Word Frequency Counter Using Multithreading

Jacob Jeffers, Sr.

MSCS 630

Dr. Brandon Bass

May 18th, 2025

Summer 2025

Word Frequency Counter Using Multithreading

The program for this lab is designed to count word frequencies in a text file using multithreading. It takes a text file and the number of segments as input, splits those into segments, spawn a thread for each segment, and processes each one concurrently. Each thread computes an intermediate word count using Python’s built-in `collections.Counter’ (*Collections — Container Datatypes*, n.d.). Once all threads finish, the main thread consolidates all the intermediate counts into a final frequency count.

Project Code

|  |
| --- |
| import threading # Import threading for parallel processing  from collections import Counter # Import Counter for counting word frequencies  import sys # Import sys for command line arguments  # Thread worker function  def count\_words(segment, results, index): # This function will be run in a separate thread      words = segment.split()      word\_count = Counter(words) # Count words in the segment      results[index] = word\_count      print(f"Thread {index} intermediate count:\n{word\_count}\n") # Print intermediate count for debugging  def main(file\_path, num\_segments): # Main function to read file and count words      # Read entire file content      with open(file\_path, 'r', encoding='utf-8') as f:          text = f.read()      # Split into segments      segment\_size = len(text) // num\_segments      segments = []      start = 0      for i in range(num\_segments): # Split text into segments          end = start + segment\_size          if i == num\_segments - 1:  # last segment takes the rest              end = len(text)          segments.append(text[start:end])          start = end      # Shared list to store thread results      results = [None] \* num\_segments      threads = []      for i in range(num\_segments): # Create and start threads          thread = threading.Thread(target=count\_words, args=(segments[i], results, i))          threads.append(thread)          thread.start()      for thread in threads:          thread.join()      # Consolidate results from all threads      final\_count = Counter()      for partial\_count in results:          final\_count.update(partial\_count)      print("Final word frequency count:")      print(final\_count)  if \_\_name\_\_ == "\_\_main\_\_": # Entry point for the script      # Check for correct number of command line arguments      if len(sys.argv) != 3:          print("Usage: python word\_count\_threads.py <file\_path> <num\_segments>")      else:          file\_path = sys.argv[1]          num\_segments = int(sys.argv[2])          main(file\_path, num\_segments) |

A screenshot of a computer program

AI-generated content may be incorrect.

Implementation and Challenges

Threads are created using Python’s ‘threading.Thread’ class (*Threading — Thread-based Parallelism*, n.d.). Each thread writes its result to a shared list. After all threads complete execution, the main thread consolidated the results. Aside from the coding portion, the biggest hurdle figuring out how to handle the shared data. I was able to get around this by writing each thread to a unique index.

How to Run the Program

1. Ensure Python 3 is installed on your system.  
2. Save the Python script as `word\_count\_threads.py`.  
3. Open a terminal or command prompt.  
4. Run the script using:  
 python word\_count\_threads.py <file\_path> <num\_segments>  
 Example: python word\_count\_threads.py sample.txt 4  
5. The output will show the intermediate word counts from each thread, followed by the final consolidated count.

Sample Output

Using the script python word\_count\_threads.py testscript.txt 4, the output below revealed the following.

A screen shot of a computer screen

AI-generated content may be incorrect.

# A black background with white text AI-generated content may be incorrect.

Link to GitHub Repository:

# References

*collections — Container datatypes*. (n.d.). Python Documentation. <https://docs.python.org/3/library/collections.html#collections.Counter>

*threading — Thread-based parallelism*. (n.d.). Python Documentation. <https://docs.python.org/3/library/threading.html#module-threading>