Description of the project

- Overview of how multiprocessing and multithreading are used.

  This project processes multiple large text files to count the occurrences of a specific word or find the top 50 most frequent words using both multithreading and multiprocessing approaches.

  Multithreading: This method divides a file into multiple chunks and assigns each chunk to a thread for processing. Each thread reads its portion of the file, counts the target word, and updates a shared word count. The threads also overlap slightly at chunk boundaries to ensure no word fragments are missed.

  Multiprocessing: This approach creates a separate process for each file, and each process counts the occurrences of the word in a single file. Results from each process are communicated to the parent process via IPC pipes.

- Consideration of the advantages and disadvantages of each approach

  Multithreading:
  Advantages:
  - Efficient for processing a single large file by splitting the work among threads.
  - Lower memory overhead as all threads share the same memory space.

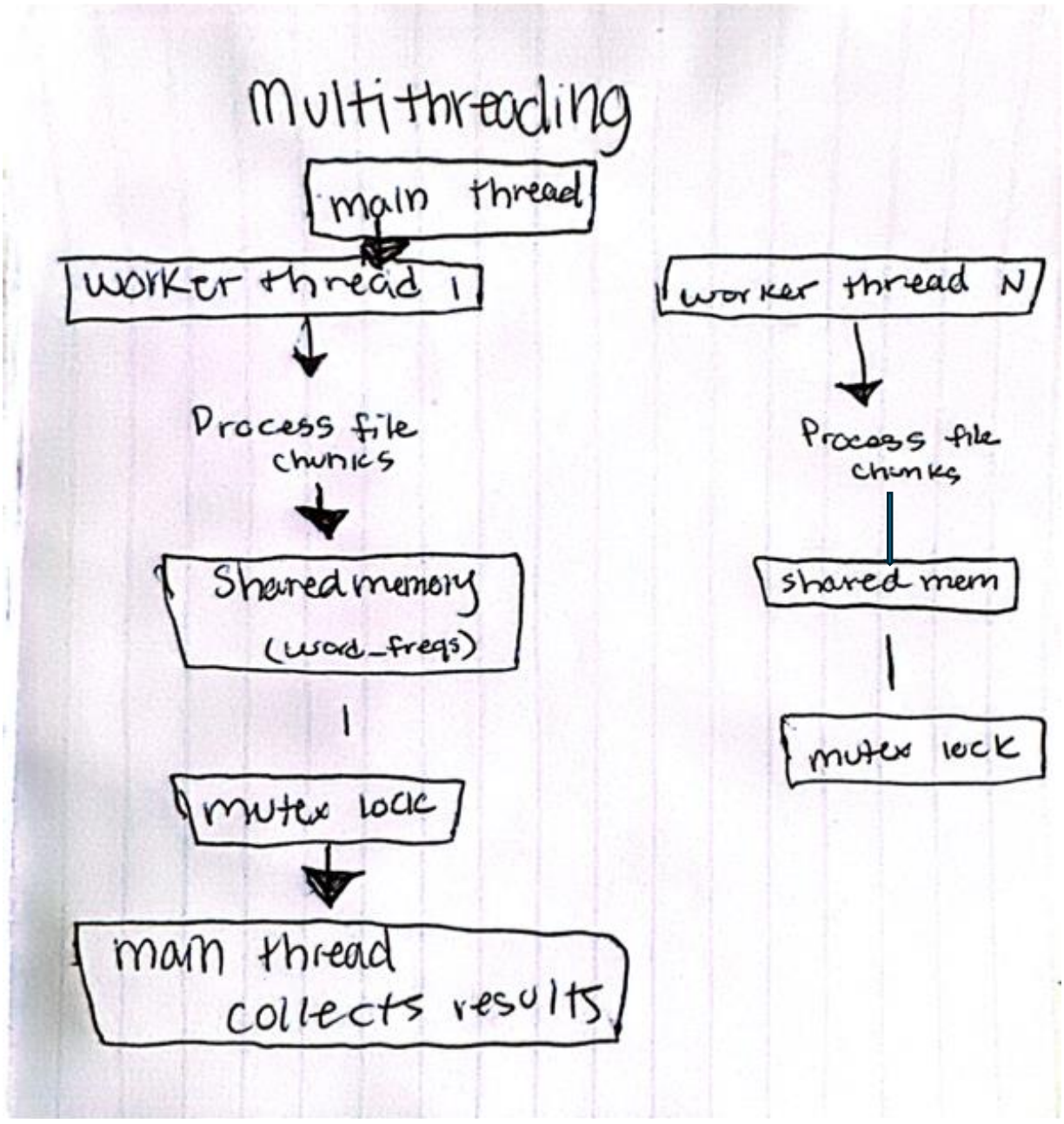
  Disadvantages:
  - Requires careful synchronization with mutexes to avoid race conditions.
  - Threads can interfere with each other, particularly when handling shared data.

