CS 331 PROJECT 10: NETWORK FILE SYSTEM

Vedant Acharya (23110010),
Haarit Chavda (23110077),
Harshil shah (23110132),
Jeet Joshi (23110148),
Akshat shah (23110293)

INTRODUCTION

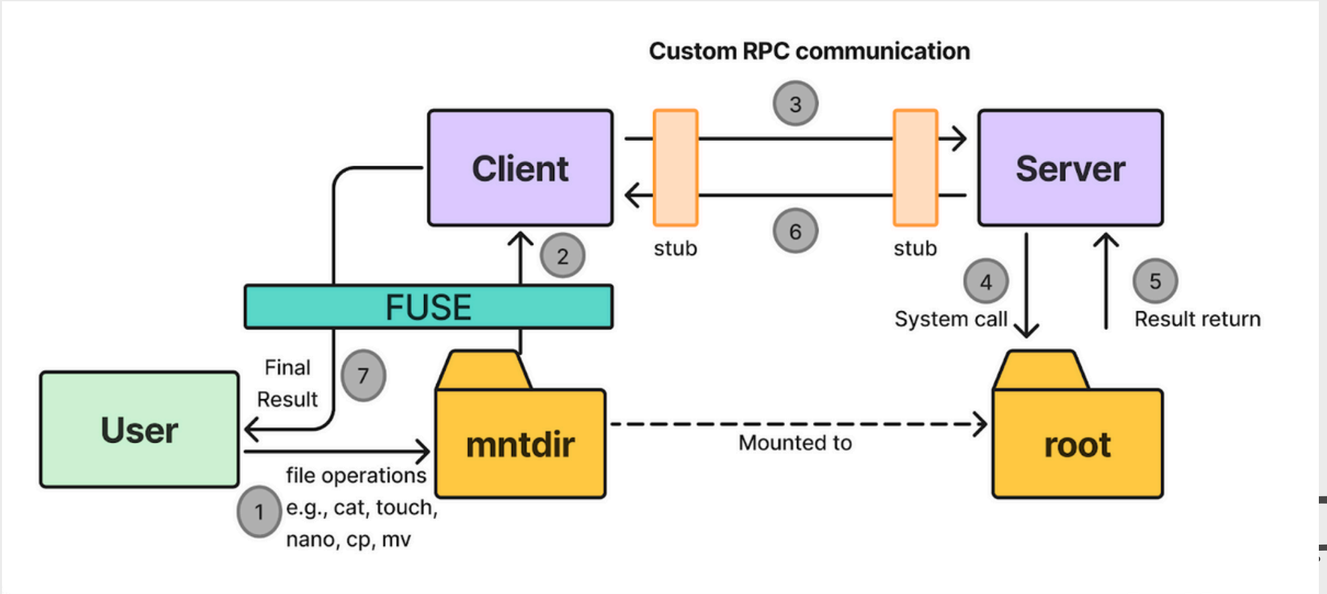
- Network File System (NFS) enables remote file access as if they were local
- Our goal: Build a simplified version of NFS, to understand NFS internals

Main Goals:

- Integrate with FUSE for real command execution (cat, cp, nano, etc.)
- Design request–response protocol (opcode-based)
- Implement client caching (LRU, attribute, write-back,read-ahead)
- Enable parallel processing on server (thread pool)

