Milestone 3: Multi-Level Generalized Reader-Writer System

# Introduction

In this milestone, we developed a Java simulation of a multi-level system based on the generalized Reader-Writer problem. The system consists of three levels: multiple writers, a processor, and multiple readers, all communicating through shared memory with proper synchronization.

# Level 1: Writers

At the first level, the system includes three writer threads, each responsible for writing data to a shared memory block called SharedMemory1. Each writer works independently and writes values (such as numbers) to the memory. Since shared memory can only be accessed by one writer at a time to avoid corruption, exclusive access is enforced using Java synchronization (synchronized, wait(), and notifyAll()). Each writer waits for its turn. Once the memory is free, it writes its data and signals that the data is ready for the next level.

# Level 2: Processor (Middle Reader)

The second level is handled by a single processor thread, which acts like a middle reader and writer. It performs the following steps:

1. Reads data from SharedMemory1 written by the writers.

2. Processes the data — in this case, it multiplies the value by 2.

3. Writes the processed result into a new shared memory block called SharedMemory2.

The processor must also use synchronization to avoid reading data too early or writing too early.

# Level 3: Readers

The third level consists of three reader threads. These threads read concurrently from SharedMemory2, which contains the final processed data. They can read at the same time because the memory at this stage supports concurrent reading — no thread modifies the data after it's written.

# Synchronization

To make the whole system work without errors, we used proper synchronization techniques:

- Writers have exclusive write access to SharedMemory1.

- The processor waits until data is available and signals after processing is done.

- Readers read in parallel from SharedMemory2, but only when new data is ready.

Java features used include: synchronized, wait(), notifyAll(), and boolean flags to indicate data availability.

# Example Flow

1. Writer 1 writes: 3

2. Processor reads 3, processes it to 6, and writes it to the second memory.

3. Readers A, B, and C all read 6 and print it.

This continues for multiple values, showing the smooth flow of data across three synchronized levels.

# Conclusion

This milestone shows a working and synchronized three-level Reader-Writer system in Java. It demonstrates how threads can communicate safely using shared memory and proper coordination. The simulation successfully ensures that:

- Writers don’t interfere with each other.

- The processor handles data safely and efficiently.

- Readers access final data without conflict.

The result is a clear, functional simulation of a real-world processing pipeline.