OPERATING SYSTEMS

CS-0205

Semester Project Report

Fall-2024

“Implementing a Basic MapReduce Framework”

Submitted By

FAMIA MALIK , ZOHA BINTE WAJAHAT, ABDULLAH BUTT
22I-2375 , 22I-0569 , 22I-0591
SECTION: C

DEPARTMENT OF COMPUTING, FAST-NUCES

ISLAMABAD

08/DECEMBER/2024

TABLE OF CONTENTS:

ABSTRACT:

This project develops a streamlined MapReduce framework to efficiently process large datasets by utilizing parallel processing through multithreading and synchronization techniques. The solution consists of three main phases: the Map Phase, where input data is divided into chunks and processed in parallel to create key-value pairs; the Shuffle Phase, where key-value pairs are grouped and aggregated based on their keys; and the Reduce Phase, where the aggregated results are combined to generate the final output.

To maintain thread safety during concurrent operations, semaphores and mutexes are employed to manage access to shared resources. The system is designed to accommodate multiple files, large datasets, and varying input sizes, offering both flexibility and scalability. The solution effectively harnesses multi-core processors to enhance performance, while robust error handling ensures smooth execution, even in the presence of missing or empty files. This implementation showcases the capabilities of the MapReduce model in data processing, providing a reliable and scalable approach for big data applications.

I. INTRODUCTION:

Overview:

The project showcases a simplified MapReduce framework that utilizes fundamental operating system principles such as **multithreading** and **synchronization**. This framework processes large datasets by breaking them into smaller segments, executing computations in parallel, and efficiently merging the outcomes. The simulation emphasizes the application of POSIX threads (pthreads) and semaphores to achieve parallel processing and maintain thread safety.

Objective Statement:

This project seeks to design, MapReduce framework using the concepts of Operating Systems.
The goal to:

1. Understand the MapReduce computational model.
2. Develop multithreaded processing to parallelize the Map, Shuffle, and Reduce stages.
3. Mimic real-world distributed data processing concepts on a single machine.
4. Reinforce knowledge of synchronization mechanisms like mutexes and semaphores.

Scope

The project aims to implement and assess the MapReduce programming model for parallel data processing. It encompasses the following main components:

Implementation:

- **Framework Development:** Creation of a MapReduce framework utilizing multithreading to efficiently process large datasets.
- **Core Phases:** Execution of the essential stages of the MapReduce process: Map, Shuffle, and Reduce.

Synchronization:

- **Mechanism Integration:** Incorporation of synchronization techniques to ensure accurate data management and prevent race conditions among threads.

Performance Evaluation:

- **Performance Analysis:** Evaluation of the implemented MapReduce framework by measuring execution time, scalability, and throughput.

Data Processing:

- **Textual Data Manipulation:** Processing and analyzing textual data to derive meaningful insights through parallel computations.
- **Demonstration of Capabilities:** Showcasing how large datasets can be divided, processed simultaneously, and effectively aggregated.

II. PROBLEM STATEMENT:

Statement:

Processing large datasets is a resource-intensive task that frequently demands considerable time and computational power. Conventional sequential processing methods are inefficient and fail to fully utilize modern multi-core processors, resulting in slower performance when managing substantial amounts of data. Furthermore, maintaining data integrity during concurrent processing presents a significant challenge in parallel computing.

III. IMPLEMENTATION STRATEGY:

Explanation:

The implementation of the MapReduce mode is structured into the following phases:

Input Data Processing:

The input data is supplied as a text file containing extensive information. This file is read and segmented into smaller chunks to enable parallel processing. Each chunk represents a portion of the input data and is allocated to a separate thread for independent processing. This approach enhances the program's scalability, as additional threads can handle more chunks when necessary, optimizing the use of system resources.

The size of each chunk is dynamically determined based on the overall input data size and the number of available threads, ensuring a balanced workload among the threads. The reading process guarantees that data chunks are neither overlapping nor missing any segments of the input data.

Map Phase:

During the map phase, the program processes each chunk of input data concurrently using multiple threads.

Each thread applies a map function to its designated chunk, extracting key-value pairs from the data. Keys typically signify unique words, while values indicate their occurrences within the chunk. For instance, for the input text "apple banana apple," the map phase for a single chunk might produce key-value pairs like (apple, 2) and (banana, 1). The output from this phase consists of intermediate key-value pairs, which are temporarily stored in thread-specific data structures to prevent contention for shared resources.

Shuffle Phase:

The shuffle phase is crucial in the MapReduce process as it groups intermediate key-value pairs by their keys.

The intermediate results from all threads are combined into a centralized data structure. Keys with the same values are clustered together, setting the stage for aggregation in the

reduce phase. Efficient algorithms are utilized to minimize the time complexity of grouping operations, ensuring that the shuffle phase does not become a performance bottleneck. This phase also includes thread-safe access to the shared grouping structure, facilitated by synchronization mechanisms.

Reduce Phase:

In the reduce phase, the grouped key-value pairs are aggregated to generate the final results.

For each key, the program iterates through its associated values and performs necessary computations, such as summing occurrences to determine word counts. This phase consolidates the intermediate results from all threads into a unified output, reflecting the overall analysis of the input data. The reduce phase ensures that the final output is well-structured, formatted, and ready for further processing or presentation.

Thread Synchronization:

Thread synchronization is a vital component of this project, as multiple threads interact with shared data structures during the shuffle and reduce phases.

Mutexes: Mutexes are used to safeguard shared data from concurrent write operations, preventing race conditions. For example, when multiple threads try to insert data into the same grouped key-value structure, mutexes ensure that only one thread can modify it at a time.

Semaphores: Semaphores are employed to efficiently manage shared resources and coordinate thread execution. They regulate access to critical sections of the code, ensuring that threads function in a specified order to avoid deadlocks or resource contention.

These synchronization mechanisms uphold the integrity of the program's execution and guarantee accurate results, even when processing large datasets.

Divide and Conquer Paradigm:

The MapReduce model adopts a divide-and-conquer approach:

Splitting Phase: Data is divided into manageable chunks for efficient processing.
Map Phase: Processes chunks independently in parallel, leveraging multithreading for improved performance.
Shuffle Phase: Aggregates intermediate results for efficient data organization.
Reduce Phase: Produces the final output sequentially by combining and reducing the intermediate results.

