Network File System (NFS)

Prezentacja rozproszonego systemu plików

Konrad Iwanicki

10 grudnia 2002

Spis treści

1 Wprowadzenie

1.1 Historia

NFS (z ang. *The Network File System*), czyli sieciowy system plików został stworzony przez firmę Sun Microsystems w latach osiemdziesiątych jako wysoce wydajne (jak na owe czasy) rozwiązanie udostępniające poprzez sieć lokalną system plików na komputerach klienckich nie wyposażonych w dyski twarde.

NFS jest otwartym standardem implementującym architekurę klient-serwer dla dzielenia plików, który jest wspierany przez rozmaite systemy: od komputerów osobistych, poprzez stacje robocze, na „mainframe'ach" kończąc. Pierwsza wersja została zaprezentowana w roku 1985. Ciekawostką jest fakt, że miała ona numer 2, ponieważ wersja o numerze 1 nie została nigdy opublikowana.

Od tego momentu NFS ewoluował i rozwijał się, by spełnić nowe wymagania, które pojawiły się na początku lat dziewięćdziesiątych (dotyczące zwłaszcza rozproszonego systemu plików dla sieci globalnych).

Upowszechnił się szeroko w środowiskach przemysłowych, akademickich oraz laboratoriach badawczych, jako że firma Sun postanowiła udostępnić definicje podstawowych interfejsów oraz przykładową implementację. Ponad 300 firm i organizacji dołączyło do swoich produktów wsparcie dla tego standardu, więc jest on dostępny na niemal wszystkie platformy programowe (m.in. VMS, MS-DOS, Windows, Unix, Solaris) i sprzętowe (m.in. Intel, SPARC, Alpha).

Aktualnie używanymi wersjami są NFS2 oraz NFS3 (z roku 1995; najpopularniejsza; używana m.in. przez Red Hat'a 7.3). Aktualnie projektowana jest czwarta wersja NFS. Prace rozpoczęły się w połowie 1999 roku. W listopadzie 2002 roku firma Sun udostępniła nową specyfikację.

1.2 Najważniejsze cechy

Najważniejszymi cechami NFS, które zadecydowały o jego sukcesie są:

- **Skalowalność** – NFS współpracuje zarówno z małymi jak i dużymi sieciami lokalnymi.
- **Przezroczystość** – W wielu różnych aspektach.
- **Wydajność** – Wersje 3 i 4 NFS mają poprawioną wydajność.
- **Buforowanie** – (z ang. *local disk caching*) – pozwala na znacznie szybszy dostęp do plików.
- **Scentralizowanie administracji** – Redukuje koszty czasowe potrzebne do wykonania rutynowych czynności administratora systemu.
- **Wsparcie różnych klientów** – NFS wspiera zarówno bezdyskowe i bezdanowe systemów klienckie, jak i nowy alternatywny system kliencki nazywany *„The Solstice AutoClient"*.

- **Bezpieczeństwo** – NFS posiada mechanizmy uniemożliwiające dostęp do danych osobom niepowołanym do ich oglądania.

1.3 Istniejące implementacje

W poniższej tabelce zostały wymienione najważniejsze implementacje NFS.

Producent	**System operacyjny**
Amdahl	UTS
Apple	A/UX
Beame and Whiteside	DOS, Windows
BSDI	Unix
Cray	UNICOS
DEC	Ultrix, VMS
Dell	SVR4
FTP Software	DOS, Windows, OS/2
Frontier Technology	DOS, Windows
Hewlett-Packard	HP-UX
IBM	AIX, MVS
Intel	Unix
ICL	Unix
Net Manage	DOS, Windows
Nixdorf	TOS 35
Novell	Netware
OSF	OSF1
Santa Cruz Operation (SCO)	SCO Unix
SunSoft	Solaris, DOS, Windows
Silicon Graphics	IRIX
Process Software	VMS
Sony	NeWS
Texas Instruments	TI SVR3.2
TGV	VMS

2 Założenia projektowe

W tym rozdziale wymienione zostały najważniejsze przesłanki, które przyświecały zespołowi projektującemu NFS.

2.1 Przezroczystość

Przezroczystość (z ang. *transparency*) oznacza, że użytkownicy i aplikacje mogą używać zdalnych plików tak, jakby znajdowały się one lokalnie (system jest postrzegany jako całość bez wyraźnie zaznaczonych składowych).

Różne rodzaje przezroczystości (wg Modelu Wzorcowego Otwartego Przetwarzania Rozproszonego (z ang. *Reference Model for Open Distributed Processing*) Biura Międzynarodowych Standardów (z ang. *International Standards Organization*):

- **dostępu** – umożliwia dostęp do lokalnych i odległych obiektów informacji za pomocą identycznych działań;
- **położenia** – umożliwia dostęp do obiektów informacji bez znajomości ich lokalizacji;
- **współbieżności** – umożliwia wielu procesom niezakłócone działanie z użyciem wspólnych obiektów informacji;
- **zwielokrotniania** – pozwala na użycie wielu kopii obiektów informacji w celu zwiększenia niezawodności i wydajności bez wiedzy użytkowników i programów użytkowych o zwielokrotnieniach;
- **awarii** – umożliwia wykrywanie uszkodzeń, pozwalając użytkownikom i programom użytkowym na kończenie zadań pomimo awarii sprzętu lub składowych oprogramowania;
- **wędrówki** – pozwala na przemieszczanie obiektów informacji w obrębie systemu bez wpływu na działania użytkowników lub programów użytkowych;
- **wydajności** – umożliwia rekonfigurowanie systemu w celu poprawy działania przy zmianie obciążenia;
- **skalowania** – umożliwia systemowi i jego zastosowaniom rozszerzanie skali bez zmiany struktury systemu lub algorytmów użytkowych.

Oto jak NFS realizuje założenia przezroczystości.

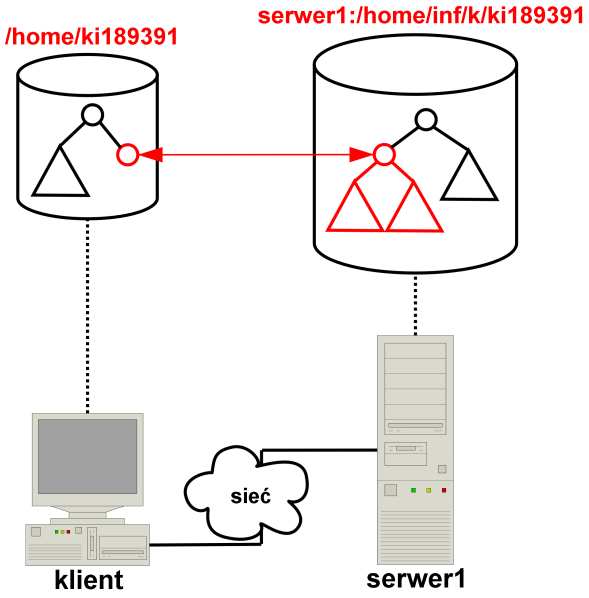
2.1.1 Przezroczystość dostępu

Moduł klienta systemu NFS dostarcza lokalnym procesom interfejsu programowania aplikacji, który jest identyczny jak interfejs lokalnego systemu operacyjnego. Na przykład linuksowy klient korzysta ze zdalnych plików używając przy tym zwykłych odwołań do (funkcji) systemu Linuks. Istniejące aplikacje mogą wykorzystywać zdalne systemy plików bez żadnych zmian kodu źródłowego.

