# Concurrent MapReduce Framework Using Operating System Concept

## From:

* Hunain Maqbool
* Hammad Ali
* Muhammad Faheem

## To: Sir Ahmed Din

# Table of Contents

## Introduction

## Activity Diagram

### Components

## Implementation Details

### Key Objectives

### MapReduce Workflow

## Code Breakdown

### Key Components

## Test Cases

## Conclusion

### Key Concepts Applied

## Introduction

This report provides a comprehensive overview of the implementation and workflow of a basic MapReduce Framework using fundamental operating system concepts. The primary objective of this project is to simulate the MapReduce algorithm on a single machine, showcasing the application of concurrent computing and synchronization in distributed data processing. The key phases—Map, Shuffle, and Reduce—are implemented in C++ with the use of pthreads for parallel processing.

## Activity Diagram

The activity diagram (referenced below) illustrates the step-by-step workflow of the project implementation, capturing key operations like initialization, multithreading in the Map and Reduce phases, and synchronization mechanisms.

### Diagram

A diagram of a computer program

Description automatically generated with medium confidence

### Components

#### Main Nodes

* **Start:** Input the sentence for processing.
* **Tokenization:** Split the sentence into individual words.
* **Map Phase:**
  + Start Map threads.
  + Process chunks of words.
* **Synchronization:** Use mutex locks to safely write intermediate key-value pairs.
* **Shuffle Phase:** Group key-value pairs by keys.
* **Reduce Phase:**
  + Start Reduce threads.
  + Aggregate values for each key.
* **Output:** Display the final result as key-value pairs.

#### Decision Points

* Check if input is valid (non-empty).
* Ensure thread count does not exceed available system resources.

#### End Nodes

* Display final output or show error messages for invalid input.

## Implementation Details

### Key Objectives

* Simulating the MapReduce algorithm on a single machine.
* Implementing concurrency using pthreads for parallel processing.
* Ensuring thread synchronization using mutex locks.
* Simulating Map, Shuffle, and Reduce phases to process data efficiently.

### MapReduce Workflow

#### Input Data

The input data is a single string sentence, which is tokenized into individual words. For example:

* **Input:** "pizza burger pasta pasta pizza"
* **Tokenized Words:** ["pizza", "burger", "pasta", "pasta", "pizza"]

#### Map Phase

* Each thread processes a portion of the input words.
* Produces intermediate key-value pairs (e.g., ("pizza", 1)).

#### Shuffle Phase

Groups all key-value pairs by key.

***Example:***

* **Intermediate Pairs:** [("pizza", 1), ("burger", 1), ("pasta", 1), ("pasta", 1), ("pizza", 1)]
* **After Shuffle:** [("pizza", [1, 1]), ("burger", [1]), ("pasta", [1, 1])]

#### Reduce Phase

Aggregates values for each key and produces the final output.

***Example:***

* **Input to Reduce:** [("pizza", [1, 1]), ("burger", [1]), ("pasta", [1, 1])]
* **Final Output:** [("pizza", 2), ("burger", 1), ("pasta", 2)]

## Code Breakdown

### Key Components

#### Tokenization Function

* Converts the input string into words.
* Allocates memory dynamically for each word.

#### Multithreaded Map Phase

* Threads distribute the workload to process chunks of words.
* Mutex locks ensure safe updating of shared data structures.

#### Shuffle Phase

* Groups key-value pairs by their keys, ensuring all occurrences of a key are together.

#### Multithreaded Reduce Phase

* Threads process grouped key-value pairs and compute the final output.

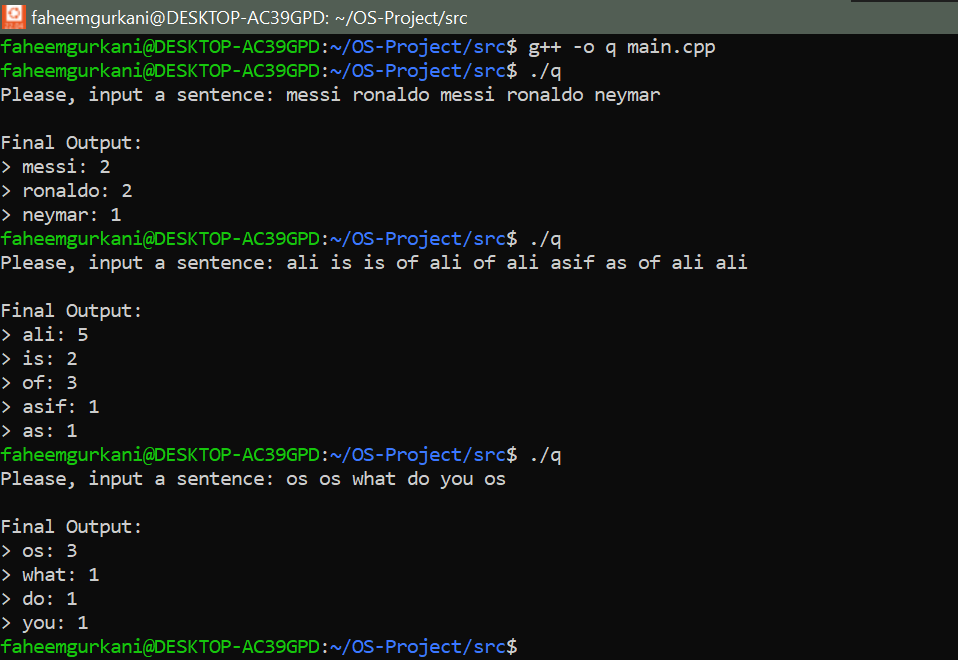
#### Synchronization

* Mutex locks manage thread-safe access to shared resources during Map and Reduce phases.

## Conclusion

The file for the test cases is included within the **testCases** directory included in the **src** directory of the folder.

### Test Cases’ Output



## Conclusion

### Key Concepts Applied

* **Concurrency:** Implementing parallel processing using pthreads.
* **Synchronization:** Using mutex locks to avoid race conditions.
* **Operating System Principles:** Applying concepts like thread management and shared memory.

This project successfully demonstrates the implementation of a basic MapReduce framework on a single machine using operating system concepts. The activity diagram provides a clear visual representation of the project's workflow, facilitating better understanding and aiding debugging efforts.