IV. CODE IMPLEMENTATION

Variables & Data Structure:

CODE:

```
// Structure to store key-value pairs
struct KeyValue
{
    string key;
    int value;

    KeyValue(string k, int v) : key(k), value(v) {}  // Constructor to initialize key and value
};

// Shared semaphore and mutex for synchronization
sem_t semaphoreLock;  // Semaphore to ensure thread-safe operations
pthread_mutex_t outputMutex = PTHREAD_MUTEX_INITIALIZER;  // Mutex to lock output section
```

EXPLINATION:

Key Value Struct

This structure represents a key-value pair. Each key is a word (string) and th value is the count of occurrences (int).

semaphoreLock:

A semaphore is initialized here for thread-safe access to shared resources. It ensures that only one thread can modify the key-value store at a time.

outputMutex:

A mutex is used to prevent race conditions when multiple threads try to print to the console simultaneously

Splitting the Text into Chunks:

CODE:

```
// Function to split input text into smaller chunks based on a defined chunk size
vector<string> splitTextIntoChunks(const string &inputText, size_t chunkSize)
{
    vector<string> chunks;  // Vector to store chunks of text
    stringstream textStream(inputText);  // Stream to read input text word by word
    string word, currentChunk;

    size_t currentSize = 0;  // Track current chunk size
    while (textStream >> word)
    {
        if (currentSize + word.size() + 1 > chunkSize)
        {  // If adding the word exceeds chunk size
            chunks.push_back(currentChunk);  // Add the chunk to the list
            currentChunk.clear();  // Reset current chunk
            currentSize = 0;
        }
        currentChunk += word + " ";  // Add word and space to the current chunk
        currentSize += word.size() + 1;
    }

    if (!currentChunk.empty())
    {  // Add the last chunk if it's not empty
        chunks.push_back(currentChunk);
    }

    return chunks;  // Return all created chunks
}
```

EXPLINATION:

Purpose:

"splitTextIntoChunks" function divides the input text into smaller segments of a specified size (chunkSize).

stringstream:

The input text is processed word by word.

Chunk Creation:

The function verifies whether adding a new word would exceed the chunk size. If it does, the current chunk is added to the chunks vector, and the function resets for the next chunk.

Final Chunk:

After the loop completes, any leftover text is appended to the chunks vector.

Mapping Phase (Thread Function):

CODE:

```
// Mapping Phase: Process a chunk of text and create key-value pairs
void *generateKeyValuePairs(void *arg)
{
    // Extract arguments
    pair<string, vector<KeyValue> *> *params = (pair<string, vector<KeyValue> *> *)arg;  // Cast arguments
    string chunk = params->first;  // Extract chunk text
    vector<KeyValue> *keyValueStore = params->second;  // Extract shared key-value storage

    stringstream chunkStream(chunk);
    string word;

    // Iterate through the chunk and generate key-value pairs
    while (chunkStream >> word)
    {
        sem_wait(&semaphoreLock);  // Lock access to shared storage
        keyValueStore->push_back(KeyValue(word, 1));  // Add (word, 1) pair
        sem_post(&semaphoreLock);  // Unlock access to shared storage
    }

    pthread_exit(NULL);  // Exit the thread
}
```

EXPLINATION:

Purpose:

"generateKeyValuePairs" function processes a segment of text in parallel and produces key-value pairs (word, 1).

Arguments:

The argument is cast to a pair that includes the text chunk and a reference to the shared key-value store.

Processing:

The function employs "stringstream" to break the chunk into individual words.

Synchronization:

Semaphores are utilized to ensure that only one thread can modify the shared key-value store at any given time.

Result:

Each word from the chunk is added as a key with a value of 1, and the results are stored in keyValueStore

Shuffling Phase (Grouping by Key):

CODE:

```
// Shuffling Phase: Group key-value pairs by key and aggregate their values
void groupAndAggregateKeyValuePairs(const vector<KeyValue> &keyValues, vector<KeyValue> &groupedResults)
{
    for (size_t i = 0; i < keyValues.size(); i++) {
        bool found = false;

        // Search for an existing key in grouped results
        for (size_t j = 0; j < groupedResults.size(); j++)
        {
            if (groupedResults[j].key == keyValues[i].key)
            {
                groupedResults[j].value += keyValues[i].value;  // Aggregate the value
                found = true;
                break;
            }
        }

        if (!found)
        {
            groupedResults.push_back(KeyValue(keyValues[i].key, keyValues[i].value));  // Add a new key-value
        }
    }
}
```

EXPLINATION:

Purpose:

"groupAndAggregateKeyValuePairs" function takes the key-value pairs produced during the map phase and organizes them by key.

Aggregation:

For each unique key, it totals the associated values (e.g., counting how many times each word appears).

Logic:

- If the key already exists in groupedResults, it adds the value to the existing entry.
- If the key is new, it creates a new key-value pair in groupedResults

Reducing Phase:

CODE:

```
// Reducing Phase: Display reduced results
void *displayReducedResults(void *arg)
{
    vector<KeyValue> *groupedResults = (vector<KeyValue> *)arg;  // Cast the grouped results

    pthread_mutex_lock(&outputMutex);  // Lock the output section
    //cout << ALIGN_SPACES << "Final Reduced Results:" << endl;

     cout << "\n" << ALIGN_SPACES << "=========================================================\n";
    cout << ALIGN_SPACES << "                FINAL REDUCED RESULTS              \n";
    cout << ALIGN_SPACES << "=========================================================\n";


    for (size_t i = 0; i < groupedResults->size(); i++)
    {
        cout << ALIGN_SPACES << "(" << (*groupedResults)[i].key << ", " << (*groupedResults)[i].value << ")"
    }
    pthread_mutex_unlock(&outputMutex);  // Unlock the output section

    pthread_exit(NULL);  // Exit the thread
}
```

EXPLINATION:

Purpose:

This function manages the final aggregation results (the reduced outcomes) and displays them in a formatted manner.

Thread Safety:

A mutex is employed to lock the output section, preventing race conditions when multiple threads attempt to print to the console simultaneously.

Output:

It prints the aggregated key-value pairs, with keys representing words and values indicating their final counts.