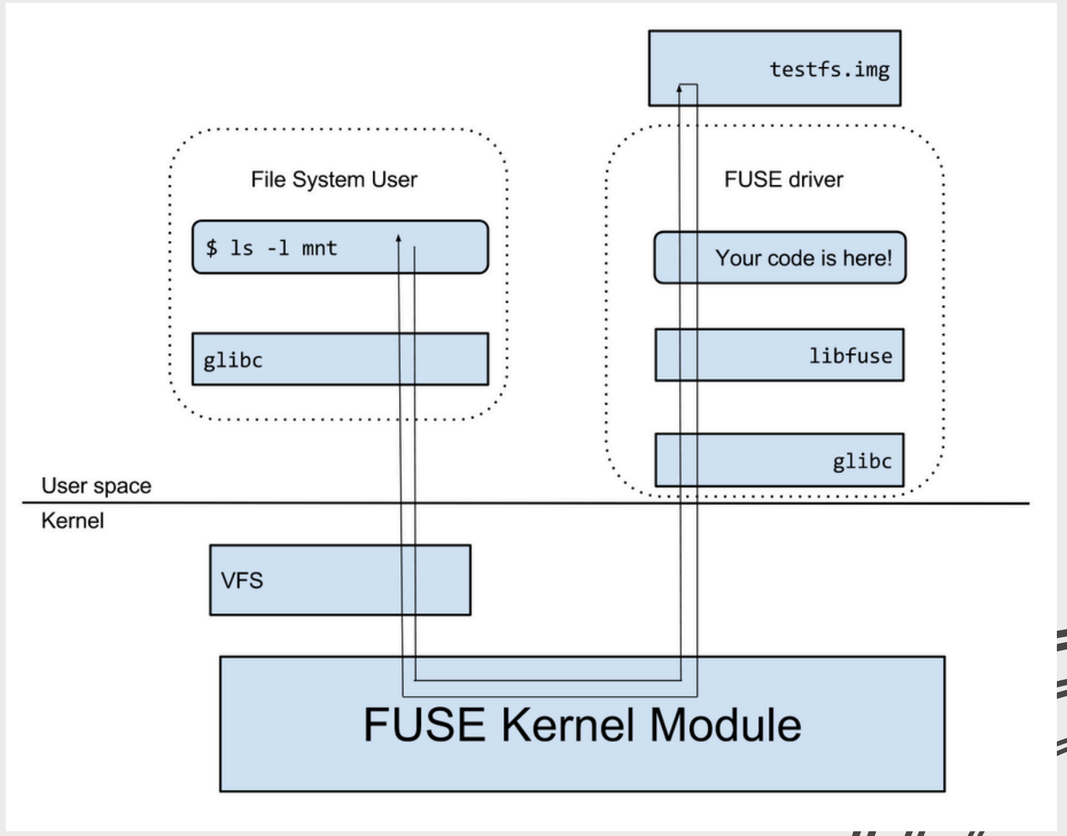
SYSTEM ARCHITECTURE

- User mounts **mntdir** via FUSE
- FUSE forwards file ops to client.cpp
- Client serializes & sends request via TCP socket
- Server executes operation on its local
- Serializes response and sends it back



FILESYSTEM IN USERSPACE (FUSE)

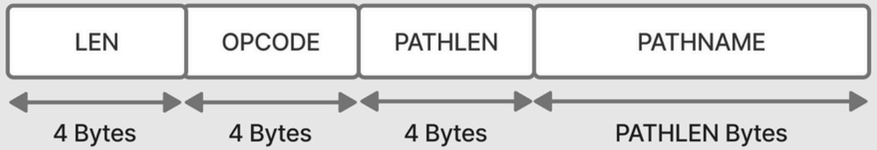
- FUSE lets user-space programs implement and mount filesystems without kernel code.
- The kernel module serializes requests and queues them via /dev/fuse.
- The user-space daemon (client) reads, processes requests, and sends replies back.
- The kernel delivers the final result to the application.



NFS PROTOCOL, RFC 1094!

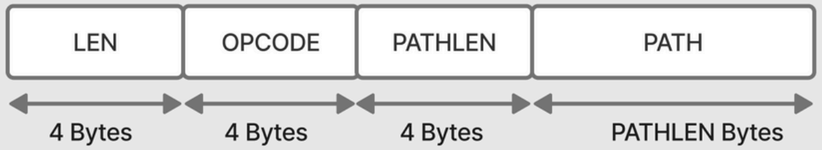
- Designed protocol based on simplified version of NFSv2 (**RFC 1094**).
- Implemented 13 main operations (READ, WRITE, OPEN, MKDIR, etc.).
- Each operation has an opcode + payload (e.g. FD, Offset, Data).