2.1.2 Przezroczystość położenia

Każdy klient określa sieciową przestrzeń nazw plików przez dodanie zdalnych systemów plików do swojej lokalnej przestrzeni nazw. Zanim procesy działające na maszynie klienta zaczną korzystać ze zdalnych plików, węzły utrzymujące te systemy plików muszą je wyeksportować, zaś klient musi je zdalnie zamontować (z ang. *remote-mounted*). Miejsce w hierarchii nazw klienta, w którym pojawi się zdalnie zamontowany system plików (tzw. punkt montowania z ang. *mount point*), jest wybierane przez klienta. Ciekawostką jest to, że w przypadku systemu MSDOS i systemów z rodziny Windows klient jest zmuszony zamontować zdalny system plików jako osobny dysk twardy. System NFS nie wymusza więc jednej, wspólnej dla całej sieci przestrzeni

Rysunek 1: Przykład montowania



nazw – każdy klient widzi zbiór zdalnych systemów plików w sposób określony lokalnie, zatem pliki odległe mogą mieć różne ścieżki dostępu u różnych klientów. Można jednak ustalić jednolitą przestrzeń nazw z odpowiednimi tabelami konfiguracyjnymi u każdego klienta osiągając przezroczystość położenia.

2.1.3 Przezroczystość awarii

Tutaj uwidaczniają się różnice pomiędzy wersją 4 a wcześniejszymi wersjami NFS.

Wersje 2 i 3
Bezstanowa i idempotentna (operacje są powtarzalne) natura protokołu dostępu do plików zapewnia, że sytuacje awaryjne awaryjne pojawiające się u klientów korzystających ze zdalnych plików są podobne do występujących podczas lokalnego dostępu do plików.

Gdy serwer ulega awarii, świadczone przez niego usługi zostają zawieszone do czasu jego ponownego uruchomienia, lecz kiedy to nastąpi procesy klientów kontynuują działanie od punktu, w którym usługi zostały przerwane, nieświadome awarii (z wyjątkiem przypadku dostępu do miękko zamontowanych – z ang. *soft-mounted* – zdalnych systemów plików).

Awaria komputera klienta lub procesu poziomu użytkowego u klienta nie wpływa na działanie żadnego używanego serwera, gdyż serwery nie przechowują żadnych informacji o stanie swoich klientów.

Wersja 4
W najnowszej wersji NFS odtwarzanie po awarii jest nieco bardziej skomplikowane, ponieważ system przechowuje informacje o blokadach założonych na fragmenty plików (patrz rozdział 3.9). Dlatego też NFS 4 nie oferuje przezroczystości awarii.

2.1.4 Przezroczystość wydajności

Zarówno klient, jak i serwer stosują pamięć podręczną w celu osiągnięcia zadowalającej wydajności. Intensywne używanie pamięci podręcznych oraz instalowanie modułów klienta i serwera w jądrze systemu Linux powodują różnice rzędu 20 procent w szybkości dostępu do zdalnych plików lekko obciążonych serwerów (dane firmy Sun) w porównaniu z dostępem do plików lokalnych na podobnych urządzeniach pamięci.

Przechowywanie w pamięci podręcznej na serwerze jest proste – w przypadku Linuksa używa się standardowego mechanizmu pamięci podręcznej bloków dyskowych. Moduł klienta utrzymuje lokalną pamięć podręczną bloków zdalnych plików, katalogów i danych opisujących atrybuty plików. Utrzymanie spójności pamięci podręcznych klientów jest bardzo złożony i dlatego zostanie szczegółowo opisany dalej.

2.1.5 Przezroczystość wędrówki

Usługi dostępu do plików NFS są uzupełniane przez oddzielną usługę montowania (z ang. *mount service*), która umożliwia osadzanie odległych systemów plików w lokalnej przestrzeni nazw plików klienta. Z usług montowania korzystają przede wszystkim programy odpowiedzialne za administrowanie systemem do ustalania przestrzeni nazw plików w komputerach klientów podczas rozruchu systemu lub podczas logowania użytkowników na stacjach roboczych. Systemy plików (w Linuksie – poddrzewa plików; w Windowsach – osobne dyski) mogą być przenoszone między serwerami, ale wtedy tabele zdalnych montaży u każdego klienta muszą być oddzielnie uaktualniane, aby umożliwić dostęp do systemu plików w nowym miejscu.

Bardzo przydatnym narzędziem używanym do montowania zdalnych systemów plików jest *automounter* opisany w dalszej części dokumentu.

2.2 Przenośność i niezależność od protokołu sieciowego

NFS jest niezależnych zarówno od platformy sprzętowej, jak i systemu operacyjnego. To sprawia, że może on być łatwo przeniesiony na wiele różnych maszyn.

Może ponadto działać z wykorzystaniem wielu istniejących protokołów (chodzi tu warstwę transportu) jak np. UDP i TCP. Ma również możliwość współpracy z protokołami wymyślonymi w przyszłości. Ta niezależność bierze się z faktu, że NFS zbudowany jest w oparciu o TI-RPC (z ang. *Transport Independent Remote Procedures Calls*), czyli zdalnego wywoływania procedur niezależnego od warstwy transportu, w której to metodologii zostały zaimplementowane wszystkie procedury.

2.3 Szybki powrót do pracy po awarii

NFS został zaprojektowany tak, aby jak najszybciej wrócić do normalnego stanu po awarii, co powoduje jedynie minimalną przerwę w dostawie usług do klientów.

2.4 Wydajność

Projekt NFS zakładał wysoką wydajność systemu, tj. taką, aby czas dostępu do zdalnych plików był porównywalny do czasu dostępu do plików lokalnych (w przypadku lokanych sieci małej i średniej wielkości).

2.5 Bezpieczeństwo

Architektura systemu pozwala na wykorzystanie wielu mechanizmów bezpieczeństwa. Administratorzy nie są więc ograniczeni do konkretnego mechanizmu, lecz mogą wybrać ten, który najbardziej odpowiada ich środowisku pracy. Rozwiązanie to pozwala także na użycie mechanizmów wymyślonych w przyszłości.