Mapping Controller (Mapping & Creating Threads):

CODE:

```
// Mapping Controller: Distribute text chunks across threads for processing
void processChunksUsingThreads(const vector<string> &chunks, vector<KeyValue> &keyValueStore)
{
    vector<pthread_t> threads(chunks.size());  // Create threads for each chunk

    for (size_t i = 0; i < chunks.size(); i++)
    {
        pair<string, vector<KeyValue> *> *params = new pair<string, vector<KeyValue> *>(chunks[i], &keyValueSt
        pthread_create(&threads[i], NULL, generateKeyValuePairs, (void *)params);  // Start a thread for each
    }

    for (size_t i = 0; i < threads.size(); i++)
    {
        pthread_join(threads[i], NULL);  // Wait for all threads to complete
    }
}
```

EXPLINATION:

Purpose:

This function generates a thread for each chunk in the chunks vector.

Parameters Passed to Threads:

For each chunk, a pair consisting of the chunk and the shared key-value store is provided to the thread.

Waiting for Threads:

pthread_join is utilized to wait for all threads to finish before proceeding to the next phase.

Example Execution:

INPUT:

For input files with the following content:

File1.txt: ` Apples are delicious fruits that come in many varieties, such as Fuji, Gala, and Granny Smith. Bananas are a great source of potassium and are often enjoyed as a quick snack. Oranges, with their juicy and tangy flavor, are rich in vitamin C. Grapes, whether green or purple, make a great addition to salads or snacks. Mangoes are tropical delights loved for their sweetness. Peaches, with their soft and fuzzy skin, are a summer favorite. Kiwis, small and green, pack a punch of flavor and nutrition. Pears, with their unique texture, are a great addition to any fruit basket. Pineapples, with their spiky skin, are a symbol of hospitality and a source of refreshing juice. Strawberries, sweet and red, are loved by many for their vibrant taste and appearance. Each of these fruits offers a unique blend of nutrients and flavors that cater to diverse tastes and dietary needs.'

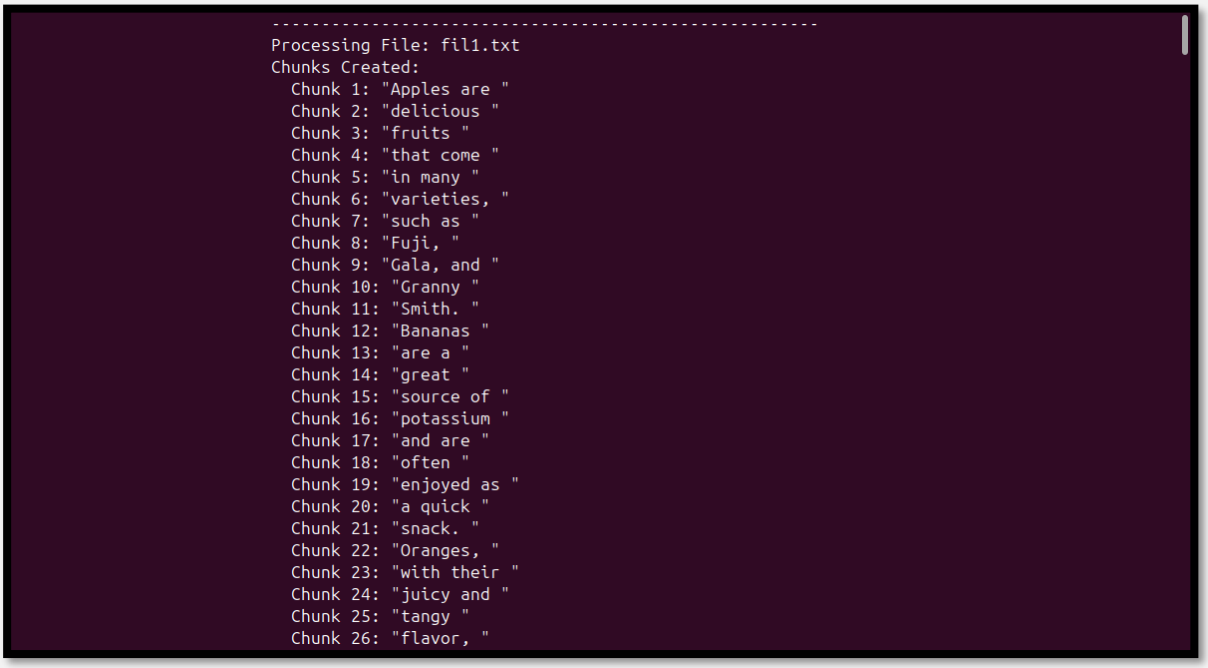
File2.txt: 'The lion, known as the king of the jungle, roars majestically as it patrols its territory. Tigers, with their orange and black stripes, are powerful and stealthy hunters. Elephants, gentle giants of the animal kingdom, have an incredible memory and social bonds. Wolves, living in packs, are intelligent and highly coordinated hunters. Foxes, with their bushy tails, are clever and adaptable creatures. Bears, massive and powerful, roam forests and mountains. Rabbits, small and quick, are a common sight in grassy meadows. Mice, tiny and resourceful, often live near human habitats. Cats, independent yet affectionate, are beloved pets worldwide. Dogs, loyal and energetic, are known as man's best friend. Each of these animals plays a vital role in their respective ecosystems, showcasing nature's diversity and balance.'

File3.txt: 'Cars, ranging from compact hatchbacks to luxury sedans, are a primary mode of transportation. Trucks, powerful and durable, are essential for carrying goods across long distances. Bicycles, eco-friendly and efficient, are used for commuting and recreation. Motorcycles, sleek and fast, appeal to those seeking an adrenaline rush. Trains, a symbol of industrial progress, connect cities and countries. Airplanes, soaring through the skies, make global travel possible in a matter of hours. Boats, gliding across rivers and lakes, are ideal for fishing and leisurely cruises. Ships, massive and robust, traverse oceans carrying passengers and cargo. Buses, reliable and affordable, serve as public transport in urban areas. Submarines, hidden beneath the

waves, perform critical military and research operations. Each vehicle has a unique role in facilitating human mobility and connectivity.'

