IP PARSER: HIGH LEVEL DESIGN

FUNCTIONAL REQUIREMENTS

Input:

- Accept the input filename as a command line argument.

IP Validation:

- Use regex matching to match V4 and V6 addresses.
- Implement an IPMatcher class encapsulating all the required features for the matcher.

Parallelism:

- Create a thread pool with a fixed number of workers(consumers) and use Mutex and signals to synchronize activities.
- The main thread acts as the producer.

Batch Processing:

- Read input line by line and split the file into batches
- The producer generates batches and adds them to a common list.
- The consumer checks for available batches to process and process them once it's available.

Total count and Unique IP processing:

- Each worker maintains its own IP counts and unique IP. Once all threads are done, the outcomes of individual workers are combined to generate the final output.

WORKFLOW (PRODUCER - CONSUMER PROBLEM)

- Parse command-line arguments to get the input file path.
- Initialize the IPMatcher class for address validation.
- Create worker threads(consumers)
- Read input line by line and generate batches of size BATCH_SIZE
- Use two conditional variables cv_free(to wait until a free slot is available, used by the producer) and cv_fill(to wait until a filled slot is available, used by the consumer)
- The producer(main thread) keeps generating batches and once a free slot is available it will signal the consumer.
- The consumer will wait until a filled slot(batch) is available and once it's available it will process the batch and signal the producer that a free slot is available.
- The workers maintain individual total count and unique IP
- Combine results of workers to generate final result.
- **HOW SLOTS ARE MAINTAINED?**
    - Maintain two arrays free_slots and filled_slots that maintain the index of the batch in the buffer batch array.

NON FUNCTIONAL REQUIREMENTS

Scalability:

- The program should handle input files of varying sizes efficiently

Concurrency:

- Use mutexes to protect shared data structures

Memory Usage:

- Manage memory usage by processing input data in manageable batches to avoid excessive memory consumption

Performance:

- Leverage parallel processing to improve performance by utilizing multiple CPU cores

Modularity:

- Organize the code into modular components: input handling, IP address validation, threading, batch processing, and output generation.

Maintainability:

- Document the code and follow coding standards to ensure readability and ease of maintenance.

Optimization:

- Optimize the regular expressions for IP address validation to ensure efficient pattern matching.