

Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 25.М41-мм

Проектирование архитектуры высокопроизводительного Userspace NFS-сервера

Патов Артемий Сергеевич

Отчёт по производственной практике
в форме «Решение»

Научный руководитель:
доцент кафедры системного программирования, к.ф.-м.н., В. И. Гориховский

Санкт-Петербург
2026

Оглавление

1. Введение	3
2. Постановка задачи	5
3. Обзор задействованных протоколов	6
4. Обзор архитектуры NFS-ganesha	8
5. Проектирование сервера nfs-mamont	11
5.1. Интеграция интерфейсов реализуемых протоколов	13
6. Заключение	14
Список литературы	15

1. Введение

Серия протоколов NFS (Network File System) является стандартом де-факто для организации прозрачного сетевого доступа к файловым системам в среде Unix и Linux. Протокол прошел длительную эволюцию: от первой публичной версии 2 (1989 г.)[5] и получившей наибольшее распространение версии 3 (1995 г.)[1] до современных спецификаций 4 (2015 г.)[3], 4.1 и 4.2 (2016 г.)[4].

Имеется множество реализаций клиентских и серверных компонентов, обеспечивающих работу через NFS. К ним относятся сервисы, работающие в пространстве ядра операционной системы: `nfsd`¹, FreeBSD NFS Server, Windows Server NFS. Наряду с ними развиваются NFS-серверы, работающие в пространстве пользователя (`userspace`), такие как NFS-Ganesha, `nfsserve`². Сервер, работающий в пространстве пользователя, в отличие от ядерных реализаций, не подвергает риску стабильность всей операционной системы в случае сбоя. Кроме того, такая архитектура обеспечивает преимущество в производительности при взаимодействии с файловыми системами, которые также вынесены за пределы ядра.

В компании YADRO в программном стеке системы хранения резервных копий TATLIN.BACKUP³ в качестве NFS сервера используется NFS-Ganesha. Несмотря на обширную функциональность (реализует NFSv3 и NFSv4) и высокие показатели производительности, код сервера NFS-Ganesha чрезвычайно сложен, его поддержка затруднительна, а вместе с тем, на странице GitHub Issues проекта⁴ заведено около 100 серьезных багов. Помимо этого, доступ к `userspace`-файловым системам происходит через FUSE, что сильно сказывается на производительности [9].

В связи с высоким спросом на высокопроизводительные NFS-серверы, работающие в пространстве пользователя, и, в частности, необ-

¹`nfsd` man page: <https://man7.org/linux/man-pages/man7/nfsd.7.html>

²`nfsserve`: <https://github.com/xetdata/nfsserve>

³TATLIN.BACKUP: <https://yadro.com/ru/tatlin/backup/>

⁴GitHub Issues проекта NFS-Ganesha: <https://github.com/nfs-ganesha/nfs-ganesha/issues>

ходимостью замены в компании YADRO монолитной и сложной в сопровождении архитектуры NFS-Ganesha, было принято решение разработать собственный NFS-сервер. В его реализации особое внимание уделяется качеству архитектуры. Для повышения производительности планируется отказаться от использования FUSE в пользу прямой интеграции с файловыми системами в пространстве пользователя. Язык Rust был выбран в качестве основного для обеспечения бесшовной интеграции сервера в программный стек СХД TATLIN.BACKUP.

2. Постановка задачи

Общей целью проекта NFS-Mamont является разработка серверных реализаций NFS-протоколов версии 3 и 4, функционирующих в пространстве пользователя. Первоначальной целью является разработка userspace-сервера версии 3 (Далее — NFSv3). Участники проекта имеют возможность свободного выбора открытых для выполнения задач. Настоящая работа посвящена формированию архитектурного базиса сервера. Стоит отметить, что фактически разработка архитектуры велась совместно со всеми участниками проекта и породила большое количество активных дискуссий.

Для погружения в предметную область и достижения поставленной цели в представленной работе будут выполнены следующие задачи:

1. Провести аналитический обзор стека протоколов: NFSv3, MOUNT, NLM и NSM.
2. Исследовать архитектурные подходы существующих userspace-реализаций (в частности, NFS-Ganesha)
3. Спроектировать структуру основных компонентов сервера и интерфейсы их взаимодействия, представив их в виде описаний и архитектурных диаграмм.