File4.txt: 'Mountains, towering and majestic, offer breathtaking views and a sense of tranquility. Rivers, flowing endlessly, provide water for agriculture, industry, and life itself. Forests, dense with trees and wildlife, act as the lungs of the planet, producing oxygen. Lakes, serene and calm, are a haven for fish and aquatic plants. Hills, smaller yet picturesque, are often dotted with trails and wildflowers. Deserts, vast and arid, display resilience through their hardy vegetation and wildlife. Oceans, covering most of the Earth's surface, are teeming with marine life and ecosystems. Plains, expansive and fertile, are ideal for farming and grazing livestock. Glaciers, icy and ancient, hold the Earth's freshwater reserves. Rainforests, rich in biodiversity, are vital for maintaining the planet's ecological balance. Nature, in its infinite forms, provides sustenance, inspiration, and wonder.'

OUTPUT:



```
Chunk 25: "tangy "
Chunk 26: "flavor, "
Chunk 27: "are rich "
Chunk 28: "in vitamin "
Chunk 29: "C. Grapes, "
Chunk 30: "whether "
Chunk 31: "green or "
Chunk 32: "purple, "
Chunk 33: "make a "
Chunk 34: "great "
Chunk 35: "addition "
Chunk 36: "to salads "
Chunk 37: "or snacks. "
Chunk 38: "Mangoes "
Chunk 39: "are "
Chunk 40: "tropical "
Chunk 41: "delights "
Chunk 42: "loved for "
Chunk 43: "their "
Chunk 44: "sweetness. "
Chunk 45: "Peaches, "
Chunk 46: "with their "
Chunk 47: "soft and "
Chunk 48: "fuzzy "
Chunk 49: "skin, are "
Chunk 50: "a summer "
Chunk 51: "favorite. "
Chunk 52: "Kiwis, "
Chunk 53: "small and "
```

```
Chunk 52: "Kiwis, "
Chunk 53: "small and "
Chunk 54: "green, "
Chunk 55: "pack a "
Chunk 56: "punch of "
Chunk 57: "flavor and "
Chunk 58: "nutrition. "
Chunk 59: "Pears, "
Chunk 60: "with their "
Chunk 61: "unique "
Chunk 62: "texture, "
Chunk 63: "are a "
Chunk 64: "great "
Chunk 65: "addition "
Chunk 66: "to any "
Chunk 67: "fruit "
Chunk 68: "basket. "
Chunk 69: "Pineapples, "
Chunk 70: "with their "
Chunk 71: "spiky "
Chunk 72: "skin, are "
Chunk 73: "a symbol "
Chunk 74: "of "
Chunk 75: "hospitality "
Chunk 76: "and a "
Chunk 77: "source of "
Chunk 78: "refreshing "
Chunk 79: "juice. "
Chunk 80: "Strawberries, "
```

```
  Chunk 79: "juice. "
  Chunk 80: "Strawberries, "
  Chunk 81: "sweet and "
  Chunk 82: "red, are "
  Chunk 83: "loved by "
  Chunk 84: "many for "
  Chunk 85: "their "
  Chunk 86: "vibrant "
  Chunk 87: "taste and "
  Chunk 88: "appearance. "
  Chunk 89: "Each of "
  Chunk 90: "these "
  Chunk 91: "fruits "
  Chunk 92: "offers a "
  Chunk 93: "unique "
  Chunk 94: "blend of "
  Chunk 95: "nutrients "
  Chunk 96: "and "
  Chunk 97: "flavors "
  Chunk 98: "that cater "
  Chunk 99: "to diverse "
  Chunk 100: "tastes and "
  Chunk 101: "dietary "
  Chunk 102: "needs. "
Shuffling Results:
  (Apples, 1)
  (are, 9)
  (delicious, 1)
  (that, 2)
```

```
  (juicy, 1)
  (tangy, 1)
  (rich, 1)
  (flavor,, 1)
  (C., 1)
  (Grapes,, 1)
  (vitamin, 1)
  (whether, 1)
  (green, 1)
  (or, 2)
  (purple,, 1)
  (make, 1)
  (snacks., 1)
  (to, 3)
  (addition, 2)
  (salads, 1)
  (Mangoes, 1)
  (tropical, 1)
  (delights, 1)
  (loved, 2)
  (for, 2)
  (sweetness., 1)
  (summer, 1)
  (fuzzy, 1)
  (soft, 1)
  (skin,, 2)
  (Kiwis,, 1)
  (favorite., 1)
  (Peaches,, 1)
```

```
(favorite., 1)
(Peaches,, 1)
(small, 1)
(green,, 1)
(pack, 1)
(punch, 1)
(flavor, 1)
(nutrition., 1)
(Pears,, 1)
(unique, 2)
(texture,, 1)
(any, 1)
(basket., 1)
(fruit, 1)
(Pineapples,, 1)
(spiky, 1)
(hospitality, 1)
(symbol, 1)
(Strawberries,, 1)
(juice., 1)
(refreshing, 1)
(sweet, 1)
(red,, 1)
(by, 1)
(vibrant, 1)
(taste, 1)
(appearance., 1)
(these, 1)
(Each, 1)
```

```
(these, 1)
(Each, 1)
(offers, 1)
(nutrients, 1)
(blend, 1)
(flavors, 1)
(diverse, 1)
(needs., 1)
(dietary, 1)
(tastes, 1)
(cater, 1)
```

```
------------------------------------------------------
Processing File: fil2.txt
Chunks Created:
  Chunk 1: "The lion, "
  Chunk 2: "known as "
  Chunk 3: "the king "
  Chunk 4: "of the "
  Chunk 5: "jungle, "
  Chunk 6: "roars "
  Chunk 7: "majestically "
  Chunk 8: "as it "
  Chunk 9: "patrols "
  Chunk 10: "its "
  Chunk 11: "territory. "
  Chunk 12: "Tigers, "
  Chunk 13: "with their "
  Chunk 14: "orange and "
  Chunk 15: "black "
  Chunk 16: "stripes, "
  Chunk 17: "are "
  Chunk 18: "powerful "
  Chunk 19: "and "
  Chunk 20: "stealthy "
  Chunk 21: "hunters. "
  Chunk 22: "Elephants, "
  Chunk 23: "gentle "
  Chunk 24: "giants of "
  Chunk 25: "the animal "
  Chunk 26: "kingdom, "
```

```
  Chunk 25: "the animal "
  Chunk 26: "kingdom, "
