Yuvraj Gupta
Id : 922933190
Github : YuvrajGupta1808
CSC415 Operating Systems

Assignment 4 – Word Blast

Description:

This program reads the text file "War and Peace" and uses multiple threads to count and tally each word that is 6 or more characters long. The program divides the file into chunks, with each thread processing a chunk, updating a shared word count array. After processing, it prints the top ten words with their frequencies and the total execution time.

Approach:

1. *Initialize and Setup:* First, the program parses command-line arguments to retrieve the filename and the number of threads. This determines which file to process and how many threads to use for parallel processing. The mutex is then initialized to synchronize access to the global word count array, ensuring that only one thread can modify the word counts at a time. The file is opened in read-only mode, and its size is determined using `lseek`. This size is crucial for dividing the file into chunks for parallel processing. Subsequently, memory is allocated for the global word counts array based on an estimated number of words to manage memory efficiently and avoid frequent reallocations.
2. *Divide File and Create Threads:* The file size is divided by the number of threads to calculate the size of each chunk, ensuring each thread processes an approximately equal portion of the file. To prevent splitting words across chunks, the start positions are adjusted to ensure each chunk begins at the next delimiter if necessary. Thread arguments are allocated and initialized for each chunk. Threads are then created and launched to process their respective file chunks in parallel, leveraging concurrent execution to enhance performance.
3. *Thread Function (process_chunk):* Each thread opens the file, seeks to its designated start position, and reads its assigned chunk into a buffer. If the start position is not at the beginning of the file, it is adjusted to ensure the chunk starts at the beginning of a word by scanning for the next delimiter. The buffer is then tokenized into words using `strtok_r`, a reentrant

function safe for multi-threaded environments. For each tokenized word that is six or more characters long, the `add_word` function is called to update the global word count array.

4. *Add Word Function (add_word):* The mutex is locked to ensure exclusive access to the global word count array, preventing data races. The array is then iterated through to check if the word already exists. If it does, its count is incremented. If it does not, the new word is added to the array and its count initialized. After updating the word count, the mutex is unlocked, allowing other threads to access the global word count array.
5. *Wait for Threads and Print Results:* The program uses `pthread_join` to wait for all threads to complete their execution, ensuring all file chunks are processed before proceeding. A header is printed indicating the filename being processed and the number of threads used, providing context for the output. The global word count array is then sorted in descending order based on word frequency using `qsort` and a custom comparison function. The top N words and their counts are printed, summarizing the most frequent words in the text.
6. *Measure and Print Execution Time:* The program measures the total execution time from start to finish using `clock_gettime`. This helps evaluate the performance of the multi-threaded implementation. The total time taken for the program to execute is printed, providing insight into its efficiency. Finally, the allocated memory for the global word counts array is freed, and the mutex is destroyed to clean up resources and prevent memory leaks, ensuring the program exits gracefully and efficiently.

Issues:

Issue 1: The initial implementation of the program treated words with different cases (e.g., "Word" and "word") as distinct entries in the word count array. This resulted in inaccurate word frequency counts, as the same word in different cases was counted separately. For example, "The" and "the" would be treated as two different words, leading to incorrect frequency analysis.

Solution: To address this issue, the program was modified to convert all words to lowercase before adding them to the global word count array. This ensures that words like "Word" and "word" are treated as the same word. The `strcasecmp` function was used for case-insensitive comparison during the word count update process.

```
if (strcasecmp(wordCounts[i].word, word) == 0) {
    wordCounts[i].count++;
    pthread_mutex_unlock(&wordCountMutex);
    return;
}
```

Issue 2: To ensure safe concurrent updates to the global word count array, a mutex (`pthread_mutex_t`) was introduced. The mutex is used to synchronize access to the word count array. Specifically, the mutex is locked before any modification to the array and unlocked immediately after the modification. This ensures that only one thread can update the array at a time, preventing race conditions and ensuring data integrity. Here's how the synchronization is implemented:

```
pthread_mutex_lock(&wordCountMutex);
```