GETATTR Request:

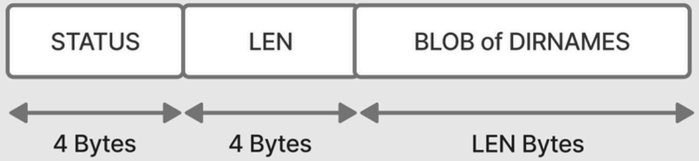


GETATTR Response:

READDIR Request:

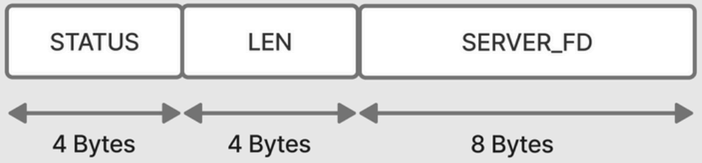


READDIR Response:

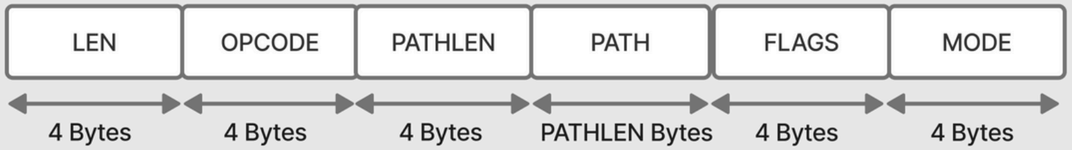


OP_OPEN Request:

OP_OPEN Response:



OP_CREATE Request:



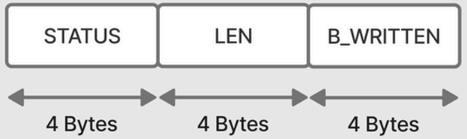
OP_CREATE Response:

OP_READ Request:

OP_READ Response:

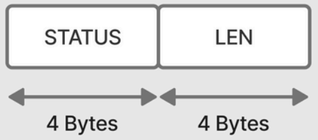
OP_WRITE Request:

OP_WRITE Response:



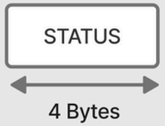
OP_MKDIR Request:

OP_MKDIR Response:



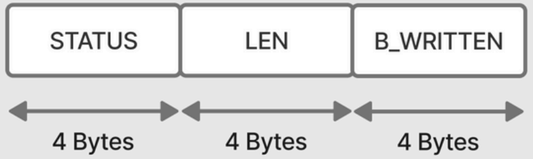
OP_RELEASE Request:

OP_RELEASE Response:

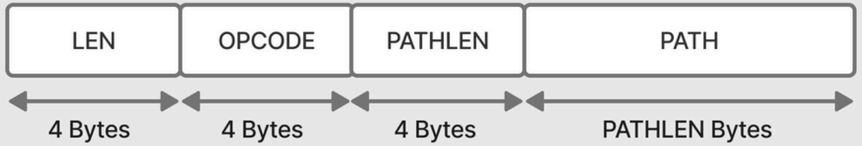


OP_WRITE_BATCH Request:

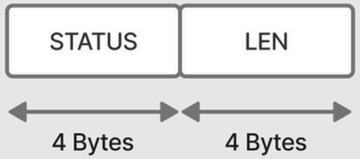
OP_WRITE_BATCH Response:



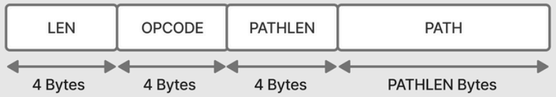
OP_UNLINK Request:



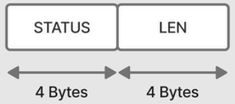
OP_UNLINK Response:



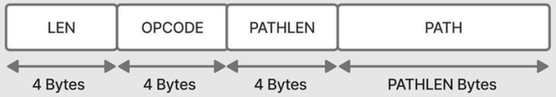
OP_RMDIR Request:



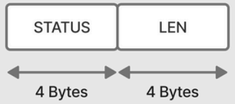
OP_RMDIR Response:



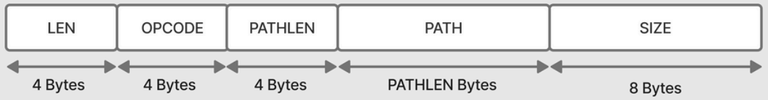
OP_RMDIR Request:



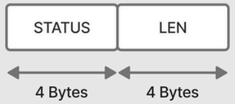
OP_RMDIR Response:



OP_TRUNCATE Request:

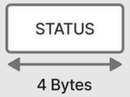


OP_TRUNCATE Response:



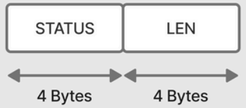
OP_RELEASE Request:

OP_RELEASE Response:

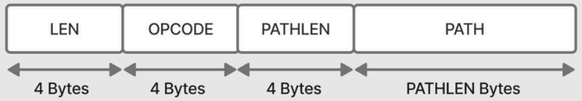


OP_UTIMENS Request:

OP_UTIMENS Response:



OP_STATFS Request:



OP_STATFS Response:

DESIGN CHOICES

Optimizations Implemented:

- Chunking: Split large files into small transfers.
- Caching: Data & attribute caching for quick access.
- Write-back: Batch writes to minimize network latency.
- Connection Pooling: Multiple TCP links for concurrency.
- Read-Ahead: Prefetch sequential blocks.
- Reader Writer Lock: Simultaneous Read and Exclusive Write

CACHING & WRITE-BACK

Caching (LRU):

- Keeps frequently used file blocks in memory.
- Evicts least recently used on overflow.

Write-Back Logic:

- Buffers sequential writes locally.
- Flushes in batches or on mismatch.
- Boosts throughput, lowers latency.

Trade-Off:

- Slight data-loss risk if client crashes.

PARALLELISM & READ OPTIMIZATION

Connection Pool:

- Pool of sockets reused for each operation.
- Avoids blocking; allows true parallel I/O.

Read-Ahead:

- Prefetch nearby data for sequential reads.
- Reduces round-trip delays and boosts throughput.

READER-WRITER LOCK

- Allows concurrent reads but exclusive writes.
- Uses shared locks for readers, unique locks for writers.
- Improves performance for read heavy workloads.

Trade-Off:

- Requires careful synchronization to prevent deadlocks.

RESULT

Read and Write Throughput Comparison (for 50 MB file using fio):

NFS Version	Read throughput (KiB/s)	Write throughput (KiB/s)
NFSv3	6603	1535
NFSv4	8878	1506
LNFSv1 (ours)	2580	1270

