Um Sistema de Arquivos Distribuídos Tolerante a Falhas para UNIX Compatível com o NFS

Mário Magalhães Leboute Taisy Silva Weber

Instituto de Informática - UFRGS
Caixa Postal 15064, 91501-970 - Porto Alegre, RS
taisy@inf.ufrgs.br, FAX (051) 319 1576

Resumo

É apresentado o projeto e a implementação de um sistema de arquivos distribuídos para o UNIX, voltado à obtenção de alta confiabilidade no armazenamento de dados, além de alta disponibilidade de acesso. Foi usado como base o sistema de arquivos NFS (*Network File System*), estendido para tolerar falhas através da replicação de arquivos e diretórios entre diversos servidores. O método de replicação empregado é baseado em um algoritmo de cópia principal, com distribuição síncrona de atualizações do servidor principal aos servidores secundários. Os clientes do sistema são capazes de chavear automaticamente o servidor no caso de falhas, com continuidade no acesso aos dados. Requisições de acesso por parte dos clientes levam à eleição de novo servidor principal, em caso colapso do principal, com proteções contra falhas de particionamento de rede. É proposta a recuperação de volumes em servidores religados de forma automática.

Abstract

This paper proposes a distributed file system for UNIX, supplying high availability on file service and high data dependability. The existing NFS (Network File System) is extended to achieve fault tolerance through file and directory replication over many servers on a network. The replication method used is a variation of the primary site algorithm, with synchronous distribution of updates from primary to secondary servers. System clients are able to automatically switch the server on the event of failures. Requests from clients are used to start the election of a new master server if required, with cares against network partitioning. The recover of the contents of a rebooted server is made automatically.

1. INTRODUÇÃO

Todas as técnicas de tolerância a falhas valem-se de redundância de componentes, seja de software ou de hardware, para reduzir a possibilidade de um colapso no sistema. Falhas são toleradas configurando-se o sistema de modo que um componente defeituoso tenha seu comportamento mascarado pelos componentes redundantes. Técnicas sofisticadas podem garantir a continuidade de processos tão críticos quanto o controle de aeronaves, mesmo com perda completa de processadores.

Entretanto, técnicas sofisticadas para a obtenção de alta confiabilidade não são necessárias na maioria da instalações, ou não são viáveis em função de seu alto custo. Na maioria das plataformas convencionais, paradas temporárias, e falhas parciais ou

totais em um sistema não são eventos catastróficos, desde que sua frequência seja baixa e o tempo de recuperação não seja longo. O que não pode ser aceita é a perda de dados críticos gerados em algum momento antes da falha. A preservação da informação, em paralelo com sua correção, é a preocupação mais séria em sistemas computacionais convencionais. Os recursos físicos usados para armazenar arquivos são suscetíveis a falhas, quer sejam falhas no próprio meio, nos dispositivos de controle, nos computadores que gerenciam estes dispositivos, e, no caso de redes, nas ligações de comunicação. O uso de redes e servidores corporativos, com dezenas ou centenas de usuários, tende a agravar este problema, pois aumenta o número de atividades que dependem dos mesmos arquivos, criando condições para que aconteçam perdas simultâneas, ou em cascata, devido a falhas em algum dos servidores, com consequências catastróficas.

Em função da elevada confiabilidade necessária no armazenamento de dados, diversas técnicas específicas ([BHI90], [BRE89], [ELA85], [HIS90], [KUM91], [LAD90], [LEV90], [LIS91], [LIS91a], [PUR87]) têm sido desenvolvidas para tornar este recurso mais seguro, sem que uma solução satisfatória tenha sido ainda assumida como definitiva. Este artigo descreve uma solução para prover tolerância a falhas transparente e automática para um dos sistemas de arquivamento distribuído mais utilizados: o NFS ([SAN85], [RFC1094], [RFC1813]) do UNIX. A solução visa alcançar tolerância a falhas pela replicação transparente de arquivos entre diversos servidores. Em caso de uma falha qualquer em um servidor, o ambiente automaticamente torna disponível réplicas em outros servidores, sem que a falha seja percebida pelo usuário. O sistema de replicação de arquivos tolerante a falhas desenvolvido denomina-se RNFS (*Reliable Network File System*) e é baseado no método de cópia primária [HUA89]. A terminologia usada no artigo é a usual na área de tolerância a falhas em sistemas distribuídos [JAL94].

2. RÉPLICAS EM ARQUIVAMENTO DISTRIBUÍDO

O aumento de velocidade das redes modernas e sua popularização permitem a adoção de técnicas de replicação de arquivos e a distribuição de réplicas por vários servidores. O algoritmo de replicação é o centro de todo sistema de replicação de dados, e determina de forma decisiva as decisões a serem tomadas no tratamento de falhas e na recuperação.

2.1 Algoritmo de controle de réplicas

Um algoritmo de replicação de arquivos deve resolver principalmente o problema da *manutenção da consistência das cópias sob escritas concorrentes*. Em um sistema multitarefa, diversos processos executam simultaneamente, e podem compartilhar o acesso aos arquivos, inclusive para escrita. Este recurso permite que arquivos sejam usados para troca dinâmica de informações entre programas e usuários. Ainda que leituras concorrentes não gerem nenhum problema, escritas concorrentes podem gerar inconsistências. A questão fundamental está em garantir que todas as alterações geradas sejam aplicadas em todas as cópias na mesma ordem, além da necessidade de que tenham a mesma efetividade. Não importa garantir que esta ordem seja a mesma ordem física em que as requisições foram geradas, nem que as imagens que cada cliente tenha do arquivo sejam as mesmas a qualquer instante, apenas que todas as cópias executem a mesma seqüência de alterações. Esta ordenação, denominada na literatura ([TAN92], [JAL94]) como *serialização como cópia única*, não pode ser garantida de forma simples. Os algoritmos desenvolvidos para isto, denominados *algoritmos de controle de réplicas*, devem também ser capazes de operar quando da ocorrência de

falhas, quando o número de nodos se reduz, ou sob outras condições, como particionamentos da rede, em função do modelo de falhas adotado.

Os algoritmos de controle de réplicas podem ser divididos [JAL94] em três grandes grupos: algoritmos baseados em votação [GIF79]; algoritmos baseados em cópia primária e algoritmos baseados em cópias ativas. Esse artigo trata do projeto de um sistema de replicação de dados baseado em algoritmos de controle de réplicas baseados em cópia primária.