  Chunk 27: "have an "
  Chunk 28: "incredible "
  Chunk 29: "memory and "
  Chunk 30: "social "
  Chunk 31: "bonds. "
  Chunk 32: "Wolves, "
  Chunk 33: "living in "
  Chunk 34: "packs, are "
  Chunk 35: "intelligent "
  Chunk 36: "and highly "
  Chunk 37: "coordinated "
  Chunk 38: "hunters. "
  Chunk 39: "Foxes, "
  Chunk 40: "with their "
  Chunk 41: "bushy "
  Chunk 42: "tails, are "
  Chunk 43: "clever and "
  Chunk 44: "adaptable "
  Chunk 45: "creatures. "
  Chunk 46: "Bears, "
  Chunk 47: "massive "
  Chunk 48: "and "
  Chunk 49: "powerful, "
  Chunk 50: "roam "
  Chunk 51: "forests "
  Chunk 52: "and "
  Chunk 53: "mountains. "
```

```
  Chunk 52: "and "
  Chunk 53: "mountains. "
  Chunk 54: "Rabbits, "
  Chunk 55: "small and "
  Chunk 56: "quick, are "
  Chunk 57: "a common "
  Chunk 58: "sight in "
  Chunk 59: "grassy "
  Chunk 60: "meadows. "
  Chunk 61: "Mice, tiny "
  Chunk 62: "and "
  Chunk 63: "resourceful, "
  Chunk 64: "often live "
  Chunk 65: "near human "
  Chunk 66: "habitats. "
  Chunk 67: "Cats, "
  Chunk 68: "independent "
  Chunk 69: "yet "
  Chunk 70: "affectionate, "
  Chunk 71: "are "
  Chunk 72: "beloved "
  Chunk 73: "pets "
  Chunk 74: "worldwide. "
  Chunk 75: "Dogs, "
  Chunk 76: "loyal and "
  Chunk 77: "energetic, "
  Chunk 78: "are known "
  Chunk 79: "as man's "
  Chunk 80: "best "
```

```
  Chunk 79: "as man's "
  Chunk 80: "best "
  Chunk 81: "friend. "
  Chunk 82: "Each of "
  Chunk 83: "these "
  Chunk 84: "animals "
  Chunk 85: "plays a "
  Chunk 86: "vital role "
  Chunk 87: "in their "
  Chunk 88: "respective "
  Chunk 89: "ecosystems, "
  Chunk 90: "showcasing "
  Chunk 91: "nature's "
  Chunk 92: "diversity "
  Chunk 93: "and "
  Chunk 94: "balance. "
Shuffling Results:
  (The, 1)
  (lion,, 1)
  (known, 2)
  (as, 3)
  (the, 3)
  (king, 1)
  (roars, 1)
  (it, 1)
  (majestically, 1)
  (jungle,, 1)
  (patrols, 1)
  (its, 1)
```

```
(coordinated, 1)
(Foxes,, 1)
(bushy, 1)
(living, 1)
(in, 3)
(tails,, 1)
(adaptable, 1)
(clever, 1)
(creatures., 1)
(massive, 1)
(Bears,, 1)
(powerful,, 1)
(forests, 1)
(roam, 1)
(mountains., 1)
(Rabbits,, 1)
(small, 1)
(quick,, 1)
(a, 2)
(common, 1)
(sight, 1)
(grassy, 1)
(meadows., 1)
(Mice,, 1)
(tiny, 1)
(resourceful,, 1)
(often, 1)
(live, 1)
(near, 1)
```

```
(habitats., 1)
(Cats,, 1)
(independent, 1)
(yet, 1)
(affectionate,, 1)
(beloved, 1)
(pets, 1)
(worldwide., 1)
(Dogs,, 1)
(loyal, 1)
(energetic,, 1)
(best, 1)
(man's, 1)
(Each, 1)
(these, 1)
(animals, 1)
(plays, 1)
(vital, 1)
(role, 1)
(friend., 1)
(ecosystems,, 1)
(showcasing, 1)
(nature's, 1)
(diversity, 1)
(balance., 1)
(respective, 1)

------------------------------------------------------
Processing File: fil3.txt
```

```
------------------------------------------------------
Processing File: fil3.txt
Chunks Created:
  Chunk 1: "Cars, "
  Chunk 2: "ranging "
  Chunk 3: "from "
  Chunk 4: "compact "
  Chunk 5: "hatchbacks "
  Chunk 6: "to luxury "
  Chunk 7: "sedans, "
  Chunk 8: "are a "
  Chunk 9: "primary "
  Chunk 10: "mode of "
  Chunk 11: "transportation. "
  Chunk 12: "Trucks, "
  Chunk 13: "powerful "
  Chunk 14: "and "
  Chunk 15: "durable, "
  Chunk 16: "are "
  Chunk 17: "essential "
  Chunk 18: "for "
  Chunk 19: "carrying "
  Chunk 20: "goods "
  Chunk 21: "across "
  Chunk 22: "long "
  Chunk 23: "distances. "
  Chunk 24: "Bicycles, "
  Chunk 25: "eco-friendly "
  Chunk 26: "and "
```

```
  Chunk 25: "eco-friendly "
  Chunk 26: "and "
  Chunk 27: "efficient, "
  Chunk 28: "are used "
  Chunk 29: "for "
  Chunk 30: "commuting "
  Chunk 31: "and "
  Chunk 32: "recreation. "
  Chunk 33: "Motorcycles, "
  Chunk 34: "sleek and "
  Chunk 35: "fast, "
  Chunk 36: "appeal to "
  Chunk 37: "those "
  Chunk 38: "seeking an "
  Chunk 39: "adrenaline "
  Chunk 40: "rush. "
  Chunk 41: "Trains, a "
  Chunk 42: "symbol of "
  Chunk 43: "industrial "
  Chunk 44: "progress, "
  Chunk 45: "connect "
  Chunk 46: "cities and "
  Chunk 47: "countries. "
  Chunk 48: "Airplanes, "
  Chunk 49: "soaring "
  Chunk 50: "through "
  Chunk 51: "the skies, "
  Chunk 52: "make "
  Chunk 53: "global "
```

```
  Chunk 54: "travel "
  Chunk 55: "possible "
  Chunk 56: "in a "
  Chunk 57: "matter of "
  Chunk 58: "hours. "
  Chunk 59: "Boats, "
  Chunk 60: "gliding "
  Chunk 61: "across "
  Chunk 62: "rivers and "
  Chunk 63: "lakes, are "
  Chunk 64: "ideal for "
  Chunk 65: "fishing "
  Chunk 66: "and "
  Chunk 67: "leisurely "
  Chunk 68: "cruises. "
  Chunk 69: "Ships, "
  Chunk 70: "massive "
  Chunk 71: "and "
