4>nfsd: LOOKUP   776/482881 right
<4>nfsd: LOOKUP   776/482881 left
<4>nfsd: GETATTR  776/370211
<4>nfsd: READDIR  776/482881 4096 bytes at 4096
```

```
<4>nfsd: LOOKUP   776/482881 left
<4>nfsd: READDIR  776/370211 4096 bytes at 0
<4>nfsd: READDIR  776/370211 4096 bytes at 4096

<4>nfsd: LOOKUP   776/482881 right
<4>nfsd: READDIR  776/177058 4096 bytes at 0
<4>nfsd: READDIR  776/177058 4096 bytes at 4096
```

図3.3.2-5　NRFSクライアントの多重サーバ対応の検証
（NRFSサーバにおける応答ログの一例、NRFSサーバ３）

```
<4>NRFS: lookup(//root)

<4>NRFS: call SERVER#0 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa4
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625a8
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 6e08d
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(//root, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(//root, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(//root, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(//root set fileid=2:1074192525)
<4>NRFS: refresh_inode(4/1074192525 ct=1)
<4>NRFS: __nrfs_fhget(4/1074192525 ct=1)


<4>NRFS: lookup(root/left)

<4>NRFS: call SERVER#0 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa6
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625aa
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 6e094
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(root/left, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(root/left, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(root/left, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(root/left set fileid=2:1074192532)
<4>NRFS: refresh_inode(4/1074192532 ct=1)
<4>NRFS: __nrfs_fhget(4/1074192532 ct=1)


<4>NRFS: lookup(left/README.kernel-sources)

<4>NRFS: call SERVER#0 to lookup README.kernel-sources
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 66aa5
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup README.kernel-sources
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 625ad
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup README.kernel-sources
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 6e097
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(left/README.kernel-sources, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(left/README.kernel-sources, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(left/README.kernel-sources, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(left/README.kernel-sources set fileid=2:1074192535)
<4>NRFS: refresh_inode(4/1074192535 ct=1)
<4>NRFS: __nrfs_fhget(4/1074192535 ct=1)


<4>nrfs: read(left/README.kernel-sources, 512@0)

<4>NRFS: nrfs_readpage (c0368ac8, left/README.kernel-sources, 4096@0)
<4>NRFS: nrfs_readpage_async(c0368ac8, left/README.kernel-sources, 4096@0)

<4>NRFS: 0 read buffer(c5dec000) for SERVER#0 allocated
<4>NRFS: 0 read buffer(c5e4b000) for SERVER#1 allocated
<4>NRFS: 0 read buffer(c5e4a000) for SERVER#2 allocated

<4>NRFS: 0 executing async read call for SERVER#0(nrpc01)
<4>NRFS: 0 executing async read call for SERVER#1(nrpc02)
<4>NRFS: 0 executing async read call for SERVER#2(nrpc03)

<4>RPC:      checksum and data read reply OK status 0
<4>RPC:      checksum and data readres OK, csum(0:4:ef20), count 425
<4>NRFS: 0 callback(xid=19) for page c5e45000 from server#0, result 425

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:dea3)
<4>NRFS: 0 callback(xid=21) for page c5e45000 from server#2, result 0

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:dea3)
<4>NRFS: 0 callback(xid=20) for page c5e45000 from server#1, result 0
```

ファイルの読み込み
FILE : root/left/README.kernel-sources

SERVER#0 : CHECKSUM_READ要求
SERVER#1 : CHECKSUM要求
SERVER#2 : CHECKSUM要求

CHECKSUMの不一致を検出

```
<4>NRFS: 0 all read call for page c5e45000 completed
<4>NRFS: 0 validate data replied from each server


<4>NRFS:  call SERVER#1 for re-read(4096@0)
<3>RPC: rpciod waiting on sync task!
<4>RPC:       readres OK status 0
<4>RPC:       readres OK count 642
<4>NRFS:  SERVER#1 reply re-read(4096@0): 642


<4>NRFS: 0 update page c5e45000 with data reply from SERVER#1

<1>NRFS: 0 INVALID DATA(README.kernel-sources) ON SERVER#0 DETECTED !

<4>NRFS: 0 Notify SERVER#0 to repair README.kernel-sources


<3>RPC: rpciod waiting on sync task!


<4>NRFS: 0 SERVER#0 reply repair: 0

<1>NRFS: 0 SERVER#0 accepted REPAIR call, 0.


<4>NRFS: refresh_inode(4/1074192535 ct=1)

<4>NRFS: 0 readpage_async done, 1 successful, 0 failures


<4>nrfs: read(left/README.kernel-sources, 512@512)

<4>NRFS: call  getattr from SERVER#0
<4>RPC:       attrstat status 0
<4>RPC:       attrstat OK type 1 mode 100644 dev 308 ino 66aa5
<4>NRFS: SERVER#0 reply getattr

<4>NRFS: call  getattr from SERVER#1
<4>RPC:       attrstat status 0
<4>RPC:       attrstat OK type 1 mode 100644 dev 308 ino 625ad
<4>NRFS: SERVER#1 reply getattr

<4>NRFS: call  getattr from SERVER#2
<4>RPC:       attrstat status 0
<4>RPC:       attrstat OK type 1 mode 100644 dev 308 ino 6e097
<4>NRFS: SERVER#2 reply getattr

<4>NRFS: refresh_inode(4/1074192535 ct=1)

<4>NRFS: nrfs_readpage (c0368ac8, left/README.kernel-sources, 4096@0)
<4>NRFS: nrfs_readpage_async(c0368ac8, left/README.kernel-sources, 4096@0)

<4>NRFS: 1 read buffer(c5dec000) for SERVER#0 allocated
<4>NRFS: 1 read buffer(c5e49000) for SERVER#1 allocated
<4>NRFS: 1 read buffer(c5e48000) for SERVER#2 allocated

<4>NRFS: 1 executing async read call for SERVER#0(nrpc01)
<4>NRFS: 1 executing async read call for SERVER#1(nrpc02)
<4>NRFS: 1 executing async read call for SERVER#2(nrpc03)

<4>RPC:       checksum read reply OK status 0
<4>RPC:       checksum readres OK, csum(0:4:dea3)
<4>NRFS: 1 callback(xid=28) for page c5e45000 from server#1, result 0

<4>RPC:       checksum read reply OK status 0
<4>RPC:       checksum readres OK, csum(0:4:dea3)
<4>NRFS: 1 callback(xid=29) for page c5e45000 from server#2, result 0

<4>RPC:       checksum and data read reply OK status 0
<4>RPC:       checksum and data readres OK, csum(0:4:ef20), count 425
<4>NRFS: 1 callback(xid=27) for page c5e45000 from server#0, result 425

<4>NRFS: 1 all read call for page c5e45000 completed
<4>NRFS: 1 validate data replied from each server


<4>NRFS:  call SERVER#1 for re-read(4096@0)
<3>RPC: rpciod waiting on sync task!
<4>RPC:       readres OK status 0
<4>RPC:       readres OK count 642
<4>NRFS:  SERVER#1 reply re-read(4096@0): 642


<4>NRFS: 1 update page c5e45000 with data reply from SERVER#1

<1>NRFS: 1 INVALID DATA(README.kernel-sources) ON SERVER#0 DETECTED !