2.2 Controle de réplicas por cópia primária

Muitos métodos de replicação baseiam-se no uso de uma entidade centralizadora para realizar a difusão de escritas. Em qualquer situação existirá uma ordem única em que os acessos são recebidos pelo centralizador, e este repetirá estes acessos às réplicas nesta mesma ordem. De maneira geral, é interessante escolher uma das cópias em questão como entidade centralizadora. Esta escolha elimina um elemento do sistema (um centralizador dedicado), e permite que, em caso de falha na cópia centralizadora, outra cópia passe a agir como centralizador, evitando a necessidade de centralizadores-estepe específicos. Em um algoritmo desta natureza, a cópia escolhida como principal se torna responsável por irradiar para todas as outras réplicas cada procedimento de escrita, de forma síncrona (ou seja, um novo procedimento de escrita só começa quando o anterior tiver sido difundido para todas as réplicas). A mesma cópia também realiza estes procedimentos em sua imagem local. Os procedimentos apenas de leitura são realizados somente no contexto da cópia principal, e não necessitam ser irradiados. Todos os clientes conhecem qual servidor é o principal, e comunicam-se apenas com este servidor. Isto implica na necessidade de mudança de servidor principal, no caso de uma falha neste servidor, e na existência de mecanismos para difundir esta informação para todos os clientes.

Uma grande quantidade de trabalhos dedicam-se ao método de cópia principal. O algoritmo básico está descrito em livros clássicos da área ([TAN92], [JAL94]). Um estudo de seu desempenho foi apresentado por Huang [HUA89].

2.3 Arquivamento distribuído em UNIX

O NFS [SAN85] é o sistema de arquivos distribuído mais utilizado em ambiente UNIX. Embora contenha algumas incorreções sem‰onticas [NEL88], é um sistema simples e eficiente. O NFS tornou-se um padrão de fato para distribuição de arquivos em redes locais baseadas em UNIX.

O NFS implementa um ambiente cliente-servidor, em que N estações clientes podem acessar simultaneamente um conjunto de M servidores, utilizando partes de diretórios (denominadas "volumes") contidas nestes servidores como se fossem locais, e implementando compartilhamento remoto de arquivos. O NFS usa um protocolo fundamentado na chamada remota de procedimentos (RPCs), implementado no núcleo do SO (nos clientes) e em processos "deamons" (nos servidores). O NFS apoia-se na camada de serviços RPC padrão do System-5 do UNIX [RFC1057], e ainda na camada de transporte disponível na rede para conectar clientes e servidores [RFC1014].

3. RNFS

O RNFS foi desenvolvido visando: (a) obter o máximo de compatibilidade com o padrão NFS atual, preservando a cultura de usuários e administradores, e utilizando o

mesmo método básico de serviço de arquivos e suas semânticas; (b) realizar automaticamente recuperações após falhas mantendo as atividades relacionadas à replicação e à recuperação invisíveis ao usuário; (c) utilizar o volume de montagem do UNIX como unidade de replicação permitindo um grau configurável de replicação para cada volume escolhido. Os objetivos deveriam ser alcançados evitando o uso de qualquer *hardware* especial.

3.1 Modelo de falhas utilizado

O modelo de falhas suportado pelo RNFS é baseado nas seguintes suposições:

- Servidores e clientes podem falhar a qualquer tempo.
- A rede de comunicação pode perder pacotes mesmo na ausência de falhas de servidores ou estações, mas este evento tem baixa probabilidade. Adicionalmente existe a possibilidade de particionamentos na rede de estações [DAV85].
- Podem ocorrer falhas simultâneas em todas as estações, como consequência de falta de energia na organização.
- Podem ocorrer pseudo-falhas, ou seja, operações que não podem ser realizadas em determinada réplica devido a alguma condição não-errÙnea nesta, como por exemplo, um sistema de arquivos cheio.

O modelo de falhas aproxima-se o mais exatamente possível das condições esperadas em uma rede de estações UNIX, em especial redes de médio porte (interligadas por *bridges* e *gateways*), em que particionamentos são mais prováveis, e em organizações que contam com fornecimento de energia não-confiável.

3.2 Ambiente da replicação

O ambiente de operação é uma rede de estações UNIX, com um número minoritário de servidores e um número majoritário de clientes. Os servidores existentes, rodando protocolo RNFS, podem conter volumes replicados e também oferecerem serviço NFS normal (não-replicado).

Cada volume replicado é denominado um **conjunto de replicação**, e os servidores que os contém formam um **grupo de replicação**. Podem existir diversos conjuntos de replicação simultaneamente, podendo também cada servidor participar de mais de um grupo de replicação. Os clientes podem estar simultaneamente conectados a vários grupos de replicação. Os servidores pertencentes a um grupo de replicação estão divididos em **servidor principal** e **servidores secundários**. A atribuição da condição de principal ou secundário é inicialmente estabelecida com base em um arquivo de configuração, mas pode mudar ao longo do uso do sistema.

Cada servidor, para cada grupo de replicação em que participar, estará em um determinado estado, cujos valores possíveis estão listado na tabela 1. Além disso, o servidor mantém uma **tabela de estados** contendo seu próprio estado, e o estado suposto (sob seu ponto de vista), de todos outros servidores membros do grupo. Esta tabela forma uma base de informação local sobre a condição corrente do grupo, e é usada pelos algoritmos descritos adiante. As tabelas são armazenadas nos servidores em memória volátil, sem necessidade de armazenamento estável. Após falhas do tipo *"crash"* ou em caso de inconsistência, as tabelas são reconstruídas explicitamente.

TABELA 1. - Valores possíveis para estados de servidores

Mnemônico	*Valor*	*Significado*
PRINCIPAL	0	Servidor mestre operacional
SECUNDçRIO	1	Servidor secundário em operação normal
ERRO_HARD	2	Servidor em falha física
ERRO_SOFT	3	Erro no sistema de arquivos remoto
RECUPERA,ÌO	4	Servidor está tendo seu conteúdo recuperado

Nos clientes, é mantido simplesmente um registro (também em memória volátil), de qual é o servidor principal suposto para cada conjunto de replicação a que o cliente estiver conectado. Esta tabela denomina-se **tabela de mestres**. Um exemplo de um cenário de replicação, envolvendo três servidores (um dos quais em falha), diversos clientes, e suas tabelas é mostrado na figura 1.