2.6 Innowacje w kolejnych wersjach

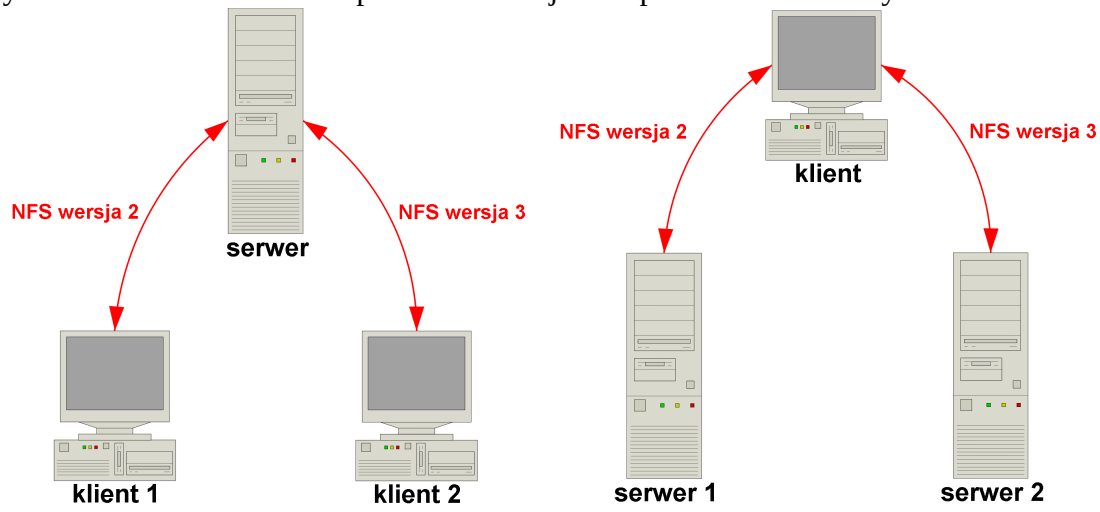
2.6.1 Wersja 3

W roku 1992 grupa organizacji (m.in. IBM, Digital, SunSoft) wyszła z propozycją zdefiniowania kolejnej wersji protokołu NFS. W rezultacie powstał NFS3 jako rozszerzenie wersji drugiej. Najważniejszymi ulepszeniami w tej wersji były:

- poprawiona wydajność operacji zapisu danych przez klienta,
- zredukowane obciążenie serwera poprzez zwiększoną skalowalność oraz poprawę wydajności,
- ulepszone wsparcie dla systemów korzystających z ACL (z ang. *Access Control List* – patrz rozdział 5),
- dodane wsparcie dla dużych (kilkugigabajtowych) plików na serwerach NFS.

Dzięki oczywistym korzyściom płynącym z wprowadzenia trzeciej wersji systemu większość jego największych użytkowników postanowiła przejść na nowy protokół. Jednakże, aby zachować zgodność z dotychczasowo używanymi wersjami możliwe jest zaimplementowanie NFS tak, aby obsługiwał równocześnie oba protokoły. Przykładowo, w Solaris'ie (wersja 2.5) klient i serwer będą mogły „wynegocjować", którego protokołu mają używać na podstawie tego, jakie protokoły wspierają (patrz rysunek 2.6.1).

Rysunek 2: Równoczesne wsparcie dla wersji 2 i 3 protokołu NFS w systemie Solaris 2.5



2.6.2 Wersja 4

NFS w wersjach 2 i 3 pomimo swoich zalet i popularności posiadał jednak znaczące wady. Uciążliwy był brak spójnego mechanizmu pozwalającego w ramach protokołu blokować fragmenty plików na poziomie rekordów. W zabezpieczeniach wykrywano coraz to nowe luki pozwalające osobom nieuprawnionym podszywać się pod innych użytkowników. Jednak najbardziej znaczącą wadą była zbyt mała wydajność systemu w zyskujących coraz większą aprobatę sieciach globalnych (jako że pierwotnie NFS miał być używany wyłącznie w sieciach LAN).

W czerwcu 1999 roku The Internet Society opublikowało artykuł napisany przez pracowników firmy Sun Microsystems zatytułowany *„NFS Version 4 Design Considerations"* zawierający informacje o nowej wersji protokołu NFS. Jego głównymi cechami miały być:

- blokowanie plików i usługa montowania wbudowana w protokół,
- ulepszony dostęp i wydajność w sieci Internet,
- zwiększone bezpieczeństwo z mechanizmem negocjacji wbudowanym bezpośrednio w protokół,
- poprawiona przenaszalność plików między platformami.

Zasadnicze właściwości wersja 4 miała odziedziczyć po poprzednikach, lecz wiele cech musiało zostać zmienionych, aby spełnić postawione wymagania.

3 Przykładowa implementacja

3.1 Podstawowe wiadomości

Jak zostało już wspomniane wcześniej, NFS działa przy wykorzystaniu architektury klient-serwer, na którą składają się:

- program serwera,
- programy klienckie,
- sieciowy protokół komunikacyjny.

3.1.1 Serwer NFS

Zasadniczym zadaniem serwera jest udostępnienie poprzez sieć swoich systemów plików dla innych maszyn (klientów). Proces ten nazywany jest eksportowaniem, zaś udostępniane katalogi – eksportowanymi systemami plików.

W celu eksportowania systemu plików, administrator musi w pliku konfiguracyjnym wyedytować następujące rzeczy:

- pełną nazwę ścieżki katalogu, który ma być eksportowany,
- maszyny klienckie, które będą miały dostęp do eksportowanego katalogu,
- ewentualne ograniczenia dostępu.

Podczas startu systemu uruchamiany jest program (na Solaris'ie 2.x jest to tzw. *share program*), który odczytuje zawartość tego pliku, a następnie informuje jądro systemu operacyjnego o prawach dostępu przypisanych każdemu eksportowanemu katalogowi. Następnie uruchamiane są specjalne demony (przeważnie *mountd* oraz *nfsd*), które rozpoczynają oczekiwanie na żądania dostępu do eksportowanych systemów plików.

3.1.2 Klient NFS

Klient NFS uzyskuje dostęp do plików serwera poprzez tzw. „montowanie" eksportowanych systemów plików. Proces montowania powoduje (podobnie jak w Linuksie) integrację zdalnych (montowanych) systemów plików z lokalnym drzewem katalogów klienta (zobacz rysunek w rozdziale 2.1.2).

Istnieje usługa ułatwiająca powyższy proces – jest to tzw. *automounter*, który na żądanie, niewidocznie dla użytkownika montuje i odmontowuje systemy plików. Z punktu widzenia użytkownika wygląda to tak, jakby system plików serwera był cały czas dostępny.

Mapowanie nazw przez automounter'a
W celu zamontowania systemu plików automounter potrzebuje tzw. map (z ang. *namespace maps*), dzięki którym „wie", jak to zrobić. Mapa łączy „nazwę" (ścieżkę) po stronie klienta ze zdalnym systemem plików (patrz rysunek 2.1.2). Te nazwy są także nazywane punktami montowania. Jeśli systemem operacyjnym klienta jest DOS lub Windows, to zamontowane systemy plików są mapowane jako dodatkowe dyski twarde.

