

MSCS-630: Lab1: Lab Report: Word Frequency Counter Using Multithreading>

Sandesh Pokharel

University of the Cumberland

MSCS-630 Advanced Operating Systems

Satish Penmatsa

February 2, 2025

Introduction

This report outlines the design, implementation, and execution of a multithreaded program that computes word frequency from a text file. The program leverages Java's multithreading capabilities to divide the file into segments and concurrently process each segment using separate threads. After thread execution, the program consolidates intermediate results into a final word frequency count.

Program Design and Structure

The program is designed with modular components to ensure efficient execution and maintainability. The key components include:

Main Class (WordFrequencyCounter): Handles input reading, file segmentation, thread management, and final result consolidation.

Task Class (WordCountTask): Implements the logic to compute word frequency for a given file segment.

Utility Methods: Support functions for file reading, segment creation, and result merging.

The program starts by reading the file and the number of segments (N) specified by the user. It then divides the file into segments and assigns each segment to a thread for processing.

File Segmentation and Word Counting Logic

To ensure balanced processing, the file is divided into N segments, where each segment consists of a nearly equal number of lines. This segmentation is performed by the `createSegments()` method. Each thread processes its segment by reading each line, splitting it into words using non-word characters as delimiters, and updating a word-frequency map.

The word counting is case-insensitive, and empty words are ignored. The intermediate word frequencies for each segment are stored in a `HashMap<String, Integer>`, which is later merged into the final result.

Multithreading Implementation

The program utilizes Java's `ExecutorService` to manage threads. The steps for multithreading are as follows:

The main process creates a thread pool with a fixed number of threads.

For each file segment, a `WordCountTask` is submitted to the executor.

The `call()` method in `WordCountTask` performs the word frequency computation.

The main process waits for all threads to complete by calling `Future.get()`.

The final consolidated word frequency is generated by merging results from all threads.

This approach ensures that threads execute concurrently, taking advantage of multi-core CPUs for better performance.

Key Challenges and Solutions

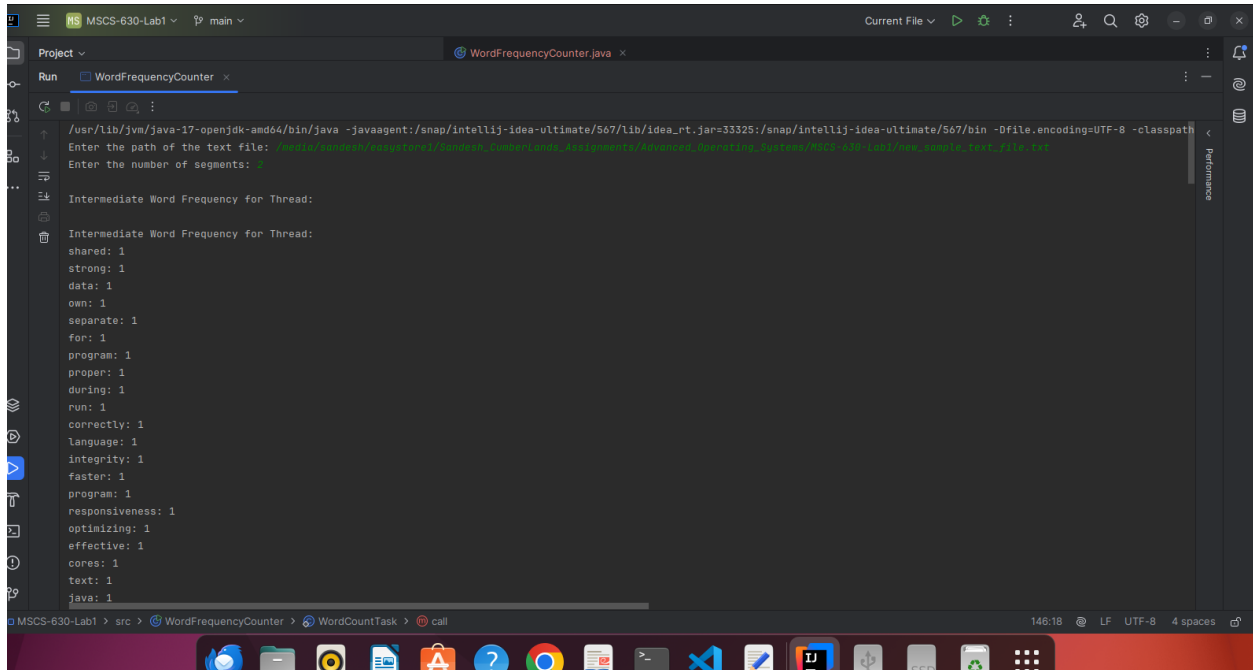
The following challenges were encountered during development:

Concurrency Management: Ensuring that all threads completed execution before final consolidation. This was addressed by using the `ExecutorService` and `Future` objects to synchronize thread completion.

File Segmentation: Determining the best way to partition the file. Line-based segmentation was chosen to avoid splitting words across segments.

Thread Safety: Although each thread operates independently on its segment, careful merging of word frequencies was necessary to avoid data inconsistencies.

Sample Run Output



```
MSCS-630-Lab1 main
Project WordFrequencyCounter.java
Run WordFrequencyCounter
/usr/lib/jvm/java-17-openjdk-amd64/bin/java -javaagent:/snap/intellij-idea-ultimate/567/lib/idea_rt.jar=33325:/snap/intellij-idea-ultimate/567/bin -Dfile.encoding=UTF-8 -classpath
Enter the path of the text file: /home/mcs630-lab1/src/WordCountTask/wordCountTask.txt
Enter the number of segments: 1
Intermediate Word Frequency for Thread:
Intermediate Word Frequency for Thread:
shared: 1
strong: 1
data: 1
own: 1
separate: 1
for: 1
program: 1
proper: 1
during: 1
run: 1
correctly: 1
language: 1
integrity: 1
faster: 1
program: 1
responsiveness: 1
optimizing: 1
effective: 1
cores: 1
text: 1
java: 1
```

```
Activities Terminal Feb 2 16:02
sandesh@sandesh-Inspiron-7373: /media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Operating_Systems/MSCS-630-Lab1/MSCS-630-Lab1/src
sandesh@sandesh-Inspiron-7373: /media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Operating_Systems/MSCS-630-Lab1/MSCS-630-Lab1/src$ java WordFrequencyCounter
Enter the path of the text file: /media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Operating_Systems/MSCS-630-Lab1/new_sample_text_file.txt
Enter the number of segments: 2

Intermediate Word Frequency for Thread:

Intermediate Word Frequency for Thread:
data: 1
separate: 1
program: 1
during: 1
correctly: 1
integrity: 1
faster: 1
optimizing: 1
text: 1
results: 1
breaking: 1
using: 1
ensure: 1
large: 1
resource: 1
un: 3
this: 2
handle: 1
each: 1
achieving: 1
processing: 1
must: 1
demonstrate: 1
vital: 1
role: 1
importance: 1
part: 1
concurrent: 1
datasets: 1
we: 2
shared: 1
segments: 1
strong: 1
example: 1
own: 1
```