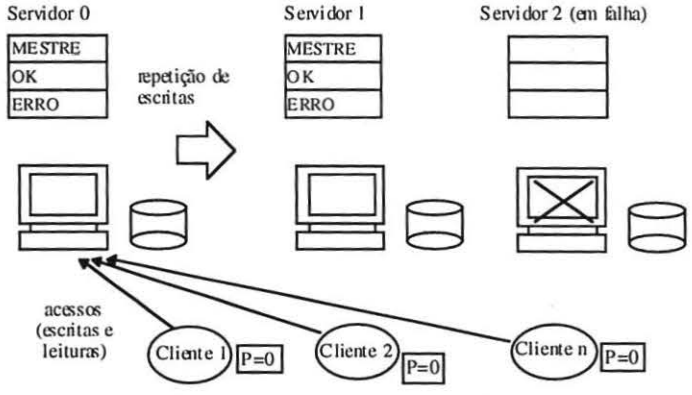

FIGURA 1 - Cenário de replicação e tabelas de estado.

3.3 Padrões interoperabilidade

No contexto UNIX, NFS significa uma especificação precisa de protocolo, documentada [RFC1094] e divulgada com vistas à interoperabilidade, e que não pode ser modificada sem concordância geral. Além disso, dentro dos padrões do SUN-RPC [RFC1057], usados como base para construção do serviço, cada aplicação RPC possui um nœmero identificador œnico, além de um nœmero de versão. Assim, para modificar o NFS é necessário definir um protocolo novo, mesmo que semelhante, afim de que os clientes possam detectar com qual tipo de servidor estão lidando.

Esta abordagem vem contra a intenção de permitir a interoperabilidade entre clientes NFS padrão e o RNFS, o que seria possível, uma vez que definimos o protocolo RNFS como uma extensão do NFS. Neste caso, os clientes não-modificados seriam capazes de acessar arquivos normalmente, inclusive mantendo replicações atualizadas, mas sem serem capazes de comutar entre servidores no caso de falhas. Esta intenção po de ser alcançada, desde que se force os números de identificação dos servidores a serem os mesmos, o que funciona corretamente, mas viola os padries do ambiente. Optamos por implementar o RNFS sem criar um novo número de aplicação, pois permite a interoperação, e não exige a criação de um conjunto completo de *drivers* novos no núcleo, embora signifique certa violação de convenções.

3.4 Estrutura de entidades

A implementação realizada modifica, além do protocolo de RPCs, o comportamento dos monitores que implementam os servidores, e o comportamento dos *drivers* de acesso remoto a arquivos no núcleo do sistema operacional de todas as estações clientes. Estes monitores [SAN85] são o rpc.nfsd e o rpc.mountd, que modificados passam a denominar-se rpc.ft-nfsd e rpc.ft-mountd (de *"fault tolerant"*).

A modificação nos clientes envolve definir um novo padrão de gerente de sistema de arquivos, denominado RNFS, dentro do nível VFS (*virtual file system*) [KLE86] do UNIX. A estrutura de clientes e servidores do RNFS está na figura 2. Para simplificar a implementação, optamos por modificar o *driver* NFS existente no núcleo das estações clientes. Desta forma evita-se definir um novo cliente VFS completo, que exigiria a produção de grande volume de código.

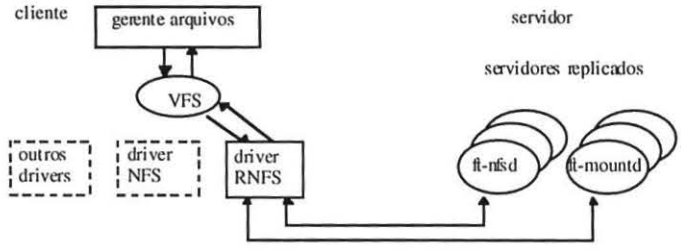

FIGURA 2 - Estrutura de serviço do RNFS

3.5 Representação de conjuntos de replicação

Para a representação dos conjuntos de replicação, definimos que todas as cópias de um volume replicado tem a mesma *path* física em cada servidor. Isto permite a fácil identificação das réplicas, formalizando que cada conjunto de replicação seja representado por uma lista de servidores, seguida de uma *path* única, válida em todos os servidores. Embora esta limitação reduza a flexibilidade de configuração, facilita a administração e simplifica a implementação dos algoritmos de replicação. Esta condição também permite que uma operação seja facilmente identificada como relativa a um volume replicado, pois a *path* indicada na operação será a mesma *path* geral escolhida para o volume replicado. A troca de servidores principais também é facilitada com esta escolha.

3.6 Configuração e administração

Todo o sistema de replicação é configurado a partir de um único arquivo de configuração adicional: o arquivo **replicate** (*\/etc/replicate*). Este arquivo descreve de forma simples cada conjunto de replicação, indicando sua *path* (igual em todos os servidores), e uma lista dos servidores que o contém. Um arquivo "replicate", como o listado na figura 3 contém blocos de texto, cada qual descrevendo um conjunto de replicação: cada bloco inicia por um linha com a palavra chave **replicate**, seguida da *"path"* completa (a partir da raiz dos servidores) que indica o volume replicado; a esta linha seguem-se outras, cada uma contendo o nome de rede de um dos servidores envolvidos na replicação.

Por princípio, todos os clientes e servidores possuem uma cópia equivalente do arquivo **replicate**. No UNIX, isto pode ser obtido com certa facilidade através do serviço NIS (*Network Information Service* [STE91]), responsável por difundir arquivos de configuração quando da partida de estações.

```
replicate /usr/data/critico        replicate /var/adm/configs
 sirius                             sirius
 antares                            antares
                                    perseus
```

FIGURA 3 - Exemplo de arquivo "replicate"

4. ALGORITMOS

Todas as atividades envolvidas na replicação são implementadas por um conjunto de cinco algoritmos cooperantes. São eles:

1. Acesso aos serviços de arquivo, com troca de servidor principal.
2. Repetição de escritas entre servidores.
3. Sinalização e monitoração de atividade dos servidores.
4. Eleição de novo servidor principal.
5. Recuperação entre servidores.

O primeiro algoritmo roda entre clientes e servidores. Os demais rodam somente nos servidores. Os quatro primeiros algoritmos são descritos a seguir. A implementação do algoritmo de recuperação definido para o RNFS é assunto específico de um projeto atualmente em andamento e não será tratado nesse artigo.