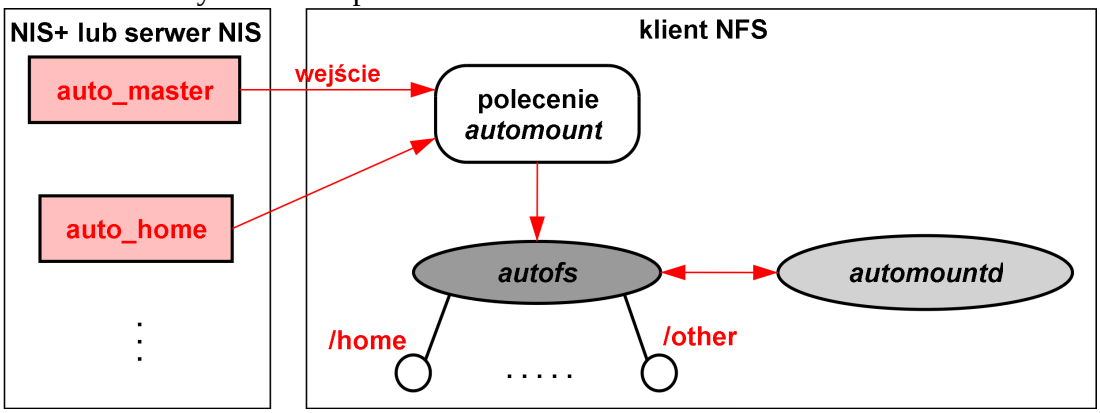
Zasada działania automounter'a
Jak zostało wspomniane wcześniej, najważniejszym zadaniem automount'era jest niewidoczne dla użytkownika zamontowanie systemu plików na żądanie. Potrafi on wykryć sytuację, w której użytkownik (klient) próbuje odwołać się do systemu plików, który nie jest zamontowany i zamontować go wykorzystując informacje z map (patrz wyżej). Ponadto automatycznie odmontowuje system plików, do którego nie było odwołań od ponad pięciu minut.

Automounter składa się z trzech komponentów (patrz rysunek 3.1.2):

- polecenia *automount*;
- demona *automountd*;
- wirtualnego systemu plików – VFS – *autofs*.

Podczas startu systemu wywoływane jest polecenie *automount*, które odczytuje główny plik z konfiguracją map (zwykle zwany *auto_master* oraz tworzy punkty montowania w tzw. tabeli punktów montowania (przeważnie w */etc/mnttab*). Jeśli teraz użytkownik odwoła się do jednego z puntów montowania, *autofs* wysyła żądanie zamontowania systemu plików do demona *automountd*. Ten nawiązuje połączenie z demonem NFS na serwerze (*nfsd*), wyszukuje odpowiednie odwołanie (w postaci: *serwer:ścieżka* np. `serwer1:/home/inf/k/ki189391`) w tabeli punktów montowania i montuje system plików w odpowiednim punkcie montowania (np. `/home/ki189391`) w wirtualnym systemie plików *autofs* klienta.

Rysunek 3: Uproszczona zasada działania Automounter'a



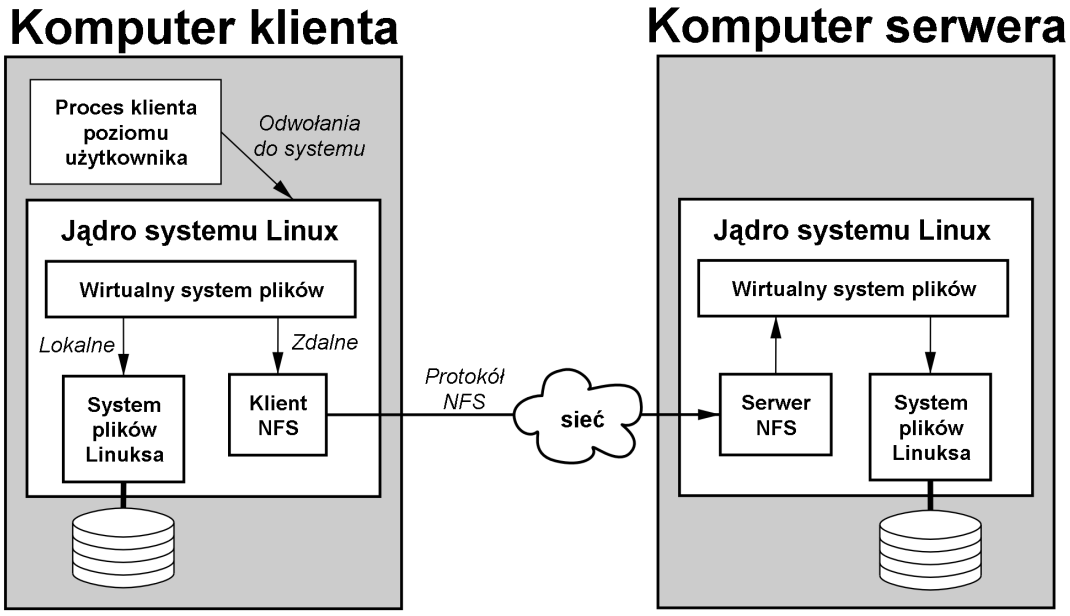
3.1.3 Protokół NFS

Protokół NFS składa się ze zbioru zdalnych procedur pozwalających klientom manipulować plikami na serwerze tak, jakby znajdowały się one lokalnie. Standardowe użycie protokołu przebiega następująco: klient wysyła żądanie operacji na pliku do serwera; serwer próbuje wykonać zlecone zadanie (oczywiście przy założeniu, że użytkownik ma odpowiednie uprawnienia), a następnie przesyła z powrotem rezultaty, bądź informacje o zaistniałych błędach. Protokół NFS zawiera standardowy zbiór operacji, m.in: pisanie, czytanie, tworzenie plików, katalogów, zmiana nazwy, itd.

3.2 Bardziej szczegółowe informacje

Rysunek przedstawia programową architekturę klientów i serwerów systemu NFS. Nie ma na nim składowych dotyczących montowania zdalnych systemów plików. Moduły klienta i serwera ko-

Rysunek 4: Architektura oprogramowania systemu NFS (rysunek pochodzi z [6])



munikują się za pomocą wywołań zdalnych procedur (mechanizm ten został opisany w dalszej części dokumentu). Warto wspomnieć, że całkiem niedawno wprowadzono usługę odwzorowywania (mapowania) portu, aby umożliwić klientom nawiązywanie kontaktu z usługami w danym komputerze za pomocą nazwy (we wczesnych wydaniach NFS klienci kontaktowali się z serwerem za pomocą z góry ustalonego numeru portu).

W celu uzyskania dodatkowego bezpieczeństwa można nakazać szyfrowanie komunikatów RPC (w wersji 4 NFS), lecz interfejs RPC do serwera NFS jest otwarty. Każdy może wysyłać zamówienia do serwera NFS i jeśli są one dozwolone i zawierają poprawne informacje dotyczące tożsamości, to zostaną wykonane.

