Benchmarking Report: memFS- A Fast, In-Memory File System

Ajay Choudhury (24CS60R85)
CS69201: Computing Lab- 1
Department of Computer Science and Engineering(CSE)
Indian Institute of Technology Kharagpur, West Bengal, India

---

1 Introduction

This report presents a comprehensive analysis of the performance characteristics of memFS, an in-memory file system implementation. The benchmarking was conducted across various workloads and thread configurations to evaluate system performance, resource utilization, and scalability.

2 Benchmark Configuration

2.1 Workload Scenarios

- Small workload: 100 operations
- Medium workload: 1,000 operations
- Large workload: 10,000 operations

2.2 Thread Configurations

- Single-threaded: 1 thread
- Multi-threaded: 2, 4, 8, and 16 threads

2.3 Test Operations

Each workload consists of a sequence of:

- File creation
- Write operations (2KB content): some content is generated.
- Read operations
- File deletion

2.4 Performance Metrics

The following metrics were measured for each configuration:

- Elapsed Time (milliseconds)
- CPU Time (seconds)
- Memory Usage (bytes)
- Average Operation Latency (microseconds)

3 Results Analysis

3.1 Results

For small workload i.e., for 100 operations,

Threads	Elapsed time	CPU time	Memory used	Average latency
1	10.5723 ms	0.009709 s	2 MB	100.86 μs
2	13.7957 ms	0.010066 s	2 MB	238.63 μs
4	16.3031 ms	0.01293 s	4 MB	488.86 μs
8	15.2214 ms	0.009428 s	4 MB	864.32 μs
16	17.3038 ms	0.010769 s	8 MB	1481.54 μs

For medium workload i.e., 1000 operations,

Threads	Elapsed time	CPU time	Memory used	Average latency
1	88.6319 ms	0.086229 s	8MB	87.454 μs
2	148.034 ms	0.165798 s	8MB	286.273 μs
4	159.748 ms	0.17793 s	8MB	596.917 μs
8	160.35 ms	0.165485 s	8MB	1039.85 μs
16	154.005 ms	0.144902 s	8MB	1875.48 μs

For heavy workload, i.e., 10000 operations,

Threads	Elapsed time	CPU time	Memory used	Average latency
1	837.194 ms	0.819753 s	8MB	82.7742 μs
2	1472.65 ms	1.63526 s	8MB	288.098 μs
4	1562.06 ms	1.77135 s	8MB	586.643 μs
8	1519.57 ms	1.71643 s	8MB	1119.3 μs
16	1607.33 ms	1.58886 s	8MB	2363.49 μs

3.2 Latency Analysis

- **Small Workload (100 operations)**
  - Single-threaded execution provides consistent latency
  - Thread overhead dominates with multiple threads

  - Optimal thread count: 8 threads
  - The CPU time takes advantage from multiple threads with increasing threads.
  - Additional threads increase coordination overhead

- **Medium Workload (1,000 operations))**
  - Benefits from parallelization become apparent
  - Sweet spot at 4-8 threads
  - Diminishing returns beyond 8 threads
  - Lower per-operation latency compared to small workload

- **Large Workload (10,000 operations))**
  - Maximum benefit from parallelization
  - Scales well up to 8 threads
  - Marginal improvements with 16 threads
  - Lowest per-operation latency across all workloads

3.3 Resource Utilization

- **CPU Utilization**
  - Single-threaded: Linear CPU usage
  - Multi-threaded: Increased CPU utilization with thread count
  - Context switching overhead becomes significant at 16 threads
  - CPU time increases with workload size but not linearly, for heavy and medium workload the CPU time decreases a little till 16 threads but for smaller workload use of 16 threads backfires and CPU time takes a minor hit for 16 threads.

- **Memory Usage**
  - Baseline memory footprint is minimal
  - Linear scaling with number of concurrent operations
  - Memory usage primarily dependent on workload size
  - Thread count has minimal impact on memory consumption

4 Key Observations

4.1 Thread Scaling

- Small workloads (100 ops): Limited benefit from multiple threads

- Medium workloads (1,000 ops): Optimal at 4-8 threads
- Large workloads (10,000 ops): Best performance with 8-16 threads

4.2 Performance Bottlenecks

- Thread creation/destruction overhead
- Memory allocation/deallocation
- Synchronization costs in multi-threaded scenarios

4.3 Resource Efficiency

- Memory usage scales linearly with data size
- CPU utilization shows diminishing returns with thread count
- System demonstrates good resource management

4.4 Optimaztions that can be helpful

- **Workload-Based Threading:**
  - Small workloads: Use 2-4 threads
  - Medium workloads: Use 4-8 threads
  - Large workloads: Use 8-16 threads
- **Resource Optimization**
  - Implement thread pooling for better thread management
  - Consider batch operations for small workloads
  - Optimize memory allocation patterns
- **Workload-Based Threading**
  - Small workloads: Use 2-4 threads
  - Medium workloads: Use 4-8 threads
  - Large workloads: Use 8-16 threads

5 Conclusion

The benchmark results demonstrate that memFS provides efficient in-memory file operations with good scalability characteristics. The system shows optimal performance when thread count is matched to workload size, with larger workloads benefiting more from increased parallelism. Memory usage remains efficient across different configurations, while CPU utilization shows expected scaling with thread count.

The implementation successfully manages to maintain the average operation times in the millisecond range, while providing thread-safe operations for concurrent access.