TABELA 2 - Procedimentos adicionados ao protocolo NFS

N.	*Nome*	*Natureza*	*Função*
18	IMALIVE	Broadcasting	Sinal periódico de sinalização de atividade
19	REQ_MASTER	Broadcasting	Requisição de novo principal a partir dos clientes
20	ELECTION	Unicasting	Pacote de inquirição usado no algoritmo de eleição
22	REQ_RECUP	Unicasting	Usado no algoritmo de recuperação de servidores
23	BVISION	Broadcasting	Difunde tabela com estado de um grupo de replicação

O sistema foi projetado visando implementar a tolerância a falhas através de extensões ao NFS. Desta forma, toda a comunicação adicional entre clientes e servidores, bem como comunicações servidor-servidor, foram implementadas usando RPCs, e adicionando cinco novos procedimentos aos 18 já existentes no protocolo NFS (tabela 2).

4.1 Acesso ao serviço de arquivos e chaveamento de servidor

Cada estação cliente no ambiente executa dois algoritmos interligados: o acesso ao serviço, e o chaveamento de servidor. O algoritmo de acesso ao serviço do RNFS é exatamente o mesmo do NFS, com a diferença de que, após um número determinado de *timeouts* sucessivos na comunicação com o servidor principal corrente (três, na implementação), este é invalidado, e a estação cliente entra em um estado de busca de novo servidor principal, através do seguinte algoritmo:

Após o servidor do volume ter sido invalidado, a estação cliente difunde um pacote REQ_MASTER, com a identificação do volume, solicitando um novo servidor mestre para o mesmo. Este pacote é retransmitido periodicamente, até que um servidor principal seja encontrado. Cada servidor que receber um pacote REQ_MASTER verifica simplesmente se é o servidor principal para o conjunto indicado, e em caso afirmativo retorna sua identificação. Em caso negativo, nenhum retorno é enviado. Quando o cliente recebe a resposta, o servidor que a tiver enviado é anotado incondicionalmente como novo servidor mestre do volume, e o cliente tenta continuar com os acessos ao volume dirigidos para este servidor. Adicionalmente, quando um servidor principal responde ao pacote de solicitação, após enviar a resposta ao cliente, é difundido um pacote BVISION, contendo sua visão da rede. Este pacote é útil para o algoritmo de eleição.

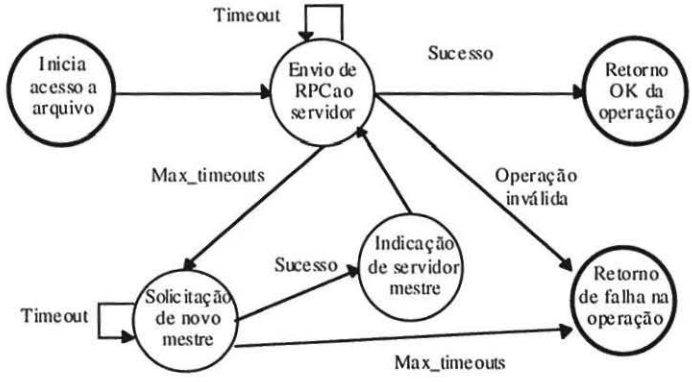

FIGURA 3 - Acesso RNFS no núcleo dos clientes

A figura 3 contém um diagrama de estados do processamento de um acesso RNFS no núcleo dos clientes, a figura 4 contém uma descrição em pseudo-código deste, e a figura 5 contém uma descrição do tratamento do pacote REQ_MASTER nos servidores.

4.2 Algoritmo de repetição de escritas entre servidores

Este é o algoritmo principal do sistema de replicação, executado em servidores com condição de principal, e responsável sobretudo por implementar a repetição de escritas nos servidores secundários. Outra atividade deste algoritmo é atualizar a tabela de estados de servidores cada vez que for detectada falha persistente em um servidor secundário. O núcleo deste algoritmo é a retransmissão das escritas recebidas no servidor principal para os demais servidores. Para este fim, o servidor realizando a repetição testa o "status" de cada chamada RPC incidente, com relação a um atributo de escrita. Caso o procedimento consista em uma escrita, é verificado através da tabela de replicação se o diretório em questão corresponde a um volume replicado. Em caso afirmativo, são substituídos os parâmetros do RPC, e o procedimento é reencaminhado a todos os demais servidores envolvidos nesta replicação, cujo estado na tabela de replicação seja "SECUNDÁRIO".

Os valores de retorno de cada procedimento repetido são anotados, e usados tanto para compor o valor final de retorno do procedimento ao cliente, como para manter o estado suposto do servidor na tabela de estados. A composição do valor de

retorno segue o princípio de "devolver o melhor resultado possível ao cliente". Assim, caso o resultado de uma escrita seja falha em algum servidor, este resultado é mascarado, e se houver pelo menos um número mínimo de servidores onde a operação possa ser realizada com sucesso, a operação retorna como corretamente finalizada ao cliente. Este número mínimo deve ser igual ou maior a uma maioria simples do número de servidores, de forma a garantir proteção contra particionamentos da rede.

```
Variáveis:
   Servidor_Mestre: Aponta o servidor mestre do
   volume na tabela de mestres.

faz_acesso_NFS( parâmetros )
{
   Loop:
   for( retrie_serv = 0; retrie_serv <3;
                        retrie_serv++ ) {
      // Solicita o acesso
      solicita_serviço( Servidor_Mestre,
                        parâmetros );
      // Aguarda resultados
      res = espera_retorno( TIMEOUT,
                        &result_data );
      se( res == OK || res == OP_INVALIDA )
         return( res );
      // TIMEOUT...
   }
   // Tenta trocar servidor
   for( retrie_troca = 0; retrie_troca < 3;
                        retrie_troca++ ) {
      difunde_solicitação_serviço(
               request_new_master, rep_id );
      res = espera_retorno( TIMEOUT,
                        &result_data );
      // Se o mestre respondeu, anota e termina
      se( res == NEW_MASTER ) { //
conseguiu
      novo mestre
         Servidor_Mestre =
               result_data.new_master_id;
         // reinicia
         goto Loop;
         }
      // nova tentativa
      }
   // falha geral
   return( NFS_ERR );
```