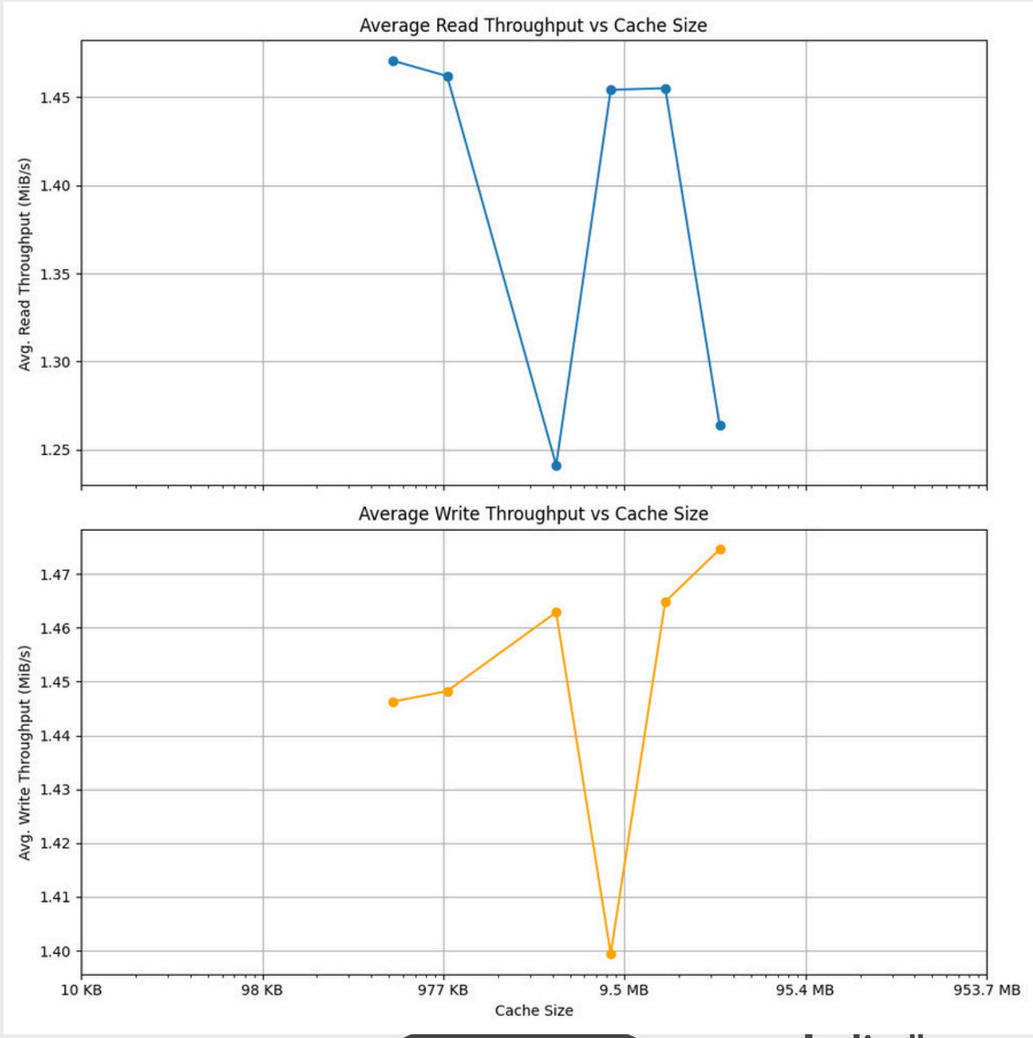
Analysis:

- NFSv4 shows the highest read throughput (8878 KiB/s), followed by NFSv3 (6603 KiB/s).
- LNFSv1 records a much lower read speed (2580 KiB/s).
- Write throughput remains comparable across all versions (≈ 1.2–1.5 MiB/s).
- Overall, LNFSv1 lags behind standard NFS versions, mainly in read performance.

RESULT

Average Throughput vs Cache Size:

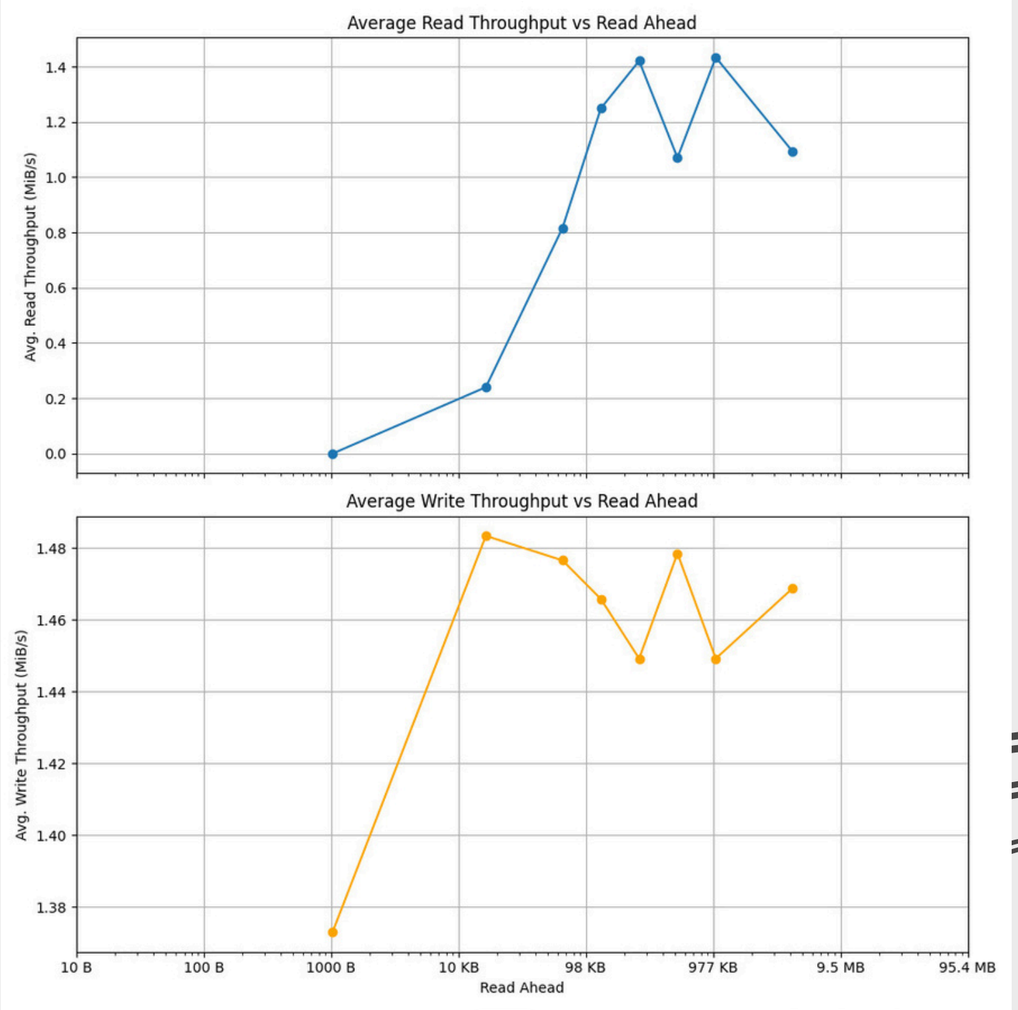
- Increasing cache size shows minimal impact on read throughput (variation within ±0.2 KiB/s).
- This is because the 10 MiB test file fits entirely in cache, limiting further gains.
- Throughput also depends on how fio performs read/write operations.
- The trend aligns with read-ahead optimization, which boosts sequential reads.
- Write performance remains largely unaffected, influenced by other system factors.



RESULT

Average Throughput vs Read-Ahead Size:

- Larger read-ahead size increases caching efficiency, resulting in higher read throughput.
- Throughput growth saturates at the network bandwidth limit.
- Write throughput remains mostly unaffected, showing only minor variations (≈0.1–0.3 KiB/s) due to network conditions.



CONCLUSION

- Implemented end-to-end Network File System using FUSE & TCP.
- Explored real-world aspects:
  - Caching
  - Write-back buffering
  - Concurrency
- Gained practical understanding of distributed file system design.

FUTURE WORK

- Distributing client requests across multiple servers to improve scalability and reduce bottlenecks.
- Maintaining file replicas across servers to ensure reliability and quick recovery from failures.
- Replacing per client threading with per request threading at server side to achieve finer grained concurrency and better resource utilisation.

THANK YOU