3. Обзор задействованных протоколов

В проекте NFS-Mamont для разработки NFSv3-сервера планируется реализовать ядро (необходимый минимум) из 4 протоколов: NFS версии 3, MOUNT, NLM, NSM. Общим у всех перечисленных протоколов является то, что все они построены на основе стандарта RPC (Remote Procedure Call) и языка описания и кодирования данных XDR. Далее приведены определения каждого из используемых протоколов.

NFS версии 3 [1] — протокол удаленного сетевого доступа к распределенным файловым системам. Не имеет состояния (stateless) и представляет собой набор процедур для удаленного вызова, порядок которого описан в RPC.

MOUNT [2] — протокол монтирования, предоставляющий клиенту корень экспортируемой по NFS древовидной файловой системы (далее — экспорт). Запрос корневого каталога, возвращающий его серверный идентификатор, является первоначальным действием для клиента, намеревающегося работать с удаленной файловой системой. Описание протокола MOUNT предписывает серверу поддержание т. н. `mount list` — списка с записями вида "клиент — экспорт", расширяющегося при каждом монтировании. Соответственно, протокол MOUNT, в отличие от NFSv3, должен иметь состояние (т. е. он stateful).

NLM [8] — протокол файловых блокировок (англ. file lock). Его необходимость обусловлена тем, что протокол NFSv3 не обладает функциональностью, поддерживающей в каком-либо виде файловые блокировки, так как лишен состояния, без которого нельзя обойтись при реализации file lock'ов.

NSM [8] — протокол мониторинга состояния хоста. Сохраняет информацию о количестве сбросов хоста, на котором запущен, и клиентов, работающих с данным хостом, на некоторое энергонезависимое хранилище, чтобы в случае перезагрузки сообщить об этом клиентам. После этого клиент, желая продолжить работу с NFS-сервером перезагруженного хоста, должен выполнить ряд действий для восстановления состояния — например, перемонтировать файловую систему и перезапросить

блокировки.

Отсутствие требования поддержки состояния в протоколе NFS значительно облегчает его реализацию. Состояние выносится в другие несамостоятельные небольшие протоколы, такие как MOUNT и NLM, что повышает модульность системы.

Во всех вышеупомянутых протоколах для вызова клиентом серверных процедур используется порядок, описанный в стандарте **RPC** (Remote Procedure Call) [6], данные описываются с помощью языка **XDR** (External Data Representation) [7].

Исходя из стандарта RPC, Каждая процедура на серверном хосте определяется номером программы (явно определяемым стандартом), номером версии программы и номером процедуры. Модель вызова удаленной процедуры аналогична модели вызова локальной процедуры: вызывающий посылает сообщение, включающее идентификатор процедуры и параметры, и ждет ответа, приостанавливая исполнение дальнейших инструкций, зависящих от ответа. Когда ответ получен, клиент декодирует данные в соответствии с XDR, и исполнение продолжается.

Таким образом, в своем простейшем виде NFSv3-сервер представляет собой набор изолированных друг от друга процедур, большинство из которых идемпотентны. Сложности возникают при попытке реализовать высокопроизводительный NFS-сервер, обладающий кэшем (следовательно, неявно поддерживающим некоторое внутреннее состояние) и эксплуатирующим множество потоков. Одна из главных сложностей — сохранить при этом гибкость и поддерживаемость всего сервера.

4. Обзор архитектуры NFS-ganesha

NFS-Ganesha⁵ — это userspace NFS-сервер, написанный на языке C, который реализует как версию 3 протокола (вместе с MOUNT, NLM, NSM, протоколами аутентификации и др.), так и версию 4. Он является самой распространённой реализацией серии NFS-протоколов, работающей в пространстве пользователя. NFS-Ganesha, в том числе, используется в составе распределённой СХД TATLIN.BACKUP в компании YADRO.