FIGURA 4 - Acesso RNFS no núcleo do clientes

```
   (...outros pacotes)
   // Bloco inserido no dispatcher de servico no
            servidor rpc.nfsd

   case REQMASTER :
      // anota identificação do conjunto de
         replicação
      Group_id =
         request_data.replication_group_id;
      se( master[ Group_id ] != MY_NUMBER
)
         // teste para iniciar algoritmo de eleição
         if( Req_master[ Group_id ]++ == 3 ){
            election();
         }
         (...)
         break; // apenas o mestre responde
      request_result(NEW_MASTER,
                        MY_NUMBER);
      // difunde visão da rede
      difunde_RPC( B_VISION, &Tabela de
                              estados );
      break;
   //
   case B_VISON :
      // cancela contagem de requisições de novo
                        servidor
      Req_master[
request_data.replication_group_id ] = 0;
      break;
   //
   (outros pacotes...)
}
```

FIGURA 5 - Tratamento do pacote REQ_MASTER no servidor RNFS

Para servidores assinalados como "SECUNDÁRIO" na tabela de estados e que não respondam a solicitações do servidor principal, a solicitação é repetida até um número máximo de vezes, após o que o servidor é anotado como em "FALHA_HARD", e deixa de receber mensagens do principal. O retorno de um estado de falha só pode ser feito pelo algoritmo de recuperação. A figura 6 contém o método de repetição de

escritas, em pseudo-linguagem. A estrutura mostrada nas listagens assim como os nomes de funções e variáveis são apenas ilustrativas, embora correspondam à lógica da implementação. Na prática, mais funções são chamadas para realizar tarefas como enviar mensagens, e traduzir endereços dos servidores de destino.

```
Constantes:
        NUM_SERVS:          Total de servidores nesta replicação
        MEU_NUMERO:                 Número do servidor dentro do conjunto
        MAX_RETRANS:                Máximo de retentativas no caso de timeout (3)
        EXPIR:              Timeout antes de fazer uma retentativa
enquanto( sempre ) {
        // aguarda solicitacao de algum cliente
        espera_servico( &dados_do_serviço );
        se( tipo_do_serviço == LEITURA ) { // leitura...
                resultado = serviço_local( &dados_do_serviço );
                retorna_serviço( resultado );
        }
        se( tipo_do_serviço == ESCRITA ) { // escrita...
                // para todos os servidores
                para( i=0; i<NUM_SERVS; i++ ) {
                        se( i == MEU_NUMERO ) continue;
                        // só transmite se OK
                        se( estado_servidor[i] != OK ) continue;
                        Loop:
                        // repete a solicitação
                        repete_requisição( servidor[i], dados_do_serviço );
                        espera_retorno( expiração, &codigo_de_retorno );
                        se( codigo_de_retorno == EXPIRA«vO ) {
                                se( max_retrans++ < MAX_RETRANS ) Loop;
                                        senao estado_servidor[i] = ERRO;
                        se( codigo_de_retorno != SUCESSO_RPC )
                                // outro erro
                                estado_servidor[i] = FALHA;
                }
        }
```

FIGURA 6 - Repetição de escritas a partir do servidor principal

4.3 Sinalização e monitoração de atividade de servidores

O algoritmo de informação de atividade visa iniciar a recuperação de um servidor que tenha estado desligado, por falha física, falta de energia, ou desconexão da rede. Este algoritmo consiste, em um lado, na emissão de um pacote periódico de "*I'm alive*", por todos os servidores que se encontram em operação. O outro lado consiste na recepção deste pacote (também por todos os servidores) resultando eventualmente na atualização da tabela de estados dos servidores.

A única função da recepção de um pacote de informação de atividade em um servidor a partir de outro é sinalizar a religada do servidor remoto e provocar a mudança do seu estado suposto de "FALHA_HARD" ou "FALHA_SOFT" para "RECUPERA-«vO". A falta na recepção deste pacote não provoca, de qualquer forma, a retirada do servidor faltoso de um eventual estado de "SECUNDÁRIO". Isto é provocado apenas pela falta em atender mensagens de irradiação de escritas do servidor principal.

O uso de pacotes periódicos de *broadcasting* é necessário, uma vez que, conforme enunciado, a religada de um servidor pode não ser sinalizada por nenhum evento

específico. Em particular pode-se citar a reconexão de um cabo de rede, que pode reintegrar um servidor ao grupo sem qualquer aviso. Desta forma, é mais adequado não depender de uma rotina especial de religada, quando se quiser que o sistema possa realizar recuperação automática.

De forma a não onerar o sistema, é razoável usar uma frequência bastante baixa na emissão dos pacotes de sinalização. Uma vez que o sistema é tolerante, o procedimento de recuperação não necessita ser iniciado imediatamente após a religada, uma vez que o acesso aos dados não depende disto. Na implementação foi utilizado um intervalo de cinco minutos entre cada pacote.

4.4 Algoritmo de eleição de novo servidor principal

O algoritmo de eleição é responsável por escolher um novo principal para um dado conjunto de replicação entre os servidores secundários restantes, no caso de falha no principal corrente. Este algoritmo também é responsável por resolver possíveis conflitos de estados que resultem em contextos inconsistentes. O algoritmo de eleição usado é uma modificação do algoritmo de eleição por domínio, descrito em [GAR82].

A escolha de algoritmos de eleição visando tolerância a falhas apresenta a dificuldade de que falhas podem ocorrer de modo arbitrário, gerando particionamentos na rede, e em especial, novas falhas podem ocorrer enquanto o algoritmo de eleição está em andamento, e devem ser tratadas. O algoritmo de eleição escolhido:

- Deve iniciar sob demanda dos clientes.
- Deve prever a possibilidade de que quaisquer mensagens possam ser perdidas.
- Deve garantir a proteção contra particionamentos, ou seja, em qualquer circunstância pode existir no máximo um servidor principal.

O início do algoritmo de eleição por demanda dos clientes é a maneira mais simples de controlar sua partida: uma eleição só inicia se um cliente tentar sem sucesso e repetidamente acessar o servidor anotado principal, e simultaneamente nenhum outro servidor se habilitar como tal. A segunda exigência é necessária de acordo com o modelo de falhas: no caso de perda de mensagens a eleição será abortada sem que nenhum estado seja alterado. Esta propriedade, combinada com as retentativas periódicas de obtenção de servidor principal por parte dos clientes, leva à tolerância a falhas no processo de eleição, e garante que em algum momento uma eleição terá sucesso, se as condições mínimas da rede o permitirem.