  Chunk 72: "robust, "
  Chunk 73: "traverse "
  Chunk 74: "oceans "
  Chunk 75: "carrying "
  Chunk 76: "passengers "
  Chunk 77: "and cargo. "
  Chunk 78: "Buses, "
  Chunk 79: "reliable "
  Chunk 80: "and "
  Chunk 81: "affordable, "
  Chunk 82: "serve as "
```

```
  Chunk 81: "affordable, "
  Chunk 82: "serve as "
  Chunk 83: "public "
  Chunk 84: "transport "
  Chunk 85: "in urban "
  Chunk 86: "areas. "
  Chunk 87: "Submarines, "
  Chunk 88: "hidden "
  Chunk 89: "beneath "
  Chunk 90: "the waves, "
  Chunk 91: "perform "
  Chunk 92: "critical "
  Chunk 93: "military "
  Chunk 94: "and "
  Chunk 95: "research "
  Chunk 96: "operations. "
  Chunk 97: "Each "
  Chunk 98: "vehicle "
  Chunk 99: "has a "
  Chunk 100: "unique "
  Chunk 101: "role in "
  Chunk 102: "facilitating "
  Chunk 103: "human "
  Chunk 104: "mobility "
  Chunk 105: "and "
  Chunk 106: "connectivity. "
Shuffling Results:
  (Cars,, 1)
  (ranging, 1)
```

```
(efficient,, 1)
(used, 1)
(commuting, 1)
(recreation., 1)
(Motorcycles,, 1)
(sleek, 1)
(fast,, 1)
(appeal, 1)
(those, 1)
(seeking, 1)
(an, 1)
(adrenaline, 1)
(Trains,, 1)
(rush., 1)
(symbol, 1)
(industrial, 1)
(progress,, 1)
(connect, 1)
(cities, 1)
(countries., 1)
(Airplanes,, 1)
(soaring, 1)
(through, 1)
(the, 2)
(skies,, 1)
(make, 1)
(global, 1)
(travel, 1)
(possible, 1)
```

```
(travel, 1)
(possible, 1)
(in, 3)
(matter, 1)
(hours., 1)
(Boats,, 1)
(gliding, 1)
(rivers, 1)
(lakes,, 1)
(ideal, 1)
(fishing, 1)
(leisurely, 1)
(cruises., 1)
(Ships,, 1)
(massive, 1)
(robust,, 1)
(traverse, 1)
(oceans, 1)
(passengers, 1)
(cargo., 1)
(Buses,, 1)
(reliable, 1)
(affordable,, 1)
(serve, 1)
(as, 1)
(public, 1)
(transport, 1)
(urban, 1)
(areas., 1)
```

```
(areas., 1)
(Submarines,, 1)
(hidden, 1)
(beneath, 1)
(waves,, 1)
(perform, 1)
(critical, 1)
(military, 1)
(research, 1)
(operations., 1)
(Each, 1)
(vehicle, 1)
(has, 1)
(unique, 1)
(role, 1)
(facilitating, 1)
(human, 1)
(mobility, 1)
(connectivity., 1)
```

```
Chunk 79: "Glaciers, "
Chunk 80: "icy and "
Chunk 81: "ancient, "
Chunk 82: "hold the "
Chunk 83: "Earth's "
Chunk 84: "freshwater "
Chunk 85: "reserves. "
Chunk 86: "Rainforests, "
Chunk 87: "rich in "
Chunk 88: "biodiversity, "
Chunk 89: "are vital "
Chunk 90: "for "
Chunk 91: "maintaining "
Chunk 92: "the "
Chunk 93: "planet's "
Chunk 94: "ecological "
Chunk 95: "balance. "
Chunk 96: "Nature, in "
Chunk 97: "its "
Chunk 98: "infinite "
Chunk 99: "forms, "
Chunk 100: "provides "
Chunk 101: "sustenance, "
Chunk 102: "inspiration, "
Chunk 103: "and "
Chunk 104: "wonder. "
Shuffling Results:
(Mountains,, 1)
(and, 14)
```

```
Chunk 25: "the "
Chunk 26: "planet, "
Chunk 27: "producing "
Chunk 28: "oxygen. "
Chunk 29: "Lakes, "
Chunk 30: "serene and "
Chunk 31: "calm, are "
Chunk 32: "a haven "
Chunk 33: "for fish "
Chunk 34: "and "
Chunk 35: "aquatic "
Chunk 36: "plants. "
Chunk 37: "Hills, "
Chunk 38: "smaller "
Chunk 39: "yet "
Chunk 40: "picturesque, "
Chunk 41: "are often "
Chunk 42: "dotted "
Chunk 43: "with "
Chunk 44: "trails and "
Chunk 45: "wildflowers. "
Chunk 46: "Deserts, "
Chunk 47: "vast and "
Chunk 48: "arid, "
Chunk 49: "display "
Chunk 50: "resilience "
Chunk 51: "through "
Chunk 52: "their "
Chunk 53: "hardy "
```

```
Chunk 52: "their "
Chunk 53: "hardy "
Chunk 54: "vegetation "
Chunk 55: "and "
Chunk 56: "wildlife. "
Chunk 57: "Oceans, "
Chunk 58: "covering "
Chunk 59: "most of "
Chunk 60: "the "
Chunk 61: "Earth's "
Chunk 62: "surface, "
Chunk 63: "are "
Chunk 64: "teeming "
Chunk 65: "with "
Chunk 66: "marine "
Chunk 67: "life and "
Chunk 68: "ecosystems. "
Chunk 69: "Plains, "
Chunk 70: "expansive "
Chunk 71: "and "
Chunk 72: "fertile, "
Chunk 73: "are ideal "
Chunk 74: "for "
Chunk 75: "farming "
Chunk 76: "and "
Chunk 77: "grazing "
Chunk 78: "livestock. "
Chunk 79: "Glaciers, "
Chunk 80: "icy and "
```

```
  Chunk 79: "Glaciers, "
  Chunk 80: "icy and "
  Chunk 81: "ancient, "
  Chunk 82: "hold the "
  Chunk 83: "Earth's "
  Chunk 84: "freshwater "
  Chunk 85: "reserves. "
  Chunk 86: "Rainforests, "
  Chunk 87: "rich in "
  Chunk 88: "biodiversity, "
  Chunk 89: "are vital "
  Chunk 90: "for "
  Chunk 91: "maintaining "
  Chunk 92: "the "
  Chunk 93: "planet's "
  Chunk 94: "ecological "
  Chunk 95: "balance. "
  Chunk 96: "Nature, in "
  Chunk 97: "its "
  Chunk 98: "infinite "
  Chunk 99: "forms, "
  Chunk 100: "provides "
  Chunk 101: "sustenance, "
  Chunk 102: "inspiration, "
  Chunk 103: "and "
  Chunk 104: "wonder. "
Shuffling Results:
  (Mountains,, 1)
  (and, 14)
```