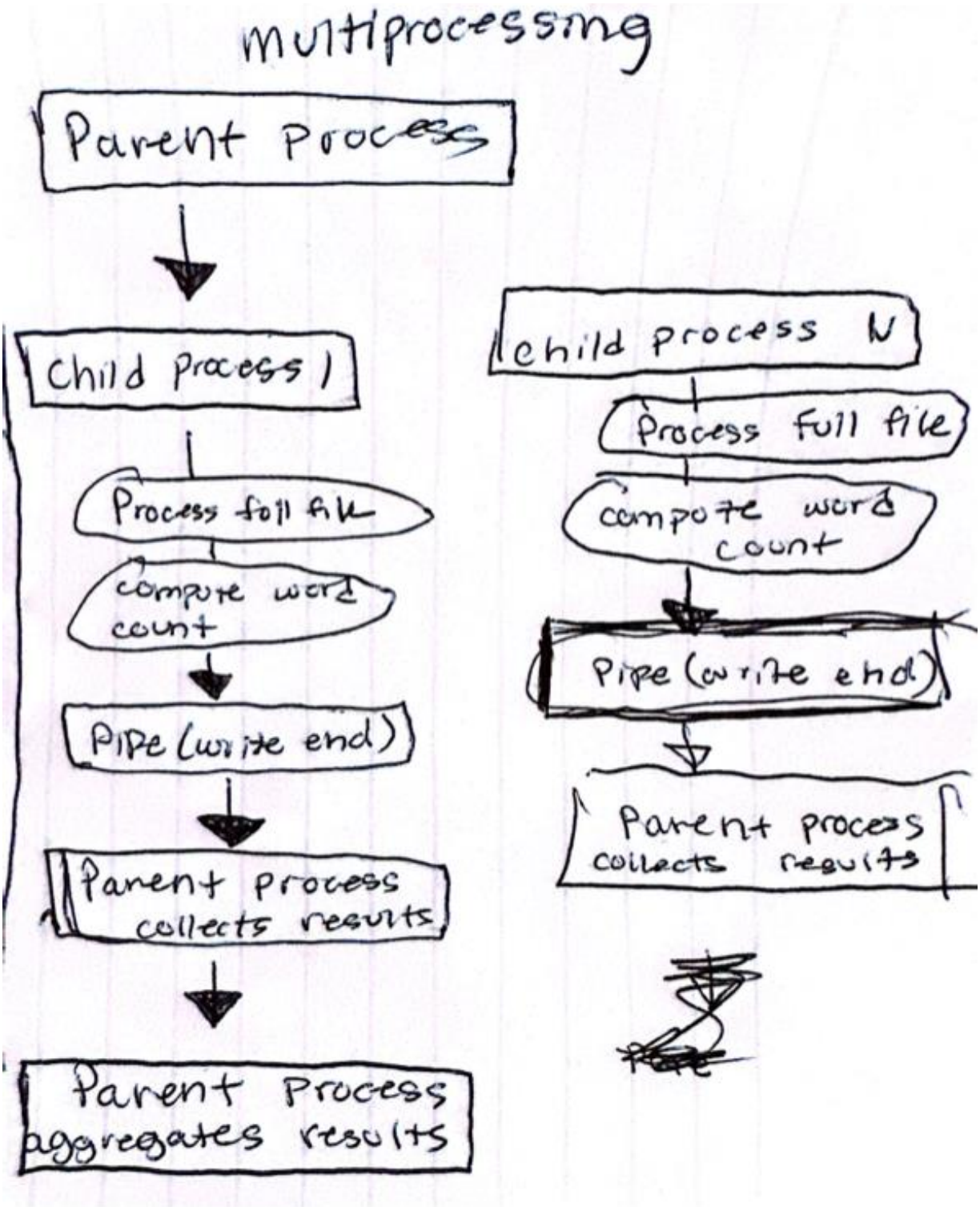
  Multiprocessing (single thread):
  Advantages:
  - Each process has its own memory space, so there is no need to worry about thread synchronization.
  - Better isolation between processes, reducing potential interference or race conditions.