O algoritmo de eleição é iniciado pela recepção por algum servidor de pacotes REQ_MASTER emitidos por qualquer cliente. Caso o servidor a receber o pacote já seja o principal para o conjunto de replicação solicitado, este servidor responde ao RPC informando seu número e sua condição de servidor principal. Quando um servidor secundário recebe o pacote REQ_MASTER, é aguardado um determinado número de retransmissyes sem que seja recebido o pacote de difusão de visão da rede BVISION. Caso o pacote não seja recebido, o servidor secundário inicia o procedimento de eleição.

O algoritmo de eleição opera da seguinte maneira: quando um servidor secundário inicia uma eleição, envia sequencialmente a todos os outros servidores do grupo

um pacote ELECTION, informando o conjunto de replicação e o seu número de prioridade dentro deste (dado pelo arquivo **replicate**). Cada servidor que receber um pacote ELECTION, pode concordar ou discordar, em função de seu próprio número de prioridade. Se seu número for menor que o número de prioridade do proponente, o servidor concorda e retorna um valor indicativo desta opção. Caso a prioridade do servidor que receber o pacote seja maior que a do proponente, o servidor retorna um indicativo de discordância, e inicia ele próprio (se ainda não tiver iniciado) um algoritmo de eleição. Desta forma, a eleição iniciada em um servidor qualquer vai sendo transferida até atingir o servidor de maior prioridade dentro do grupo que se mantiver conectado.

Uma eleição é completada quando um servidor obtiver pelo menos uma maioria simples de concordâncias (metade do número de servidores configurados para o grupo, mais um). Ao obter vitória na eleição, o servidor torna-se principal imediatamente. Caso não seja obtida maioria simples, ou ocorra *timeout* enquanto estiver aguardando por um número suficiente de respostas que permita tomar uma decisão, a eleição falha, e o servidor mantém o seu estado anterior, podendo reiniciar o algoritmo de eleição, caso a condição de partida se repita.

Se um servidor que receber o pacote ELECTION, por qualquer razão, já for o servidor principal do grupo, envia como resposta uma mensagem solicitando o cancelamento da eleição. Uma alternativa é: caso a prioridade deste servidor seja menor que a do proponente, inicie-se um processo de **demoção** do servidor principal.

Uma tabela de transições de estados para o algoritmo de eleição é mostrado na figura 7. A figura 8 (parte 1) inclui o código responsável pelo início da eleição, e a parte 2 da mesma figura mostra o tratamento dos pacotes de eleição nos servidores.

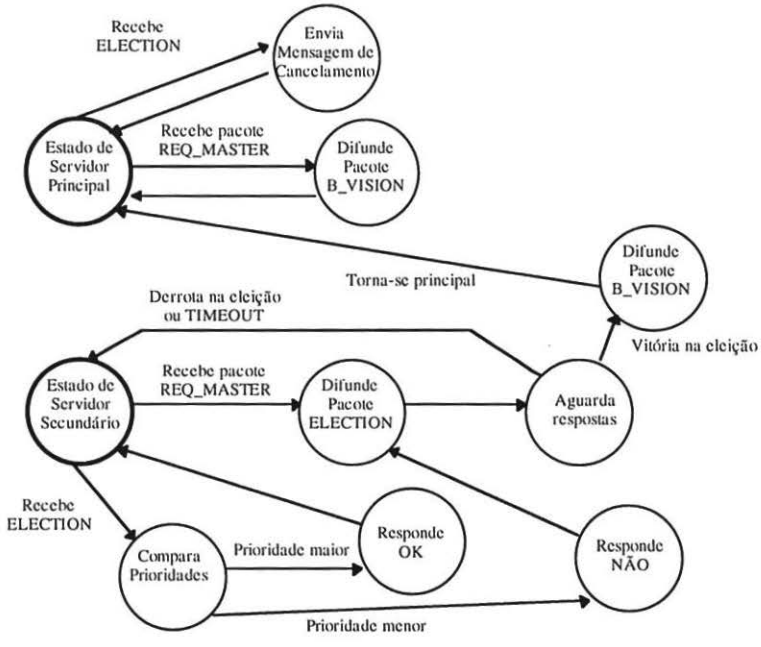

FIGURA 7 - Transições de estados no algoritmo de eleição

```
Parte 1: Coordenação da eleição
eleição( )
{
        // difunde ELECTION
        difunde_solicitação_serviço( ELECTION, rep_id );
        // aguarda retornos
        ok = 0; itemout = 0; cnt = 0;
        while( cnt < NSERVERS ) {
                res = espera_retorno( TIMEOUT, &result_data );
                if( res == IS_TIMEOUT ) {
                        timeout++;
                        break;
                }
                if( result_data.res = ELECTION_OK ) ok++;
                cnt++;
        }
        // testa se obteve quórum
        if( ok >= NSERVERS / 2 )
                // torna-se mestre
                Estado[ MY_NUMBER ] = MESTRE;
}
Parte 2: Tratamento do pacote ELECTION nos servidores
        (...outros pacotes)
        // Bloco inserido no dispatcher de servico no servidor rpc.nfsd
        case ELECTION :
                // anota identificação do conjunto de replicação
                Group_id = request_data.replication_group_id;
                se( master[ Group_id ] == MY_NUMBER ) {
                        // já e'o servidor principal
                        request_result( END_ELECTION );
                        break; // apenas o mestre responde
                }
                // é servidor secundário: compara prioridades
                if( request_data.priority > MY_NUMBER )
                        request_result( ELECTION_OK );
                                else    {
                                        request_result( ELECTION_NO );
                                        // inicia uma eleição
                                        election();
                                }
                break;
        (outros pacotes...)
```

FIGURA 8 - Procedimento de eleição

4.5 Tolerância a falhas no procedimento de montagem

Os procedimentos de montagem e desmontagem de volumes devem preceder (e suceder, respectivamente) o uso de qualquer volume local ou remoto do sistema de arquivos. No caso de volumes NFS, o procedimento de montagem tem pouco significado, limitando-se a retornar autorizações de acesso aos volumes solicitados ao servidor, e o de desmontagem simplesmente remove o cliente de uma lista de clientes ativos mantida em cada servidor. De qualquer forma, para maior confiabilidade, o atendimento a solicitações de montagem e desmontagem deve também ser tornado tolerante a falhas, pois o uso de um volume remoto NFS pelo cliente não inicia caso o

procedimento de montagem correspondente não ocorra com sucesso. Da mesma forma, a falha no atendimento à requisições de desmontagem leva ao bloqueio, ou a grandes atrasos, no procedimento de *shutdown* das estações.