<1>NRFS: 1 REPAIR CALL(642@0) TO SERVER#0 SKIPPED!


<4>NRFS: refresh_inode(4/1074192535 ct=1)

<4>NRFS: 1 readpage_async done, 2 successful, 0 failures


<4>nrfs: read(left/README.kernel-sources, 512@642)

<4>NRFS: call  getattr from SERVER#0
<4>RPC:       attrstat status 0
<4>RPC:       attrstat OK type 1 mode 100644 dev 308 ino 66aa5
<4>NRFS: SERVER#0 reply getattr
```

確認のためNRFSサーバ（SERVER#1）に対し
『READ要求』を発行

ファイル内容の不整合を検出
SERVER : SERVER#0
FILE : root/left/README.kernel-sources

NRFSサーバ（SERVER#0）に対し
『訂正要求』を発行

NRFSサーバ（SERVER#0）において
『訂正処理を正常に開始』

ファイルの読み込み
FILE : root/left/README.kernel-sources

SERVER#0 : CHECKSUM_READ要求
SERVER#1 : CHECKSUM要求
SERVER#2 : CHECKSUM要求

CHECKSUMの不一致を検出

既に訂正処理は動作しているものの、この時点ではまだ、処理が完了しておらず、不整合が残っているため

確認のためNRFSサーバ（SERVER#1）に対し
『READ要求』を発行

ファイル内容の不整合を検出
SERVER : SERVER#0
FILE : root/left/README.kernel-sources

当該障害については、既に訂正処理が動作しているため、SKIP

```
<4>NRFS: call  getattr from SERVER#1
<4>RPC:       attrstat status 0
<4>RPC:       attrstat OK type 1 mode 100644 dev 308 ino 625ad
<4>NRFS: SERVER#1 reply getattr

<4>NRFS: call  getattr from SERVER#2
<4>RPC:       attrstat status 0
<4>RPC:       attrstat OK type 1 mode 100644 dev 308 ino 6e097
<4>NRFS: SERVER#2 reply getattr

<4>NRFS: refresh_inode(4/1074192535 ct=1)

<4>nrfs: flush(4/2:1074192535)

<4>NRFS dentry_delete(left/README.kernel-sources, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/2:1074192535)
<4>NRFS: delete_inode(4/2:1074192535)

<4>NRFS dentry_delete(root/left, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/2:1074192532)
<4>NRFS: delete_inode(4/2:1074192532)

<4>NRFS dentry_delete(//root, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/2:1074192525)
<4>NRFS: delete_inode(4/2:1074192525)

<4>NRFS dentry_delete(///, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/0:402676)
<4>NRFS: delete_inode(4/0:402676)
```

図3.3.2-6　障害検出機能並びに訂正機能の検証（NRFSクライアントの応答ログ）
障害『ファイル内容の不整合』

```
<4>nfsd: GETATTR  776/402676


<4>nfsd: LOOKUP   776/402676 root
<4>nfsd: LOOKUP   776/420516 left
<4>nfsd: LOOKUP   776/420518 README.kernel-sources

<4>nfsd: CHECKSUM READ : 0-425(4096)-4096 byte/SUM : 0-4-20ef0000


<4>nfsd: NRFS REPAIR :

<4>nfsd: CLIENT   : 172.20.37.143
<4>nfsd: FILE     : /home/nrfs/root/left/README.kernel-sources(42)
<4>nfsd: E_TYPE   :    1
<4>nfsd: I_MODE   : 81A4
<4>nfsd: UID      : 65534
<4>nfsd: GID      : 65534

<4>nfsd: ESERV    : >>
<4>nfsd: SERVER   : 172.20.37.140
<4>nfsd: PATH     : /home/nrfs(10)

<4>nfsd: NCNT     : 2

<4>nfsd: NSERV[0] : >>
<4>nfsd: SERVER   : 172.20.37.141
<4>nfsd: PATH     : /home/nrfs(10)

<4>nfsd: NSERV[1] : >>
<4>nfsd: SERVER   : 172.20.37.142
<4>nfsd: PATH     : /home/nrfs(10)


<4>nfsd: GETATTR  776/420517

<4>nfsd: CHECKSUM READ : 0-425(4096)-4096 byte/SUM : 0-4-20ef0000


<4>nfsd: GETATTR  776/420517
```

ファイルへの『CHECKSUM_READ要求』
FILE : root/left/README.kernel-sources

クライアントからの障害情報に基づく訂正処理
障害を検出したクライアント
172.20.37.143
障害検出箇所
/home/nrfs/root/left/README.kernel-sources
障害内容
ファイル内容の不整合
障害が発生したサーバと共有リソース
172.20.37.140:/home/nrfs
正常なサーバと共有リソース
172.20.37.141:/home/nrfs
172.20.37.142:/home/nrfs

ファイルへの『CHECKSUM_READ要求』
FILE : root/left/README.kernel-sources

障害が発生したサーバ（SERVER#0、172.20.37.140）

```
<4>nfsd: GETATTR  776/370445


<4>nfsd: LOOKUP   776/370445 root
<4>nfsd: LOOKUP   776/402856 left
<4>nfsd: LOOKUP   776/402858 README.kernel-sources

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 0-642(4096)-4096 byte/SUM : 0-4-a3de0000
<4>nfsd: <<-----------------<<



<4>nfsd: READ     776/402861 4096 bytes at 0



<4>nfsd: GETATTR  776/402861

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 0-642(4096)-4096 byte/SUM : 0-4-a3de0000
<4>nfsd: <<-----------------<<