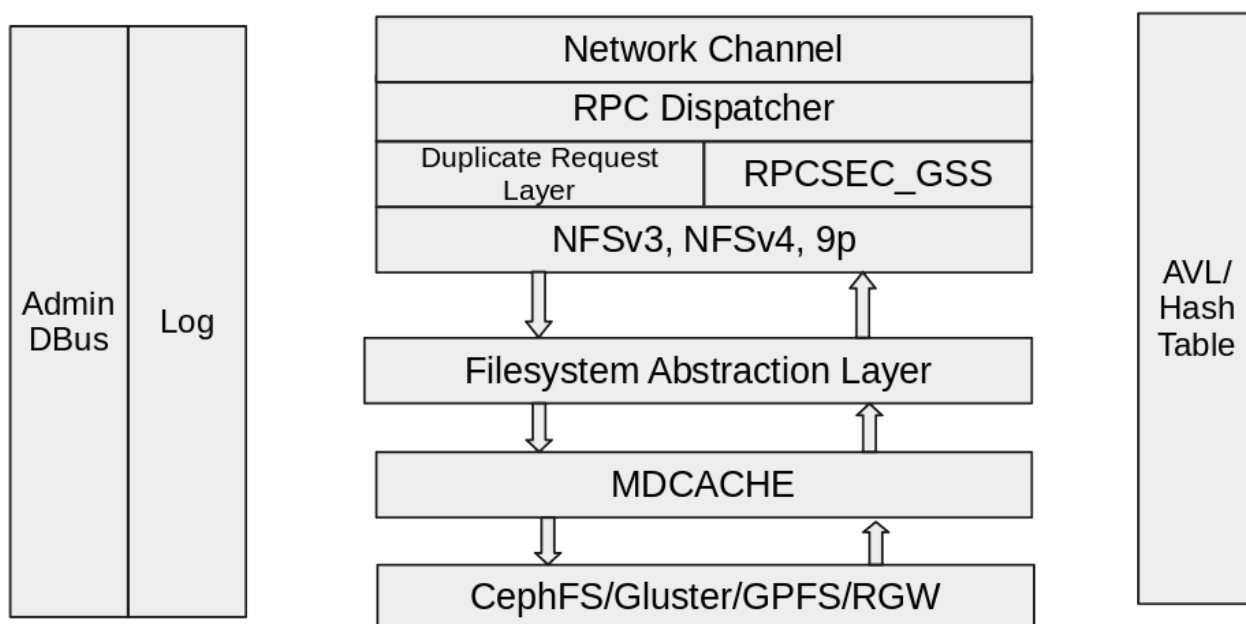


Рис. 1: Составляющие сервера NFS-Ganesha. Диаграмма взята из GitHub Wiki проекта⁶

Основными протоколами, которые реализует NFS-Ganesha, являются NFS версий 3, 4 и 4.1. Кроме того, поддерживается узкоспециализированный 9P. Для поддержки экспортирования различных типов файловых систем используется слой абстракции FSAL (File System Abstraction Layer) с различными реализациями для разных файловых систем: VFS (в т. ч. для FUSE), Ceph, Gluster и др. Необходимый FSAL подгружается как динамическая библиотека (посредством `dlopen` и `dlsym`) при инициализации сервера. Дополнительная функциональ-

⁵GitHub репозиторий проекта NFS-Ganesha: <https://github.com/nfs-ganesha/nfs-ganesha>

⁶GitHub Wiki проекта NFS-Ganesha: <https://github.com/nfs-ganesha/nfs-ganesha/wiki>

ность, поддерживаемая NFS-Ganesha включает

- DRC (Duplicate Request Cache) — кеш, сохраняющий информацию о недавних запросах, чтобы избежать их повторную обработку;
- MDCACHE (Metadata Cache) — кеш, хранящий метаданные недавно измененных файлов;
- D-Bus для динамического управления экспортами;
- Система логирования, поддерживающая разные уровни информационной детализации (verbosity).

За обработку сетевых запросов отвечает рутина `xprt`, за диспетчеризацию и обработку полученных запросов, а также работу с локальными для сервера файловыми системами — рутина `svc`. Для распределения задач по нескольким потокам `nfs-Ganesha` использует две собственные реализации пула потоков на основе `pthread`: первая реализация выполняет отложенную работу и используется во вспомогательных подсистемах (кеши, логер и др.), вторая реализация используется рутиной `svc`.