O atendimento às operações de montagem é implementado no NFS através dos monitores *mountd*, e a solução consiste em replicar também o serviço destes monitores. A princípio, a replicação dos monitores *mountd* constitui uma tarefa mais simples do que a do serviço de arquivos do NFS, pois estes monitores armazenam muito pouca informação, e seu protocolo com os clientes também é *stateless*.

Uma primeira solução é utilizar os monitores *mountd* apenas como servidores de autorização (sua única função prática), e implementar tolerância a falhas apenas para o procedimento de montagem. A tolerância a falhas seria implementada nos clientes através da varredura de todos os possíveis servidores de um volume requisitado (com base no arquivo **replicate**) até que seja encontrado algum capaz de fornecer as autorizações. Este método parece suficiente, sem necessidade de qualquer difusão de operações entre os servidores, pois (conforme mencionado) a informação de quais clientes estão montados em quais servidores tem pouca ou nenhuma utilidade prática. Além disso, o procedimento de desmontagem poderia ser implementado com um *timeout* curto, retornando apenas uma advertência caso o servidor esteja inativo, uma vez que sua utilidade é quase simbólica. Um método mais estrito, todavia, deveria incluir a difusão dos eventos de montagem/desmontagem entre todos os servidores *mountd*, de forma a manter consistentes as tabelas de usuários de volumes, e realizar varredura de servidores também na desmontagem.

A implementação da primeira solução é relativamente simples, pois as mudanças se restringem aos clientes. Nos clientes, a montagem e desmontagem é realizada por um programa executável especial, denominado **comando mount**. Este comando realiza chamadas RPC aos servidores *mountd*, e caso haja retorno de autorizações, gera as chamadas de sistema necessárias ao registro do volume de arquivos no *kernel* local. A mudança consiste apenas em implementar uma consulta ao arquivo **replicate**, com varredura dos servidores listados, caso o primeiro servidor não esteja disponível. O comando *mountd* falhará apenas no caso de falha em todos os servidores do grupo.

5. AVALIAÇÃO DO RNFS

5.1 Algoritmos de replicação e tratamento de falhas

O algoritmo de cópia principal parece ser atualmente o melhor algoritmo para replicação de arquivos, o que é confirmado tanto pelas experiências realizadas como pelo volume de trabalhos que o utilizam [BRE86], [LAD90], [LIS91], [PUR87]. Especial cuidado deve ser tomado na escolha do método de difusão de escritas empregado dentro do algoritmo, de forma a tentar-se reduzir a perda de desempenho para escritas, implícita a esta família de algoritmos. Embora seja o centro de toda discussão sobre replicação, o algoritmo de replicação por si só não completa o funcionamento de um sistema de arquivos replicados. As técnicas adicionais responsáveis pelo efetivo mascaramento de falhas, quais sejam, a troca de servidor principal, a eleição de novo servidor principal, e o controle de estados dos servidores são igualmente essenciais ao funcionamento do sistema, e na prática se revelaram de projeto mais difícil, em parte por haver mais alternativas para sua implementação. Em particular, um modelo de falhas amplo aplicado a um sistema distribuído leva a um número muito grande de possíveis combinações de falhas, o que torna difícil assegurar a correção de um

algoritmo de cobertura para todas circunstâncias. Os algoritmos aqui propostos são aparentemente corretos para todos os casos estudados, mas existe carência de um método formal para provar a correção de algoritmos distribuídos sob a presença de falhas. Prevê-se aplicar técnicas de injeção de falhas [SOT97] para validar o RNFS.

5.2 Manutenção de semânticas de operações sob replicação

As semânticas para operações em arquivos nativas do sistema operacional UNIX são violadas a partir do sistema NFS utilizado como base, no que diz respeito à consistência para acessos concorrentes. Estas violações, embora afetem a correção formal do sistema, tem pouco efeito prático e quase nunca são percebidas pelos usuários. O RNFS não piora nem melhora estas violações semânticas, uma vez que toda concorrência de acesso é resolvida a nível de servidor principal, da mesma forma que no NFS padrão, como demonstrado [SRI89]. Um protocolo explícito de consistência de cache nos clientes traria correção ao NFS (e ao RNFS), com pouco ou nenhum prejuízo de performance, mas não proveria nenhuma vantagem direta a nível de replicação de dados.

Outra questão diz respeito à semântica de operações na presença de falhas. Vários sistemas replicados buscam obter consistência a nível de transações, o que significa que toda operação que retorne ao cliente como bem sucedida efetivamente estará gravada em meio estável (tolerante a falhas), mesmo que a operação ocorra no momento de uma falha. Esta especificação, exigiria um algoritmo de operação atÙmica na distribuição de atualizações, o que reduz muito a performance do sistema [LAD90]. O RNFS assume que esta consistência é pouco importante na maioria dos casos, e pode ser substituída por um algoritmo que assegure uma probabilidade de erro muito baixa, com eficiência bem maior.

5.3 Controle de versyes nos arquivos

O uso de um sistema de arquivos de base que contenha números de versão para todos os arquivos e diretórios do sistema sem dúvida facilitaria a operação do sistema de replicação de dados. Em especial, seria facilitada a recuperação de servidores, tornando fácil a implementação do modelo de recuperação diferencial. A escolha do servidor a ser usado como fonte de uma recuperação pode também ser tomada a partir de números de versão gerais de volumes, na falta de outro critério. O controle de versyes, contudo, não é essencial à implementação do sistema, apenas aumenta a quantidade de informações disponíveis para controle da consistência. Sua inclusão depende de um balanceamento de custo entre a perda de eficiência que acarreta e as vantagens objetivas. A abordagem mais simples para adicionar números de versão é introduzi-los entre os dados contidos no *inode* do arquivo. Para a atualização dos números, pode-se utilizar um protocolo de *write-ahead-log*. Este protocolo garante atomicidade, com menor desempenho. Alternativamente, pode-se utilizar um algoritmo de "escrever primeiro, incrementar depois", que assegura que os números refletem *pelo menos* a última operação corretamente realizada, o que pode ser suficiente para o algoritmo de recuperação.