  Disadvantages:
  - Higher memory usage, as each process has its own memory space.
  - Communication between processes is slower than between threads due to the need for IPC mechanisms like pipes.

Structure of the code:

- **Include diagrams showing how processes and threads are created and how communication happens via IPC.**





Description of the implementation required by project requirements

1. **Process management**
   The code uses fork() to create child processes for each file during the multiprocessing mode. In each process, a file is read and processed, and results are communicated back using pipes.

2. **IPC mechanism**
   Inter-Process Communication (IPC) is achieved via pipes. The parent process creates a pipe for each child process, and the children write the results of word counts to the pipe, which the parent process reads and aggregates.

3. **Threading**
   Multithreading is implemented using pthread_create() to split file processing among threads. Threads process chunks of a file and synchronize updates to a shared counter using a mutex.

4. **Error handling**
   - **File handling**: Each file operation checks for errors (file not found) and exits if an error occurs.
   - **Thread creation**: Threads check for errors during creation and handle failures by printing errors and terminating cleanly.
   - **Mutex handling**: Proper initialization and destruction of mutexes ensure no memory leaks or deadlocks.

5. **Performance evaluation**
   - Performance is evaluated by measuring the time taken for each approach using clock(). This enables comparing the efficiency of multithreading versus multiprocessing.

Instructions: Provide a clear guide on how to run the system and test different scenarios.

First, ensure the directories are setup as they are on the GitHub. It must be as follows:

```
|--docs/
|--files/
|    --bib
|    --paper1
|    --paper2
|    --etc..
|--src/
|    --wordcount.c
```

Compile Code using gcc:
gcc wordcount.c -o wordcount -lpthread

Run in different modes:
Multiprocessing:
/wordcount <directory> <word> --multiprocessing
Ex. ./wordcount ../files hello –multiprocessing

Multithreading:
./wordcount <directory> <word> --multithreading
Ex. ./wordcount ../files hello –multithreading

Top 50 words:
./ wordcount <directory> --top50
Ex ./wordcount ../files –top50

Performance evaluation

include screenshots or examples of file processing.