<4>nfsd: READ     776/402861 4096 bytes at 0



<4>nfsd: GETATTR  776/402861
```

ファイルへの『CHECKSUM』要求
FILE : root/left/README.kernel-sources

『ファイル内容の不整合』を検出したクライアントより
確認のための『READ要求』

ファイルへの『CHECKSUM要求』
FILE : root/left/README.kernel-sources

『ファイル内容の不整合』を検出したクライアントより
確認のための『READ要求』

正常に機能しているサーバ（SERVER#1、172.20.37.141）

```
<4>nfsd: GETATTR  776/482883


<4>nfsd: LOOKUP   776/482883 root
<4>nfsd: LOOKUP   776/450701 left
<4>nfsd: LOOKUP   776/450708 README.kernel-sources

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 0-642(4096)-4096 byte/SUM : 0-4-a3de0000
<4>nfsd: <<-----------------<<


<4>nfsd: GETATTR  776/450711

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 0-642(4096)-4096 byte/SUM : 0-4-a3de0000
<4>nfsd: <<-----------------<<


<4>nfsd: GETATTR  776/450711
```

ファイルへの『CHECKSUM要求』
FILE : root/left/README.kernel-sources

ファイルへの『CHECKSUM要求』
FILE : root/left/README.kernel-sources

正常に機能しているサーバ（SERVER#2、172.20.37.142）

図3.3.2-7　障害検出機能並びに訂正機能の検証（NRFSサーバの応答ログ）
障害『ファイル内容の不整合』

```
NRFS server recovery message

CLIENT    : 172.20.37.143
FILE      : /home/nrfs/root/left/README.kernel-sources(42)
E_TYPE    : 1
I_MODE    : 81a4
UID       : 65534
GID       : 65534

ESERV     : 172.20.37.140
PATH      : /home/nrfs(10)

NCOUNT    : 2
NSERV     : 172.20.37.141
PATH      : /home/nrfs(10)
NSERV     : 172.20.37.142
PATH      : /home/nrfs(10)

Recovery request to 172.20.37.141:
Recovery request target:/home/nrfs/root/left/README.kernel-sources
```

障害情報（ファイル内容の不整合）
- 障害を検出したクライアント
  - 172.20.37.143
- 障害発生箇所
  - /home/nrfs/root/left/README.kernel-sources
- 障害が発生したサーバと共有リソース
  - 172.20.37.140:/home/nrfs
- 正常なサーバと共有リソース
  - 172.20.37.141:/home/nrfs
  - 172.20.37.142:/home/nrfs

正常なサーバ（172.20.37.141）に対し『訂正処理』を要求

障害が発生したサーバ（SERVER#0、172.20.37.140）

```
SERVER CLOSE

Recovery request from 0.0.0.16:
REQUEST PATH : /home/nrfs
REQUEST FILE : root/left/README.kernel-sources
```

障害訂正処理情報
- 障害が発生したサーバと共有リソース
  - 172.20.37.140:/home/nrfs
- 訂正処理対象
  - root/left/README.kernel-sources

障害訂正処理を実行した正常サーバ（SERVER#1、172.20.37.141）

図3.3.2-8　障害検出機能並びに訂正機能の検証（NRFSサーバ訂正機能のログ）
障害『ファイル内容の不整合』

```
<4>NRFS: call  getattr from SERVER#0
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 624f4
<4>NRFS: SERVER#0 reply getattr

<4>NRFS: refresh_inode(4/402676 ct=1)
<4>NRFS: __nrfs_fhget(4/402676 ct=1)

<4>NRFS: call  getattr from SERVER#1
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 5a70d
<4>NRFS: SERVER#1 reply getattr

<4>NRFS: call  getattr from SERVER#2
<4>RPC:      attrstat status 0
<4>RPC:      attrstat OK type 2 mode 40777 dev 308 ino 75e43
<4>NRFS: SERVER#2 reply getattr

<4>NRFS: lookup(//root)

<4>NRFS: call SERVER#0 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa4
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625a8
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 6e08d
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(//root, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(//root, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(//root, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(//root set fileid=2:1074192525)
<4>NRFS: refresh_inode(4/1074192525 ct=1)
<4>NRFS: __nrfs_fhget(4/1074192525 ct=1)


<4>NRFS: lookup(root/right)

<4>NRFS: call SERVER#0 to lookup right
<4>RPC:      diropres status 2
<4>NRFS: SERVER#0 reply lookup: -2
<1>NRFS: Can't find right on SERVER#0 !!!

<4>NRFS: call SERVER#1 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625a9
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup right
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 6e093
<4>NRFS: SERVER#2 reply lookup: 0


<1>NRFS: Notify SERVER#0 to repair(20) right


<1>NRFS: SERVER#0 reply repair: 0


<4>NRFS: nrfs_fhget(root/right, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(root/right, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(root/right, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(root/right set fileid=2:1074192531)
<4>NRFS: refresh_inode(4/1074192531 ct=1)
<4>NRFS: __nrfs_fhget(4/1074192531 ct=1)

<4>NRFS dentry_delete(root/right, d_flags=0 means do nothing)


<4>NRFS: nrfs_readdir(root/right)

<4>NRFS: call SERVER#0 to readdir 0
<4>NRFS: SERVER#0 reply readdir: 4

<4>NRFS: call SERVER#1 to readdir 0
<4>NRFS: SERVER#1 reply readdir: 3

<4>NRFS: call SERVER#2 to readdir 0
<4>NRFS: SERVER#2 reply readdir: 3


<1>NRFS: dir-contents on SERVER#0 invalid!


<4>NRFS: nrfs_readdir(root/right)

<4>NRFS: call SERVER#0 to readdir 4096
<4>NRFS: SERVER#0 reply readdir: 0

<4>NRFS: call SERVER#1 to readdir 4096
<4>NRFS: SERVER#1 reply readdir: 0

<4>NRFS: call SERVER#2 to readdir 4096
```

ディレクトリの欠損を検出
SERVER : NRFSサーバ１（SERVER#0）
DIRECTORY : root/right

NRFSサーバ１（SERVER#0）に対し『訂正要求』を発行

NRFSサーバ１（SERVER#0）において『訂正処理を正常に開始』

ディレクトリの内容に不整合を検出
SERVER : NRFSサーバ１（SERVER#0）
DIRECTORY : root/right

既に訂正処理は動作しているものの、この時点ではまだ、処理が完了しておらず、不整合が残っているため

```
<4>NRFS: SERVER#2 reply readdir: 0

<4>NRFS dentry_delete(root/right, d_flags=0 means do nothing)

<4>NRFS: call SERVER#0 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa4
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625a8
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup root
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 6e08d
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: refresh_inode(4/1074192525 ct=1)


<4>NRFS: lookup(root/left)

<4>NRFS: call SERVER#0 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 66aa6
<4>NRFS: SERVER#0 reply lookup: 0

<4>NRFS: call SERVER#1 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 625aa
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup left
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 2 mode 40775 dev 308 ino 6e094
<4>NRFS: SERVER#2 reply lookup: 0

<4>NRFS: nrfs_fhget(root/left, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(root/left, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(root/left, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(root/left set fileid=2:1074192532)
<4>NRFS: refresh_inode(4/1074192532 ct=1)
<4>NRFS: __nrfs_fhget(4/1074192532 ct=1)


<4>NRFS: lookup(left/README.kernel-sources)

<4>NRFS: call SERVER#0 to lookup README.kernel-sources
<4>RPC:      diropres status 2
<4>NRFS: SERVER#0 reply lookup: -2
<1>NRFS: Can't find README.kernel-sources on SERVER#0 !!!