В целом, система выглядит крайне сложной, что объясняется тем, что проект имеет несколько сотен участников и решения принимаются не централизованно. Многократные попытки разобраться в коде `NFS-Ganesha` привели к выводу, что архитектура сервера представляет собой множество отделимых компонентов лишь на самом верхнем уровне абстракции (см. рис. 1). *Внутри компонентов модульность и прозрачная иерархия отсутствуют, их архитектура являет собой недисциплинированный набор объектов и их связей. Поддержка проекта новыми участниками почти не представляется возможной.* В GitHub Issues данного проекта⁷ заведено около 100 серьезных багов, связанных с утечкой памяти и неопределенным поведением, возникающими при работе сервера. Их исправление происходит медленно: последний баг был исправлен 8 ноября, т. е. около 2-х месяцев назад.

⁷GitHub Issues проекта NFS-Ganesha: <https://github.com/nfs-ganesha/nfs-ganesha/issues>

При разработке сервера NFS-Matont важно не допустить перечисленные ошибки, поэтому проектированию и поддержанию читаемости кода уделяется особое внимание.

При экспорте userspace-файловых систем через NFS-Ganesha используется⁸ фреймворк FUSE⁹ предоставляет userspace API для доступа к нижележащим файловым системам. Однако при обращении через пользовательский интерфейс фреймворк перенаправляет запрос к драйверу ядра, что влечёт множество переключений контекста (context switch) и копирований данных [9]. Это сильно сказывается на производительности. В *nfs-matont* будет предпринята попытка избавиться от необходимости использовать FUSE для доступа к файловым системам, работающим в пространстве пользователя.

⁸FUSE в NFS-Ganesha: <https://www.linux.com/news/run-your-nfs-server-user-address-space-nfs-gane>

⁹FUSE: <https://docs.kernel.org/filesystems/fuse/fuse.html>

5. Проектирование сервера nfs-mamont

Код nfs-mamont доступен в GitHub репозитории проекта¹⁰. Сервер реализуется на языке программирования Rust. Для распределения внутренних задач сервера по нескольким потокам используется асинхронный высокопроизводительный рантайм Tokio¹¹. Для передачи по сети используются транспортные протоколы TCP и UDP.

Первично реализуемым набором протоколов (и работоспособным минимумом) является стек NFSv3, MOUNT, NLM, NSM. Такой сервер предоставляет функциональность для экспорта файловых систем, мониторингирования их на хосте клиента, блокировки файла или части файла, а также восстановления состояния в случае сбоев серверного хоста.