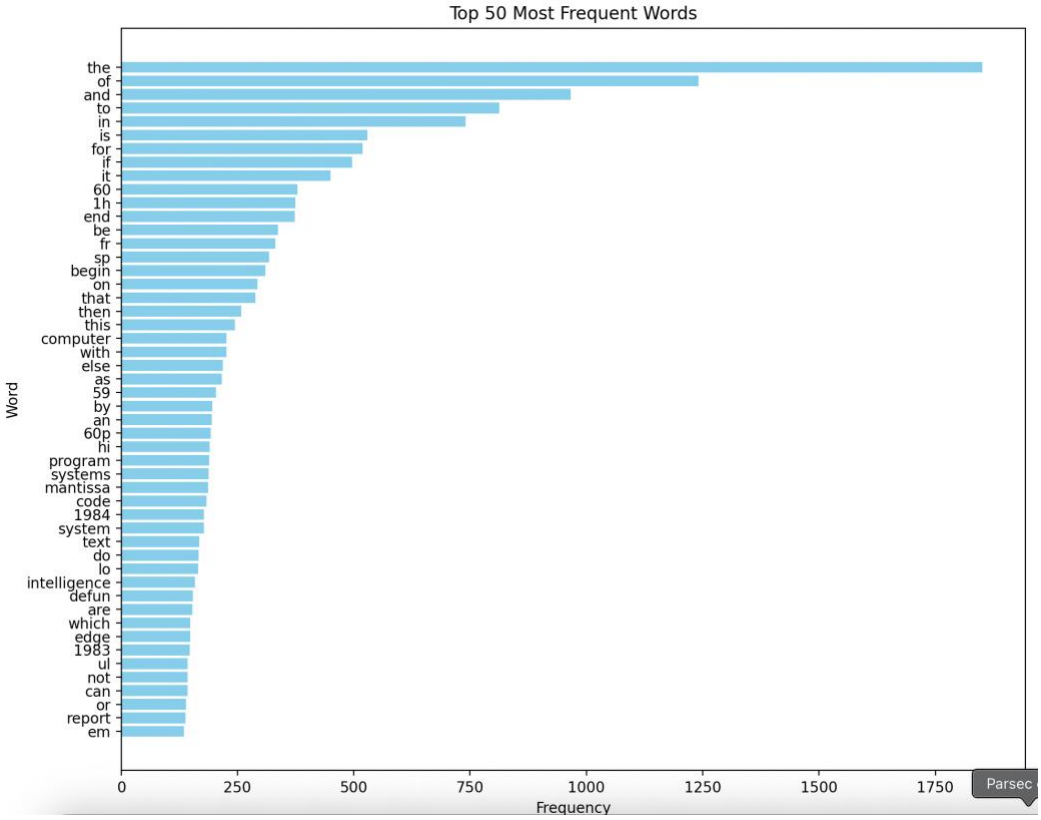
Top 50:

```
Top 50 most frequent words:
Word                 Frequency
-----------------------------------------
the                  1851
of                   1241
and                  966
to                   813
in                   741
is                   529
for                  519
if                   496
it                   450
60                   379
1h                   374
end                  373
be                   337
fr                   332
sp                   318
on                   293
that                 289
then                 258
this                 244
computer             227
with                 226
else                 219
as                   216
by                   196
an                   195
60p                  192
hi                   190
program              189
systems              188
mantissa             187
code                 184
1984                 178
system               178
text                 168
lo                   166
intelligence         159
defun                154
are                  153
edge                 148
which                148
1983                 147
not                  143
can                  143
ul                   143
or                   139
report               138
em                   135
will                 135
pp                   133
artificial           133
```

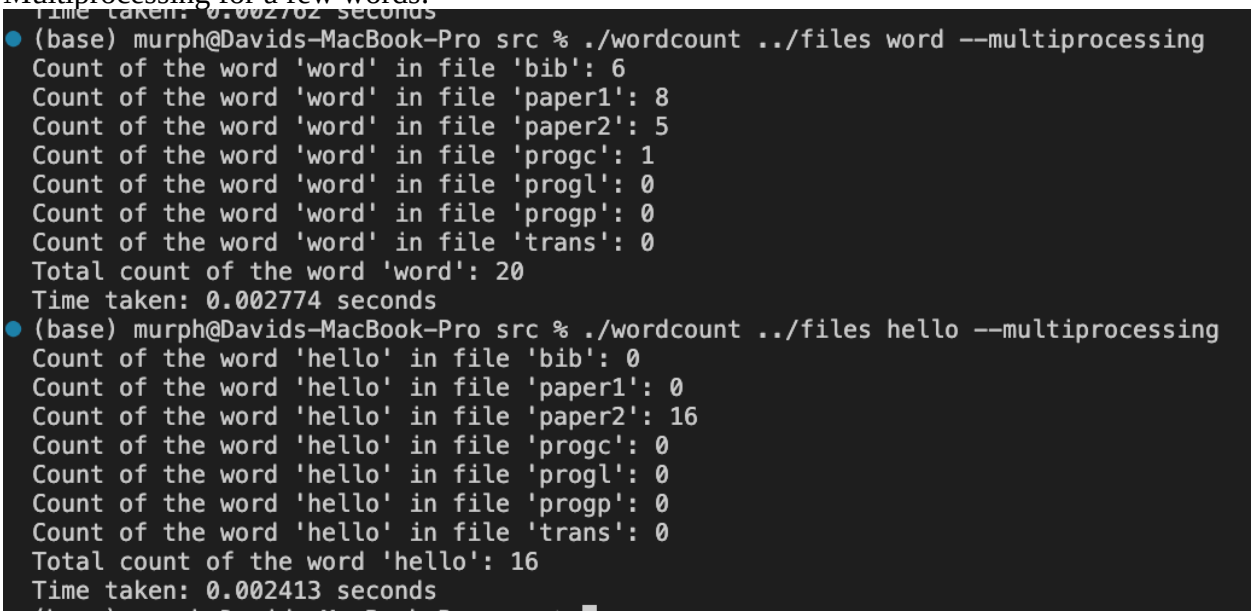
Histogram of top 50 and their relative frequencies
using python from .txt file generated by the following code after determining top 50:

```
FILE *output = fopen("word_frequencies.txt", "w");
    if (output) {
        fprintf(output, "word,frequency\n");
        for (int i = 0; i < word_count; i++) {
        fprintf(output, "%s,%d\n", word_freqs[i].word, word_freqs[i].frequency);
        }
        fclose(output);
        printf("\nWord frequencies saved to 'word_frequencies.txt'\n");
    }
```