<4>NRFS: call SERVER#1 to lookup README.kernel-sources
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 625ad
<4>NRFS: SERVER#1 reply lookup: 0

<4>NRFS: call SERVER#2 to lookup README.kernel-sources
<4>RPC:      diropres status 0
<4>RPC:      diropres OK type 1 mode 100644 dev 308 ino 6e097
<4>NRFS: SERVER#2 reply lookup: 0


<1>NRFS: Notify SERVER#0 to repair(20) README.kernel-sources


<1>NRFS: SERVER#0 reply repair: 0


<4>NRFS: nrfs_fhget(left/README.kernel-sources, intall fh from SERVER#0, 0)
<4>NRFS: nrfs_fhget(left/README.kernel-sources, intall fh from SERVER#1, 0)
<4>NRFS: nrfs_fhget(left/README.kernel-sources, intall fh from SERVER#2, 1)

<4>NRFS: nrfs_fhget(left/README.kernel-sources set fileid=2:1074192535)
<4>NRFS: refresh_inode(4/1074192535 ct=1)
<4>NRFS: __nrfs_fhget(4/1074192535 ct=1)


<4>nrfs: read(left/README.kernel-sources, 512@0)

<4>NRFS: nrfs_readpage (c0367f60, left/README.kernel-sources, 4096@0)
<4>NRFS: nrfs_readpage_async(c0367f60, left/README.kernel-sources, 4096@0)

<4>NRFS: 10 read buffer(c5f2f000) for SERVER#0 allocated
<4>NRFS: 10 read buffer(c5f3c000) for SERVER#1 allocated
<4>NRFS: 10 read buffer(c5f3d000) for SERVER#2 allocated

<4>NRFS: 10 executing async read call for SERVER#1(nrpc02)
<4>NRFS: 10 executing async read call for SERVER#0(nrpc01)
<4>NRFS: 10 executing async read call for SERVER#2(nrpc03)

<4>RPC:      checksum and data read reply OK status 0
<4>RPC:      checksum and data readres OK, csum(0:4:dea3), count 642
<4>NRFS: 10 callback(xid=4772) for page c5dfc000 from server#1, result 642

<4>RPC:      checksum read reply OK status 15
<4>NRFS: 10 callback(xid=4773) for page c5dfc000 from server#0, result -21

<4>RPC:      checksum read reply OK status 0
<4>RPC:      checksum readres OK, csum(0:4:dea3)
```

ファイルの欠損を検出
SERVER : NRFSサーバ１（SERVER#0）
FILE : root/left/README.kernel-sources

NRFSサーバ１（SERVER#0）に対し
『訂正要求』を発行

NRFSサーバ１（SERVER#0）において
『訂正処理を正常に開始』

ファイルの読み込み時にエラーを検出
SERVER : NRFSサーバ１（SERVER#0）
FILE : root/left/README.kernel-sources

既に訂正処理は動作しているものの、この時点ではまだ、処理が完了していないため、エラーが発生

```
<4>NRFS: 10 callback(xid=4774) for page c5dfc000 from server#2, result 0

<4>NRFS: 10 all read call for page c5dfc000 completed
<4>NRFS: 10 validate data replied from each server

<4>NRFS: 10 update page c5dfc000 with data reply from SERVER#1
<1>NRFS: 10 INVALID DATA(README.kernel-sources) ON SERVER#0 DETECTED !
<1>NRFS: 10 REPAIR CALL(642@0) TO SERVER#0 SKIPPED!

<4>NRFS: refresh_inode(4/1074192535 ct=1)

<4>NRFS: 10 readpage_async done, 10 successful, 0 failures

<4>nrfs: read(left/README.kernel-sources, 512@512)
<4>nrfs: read(left/README.kernel-sources, 512@642)

<4>nrfs: flush(4/2:1074192535)

<4>NRFS dentry_delete(left/README.kernel-sources, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/2:1074192531)
<4>NRFS: delete_inode(4/2:1074192531)

<4>NRFS: put_inode(4/2:1074192535)
<4>NRFS: delete_inode(4/2:1074192535)

<4>NRFS dentry_delete(root/left, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/2:1074192532)
<4>NRFS: delete_inode(4/2:1074192532)

<4>NRFS dentry_delete(//root, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/2:1074192525)
<4>NRFS: delete_inode(4/2:1074192525)

<4>NRFS dentry_delete(///, d_flags=0 means do nothing)

<4>NRFS: put_inode(4/0:402676)
<4>NRFS: delete_inode(4/0:402676)
```

正常なサーバからの応答に基づいて処理を継続

図3.3.2-9　障害検出機能並びに訂正機能の検証（NRFSクライアントの応答ログ）
障害『ディレクトリあるいはファイルの欠損』

```
<4>nfsd: GETATTR  776/402676
```

```
<4>nfsd: LOOKUP   776/402676 root
<4>nfsd: LOOKUP   776/420516 right
```

```
<4>nfsd: NRFS REPAIR :

<4>nfsd: CLIENT   : 172.20.37.143
<4>nfsd: FILE     : /home/nrfs/root/right(21)
<4>nfsd: E_TYPE   :    0
<4>nfsd: I_MODE   : 41FD
<4>nfsd: UID      : 1024
<4>nfsd: GID      : 1024

<4>nfsd: ESERV    : >>
<4>nfsd: SERVER   : 172.20.37.140
<4>nfsd: PATH     : /home/nrfs(10)

<4>nfsd: NCNT     : 2

<4>nfsd: NSERV[0] : >>
<4>nfsd: SERVER   : 172.20.37.141
<4>nfsd: PATH     : /home/nrfs(10)