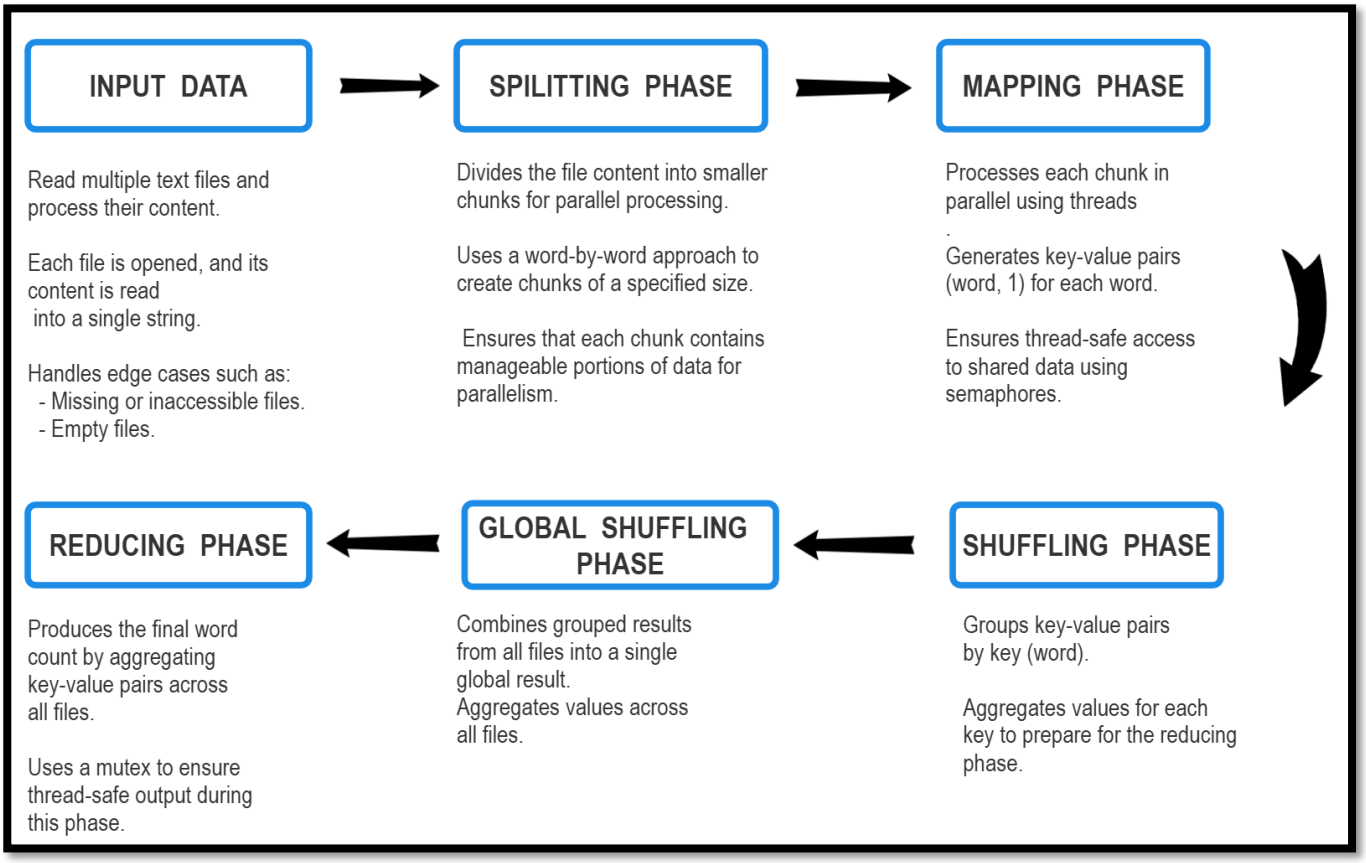
```
  (Mountains,, 1)
  (and, 14)
  (towering, 1)
  (offer, 1)
  (views, 1)
  (breathtaking, 1)
  (tranquility., 1)
  (a, 2)
  (sense, 1)
  (of, 3)
  (Rivers,, 1)
  (flowing, 1)
  (majestic,, 1)
  (provide, 1)
  (endlessly,, 1)
  (water, 1)
  (for, 4)
  (agriculture,, 1)
  (industry,, 1)
  (itself., 1)
  (Forests,, 1)
  (trees, 1)
  (wildlife,, 1)
  (dense, 1)
  (with, 3)
  (the, 5)
  (planet,, 1)
  (lungs, 1)
  (oxygen., 1)
```

```
(lungs, 1)
(oxygen., 1)
(life, 2)
(act, 1)
(as, 1)
(producing, 1)
(Lakes,, 1)
(calm,, 1)
(are, 5)
(fish, 1)
(serene, 1)
(haven, 1)
(aquatic, 1)
(plants., 1)
(yet, 1)
(picturesque,, 1)
(smaller, 1)
(Hills,, 1)
(dotted, 1)
(often, 1)
(trails, 1)
(wildflowers., 1)
(Deserts,, 1)
(vast, 1)
(arid,, 1)
(display, 1)
(resilience, 1)
(through, 1)
(their, 1)
```

```
(through, 1)
(their, 1)
(vegetation, 1)
(Oceans,, 1)
(wildlife., 1)
(covering, 1)
(hardy, 1)
(most, 1)
(Earth's, 2)
(surface,, 1)
(teeming, 1)
(marine, 1)
(ecosystems., 1)
(Plains,, 1)
(fertile,, 1)
(ideal, 1)
(expansive, 1)
(farming, 1)
(grazing, 1)
(livestock., 1)
(Glaciers,, 1)
(icy, 1)
(ancient,, 1)
(hold, 1)
(freshwater, 1)
(Rainforests,, 1)
(rich, 1)
(in, 2)
(biodiversity,, 1)
```

```
(in, 2)
(biodiversity,, 1)
(vital, 1)
(maintaining, 1)
(planet's, 1)
(ecological, 1)
(reserves., 1)
(balance., 1)
(Nature,, 1)
(infinite, 1)
(its, 1)
(forms,, 1)
(provides, 1)
(sustenance,, 1)
(inspiration,, 1)
(wonder., 1)
```