Prezentowana implementacja przeznaczona jest dla systemu Linuks/UNIX. Interfejs RPC dostarczany przez serwer NFS przedstawiony został w rozdziale dotyczącym specyfikacji.

3.3 Zintegrowanie klienta

Moduł klienta dostarcza interfejs odpowiedni do stosowania w programach użytkowych. Emuluje on semantykę elementarnych operacji standardowego systemu plików Linuksa (z pewnymi wyjątkami) i jest zintegrowany z jądrem systemu. Integracja z jądrem (zamiast wykorzystywania biblioteki) została wykonana w celu:

- umożliwienia aplikacjom użytkownika dostępu do plików systemu Linuks bez ponownej kompilacji lub ponownego ładowania;
- obsługi wszystkich procesów użytkownika przez jeden moduł z dzieloną pamięcią podręczną ostatnio używanych bloków;
- zachowania w jądrze klucza szyfrowania używanego do ochrony identyfikatorów użytkowników przekazywanych serwerowi.

Moduł klienta NFS współpracuje z wirtualnym systemem plików w każdej maszynie klienta. Jego działanie przypomina standardowy uniksowy system plików przesyłający bloki plików do i od serwera oraz przechowujący doraźnie bloki w lokalnej pamięci, kiedy tylko jest to możliwe. Dzieli on ten sam bufor pamięci podręcznej, którego używa lokalny system wejścia-wyjścia. Ponieważ kilku klientów może jednocześnie używać tego samego zdalnego pliku, pojawia się problem spójności pamięci podręcznej.

3.4 Zintegrowanie serwera

Moduł serwera także jest zintegrowany z jądrem systemu Linuks, lecz ten krok podyktowany jest głównie wydajnością.

Ciekawostką jest, że powstała wersja implementacji serwera NFS działająca na poziomie użytkownika. Osiągała ona około 80 procent wydajności wersji zamkniętej w jądrze.

3.5 Kontrola dostępu i tożsamości

Rozproszone sieci są bardzo wrażliwe na nielegalne włamania. Jest to powodowane faktem, że ich komponenty nie stanowią całości, lecz są oddzielone „niebezpiecznym interfejsem”, jakim jest sieć. Bardzo ważnym elementem jest zatem realna ocena wartości przechowywanych danych, a dopiero potem wybór odpowiedniego mechanizmu zabezpieczeń.

NFS może zostać zaimplementowany tak, aby wspierał wiele różnych rodzajów usług zabezpieczających. Dalsze punkty będą dotyczyły usług dostępnych dla NFS w systemie Solaris firmy Sun.

3.5.1 Autentykacja

Autentykacja dostarcza mechanizmu do zidentyfikowania użytkownika w sieci (czyli stwierdzenia, że użytkownik jest tym, za kogo się podaje) przed udostępnieniem mu jakichkolwiek zasobów. NFS może zostać skonfigurowany tak, że może używać następujących sposobów autentykacji:

- protokół wymiany kluczy publicznych Diffie'go-Hellman'a,
- autentykacja Kerberos,
- prosta autentykacja w stylu Uniksa.

W czwartej wersji dodatkowo zaimplementowano możliwość korzystania z następujących metod:

- SPKM-3,
- LIPKEY.

3.5.2 Autoryzacja

Głównym celem autoryzacji jest zapewnienie, że użytkownicy mają prawo do wykonania danej operacji po tym, jak przeszli przez fazę autentykacji. NFS wspiera dwa mechanizmy autoryzacji:

- ACLs – (z ang. *Access Control Lists*) listy kontroli dostępu,
- autoryzacja a'la Uniks.

W najprostszej formie mechanizmu kontroli dostępu (autentykacja i autoryzacja w stylu Uniksa) występuje furtka umożliwiająca obejście zasad bezpieczeństwa. Otóż klient może tak zmienić wywołania RPC, aby zawierały identyfikator dowolnego użytkownika, podszywając się pod użytkowników bez ich wiedzy i pozwolenia. Ową furtkę zamknięto w wersji 4 NFS przez zaopatrzenie protokołu RPC w możliwość szyfrowania informacji o tożsamości użytkownika w standardzie DES.

3.6 Tłumaczenie nazwy ścieżki

Linuksowe systemy plików tłumaczą wieloczłonowe nazwy ścieżek prowadzących do plików na odniesienia dane przez i-węzły w wielokrokowym procesie przy każdym wykonywaniu systemowych wywołań *open*, *creat* lub *stat*. W systemie NFS nazwy ścieżek nie mogą być tłumaczone na serwerze, ponieważ nazwa może przekraczać punkt zamontowania u klienta. W tej sytuacji analiza nazw ścieżek oraz ich tłumaczenie odbywa się pod kontrolą klienta. Każda część nazwy, która odnosi się do zdalnie zamontowanego katalogu, jest przekładana na uchwyt (z ang. *handle*) pliku w osobnym żądaniu do zdalnego serwera. W każdym kroku jest sprawdzana tabela zdalnego montowania u klienta, aby rozstrzygnąć czy będzie konieczny kontakt z następnym zdalnie zamontowanym systemem plików. Przechowywanie rezultatów każdego kroku tłumaczenia nazwy ścieżki w pamięci podręcznej łagodzi niesprawność tego procesu (użytkownicy zwykle korzystają z niewielkiej liczby katalogów).

3.7 Pamięć podręczna serwera

Stosowanie pamięci podręcznej, zarówno na stacjach klienckich, jak i na serwerze, jest konieczne, by zachować odpowiednią wydajność operacji na plikach.

Standardowymi technikami (stosowanymi np. w Linuksie) jest przechowywanie ostatnio używanych stron w pamięci fizycznej, co umożliwia dostęp do nich bez konieczności dostępu do dysku twardego (przy kolejnych operacjach na plikach). Inną optymalizacją jest odczyt z wyprzedzeniem (z ang. *read-ahead*) pozwalający pobrać kilka stron z dysku do których jeszcze nie było odwołań, jeśli spodziewane jest, że kolejne będą ich dotyczyć. Można także opóźniać zapis strony na trwały nośnik (z ang. *delayed-write*) do momentu, gdy w pamięci fizycznej zacznie brakować miejsca na nowe strony. Te rozwiązania działają na pojednynczej maszynie, ponieważ żądania czytania i pisania dotyczą jednej pamięci podręcznej, która jest stale aktualna.

Serwery NFS korzystają ze swojej pamięci podręcznej w sposób analogiczny. Jest jednak jeden wyjątek – operacje pisania są zaimplementowane tak, aby działały jako tzw. przepisywalne (z ang. *write-through*). Wtedy użycie pamięci podręcznej serwera nie powoduje żadnych komplikacji ze spójnością. Ma jednak pewną cenę – gdy serwer otrzyma żądanie pisania, wówczas każda modyfikacja pliku musi być zapisana na trwały nośnik, ponieważ awaria serwera mogłaby spowodować utratę danych przez klienta. Co gorsza, klient mógłby nawet o tym nie wiedzieć.