<4>nfsd: NSERV[1] : >>
<4>nfsd: SERVER   : 172.20.37.142
<4>nfsd: PATH     : /home/nrfs(10)
```

```
<4>nfsd: READDIR  776/420516 4096 bytes at 0
<4>nfsd: READDIR  776/420516 4096 bytes at 4096
```

ディレクトリへの『READDIR要求』
　DIRECTORY : root/right
ただし、『LOOKUP』時点で障害が検出される

クライアントからの障害情報に基づく訂正処理
　障害を検出したクライアント
　　172.20.37.143
　障害検出箇所
　　/home/nrfs/root/right
　障害内容
　　ディレクトリあるいはファイルの欠損
　障害が発生したサーバと共有リソース
　　172.20.37.140:/home/nrfs
　正常なサーバと共有リソース
　　172.20.37.141:/home/nrfs
　　172.20.37.142:/home/nrfs

『READDIR要求』への応答

```
<4>nfsd: LOOKUP   776/402676 root
<4>nfsd: LOOKUP   776/420516 left
<4>nfsd: LOOKUP   776/420518 README.kernel-sources
```

```
<4>nfsd: NRFS REPAIR :

<4>nfsd: CLIENT   : 172.20.37.143
<4>nfsd: FILE     : /home/nrfs/root/left/README.kernel-sources(42)
<4>nfsd: E_TYPE   :    0
<4>nfsd: I_MODE   : 81A4
<4>nfsd: UID      : 1024
<4>nfsd: GID      : 1024

<4>nfsd: ESERV    : >>
<4>nfsd: SERVER   : 172.20.37.140
<4>nfsd: PATH     : /home/nrfs(10)

<4>nfsd: NCNT     : 2

<4>nfsd: NSERV[0] : >>
<4>nfsd: SERVER   : 172.20.37.141
<4>nfsd: PATH     : /home/nrfs(10)

<4>nfsd: NSERV[1] : >>
<4>nfsd: SERVER   : 172.20.37.142
<4>nfsd: PATH     : /home/nrfs(10)
```

```
<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 0-4096(4096)-4096 byte/SUM : 0-4-624f4
<4>nfsd: <<-----------------<<
```

ファイルへの『CHECKSUM要求』
　FILE : root/left/README.kernel-sources
ただし、『LOOKUP』時点で障害が検出される

クライアントからの障害情報基づく訂正処理
　障害を検出したクライアント
　　172.20.37.143
　障害検出箇所
　　/home/nrfs/root/left/README.kernel-sources
　障害内容
　　ディレクトリあるいはファイルの欠損
　障害が発生したサーバと共有リソース
　　172.20.37.140:/home/nrfs
　正常なサーバと共有リソース
　　172.20.37.141:/home/nrfs
　　172.20.37.142:/home/nrfs

『CHECKSUM要求』への応答

障害が発生したサーバ（SERVER#0、172.20.37.140）

```
<4>nfsd: GETATTR  776/370445
```

```
<4>nfsd: LOOKUP   776/370445 root
<4>nfsd: LOOKUP   776/402856 right
<4>nfsd: READDIR  776/402857 4096 bytes at 0
<4>nfsd: READDIR  776/402857 4096 bytes at 4096
```

ディレクトリへの『READDIR要求』
　DIRECTORY : root/right

```
<4>nfsd: LOOKUP   776/370445 root
<4>nfsd: LOOKUP   776/402856 left
<4>nfsd: LOOKUP   776/402858 README.kernel-sources

<4>nfsd: CHECKSUM READ : 0-642(4096)-4096 byte/SUM : 0-4-a3de0000
```

ファイルへの『CHECKSUM_READ要求』
　FILE : root/left/README.kernel-sources

正常に機能しているサーバ（SERVER#1、172.20.37.141）

```
<4>nfsd: GETATTR  776/482883

<4>nfsd: LOOKUP   776/482883 root
<4>nfsd: LOOKUP   776/450701 right
<4>nfsd: READDIR  776/450707 4096 bytes at 0
<4>nfsd: READDIR  776/450707 4096 bytes at 4096
```

ディレクトリへの『READDIR要求』
DIRECTORY : root/right

```
<4>nfsd: LOOKUP   776/482883 root
<4>nfsd: LOOKUP   776/450701 left
<4>nfsd: LOOKUP   776/450708 README.kernel-sources

<4>nfsd: >>- CHECKSUM ONLY ->>
<4>nfsd: CHECKSUM READ : 0-642(4096)-4096 byte/SUM : 0-4-a3de0000
<4>nfsd: <<-----------------<<
```

ファイルへの『CHECKSUM要求』
FILE : root/left/README.kernel-sources

正常に機能しているサーバ（SERVER#2、172.20.37.142）

図3.3.2-10　障害検出機能並びに訂正機能の検証（NRFSサーバの応答ログ）
障害『ディレクトリあるいはファイルの欠損』

```
NRFS server recovery message

CLIENT    : 172.20.37.143
FILE      : /home/nrfs/root/right(21)
E_TYPE    : 0
I_MODE    : 41fd
UID       : 1024
GID       : 1024

ESERV     : 172.20.37.140
PATH      : /home/nrfs(10)

NCOUNT    : 2
NSERV     : 172.20.37.141
PATH      : /home/nrfs(10)
NSERV     : 172.20.37.142
PATH      : /home/nrfs(10)

Recovery request to 172.20.37.141:
Recovery request target:/home/nrfs/root/right
```

障害情報（ディレクトリの欠損）
- 障害を検出したクライアント
  - 172.20.37.143
- 障害発生箇所
  - /home/nrfs/root/right
- 障害が発生したサーバと共有リソース
  - 172.20.37.140:/home/nrfs
- 正常なサーバと共有リソース
  - 172.20.37.141:/home/nrfs
  - 172.20.37.142:/home/nrfs

正常なサーバ（172.20.37.141）に対し『訂正処理』を要求

```
NRFS server recovery message

CLIENT    : 172.20.37.143
FILE      : /home/nrfs/root/left/README.kernel-sources(42)
E_TYPE    : 0
I_MODE    : 81a4
UID       : 1024
GID       : 1024

ESERV     : 172.20.37.140
PATH      : /home/nrfs(10)

NCOUNT    : 2
NSERV     : 172.20.37.141
PATH      : /home/nrfs(10)
NSERV     : 172.20.37.142
PATH      : /home/nrfs(10)

