Real-time Chat System with Message Processing

Programming Data Structures and Algorithms

Higher National Diploma in Software Engineering 24.2F

RATHNAYAKE R M I R	COHNDSE242F-012
PARANAGAMA H M T	COHNDSE242F-014
DASANAYAKA S A R S	COHNDSE242F-098
PERERA K P R	COHNDSE242F-110

School of Computing and Engineering
National Institute of Business Management
Colombo-7

1. Introduction

Modern chat systems face issues such as message prioritization, failed delivery, and network efficiency. Our solution addresses these challenges using multiple queue-based data structures, ensuring fast, reliable, and scalable communication. Target users include educational institutions, small businesses, and development teams.

2. Functional Requirements

2.1 Input Requirements

- Text messages (≤ 2000 characters)
- File attachments (≤ 10 MB)
- Priority selection (Urgent, High, Normal, Low) + auto-priority detection (@mentions, keywords)
- User authentication credentials
- Chat room join/leave requests

2.2 Process Requirements

- Priority Queue (Min-Heap): orders messages by urgency
- Circular Queue: stores last 1000 messages as history
- Batch Queue (FIFO): groups 5–50 messages for network efficiency
- Retry Queue (Deque): handles failed messages with exponential backoff
- Session Queue: processes login/authentication
- Spam Detection Queue: filters suspicious messages
- Offline Queue: stores messages for disconnected users

2.3 Output Requirements

- Messages delivered in priority order with indicators
- Real-time batch transmission with efficiency stats
- Queue monitoring (size, rate, retries)
- Delivery confirmations and failure notifications
- Performance reports (latency, throughput, resource use)

3. Data Structures and Justification

Our system is built entirely on the **Queue Data Structure**, applied in different forms (priority queue, circular queue, batch queue, retry deque, session queue, etc.). **Justification :**

1. They naturally follow **FIFO (First-In, First-Out)** order, which matches message processing.
2. Variants like **Priority Queue** allow urgent-first delivery.
3. **Circular Queues** provide efficient message history with constant-time operations.
4. **Batch and Retry Queues** improve network efficiency and ensure reliability.
5. Queues offer predictable **time complexity (O(1) or O(log n))**, ensuring scalability.