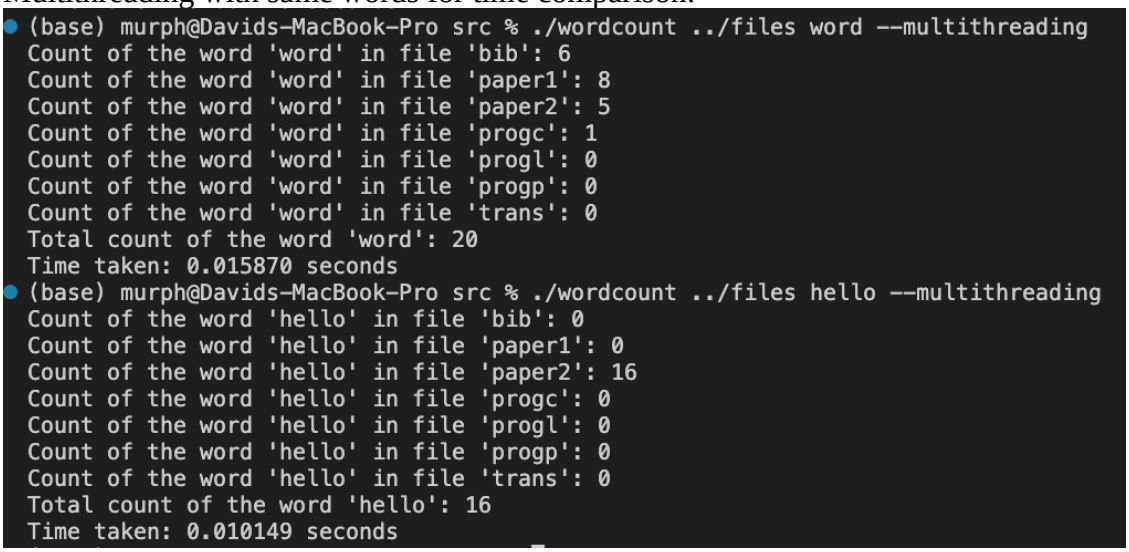
With this text file, I used pandas and matplotlib in python to read the file and generate a histogram.



Multiprocessing for a few words:



Multithreading with same words for time comparison:



Provide sample scenarios for word counting, showing the difference in performance between multiprocessing and multithreading.

More examples:

Multithreading –

Project 1 Report – David Murphy

```
(base) murph@Davids-MacBook-Pro src % ./wordcount ../files name --multithreading
Count of the word 'name' in file 'bib': 0
Count of the word 'name' in file 'paper1': 0
Count of the word 'name' in file 'paper2': 6
Count of the word 'name' in file 'progc': 1
Count of the word 'name' in file 'progl': 52
Count of the word 'name' in file 'progp': 0
Count of the word 'name' in file 'trans': 0
Total count of the word 'name': 59
Time taken: 0.017554 seconds
(base) murph@Davids-MacBook-Pro src % ./wordcount ../files sample --multithreading
Count of the word 'sample' in file 'bib': 0
Count of the word 'sample' in file 'paper1': 3
Count of the word 'sample' in file 'paper2': 0
Count of the word 'sample' in file 'progc': 0
Count of the word 'sample' in file 'progl': 0
Count of the word 'sample' in file 'progp': 0
Count of the word 'sample' in file 'trans': 0
Total count of the word 'sample': 3
Time taken: 0.010927 seconds
(base) murph@Davids-MacBook-Pro src % ./wordcount ../files test --multithreading
Count of the word 'test' in file 'bib': 0
Count of the word 'test' in file 'paper1': 2
Count of the word 'test' in file 'paper2': 1
Count of the word 'test' in file 'progc': 0
Count of the word 'test' in file 'progl': 0
Count of the word 'test' in file 'progp': 0
Count of the word 'test' in file 'trans': 0
Total count of the word 'test': 3
Time taken: 0.013123 seconds
(base) murph@Davids-MacBook-Pro src % ./wordcount ../files at --multithreading
Count of the word 'at' in file 'bib': 4
Count of the word 'at' in file 'paper1': 14
Count of the word 'at' in file 'paper2': 43
Count of the word 'at' in file 'progc': 2
Count of the word 'at' in file 'progl': 11
Count of the word 'at' in file 'progp': 0
Count of the word 'at' in file 'trans': 0
Total count of the word 'at': 74
Time taken: 0.010086 seconds
```

Multiprocessing –

```
(base) murph@Davids-MacBook-Pro src % ./wordcount ../files name --multiprocessing
Count of the word 'name' in file 'bib': 0
Count of the word 'name' in file 'paper1': 0
Count of the word 'name' in file 'paper2': 6
Count of the word 'name' in file 'progc': 1
Count of the word 'name' in file 'progl': 52
Count of the word 'name' in file 'progp': 0
Count of the word 'name' in file 'trans': 0
Total count of the word 'name': 59
Time taken: 0.002904 seconds
(base) murph@Davids-MacBook-Pro src % ./wordcount ../files sample --multiprocessing
Count of the word 'sample' in file 'bib': 0
Count of the word 'sample' in file 'paper1': 3
Count of the word 'sample' in file 'paper2': 0
Count of the word 'sample' in file 'progc': 0
Count of the word 'sample' in file 'progl': 0
Count of the word 'sample' in file 'progp': 0
Count of the word 'sample' in file 'trans': 0
Total count of the word 'sample': 3
Time taken: 0.003139 seconds
(base) murph@Davids-MacBook-Pro src % ./wordcount ../files test --multiprocessing
Count of the word 'test' in file 'bib': 0
Count of the word 'test' in file 'paper1': 2
Count of the word 'test' in file 'paper2': 1
Count of the word 'test' in file 'progc': 0
Count of the word 'test' in file 'progl': 0
Count of the word 'test' in file 'progp': 0
Count of the word 'test' in file 'trans': 0
Total count of the word 'test': 3
Time taken: 0.004915 seconds
(base) murph@Davids-MacBook-Pro src % ./wordcount ../files at --multiprocessing
Count of the word 'at' in file 'bib': 4
Count of the word 'at' in file 'paper1': 14
Count of the word 'at' in file 'paper2': 43
Count of the word 'at' in file 'progc': 2
Count of the word 'at' in file 'progl': 11
Count of the word 'at' in file 'progp': 0
Count of the word 'at' in file 'trans': 0
Total count of the word 'at': 74
Time taken: 0.002253 seconds
```

Discussion

Explain challenges faced during the implementation, the observed pros and cons of multiprocessing and multithreading, and any limitations or possible improvements.

Multithreading:

- **Pros**: Since threads share memory, multithreading had the advantage of faster communication between threads without the overhead of copying data. This made updating the shared word_freqs array more efficient. The screenshots show that multithreading, although slower than multiprocessing in certain cases, was still able to process the word count in a reasonable time.
- **Cons**: Multithreading suffered from potential bottlenecks due to mutex locking, especially when many threads tried to update the shared data simultaneously. In I/O-bound tasks (such as reading files), this overhead may be minimal, but in CPU-bound tasks, the cost of synchronization could become a limiting factor. The slight delays seen in multithreading results, such as 0.017 seconds compared to 0.002 seconds in multiprocessing, might be attributed to this synchronization overhead.