Recovery request to 172.20.37.141:
Recovery request target:/home/nrfs/root/left/README.kernel-sources
```

障害情報（ファイルの欠損）
- 障害を検出したクライアント
  - 172.20.37.143
- 障害発生箇所
  - /home/nrfs/root/left/README.kernel-sources
- 障害が発生したサーバと共有リソース
  - 172.20.37.140:/home/nrfs
- 正常なサーバと共有リソース
  - 172.20.37.141:/home/nrfs
  - 172.20.37.142:/home/nrfs

正常なサーバ（172.20.37.141）に対し『訂正処理』を要求

障害が発生したサーバ（SERVER#0、172.20.37.140）

```
SERVER CLOSE

Recovery request from 0.0.0.16:
REQUEST PATH : /home/nrfs
REQUEST FILE : root/right


SERVER CLOSE

Recovery request from 172.20.37.140:
REQUEST PATH : /home/nrfs
REQUEST FILE : root/left/README.kernel-sources
```

障害訂正処理情報
- 障害が発生したサーバと共有リソース
  - 172.20.37.140:/home/nrfs
- 訂正処理対象
  - root/right

障害訂正処理情報
- 障害が発生したサーバと共有リソース
  - 172.20.37.140:/home/nrfs
- 訂正処理対象
  - root/left/README.kernel-sources

障害訂正処理を実行した正常サーバ（SERVER#1、172.20.37.141）

図3.3.2-11　障害検出機能並びに訂正機能の検証（NRFSサーバ訂正機能のログ）
障害『ディレクトリあるいはファイルの欠損』

表 3.3.2-1　NRFS と NFS のパフォーマンス比較

操作	所要時間 秒	パフォーマンス Mbyte／秒
ローカルファイルシステム上からNFSへ	1400	1.66
NFS上からローカルファイルシステムへ	1380	1.69
ローカルファイルシステム上からNRFSへ	2450	0.95
NRFS上からローカルファイルシステムへ	1760	1.32

約2.3GByteのディレクトリ（比較的容量が小さい無数のファイルで構成）のコピー

操作	所要時間 秒	パフォーマンス Mbyte／秒
ローカルファイルシステム上からNFSへ	1000	2.05
NFS上からローカルファイルシステムへ	1010	2.03
ローカルファイルシステム上からNRFSへ	1700	1.21
NRFS上からローカルファイルシステムへ	1220	1.68

約2.1GByteの単一ファイルのコピー処理

3.3.3　NRFS プロトタイプに関するまとめ

今期は開発期間が十分に確保できないことから、NRFS 基本部の設計とその実装に注力し、サーバの多重化による冗長性の獲得、多数決による障害検出方式の整備（チェックサム情報の活用による効率化についても実現）、さらには NRFS サーバの協調による障害訂正機能の実装といった、NRFS に要求される基本機能について不完全ではあるものの一通り具現化した。

ただし、作成した NRFS プロトタイプでは、

① ファイル内容の不整合

② ディレクトリあるいはファイルの欠損

といった障害の発生を想定し、障害検出機能並びに訂正機能を整備しているものの、実際の運用環境では、その発生頻度は低いと考えられるが、様々な障害に遭遇し得る。また、LAN の障害や電源系統のトラブル等、外的な要因による障害に対しても、可能な限り安定した稼動が求められる。

このように、NRFS の実用化に向けては、クライアント並びにサーバの品質向上に加え、遭遇し得る障害を綿密に分析し、対応する障害検出訂正機能を設計・実装する等、機能面の充実並びに信頼性・安定性のより一層の向上が求められる。

4. 今期の成果と今後の課題

今期は、LAN 環境上で遊休資源化したディスクを有効活用し、低コストで高信頼分散共有ファイルシステムを可能とするネットワーク RAID ファイルシステム（NRFS）の基本部の設計とプロトタイプの開発を行った。

NRFS の基本部の設計に先立ち、多くのプラットフォーム上でサポートされている Network File System（NFS）について、その実装の詳細を解析した。その調査結果に基づいて、NFS を拡張する形で NRFS 基本部の設計を行った。NFS のサーバ実装を活用し、クライアントが複数のサーバに対してファイルアクセスを行うことにより、NRFS の高信頼性の原点となる冗長性を獲得することにした。もちろん、この複数のサーバ上のファイルシステムはマシンやディスクに障害がない限り同一の構成を持つように管理される。つまり、ファイル書き込みはすべてのサーバに対して行われ、ファイル読み出しはすべてのサーバからデータが読み出される。ただし、ファイル読み出しに関してはクライアントの処理を軽減するため、サーバ側でチェックサムを計算し、クライアントはチェックサムが一致していることでデータ一致の認証に代えている。チェックサム計算自体をネットワークハードウェアに担当させることを検討していたが、現在主流の Fast Ethernet カードではハードウェアチェックサム機構が標準装備されているとは言いがたいため、今期の開発ではソフトウェアによるチェックサム計算を実装した。ソフトウェア実装である利点を活かし、ファイル読み出し時は１台のサーバに対してのみデータとチェックサムを要求し、他のサーバに対しては対応するファイルデータのチェックサムだけを要求して、ネットワークトラフィックを大幅に削減する最適化を行っている。ディレクトリ情報やデータに誤りを発見した場合には、クライアントはエラーを起こしたサーバに対してエラー修正のヒントを発行する。従来の NFS 用の Remote Procedure Call (RPC)に対して、NRFS 特有のこれらのチェックサム読み出しや修理要求のプロトコルを拡張して、NRFS 用の RPC を確立した。

続いて、NRFS 基本部の実装方式を検討し、NRFS の実用性を実証するために、NRFS プロトタイプの作成を行った。このプロトタイプでは、サーバ多重化に対応したクライアント並びにサーバを整備するとともに、特定の障害発生に対応した障害検出訂正機能を実装した。具体的には、ファイル上のデータ誤りとファイルもしくはディレクトリの欠損（ディスククラッシュを含む）に対応する NRFS プロトタイプを作成した。

作成したプロトタイプについて、実際にテストを行い、NRFS 基本部が設計通りに機能することを確認するとともに、そのパフォーマンスを測定し、実用上支障がないレベルで NRFS が実装できることを確認した。

今期は開発期間が十分に確保できないことから、障害検出訂正機能に関して、対応可能な障害を特定のものと限定した。しかしながら、実際の運用環境では、その発生頻度は低いと考えられるものの、様々な障害に遭遇し得る。また、LAN の障害や電源系統のトラブ

ル等、外的な要因による障害に対しても、可能な限り安定した稼動が求められる。特に、今回は対応を見送ったサーバの停止やサーバとの通信経路のダウンといった障害に対しては、実用に向けて必ず対応する必要がある。また、性能に関しても、可能な限り NFS と同等の性能を達成したいと考えている。

このように、NRFS の実用化に向けては、クライアント並びにサーバの品質向上に加え、遭遇し得る障害を綿密に分析し、対応する障害検出訂正機能を設計・実装する等、機能面の充実並びに信頼性・安定性のより一層の向上が求められる。さらに、現状では、PC 機の Linux 環境並びに開発者が研究開発中である SPARC 機の SSS-CORE 環境上で開発を行っているが、その配布が容易となるよう配慮する必要がある。

また、現行の NRFS は、UDP/IP（User Datagram Protocol/Internet Protocol）ベースの実装であるが、開発者が研究開発中の Memory Based Control Facility（MBCF）に移行することで、パフォーマンスの向上を図るといったアプローチも有効である。

したがって、本プロジェクト終了後も、引き続き開発作業を継続し、NRFS の実用化に向けて注力したい。

添付 A. ハードウェアチェックサム機構の流用についての調査

各種 NIC および MAC に関する調査