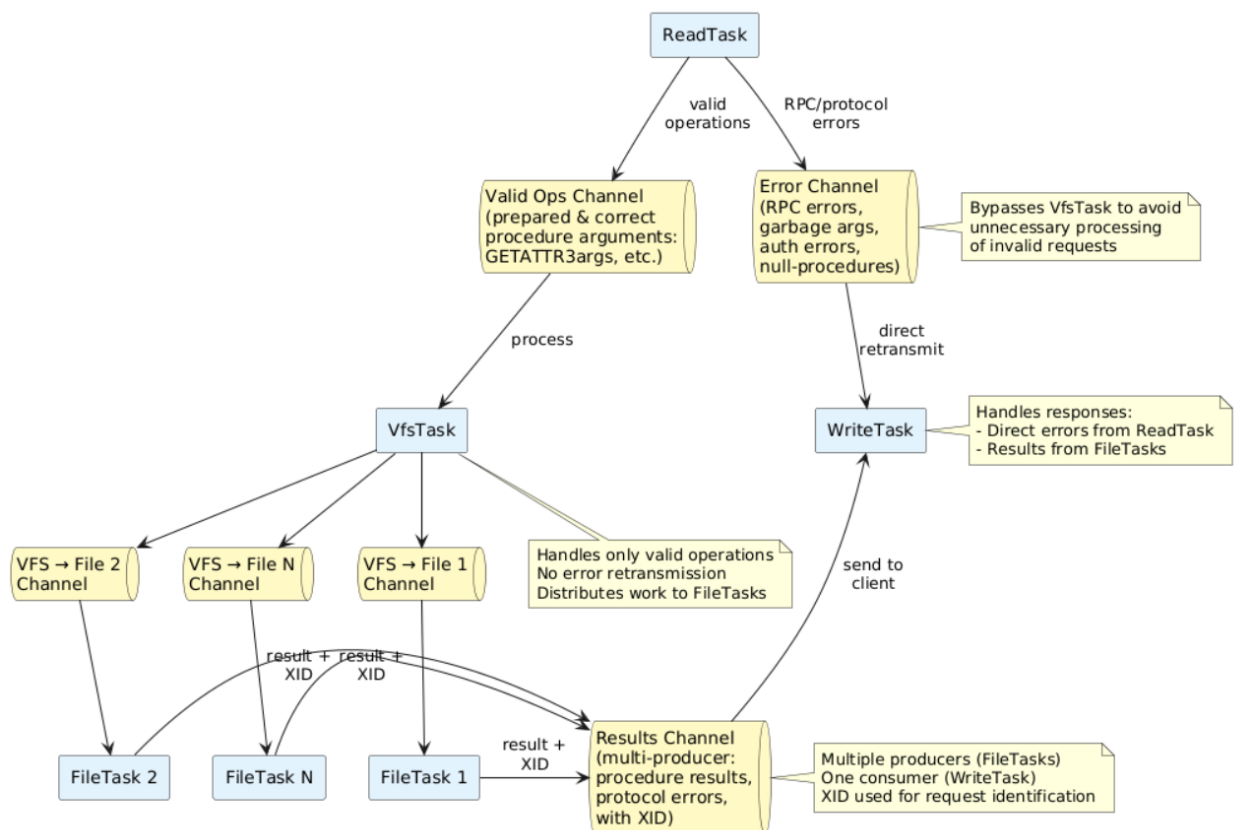


Рис. 2: Составляющие сервера nsf-mamont. Диаграмма взята из GitHub Wiki проекта¹²

¹⁰nfs-mamont: <https://github.com/RMamonts/nfs-mamont>

¹¹Tokio runtime: <https://docs.rs/tokio/latest/tokio/>.

¹²GitHub Wiki проекта nfs-mamont: <https://github.com/RMamonts/nfs-mamont/wiki/NFS-Mamont-architecture>.

На текущий момент достигнут консенсус среди участников проекта по верхнеуровневым компонентам сервера и их связям (рис. 2).

Архитектура представляется в виде нескольких акторов¹³ (англ. actor), общающихся через полудуплексные и полнодуплексные каналы. Каждый актор — это `tokio::task`¹⁴, исполняющийся на пуле потоков Tokio.

`ReadTask` прослушивает сокет и обрабатывает входящие соединения и запросы от клиентов. Стандартной последовательностью действий для `ReadTask` является получение запроса, аутентификация хоста-отправителя, аллокация буфера под десериализацию запроса, валидация запроса, передача запроса в `VfsTask` по одностороннему каналу.

`VfsTask` занимается диспетчеризацией запросов к файловым системам. Принимает структурированные входящие запросы по одностороннему каналу от `ReadTask` и, в зависимости от вызываемой процедуры и файла, перенаправляет этот запрос в некоторый `FileTask`.

`FileTask` принимает запрос от `VfsTask` и выполняет работу над конкретными объектами нижележащей файловой системы, формирует результат в структурированном виде и отправляет в `WriteTask`. `FileTask` создается для каждого файла при формировании экспорта и во время работы с файловой системой.

`WriteTask` принимает ответы от всех `FileTask`, сериализует их в соответствии с XDR и отправляет нужным клиентам.

Обработка RPC-запроса от клиента представляет собой цепь односторонних переходов от актора к актору. `ReadTask` проверяет соответствие запроса RPC и XDR, и возвращает код RPC-ошибки в случае несоответствия. `VfsTask` предполагает, что запрос провалидирован и его структура соответствует стандарту, `VfsTask` работает с файловой системой и может вернуть только NFS-ошибку, отправив ее в качестве результата операции в `WriteTask`.

¹³Actors with Tokio: <https://ryhl.io/blog/actors-with-tokio/>.