3.8 Pamięć podręczna klienta

Pamięć podręczną klienta stosuje się w celu zmniejszenia liczby żądań dla serwera. Moduł klienta cache'uje wyniki najważniejszych operacji, czyli *read, write, getattr, lookup* oraz *readdir*. Wykonywane przez klienta operacje pisania nie powodują natychmiastowego uaktualnienia kopii plików przechowywanych w pamięciach podręcznych innych klientów, co prowadzi do problemu niespójności tych pamięci. Aby zminimalizować takie niespójności, do oznaczenia ważności bloków w pamięciach podręcznych stosuje się przy każdym ich użyciu metodę opartą na znacznikach czasu. Ładując do pamięci podręcznej blok z pliku, klient zapamiętuje w niej także stempel czasowy wskazujący czas ostatniej modyfikacji tego pliku na serwerze. Kontrola ważności wszystkich bloków z pamięci podręcznej odbywa się poprzez żądanie podania przez serwer czasu ostatniej modyfikacji i porównanie go z zapamiętaną wartością stempla. Jeśli ten czas jest nowszy, to wszystkie dane z pamięci podręcznej zostaną unieważnione i przy następnych żądaniach muszą być sprowadzone z serwera.

Kontrola ważności odbywa się przy każdym otwarciu pliku oraz żądaniu sprowadzenia bloku z serwera. Po sprawdzeniu ważności zakłada się, że bloki przechowywane w pamięci podręcznej są ważne przez pewien (ustalony podczas montowania) czas. Sensowne wartości to 3 sekundy dla plików i 30 – dla katalogów.

Trochę inaczej są obsługiwane operacje pisania. Zapisywaną w cache'u stronę oznacza się jako zmienioną (z ang. *dirty*) i kieruje asynchronicznie do wysłania do serwera. Wysyłanie takich stron do serwera odbywa się podczas zamykania pliku lub gdy klient wywoła operację *sync*.

Widać więc, że klienci NFS nie mogą zorientować się czy plik jest dzielony, czy też nie. W związku z tym operacja kontroli ważności musi towarzyszyć każdej operacji dostępu do pliku.

To pociąga za sobą zwiększenie kosztów.

Realizacja czytania z wyprzedzeniem oraz opóźnionego zapisywania odbywa się poprzez użycie pewnych asynchronicznych operacji. W implementacjach klienta NFS działających w środowisku Linuksowym osiąga się to dzięki wykorzystaniu tzw. biodemonów (z ang. *block input-output daemon*).

Biodemon jest powiadamiany po każdym żądaniu czytania i przesyła do serwera żądanie transmisji do pamięci podręcznej klienta następnego bloku – czytanie z wyprzedzeniem. W przypadku pisania biodemon wysyła blok do serwera po każdym zapełnieniu go przez proces działający na maszynie klienta. Ciekawostką jest, że bloki katalogów są przesyłane po każdej modyfikacji.

3.9 Blokowanie plików (z ang. *file locking*)

Blokowanie plików jest bardzo skomplikowanym zadaniem. W wersji 2 i 3 systemu NFS było ono realizowane poprzez osobny mechanizm – *Network Lock Manager* (NLM). Najnowsza wersja jest pod tym względem rewolucyjna – blokowanie plików zostało zaimplementowane bezpośrednio w protokole.

3.9.1 Protokół NLM

Ponieważ protokół NFS3 był bezstanowy, niezbędne było zastosowanie dodatkowego protokołu – NLM (z ang. *Network Lock Manager*) – zapewniającego wsparcie dla blokowania plików zamontowanych poprzez NFS. Poniższa tabela ilustruje kompatybilność poszczególnych wersji protokołu NLM z NFS.

Wersja NFS	**Wersja NLM**
2	1, 3
3	4

3.9.2 Blokowanie plików w NFS4

Najnowsza wersja protokołu wspiera blokowanie konkretnych fragmentów plików. Implementacja blokowania plików bezpośrednio w protokole wykluczyła możliwość jego zupełnej bezstanowości. Jednak jasne założenia twórców dotyczące tego mechanizmu umożliwiają sprawne zarządzanie blokadami. Są to:

- jasny podział pomiędzy klientem i serwerem,
- zapewnienie zdolności do wykrycia niespójności pomiędzy klientem i serwerem,
- prosty i dość odporny na awarie mechanizm.

Stanem blokad założonych na pliki zarządza serwer. Klient przesyła do niego odpowiednie informacje tylko wtedy, gdy są one niezbędne. Potrafi ponadto wykryć niespójności pomiędzy

własnymi informacjami a informacjami przechowywanymi przez serwer. Mechanizm blokad jest oparty na modelu dzierżaw – z ang. *lease-based model* i działa na następującej zasadzie:

- serwer definiuje pewien okres dzierżawy (z ang. *single lease period*) dla wszystkich przechowywanych stanów;
- jeśli klient nie odświeży swoich dzierżaw w zadanym okresie, wszystkie dane z nim związane zostaną zwolnione z serwera;
- klient może odświeżyć w./wym. dane używając operacji *renew* lub implicite – poprzez inne operacje (głównie *read*).

Podstawowym założeniem dla tego mechanizmu jest fakt, że operacja blokowania plików jest wykonywana bardzo rzadko w porównaniu z np. operacjami *read* oraz *write*. Ponadto zakłada się, że awarie systemu i sieci nie są częste.

Bardzo ważne jest, aby operacje *read* oraz *write* były lekkie, lecz jednocześnie zawierały informacje o blokadzie założonej na daną część pliku. Z drugiej strony żądanie założenia blokady na plik musi zawierać odpowiednio wiele informacji, by serwer mógł ją stworzyć, a następnie efektywnie nią zarządzać (to wyjaśnia powyższe założenia).

Problemem pojawiającym się podczas używania blokad jest ich szeregowanie. Otóż wysłana poprzez np. TCP sekwencja założenia blokad może zostać rozdzielona pomiędzy różne połączenia i w rezultacie kolejność napływania tych żądań może być zupełnie inna niż kolejność ich wysyłania. Sposobem na rozwiązanie tego problemu jest numerowanie blokad (każdy właściciel blokad ma swój zbiór dozwolonych numerów). Serwer utrzymuje ostatni numer żądania blokowania. Jeśli nowe żądanie ma numer mniejszy niż ostatnie, to jest ono odrzucane.

Tak więc serwer przechowuje informacje postaci <właściciel blokady, ostatni numer żądania>. Operacje takie jak *open, lock, locku* w wywołaniach przekazują numer żądania.

NFS 4 udostępnia blokujące blokady plików, lecz działają one na zasadzie aktywnego oczekiwania, po to, aby ułatwić odtworzenie stanu serwera po awarii.