Gigabit Ethernet に関しては、ハードウェアチェックサム機構が NIC に標準装備されているケースが多い。これは Gigabit Ethernet ではプロセッサがチェックサム計算を行っていてはその計算オーバヘッドによって通信性能が大幅に低下してしまうためだと考えられる。これに対して、Fast Ethernet (100BASE-T, 100BASE-TX)の初期の NIC にはあまりハードウェアチェックサム機構は搭載されていない。最近の Fast Ethernet NIC にはメインプロセッサの通信負荷を緩和するために、ハードウェアチェックサム機構を搭載するタイプのものが増えている。以下に具体的な NIC または MAC（Ethernet コントローラチップ）に関するハードウェアチェックサム機構に関する調査結果を示す。

1. Intel 社製（元 DEC 製）21140A（種別 MAC）過去多くの NIC に採用された
   ハードウェアチェックサム機構なし、詳細マニュアル入手可能
2. Intel 社製（元 DEC 製）21143（種別 MAC）
   ハードウェアチェックサム機能なし、詳細マニュアル入手可能
3. Intel 社製　82559ER（種別 MAC）
   ハードウェアチェックサム機構なし、詳細マニュアル入手不可能
4. Intel 社製　82559（種別 MAC）Ether Express Pro 10/100B に採用
   ハードウェアチェックサム機構あり、詳細マニュアル入手不可能
5. Intel 社製　82550（種別 MAC）
   ハードウェアチェックサム機構あり、詳細マニュアル入手不可能
   ハードウェア IPSec サポート機構あり
6. AMD 社製　Am79C971（種別 MAC）
   ハードウェアチェックサム機能なし、詳細マニュアル入手可能
7. 3Com 社製　3C905C（種別 NIC）
   ハードウェアチェックサム機構あり、詳細マニュアル入手可能
   ただし、チェックサム機能は IP V4 TCP/IP, UDP/IP パケット形式に厳密に限定
8. Sun Microsystems 社製　STP2002QFP（別名 hme, 種別 MAC）
   ハードウェアチェックサム機構あり、詳細マニュアル入手可能
   自由度が高く NRFS 用チェックサムに効率良く流用が可能
9. Sun Microsystems 社製　SME2005PBGAC（別名 gem, 種別 Gigabit MAC）
   ハードウェアチェックサム機構あり、詳細マニュアル入手可能(要 NDA)
   自由度が高く NRFS 用チェックサムに効率良く流用が可能

10. Alteon Networks 社製　Gigabit Ethernet/ NIC（別名 Tigon, 種別 Gigabit NIC）
    ハードウェアチェックサム機構あり、詳細マニュアル入手可能
    ただし、チェックサム機能は IP V4 TCP/IP, UDP/IP パケット形式に厳密に限定
11. SysKonnect 社製　SK-NET GENESIS NIC（種別 Gigabit NIC）
    ハードウェアチェックサム機構あり、詳細マニュアル入手可能
    自由度が高く NRFS 用チェックサムに効率良く流用が可能

なお、9., 10., 11.が Gigabit Ethernet 用の MAC ならびに NIC である。

今期の NRFS の開発においては、まだ Gigabit Ethernet 用スイッチや NIC が普及しているとは言いがたく、価格もまだ高価であるため、Fast Ethernet ベースの NIC を採用して開発することにした。調査結果よりハードウェアチェックサム機構を有しており、詳細マニュアルが入手可能である MAC もしくは NIC は 3Com 社製の 3C905C NIC もしくは Sun Microsystems 社製の STP2002QFP MAC のみである。NRFS への流用を考えると STP2002QFP の方が機能的には格段に優れている。しかし、Sun Microsystems 社製 MAC は同社製ワークステーション用の NIC しか販売しておらず、PC 互換機用 Linux に対応したデバイスドライバが存在しない。このため、PC 互換機用には 3Com 社製の 3C905C NIC を採用することにした。Sun ワークステーション上の SSS-CORE オペレーティングシステムでは STP2002QFP を採用することとする。

UDP チェックサム機構流用に関する検討

PC 互換機に採用した Fast Ethernet NIC 3C905C はハードウェアチェックサム機構を有しているが、厳密に IP V4 TCP/IP もしくは UDP/IP パケット形式に基づいたパケットしかチェックサムを計算してくれない。このため、3C905C のチェックサム機構を NRFS のデータチェックサムに流用するためには、チェックサムの値を補正するためのソフトウェアが必要である。本節では、その補正方式について説明する。NRFS は NFS のサーバを冗長に接続可能にして、高信頼性を達成するファイルシステム構成方式である。NFS はデータ通信を通常は UDP/IP で行うため、NRFS も基本的には UDP/IP によって通信を行う（ただし、将来的にはより効率の良い MBCF/IP もしくは MBCF プロトコルの採用を検討している）。そこで、まず UDP/IP の UDP チェックサム計算方式について述べる。

図 A1 に各種パケットヘッダの形式を示す。Ethernet 上（Fsat Ethernet, Gigabit Ethernet もパケット形式は同じ）の UDP/IP パケットは先頭に Ether Header があり、次に IP Header があり、その次に UDP Header があり、その後に UDP のペイロード（送信データ）が付随する形式となっている。Ether Header は 14 バイトであり、4 バ

イトの倍数ではなく、IP Header や UDP Header は 2 バイト境界から開始される。

Ether Header

00 – 03	Destination Ether Address (1-4)	
04 – 07	Destination Ether Addr (5-6)	Source Ether Address (1-2)
08 – 11	Source Ether Address (3-6)	
12 – 15	EtherType	

EtherType: IP 0x0800, ARP 0x0806, RARP 0x8035, MBCF 0x6666

IP Header

00 – 03	VERS | HLEN	ServiceType	Total Length	
04 – 07	IDENTIFICATION		FLAGS	Fragment Offset
08 – 11	TimeToLive	Protocol	HeaderChecksum	
12 – 15	Source IP Address			
16 – 19	Destination IP Address			
20 – 20+4n	IP Options (IF ANY)			Padding

Protocol : ICMP 1, TCP 6, UDP 17, ESP 50, AH 51

UDP Header

00 – 03	UDP Source Port	UDP Destination Port
04 – 07	UDP Message Length	UDP Checksum

Pseudo Header for Checksum

00 – 03	Source IP Address		
04 – 07	Destination IP Address		
08 – 11	0	Protocol	TCP/UDP Length

図 A1　　Ethernet 上 UDP/IP の各種ヘッダ

IP Header 内には Header Checksum と呼ばれるフィールドが存在するが、このフィールドは IP Header 部についてのみのチェックサムを表しているため、NRFS のデータチェックサムとは無関係である。NRFS に流用できる UDP チェックサムは UDP Header 内にあり、UDP Header と後続の UDP ペイロードと図 A1 の最後に表示してある Pseudo Header のチェックサムを取ったものになっている。厳密には 2 バイトずつ 1 の補数和をとり、その結果の 1 の補数を UDP Checksum フィールドに格納する。補数和をとる前には UDP Checksum フィールドを 0 にしておく。参考のため、UDP チェックサム計算のためのアルゴリズムを図 A2 と図 A3 に示す。