¹⁴Tokio Task: <https://docs.rs/tokio/latest/tokio/task/>

5.1. Интеграция интерфейсов реализуемых протоколов

Описания типов данных и процедур протокола на языке XDR необходимо интегрировать в проект, оформив их в виде идиоматических структур и интерфейсов на языке Rust. В течение семестра была реализована поддержка протоколов MOUNT и NSM¹⁵. Параллельно другим участником проекта велась работа по переносу спецификаций протокола NFSv3¹⁶.

¹⁵Код интерфейсов MOUNT и NSM: <https://github.com/RMamonts/nfs-mamont/pulls/artemiipatov>

¹⁶Код интерфейса NFSv3: <https://github.com/RMamonts/nfs-mamont/pull/21>

6. Заключение

В ходе работы были получены следующие результаты:

1. Проведен детальный анализ протоколов NFS версии 3, MOUNT, NLM и NSM.
2. Выполнен критический обзор архитектуры сервера NFS-Ganesha. В результате анализа исходного кода проекта были выявлены ключевые недостатки:
 - Высокая сложность сопровождения и накопленный технический долг;
 - Ограничения производительности, обусловленные использованием интерфейса FUSE.

Обоснована необходимость перехода к прямой интеграции с файловыми системами в пространстве пользователя для минимизации задержек.

3. Спроектирована архитектура основных компонентов сервера, основанная на акторной модели.
4. Реализованы¹⁷ базовые интерфейсы протоколов MOUNT и NSM на языке Rust. Код выполнен в идиоматическом стиле с использованием строгой типизации, что гарантирует безопасность работы с памятью и корректную десериализацию XDR-структур.

¹⁷Код доступен в репозитории проекта (имя пользователя: artemiipatov): <https://github.com/RMamonts/nfs-mamont/pulls/artemiipatov>

Список литературы

- [1] Callaghan Brent, Pawlowski Brian, Staubach Peter. NFS Version 3 Protocol Specification. — IETF, 1995. — . — URL: <https://datatracker.ietf.org/doc/html/rfc1813> (online; accessed: 2026-01-08).
- [2] Callaghan Brent, Pawlowski Brian, Staubach Peter. NFS Version 3 Protocol Specification, Appendix I: Mount Protocol. — 1995. — URL: <https://datatracker.ietf.org/doc/html/rfc1813#appendix-I> (online; accessed: 2026-01-08).
- [3] Haynes Thomas, Noveck David. Network File System (NFS) Version 4 Protocol. — IETF, 2015. — . — URL: <https://datatracker.ietf.org/doc/html/rfc7530> (online; accessed: 2026-01-08).
- [4] Haynes Thomas, Noveck David. Network File System (NFS) Version 4 Minor Version 2 Protocol. — IETF, 2016. — . — URL: <https://datatracker.ietf.org/doc/html/rfc7862> (online; accessed: 2026-01-08).
- [5] Nowicki Bill. NFS: Network File System Protocol Specification (Version 2). — IETF, 1989. — . — URL: <https://datatracker.ietf.org/doc/html/rfc1094> (online; accessed: 2026-01-08).
- [6] Srinivasan Raj. RPC: Remote Procedure Call Protocol Specification Version 2. — IETF, 1995. — . — URL: <https://datatracker.ietf.org/doc/html/rfc1831> (online; accessed: 2026-01-08).
- [7] Srinivasan Raj. XDR: External Data Representation Standard. — IETF, 1995. — . — URL: <https://datatracker.ietf.org/doc/html/rfc1832> (online; accessed: 2026-01-08).
- [8] The Open Group. — Protocols for Interworking: XNFS, Issue 3. — The Open Group, 1998. — Описывает спецификации протоколов NLM (Network Lock Manager) и NSM (Network Status Monitor). URL: <https://publications.opengroup.org/c702>.

- [9] Vangoor Bharath Kumar Reddy, Tarasov Vasily, Zadok Erez. To FUSE or not to FUSE: performance of user-space file systems // Proceedings of the 15th Usenix Conference on File and Storage Technologies. — FAST'17. — USA : USENIX Association, 2017. — P. 59–72.