W wypadku awarii klienta sytuacja jest prosta – nie odnowi on swoich dzierżaw i w rezultacie serwer zwolni wszystkie informacje o blokadach, których był właścicielem (o ile klient nie zaczął działać przed upływem terminu ważności jego dzierżaw). Jeśli natomiast klient zaczął działać przed upływem tego terminu, to serwer stwierdzi, że jest to nowa instancja programu klienta i także zwolni przydzielone przez niego blokady.

Podobna sytuacja wystąpi w przypadku uszkodzenia sieci.

Jeśli natomiast serwer utraci informacje o przechowywanych blokadach, to musi on dać czas klientom na wykrycie tej sytuacji i odtworzenie stanu blokad sprzed awarii. Klient może uzyskać informację o tym w momencie otrzymania specjalnego błędu. Podczas operacji odtwarzania stanu serwer odrzuca wszystkie żądania założenia blokad, czytania, bądź pisania, które pochodzą z okresu poawaryjnego do momentu powrotu do pracy. Stosowanymi usprawnieniami jest przechowywanie liczników blokad dotyczących danego pliku na trwałym nośniku. Wtedy serwer „wie”, że stan sprzed awarii został przywrócony.

3.10 Wydajność

Z pomiaru wydajności przeprowadzonego przez Sandberga i opisanego w książce numer 8 (patrz rozdział numer 6) wynika, że po zastosowaniu pamięci podręcznych i innych opisanych wcześniej optymalizacji używanie NFS nie pociąga za sobą istotnego spadku wydajności.

Do rozwiązania pozostają jednak problemy:

- częste używanie *getattr* w celu pobrania z serwera znaczników czasu do sprawdzania ważności pamięci podręcznej;
- stosunkowo słaba wydajność operacji *write*, powodowywana przez przepisywanie danych z pamięci podręcznej serwera na dysk.

Stosunkowo słabą wydajność operacji zapisu można poprawić poprzez zastosowanie zasilanej z baterii pamięci RAM używanej w jednostce sterującej dysku serwera. Zastosowanie tej pamięci pozwala serwerowi meldować o pomyślnym zakończeniu zapisu przed przesłaniem danych na dysk jednocześnie gwarantując zachowanie ich w pamięci trwałej.

4 Specyfikacja

Zdecydowałem się na opisanie specyfikacji NFS4 (mimo ogromnych rozmiarów), ponieważ jest to najnowsza wersja protokołu.

Celem tego rozdziału nie jest opisanie pełnej specyfikacji, lecz jednynie pokazanie najciekawszych jej aspektów oraz poziomu formalizacji.

4.1 Dodatkowe elementy

4.1.1 RPC (z ang. *Remote Procedure Call*)

Specyfikacja RPC firmy Sun dostarcza proceduralnego interfejsu dla zdalnych usług. Każdy serwer posiada program będący zbiorem możliwych do wywołania procedur. NFS jest jednym z takich programów. Kombinacja adresu serwera (z ang. *host address*), numeru programu, numeru wersji oraz numeru procedury pozwala zidentyfikować dokładnie jedną zdalną procedurę. Dzięki takiemu podejściu serwer potrafi wspierać kilka wersji danego programu – poprzez użycie różnych wersji protokołu.

NFS został zaprojektowany tak, aby nie wymagał żadnych specyficznych poziomów zapewniania dostarczania komunikatów przez protokoły z niższych warstw. Może być zatem używany z wieloma różnymi protokołami transportu. Użycie RPC powoduje stworzenie pewnej abstrakcji ponad niskopoziomowymi protokołami sieciowymi (warstwy transportu).

4.1.2 XDR (z ang. *eXternal Data Representation*)

XDR, czyli tzw. zewnętrzna reprezentacja danych – to metodologia specyfikująca standard reprezentacji różnych zbiorów danych poprzez sieć. Rozwiązuje ona problem kolejności bajtów

w słowie, wyrównywania struktur do „pełnych” rozmiarów oraz różnej reprezentacji poszczególnych typów danych na innych maszynach.

W oryginalnej specyfikacji firmy Sun do opisania parametrów formatu XDR używany jest tzw. RPC Data Description Language. Jest on podobny do deklaracji w języku C z kilkoma dodanymi konstrukcjami.

Na przykład konstrukcja:

```
string name[SIZE];
string data<DSIZE>;
```

definiuje *name* jako tablicę o stałym rozmiarze – *SIZE* bajtów, zaś *data* – to tablica o zmiennym rozmiarze nie przekraczającym *DSIZE* bajtów. Jeśli w definicja zmiennej *date* pominiętoby maksymalny rozmiar, to tablica miałaby potencjalnie nieograniczoną długość.

Inny przykład:

```
union example switch (enum status) {
    case OK:
       struct {
          filename        file1;
          filename        file2;
          integer         count;
       }
    case ERROR:
       struct {
          errstat         error;
          integer         errno;
       }
    default:
       void;
}
```

Definiuje on strukturę, której pierwszym elementem jest pole typu *enum* nazywane *status*. Jeśli jego wartością jest *OK*, to drugim polem będzie struktura zawierająca pola *file1*, *file2* oraz *count*. W przeciwnym przypadku, jeśli wartością pola *status* jest *ERROR*, to drugim polem będzie struktura zawierająca pola *error* oraz *errno*. Jeśli wartością pola *status* nie jest ani *OK*, ani *ERROR*, to w strukturze nie będzie więcej danych.

Typ XDR – *hyper* oznacza wartości 64-bitowe i jest używany w ten sam sposób, jak liczby całkowite.

```
hyper              sgn;
unsigned hyper     usg;
```

sgn jest 8-bajtową liczbą całkowitą ze znakiem, zaś *usg* 8-bajtową wartością bez znaku.

Ciekawostką jest fakt, iż pomimo istnienia kompilatorów RPC/XDR potrafiących generować odpowiedni kod dla klienta i serwera z pliku w formacie RPC Data Description Language, implementacje NFS nie wymagają ich użycia.

4.2 Dostarczane procedury

Nowością w czwartej wersji protokołu jest kompletna zmiana puli dostarczanych procedur. Najnowsza specyfikacja NFS dopuszcza tylko dwie:

- **NULL**,
- **COMPOUND**.

Pozostała funkcjonalność protokołu została zdefiniowana jako zbiór operacji w normalnej składni XDR i semantyce RPC. Wszystkie te operacje są zamknięte w procedurze *COMPOUND*. To innowacyjne rozwiązanie sprawia, że klient może w pojedynczym żądaniu do serwera wykonać kilka operacji.

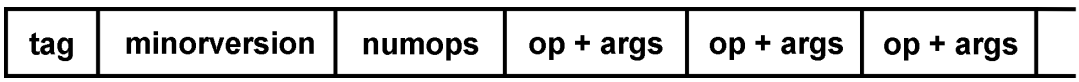
4.2.1 Procedura 0: NULL

Ta procedura nie pobiera i nie zwraca żadnych parametrów, ani też nie wykonuje żadnego konkretnego żądania. Jej głównym zadaniem jest umożliwienie sprawdzenia połączenia z serwerem oraz pomiaru czasu odpowiedzi. Istnienie takiej procedury jest wymagane przez protokoły działające w oparciu o RPC.