V. WORK FLOW DIAGRAM:



Why this is a Map-Reduce Solution?

This solution implements the core phases of MapReduce

Map Phase:

Processes input chunks in parallel to generate intermediate key-value pairs.

Shuffle Phase:

Groups key-value pairs by key and aggregates intermediate results.

Reduce Phase:

Produces the final output by aggregating results across all files.

VI. ADVANTAGES:

Specifications Of Each Block:

Parallelism:

The solution utilizes threads to concurrently process multiple chunks of input data. By breaking the task into smaller sub-tasks (chunks), the program can execute them in parallel across various CPU cores. This parallelism greatly accelerates data processing compared to sequential execution, particularly for large datasets.

Impact: It significantly reduces overall processing time by harnessing the capabilities of multi-core processors.

Thread Safety:

Synchronization tools like semaphores and mutexes are implemented to ensure safe access to shared data structures by multiple threads, preventing race conditions and data corruption. Semaphores manage access to the shared key-value store during the map phase, while mutexes ensure that console output operations are conducted safely.

Impact: This ensures the program functions correctly even when multiple threads interact with shared resources simultaneously, avoiding data inconsistencies.

Flexibility:

The solution is adaptable to handle different input sizes and file counts. It can process multiple files by iterating through a list of file names and reading their contents. It efficiently manages input data, whether small or large.

Impact: This adaptability allows the solution to scale from small files to large datasets, accommodating various data sizes and types.

Error Handling:

The program incorporates error handling mechanisms to effectively manage missing or empty files. If a file is not found or is empty, the program continues processing the remaining files without crashing or producing incorrect results.

Impact: This robustness enables the solution to handle real-world scenarios where input data may be inconsistent or incomplete.

VII. SCALIBILITY:

Handling More Files:

The solution is designed to process multiple files by iterating through a list (vector) of files. For each file, the program reads its content, divides it into chunks, and processes it concurrently using threads.

Impact: This method allows the program to easily scale as the number of files increases. By simply adding more files to the vector, the solution can manage larger datasets without needing significant changes to the code structure.

Managing Large Inputs:

The input data is split into smaller chunks to ensure efficient processing of even large datasets. This chunking mechanism enables the program to handle data more effectively by breaking it down into smaller, manageable pieces, with each chunk processed independently by a separate thread.

Impact: This approach allows the solution to scale to larger inputs without encountering memory or performance issues, as data is processed in parallel and distributed efficiently.

Utilizing Multi-Core Processors:

By employing threads, the solution fully leverages multi-core processors. Each thread can be assigned to process a chunk, distributing the workload across available CPU cores and resulting in faster execution.

Impact: The capability to utilize multi-core processors significantly enhances performance, enabling the solution to process large datasets more efficiently as hardware resources expand.

VIII. TEST CASE RESULT:

TEST CASES STATUS: (PASSED)

```
zoha@zoha-VirtualBox:~/OS Project/OS finalized$ g++ -o test1 q.cpp -lgtest -lgte
st_main
zoha@zoha-VirtualBox:~/OS Project/OS finalized$ ./test1
[==========] Running 5 tests from 4 test suites.
[----------] Global test environment set-up.
[----------] 2 tests from TextProcessingTest
[ RUN      ] TextProcessingTest.SplitTextTest
[       OK ] TextProcessingTest.SplitTextTest (0 ms)
[ RUN      ] TextProcessingTest.InvalidInputTest
[       OK ] TextProcessingTest.InvalidInputTest (0 ms)
[----------] 2 tests from TextProcessingTest (0 ms total)

[----------] 1 test from ShufflePhaseTest
[ RUN      ] ShufflePhaseTest.GroupTextTest
```

IX. CONCLUSION:

In conclusion, the implementation of the MapReduce framework using multithreading and synchronization mechanisms effectively tackles the challenges of processing large datasets. By dividing the task into manageable chunks and utilizing parallelism, the solution processes data efficiently, significantly decreasing processing time. The incorporation of semaphores and mutexes ensures thread safety, preserving data integrity during concurrent processing.

The solution's flexibility enables it to accommodate a wide variety of input sizes and file counts, while robust error handling guarantees that missing or empty files do not disrupt the workflow. Additionally, its scalability allows for handling larger datasets by leveraging multi-core processors, making it well-suited to meet increasing data demands.

This implementation highlights the effectiveness of the MapReduce model for distributed data processing in a single-machine environment, demonstrating its potential for real-world applications involving big data. The system is efficient, scalable, and capable of processing diverse datasets, making it a valuable strategy for addressing large-scale data processing challenges.

REFERENCES:

Dean, Jeffrey, and Sanjay Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*. *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI '04)*, 2004, pp. 137-150.

"MapReduce Explained." *YouTube*, uploaded by **Tech With Tim**, 21 Oct. 2020, www.youtube.com/watch?v=cHGaQz0E7AU.