Multiprocessing:

- **Pros**: Each process in the multiprocessing implementation had its own memory space, eliminating the need for locks and reducing the risk of race conditions. This allowed processes to work independently and avoid the synchronization overhead faced in multithreading. As seen in the screenshots, multiprocessing was consistently faster (0.002 seconds) since there was no contention over shared resources.

- **Cons**: The main disadvantage of multiprocessing was the higher memory usage. Since each process has its own memory space, duplicating large datasets (like the word frequencies) across processes could lead to significant memory overhead. IPC introduces overhead when processes need to share results with the parent process, though this was relatively minimal in the current implementation.

Balancing Thread/Process Creation Overhead

- **Thread Creation Overhead**: Creating too many threads (in multithreading mode) could overwhelm the system, especially if the number of files is small. For example, if there are only a few files but a large number of threads, the system spends more time creating and managing threads than actually processing the files.

- **Process Creation Overhead**: Similarly, creating too many processes (in multiprocessing mode) incurs significant overhead in terms of memory and context switching. However, the current system limits this by only creating one process per file, which reduces overhead but could still result in performance issues with larger datasets.

Limitations and Possible Improvements

1. **Optimization of Mutex Locking**:
   - Currently, the shared word_freqs array is locked every time a word frequency is updated. This introduces a bottleneck when multiple threads frequently access the shared data.

   - **Improvement**: A possible improvement would be to reduce contention on the lock. For example, the frequency array could be split into smaller chunks, with each thread responsible for updating a specific chunk. This would reduce the frequency of lock contention.

2. **Improved Load Balancing for Threads**:
   - The current implementation assigns threads to files based on simple chunking. However, if files have significantly different sizes, some threads may finish early while others are still processing large files.

   - **Improvement**: Dynamic load balancing could be introduced where threads pick up new tasks (processing additional files) as soon as they finish their current task. This would ensure better utilization of CPU resources.

3. **Hybrid Approach**:
- Combining both multiprocessing and multithreading could further optimize performance. For example, each process could handle a separate file, while multiple threads within each process could process different parts of the file concurrently. This would allow the program to scale better with large datasets and systems with many CPU cores.

4. **Error Handling and Robustness**:
- While the current implementation handles errors such as file opening failures, more comprehensive error handling could be introduced. For instance, detecting and handling cases where a file is locked or unavailable would make the program more robust.

5. **Memory Efficiency**:
- In the multiprocessing version, each process creates its own memory space, leading to duplication of data. A possible improvement would be to use shared memory to allow processes to share some of the data, reducing memory overhead while maintaining process independence.

Conclusion

Overall, both multithreading and multiprocessing have their advantages and limitations. Multithreading is advantageous when memory sharing and frequent communication between threads are required, but it suffers from synchronization overhead. Multiprocessing excels in terms of speed, especially for CPU-bound tasks, but at the cost of higher memory usage and more complex data sharing mechanisms.

References on next page

References

Concurrency and parallelism concepts:

- https://www.geeksforgeeks.org/concurrency-in-operating-system/
- https://www.tutorialspoint.com/ios_development_with_swift2/ios_development_with_swift2_concurrency_control.htm#:~:text=Concurrency%20is%20a%20way%20to,better%20experience%20to%20the%20user.

Process Management:

- https://www.geeksforgeeks.org/fork-system-call/
- https://www.geeksforgeeks.org/inter-process-communication-ipc/

Multithreading and Multiprocessing:

- https://www.geeksforgeeks.org/multithreading-in-c/
- https://www.geeksforgeeks.org/ipc-technique-pipes/

Word Counting in C:

- https://stackoverflow.com/questions/71635384/how-do-you-count-the-frequency-of-which-a-word-of-n-length-occurs-within-a-strin
- https://www.geeksforgeeks.org/calculate-the-frequency-of-each-word-in-the-given-string/

Performance Optimization:

- https://www.geeksforgeeks.org/difference-between-multiprocessing-and-multithreading/
- https://medium.com/@pankaj_pandey/boosting-python-performance-with-multithreading-and-multiprocessing-basic-understanding-and-uses-63bf73ec6f5f