Rate Limiter - Final Report

Rate Limiter

Abstract

In the age of cloud services and API-driven platforms, rate limiting is essential to prevent abuse and ensure fair usage of system resources. This mini-project aims to implement a user-specific rate limiter in C that efficiently tracks and controls API request frequencies using hash tables and queues. The final solution offers fast lookups and memory-efficient management of request timestamps.

Problem Description

Web applications often expose APIs to multiple users. Without limitations, some users can overload the service, leading to denial of service for others. A rate limiter is required to allow only N requests per T seconds per user. If a user exceeds this rate, further requests should be blocked temporarily. The goal is to develop a rate limiter that tracks users and their request times, blocking any user exceeding the threshold.

Implementation Details

The solution is implemented in C using standard libraries. Each user is associated with a circular queue that stores request timestamps. A hash table maps user IDs to their respective queues, allowing quick lookup. On every request, expired timestamps are removed, and the request is allowed only if the queue has space (i.e., within the allowed limit). The final demo tests a single user with 5 request attempts over 5 seconds, enforcing a 3-requests-per-5-seconds limit.

Data Structures Used

Rate Limiter - Final Report

1. Hash Table: Maps each user ID to a queue for constant-time lookup.

2. Circular Queue: Manages timestamps efficiently using FIFO logic. Older timestamps are dequeued when they fall outside the valid time window, ensuring accurate tracking.

Challenges Faced

- Ensuring accurate time tracking using the C `time.h` library.
- Managing memory dynamically for queues and nodes to avoid leaks.
- Debugging edge cases such as simultaneous expirations or queue overflow.
- Simplifying the design for clarity while retaining essential functionality.

Conclusion and Learnings

This project deepened understanding of how real-world systems manage resource constraints. By working with fundamental data structures like hash tables and queues in a low-level language like C, the experience emphasized memory management, performance optimization, and clean code organization. The final product is modular, easy to test, and demonstrates core data structure concepts effectively.