```
UdpChecksumCalc(buf,len)
unsigned char *buf;
int len;
{
        unsigned char p[12];

        *(p) = source_ipa >> 24;
        *(p+1) = source_ipa >> 16;
        *(p+2) = source_ipa >> 8;
        *(p+3) = source_ipa;
        *(p+4) = dest_ipa>>24;
        *(p+5) = dest_ipa>>16;
        *(p+6) = dest_ipa>>8;
        *(p+7) = dest_ipa;
        p[8] = 0;    p[9] = IP_PROTO_UDP; /* p[9] = IP_PROTO_TCP; */
        *(p+10) = udp_len>>8; /* tcp_len */
        *(p+11) = udp_len;

        s = IPCheckSum2(p, 12, buf,len );
        return s & 0xFFFF;
}
```

図 A2　UDP(TCP)**チェックサム計算**

```
IPCheckSum2( buf , len ,buf2,len2)
unsigned char   *buf;
int     len;
unsigned char   *buf2;
int     len2;
{
        int     i;
        unsigned int    h,l;
        unsigned long   sum;

        sum=0;
        for( i=0 ; i<len-2 ; i+=2 ){
          h= *buf++;
```

```
        l= *buf++;
        sum += ((h << 8) | l);
      }
      if(len & 1){
        h= *buf++;
        l= *buf2++;
        len2--;
      }else{
        h= *buf++;
        l= *buf++;
      }
      sum += ((h << 8) | l);

      for( i=0 ; i<len2-2 ; i+=2 ){
        h= *buf2++;
        l= *buf2++;
        sum += ((h << 8) | l);
      }
      if((len+len2) & 1){
        h= *buf2++;
        l= 0;
      }else{
        h= *buf2++;
        l= *buf2++;
      }
      sum += ((h << 8) | l);
      sum = (sum & 0xFFFF) + (sum>>16);
      sum = (sum & 0xFFFF) + (sum>>16);
      return ((~sum) & 0xFFFF);
}
```

図 A3　IP **チェクサム計算**

NRFS は NFS が使用する Sun RPC を拡張した RPC によってサーバクライアント間の通信を行っている。RPC は UDP/IP 上のプロトコルとして実装されているため、RPC のヘッダ部分およびデータ部分は共に UDP/IP にとってはペイロードとして扱われる。NRFS では Read データ部分のみのチェックサムを使用するため、Pseudo Header と

IP Header と RPC Header 部および NFS 用 RPC のデータ以外の部分のチェックサムを計算し、UDP Checksum フィールドの補数から差し引いて Read データ部分のみのチェックサムを求める必要がある。データ部分のチェックサム計算の必要はないため、チェックサム計算でアクセスするメモリ領域は４分の１程度に抑えることが可能である。ただし、今期の NRFS プロトタイプの実装では、Fast Ethernet 上の通信量の削減に重きをおいて、RPC にチェックサムフィールドを追加して、データとチェックサムを１台のサーバから返送し、他のサーバからはチェックサムの値だけを返送するソフトウェアを実装する。ハードウェアチェックサムの流用に関しては来期以降の課題とする。転送容量の大きい Gigabit Ethernet のような高速 LAN でこそ大きな効果を上げると考えられる。

STP2002QFP チェックサム機構流用に関する検討

STP2002QFP や SME2005PBGAC や SK-NET GENESIS では Ethernet パケットの先頭から Checksum Start Offset だけずらした場所から最後までチェックサム計算を行う仕組みになっている。そして、Checksum Stuff Offset だけずらした部分に計算結果を格納する仕様になっている。特に、GENESIS は自由度が非常に高く、両 Offset を 16bit のサイズ（バイト単位）で指定できる。このため、RPC メッセージの最後部に Read データが搭載されていれば、Read データ部のみのチェックサムを計算して、RPC のペイロード部（UDP/IP にとってももちろんペイロード部）に格納することが可能である。ただし、この場合は UDP チェックサムを計算するハードウェアが流用されているため、UDP チェックサムフィールドには 0 を格納して送信する。UDP/IP において UDP チェックサムが 0 というのはチェックサムによる認証をスキップすることを意味する。

STP2002QFP や SME2005PBGAC では Checksum Start Offset と Checksum Stuff Offset のサイズがそれぞれ 6bit と 8bit に制限されている。このため、Read データが Ether パケットの先頭から 62 バイト目以内に始まるようにパケットフォーマットを定義しないと 3C905C NIC 同様にチェックサム値を補正するソフトウェアがないとハードウェアチェックサム機構を流用できない。NFS の RPC の Read Reply は返送されるパラメータが多く、この条件を満たすことができない。よって、高速な NRFS ルーチンを開発するためには NRFS 用 Read Reply のうちの不要な（冗長な）パラメータを省略する必要がある。これは来期以降の本格 NRFS システムの開発の課題の一つである。