5.4 Estado da implementação

No momento está finalizada a implementação dos servidores replicadores (rpc.rnfsd), estando operacional a difusão de escritas do servidor principal para os secundários. A tradução de parâmetros nos acessos é realizada pela substituição de *strings*

nos procedimentos que as utilizam, e pela utilização de *handlers* temporários para cada operação de escrita a ser difundida. Esta sob implementação um mecanismo mais eficiente para tratamento de *handlers* de arquivos nos servidores, através da extensão do mecanismo de cache de *handlers* existente para uma versão capaz de gerenciar *handlers* múltiplos, relacionados a arquivos replicados.

Os núcleos das máquinas clientes também foram modificados para opcionalmente gerarem o pacote de solicitação de novo servidor principal após um número determinado de falhas de acesso, sendo capazes também de receber e processar a indicação de novo servidor, embora o protocolo de troca não esteja ainda implementado nos servidores.

A fim de completar a implementação da especificação para o RNFS falta sobretudo realizar a implementação do algoritmo de eleição que comanda a troca de servidor principal, e a conexão dos servidores com os pacotes de solicitação vindos dos clientes. Estando verificada esta parte, resta implementar o algoritmo básico de recuperação por transferência de volumes, com o protocolo de inicialização que o acompanha.

6. BIBLIOGRAFIA

[BHI90] BHIDE, A.; ELNOZAHY, E.; MORGAN, S. Implicit Replication in a Network File Server. In: WORKSHOP ON MANAGEMENT OF REPLICATED DATA, 1990, Houston. Washington: IEEE Computer Society Press, c1990.

[BRE86] BRERETON. O . P. Management o f Replicated Files in a UNIX Environment. **Software - Practice and Experience,** Sussex, England, v.16, p.771-780, Aug. 1986.

[DAV85] DAVIDSON, S. B.; GARCIA-MOLINA, H. Consistency in Partitioned Networks. **ACM Computing Surveys,** New York, v. 17, n. 3, p.341-370, Sept. 1985.

[ELA85] EL-ABBADI, A; SKEEN, A. D; CRISTIAN, F. An Efficient Fault-Tolerant Protocol for Replicated Data Management. In: SYMP. ON PRINCIPLES OF DATABASE SYSTEMS, 4., [S.l.], 1985. **Proceedings...** NY ACM Press, 1985. p.215-229.

[GAR82] GARCIA-MOLINA, H. Elections in a Distributed Computing System. **IEEE Trans. on Computers,** New York, v. C-31, n.1, p.48-59, Jan. 1982.

[GIF79] GIFFORD, D. K. Weighted Voting for Replicated Data. In: SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 7., 1979, New York. **Proceedings...** New York; ACM Press, 1979. p.150-162.

[HIS90] HISGEN, A. et al. Granularity and Semantic Level of Replication in the Echo Distributed File System. In: WORKSHOP ON MANAGEMENT OF REPLICATED DATA, 1990, Houston. Washington: IEEE Computer Society Press, c1990.

[HUA89] HUANG, Y.; JALOTE, P. Avaliability Analysis of the Primary Site Approach for Fault Tolerance. **Acta Informatica,** Berlin, n. 26, p.543-557,1989.

[JAL94] JALOTE, Pankaj. **Fault Tolerance in Distributed Systems.** Englewood Cliffs, NJ.: Prentice-Hall, 1994. 432p.

[KLE86] KLEIMAN, S. Vnodes: An Architecture for Multiple File System Types in SUN UNIX. In: USENIX SUMMER '86 CONFERENCE, 1986. [S.l.: s.n.], p. 238-247.

[KUM91] KUMAR, A. Hierarchical Quorum Consensus: A New Alghoritm for Managing Replicated Data. **IEEE Trans. on Computers,** New York, v. 40, n. 9, p.996-1004, Sept. 1994.

[LAD90] LADIN, Rivka; LISKOV, Barbara; SHRIRA, Liuba. Lazy Replication: Exploiting the Semantics of Distributed Services. **Operating Systems Review,** New York, v.25, n.1, p.49-55, Jan. 1991.

[LEV90] LEVY, E; SILBERSCHATZ, A. Distributed File Systems: Concepts and Examples. **Computing Surveys,** New York, v. 22, n. 4, p.321-374, Dec. 1990.

[LIS91a] LISKOV, Barbara et al. Replication in the Harp File System. **Operating Systems Review,** New York, 1991, v. 25, n 5, p. 226-238.

[LIS91] LISKOV, Barbara et al. A Replicated Unix File System. **Operating Systems Review,** New York, v.25, n.1, p.60-64, Jan. 1991.

[NEL88] NELSON, M. N; WELCH, B. B; OUSTERHOUT, J.K. Caching in the Sprite Network File System. **ACM Trans. on Computer Systems,** New York, v. 6, n.1, p.134-154, Feb. 1988.

[PUR87] PURDIN, T. D. M.; SCHLICHTING, R. D.; ANDREWS, G.R. A File Replication Facility for Berkeley UNIX. **Software - Practice and Experience,** Sussex, England, v.17, p.923-940, Dec. 1987.

[RFC1014] IETF Request For Comments 1014. **XDR: External Data Representation Standard.** Disponível no servidor www.internic.net.

[RFC1057] IETF Request For Comments 1057. SUN RPC Protocol Especification. Disponível no servidor www.internic.net.

[RFC1094] IETF Request For Comments 1094. **The NFS Protocol Especification.** Disponível no servidor www.internic.net.

[RFC1813] IETF Request For Comments 1813. **NFS Version 3 Protocol.** Disponível no servidor www.internic.net.

[SAN85] SANDBERG, R. et al. Design and Implementation of the Sun Network File System. In: USENIX ASSOCIATION CONFERENCE, 1985, Berkeley. **Proceedings...** Berkeley: USENIX, 1985.

[SOT97] SOTOMA, IRINEU; WEBER, TAISY SILVA. AFIDS - Arquitetura para Injeção de Falhas em Sistemas Distribuídos. In: **Anais** do 15 SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES - XV SBRC. UFScar - Universidade Federal de São Carlos. 19-22 de maio de 1997. São Carlos - SP. p294-309.

[SRI89] SRINIVASAN, V; MOGUL, J. C. Spiritely NFS: Experiments with Cache-consistency Protocols. **Operating Systems Review,** New York, v. 23, n.5, p. 45-57, Dec. 1989.

[STE91] STERN, Hal. **Managing NFS and NIS.** Sebastopol, Ca: O'Reilly & Associates, 1991. 410p.

[TAN92] TANENBAUM, Andrew S. **Modern Operating Systems.** Englewood Cliffs, NJ. Prentice-Hall, 1992. 728p.