4.2.2 Procedura 1: COMPOUND

Procedura *COMPOUND* pozwala osiągnąć lepszą wydajność w sieciach o dużych opóźnieniach transmisji. Klient unika kumulowania opóźnień wytwarzanych podczas każdorazowej transmisji żądania do serwera poprzez zgrupowanie operacji, które chce wykonać w jeden zbiór, który następnie jest przesyłany jako pojedyncze żądanie.

„Struktura” operacji *COMPOUND* jest następująca: Zaś struktura odpowiedzi dla klienta:



Serwer będzie wykonywał procedurę COMPOUND wykonując kolejno każdą operację, którą

ona zawiera. Każdy komponent reprezentujący operację składa się z 32 bitowego kodu operacji, a po nim argumentu, którego długość jest determinowana typem operacji. Jeśli któraś z operacji

się nie powiedzie, to obliczenia są przerywane i sygnalizowany jest błąd. W gestii klienta leży przywrócenie stanu danych po częściowym wykonaniu zbioru operacji procedury *COMPOUND*.

Nie ma wymagań dotyczących niepodzielności operacji zawartych w pojedynczym wywołaniu procedury *COMPOUND*. W szczególności mogą być one wykonywane współbieżnie z operacjami innej procedury *COMPOUND* pochodzącej z innego żądania (np. drugiego klienta).

Poniżej przedstowione zostały najciekawsze dostępne operacje wraz z przypisanymi im numerami.

- **Operacja 3: ACCESS** – odpowiada za kontrolę praw dostępu użytkownika;
- **Operacja 4: CLOSE** – zamknięcie pliku;
- **Operacja 5: COMMIT** – zapisanie na trwałym nośniku danych, które wcześniej były zapisywane z flagą *UNSTABLE*.
- **Operacja 6: CREATE** – tworzy obiekt nieregularny (aby stworzyć regularny plik należy użyć operacji *OPEN*.
- **Operacja 9: GETATTR** – pobiera aktualne atrybuty pliku;
- **Operacja 10: GETFH** – zwraca aktualny uchwyt pliku;
- **Operacja 11: LINK** – tworzy dowiązanie do pliku znajdującego się w danym katalogu; plik i katalog muszą znajdować się fizycznie w tym samym systemie plików;
- **Operacja 12: LOCK** – zakłada blokadę na fragment pliku; można założyć blokadę na jeszcze nie przydzielone bajty;
- **Operacja 13: LOCKT** – sprawdza czy została założona blokada na przydzielony obszar;
- **Operacja 14: LOCKU** – zdejmuje określoną blokadę;
- **Operacja 15: LOOKUP** – wyszukuje danego obiektu w systemie plików w katalogu przekazanym jako parametr;
- **Operacja 18: OPEN** – otwarcie zwykłego pliku;
- **Operacja 25: READ** – odczyt zwykłego pliku; jako parametr przekazywany jest identyfikator pozwalający serwerowi sprawdzić blokady założone na plik;
- **Operacja 26: READDIR** – odczyt zawartości katalogu;
- **Operacja 27: READLINK** – odczyt zawartości dowiązania symbolicznego;
- **Operacja 28: REMOVE** – usunięcie obiektu z systemu plików; jeśli nie ma odwołań do pliku, to jest on fizycznie usuwany, w przeciwnym przypadku serwer pozwala na dostęp do niego;

- **Operacja 29: RENAME** – zmiana nazwy pliku lub katalogu;
- **Operacja 38: WRITE** – zapis do pliku; można wybrać sposób zapisu danych:
  - *FILE_SYNC* – zanim serwer zwróci wynik wszystkie dane oraz metadane muszą zostać zapisane na trwałym nośniku,
  - *DATA_SYNC* – j.w. tylko, że dotyczy samych danych oraz takiej ilości metadanych, aby możliwe było odczytanie danych w wypadku awarii,
  - *UNSTABLE* – serwer może, ale nie musi zapisać danych na trwały nośnik (można później wymusić zapis tych danych za pomocą operacji *COMMIT*);
- **Operacja 10044: ILLEGAL** – ta operacja jest niezbędna, gdy klient w wywołaniu *COMPOUND* umieści operację, która nie jest wspierana przez serwer; wtedy w jej miejsce w wyniku podstawiana jest właśnie wartość oznaczająca operację *ILLEGAL*.

5 Definicje

Poniżej wyjaśniono znaczenie niektórych pojęć użytych w dokumencie.

Access Control Lists (ACLs)
: List wskazująca, jakie typy dostępu są dozwolone dla danego pliku. Na przykład wskazuje czy konkretni użytkownicy lub grupy mogą wykonywać konkretne operacje na pliku, tj. odczytywanie, zapisywanie, wykonywanie, itd.

TCP
: Warstwa transpotu protokołu sieciowego opracowanego przez Departament Obrony Stanów Zjednoczonych i wykorzystywany przez NFS do komunikacji poprzez sieci LAN oraz WAN. Jego najważniejszą zaletą jest zapewnianie o dostarczeniu komunikatów.

UDP
: Warstwa transpotu protokołu sieciowego opracowanego przez Departament Obrony Stanów Zjednoczonych nie zapewniająca docierania komunikatów do adresata. Wykorzystywana w wielu implementacjach NFS do komunikacji przez LAN.

VFS
: (Z ang. *Virtual File System*) – abstrakcyjny interfejs pomiędzy systemem operacyjnym oraz systemem plików, który pozwala zintegrować systemy plików wielu różnych plików (zarówno lokalnych jak też zdalnych).

6 Literatura

1. S. Shepler *„NFS version 4 Protocol”*, Internet Draft, Sun Microsystems, listopad 2002.

2. „*NFSv3 Overview*”, Internet presentation, Sun Microsystems 1996.

3. B. Callaghan, B. Pawlowski, P. Staubach „*NFS Version 3 Protocol Specification*”, RFC 1813, Sun Microsystems 1996.

4. R. Srinivasan „*RPC: Remote Procedure Call Protocol Specification Version 2*”, RFC 1831, Sun Microsystems 1995.

5. R. Srinivasan „*XDR: External Data Representation Standard*”, RFC 1832, Sun Microsystems 1995.

6. T. Kindberg „*Systemy rozproszone – podstawy i projektowanie*”, WNT 1998.

7. „*Red Hat Linux 7.3: The Official Red Hat Linux Reference Guide*”, rozdział 18 – „*Network File System (NFS)*”.

8. R. Sandberg „*The Sun Network File System: Design, Implementation and Experience*”, Technical Report, Mountain View, CA, Sun Microsystems 1987.