Analysis:

The provided output from running the Gupta_Yuvraj_HW4_main program shows the word frequency count for words of 6 or more characters in "WarAndPeace.txt" using different numbers of threads. Here is an analysis of what is happening during each run:

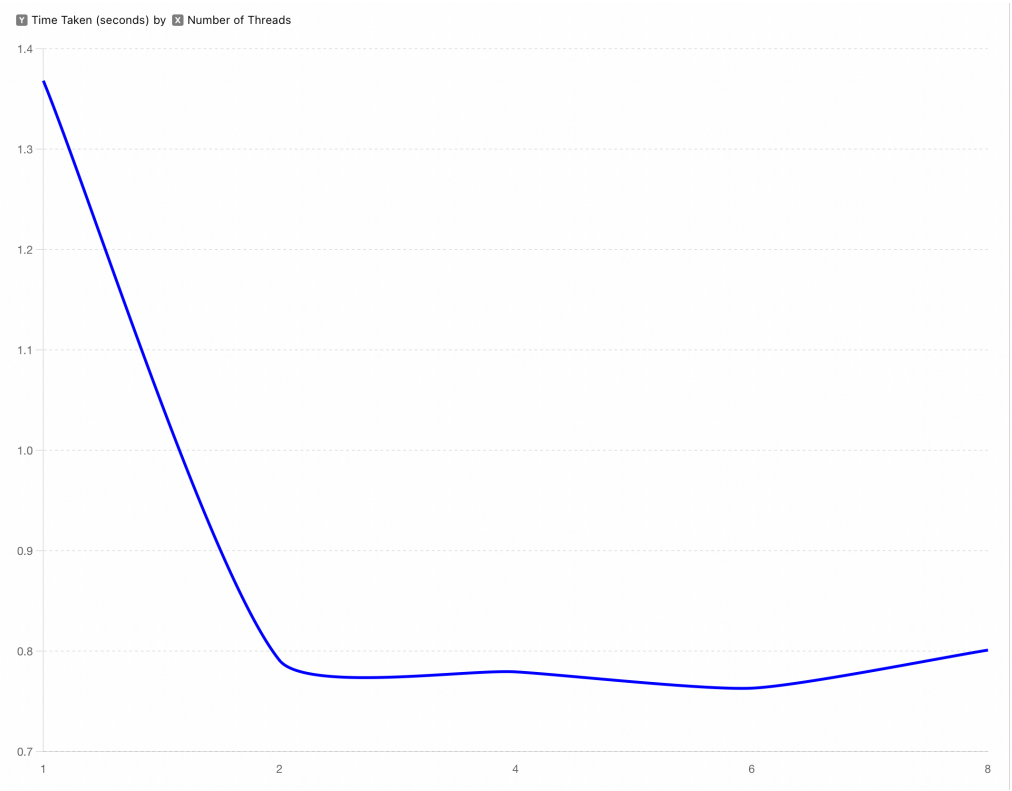
Understanding Thread Division and Execution

Threads are units of execution within a program. In a multithreaded program, multiple threads can run concurrently, allowing tasks to be processed in parallel. This parallelism can significantly reduce the time required for computations, especially for large tasks like processing a big text file.

In this program, the file is divided into chunks, each processed by a separate thread. Each thread processes its assigned chunk by reading it, tokenizing it into words, and updating a global word count array.

How Threads Use CPU Cores

Modern CPUs have multiple cores, and each core can run one or more threads simultaneously. By dividing tasks among multiple threads, the program can utilize multiple cores to perform computations in parallel. This can reduce the overall time required for processing.



Performance Gains with Increasing Threads

1. 1 thread: The entire file is processed by a single thread, which uses only one core of the CPU. This results in the longest processing time.
2. 2 threads: The file is divided into two parts, each processed by a separate thread. This significantly reduces the processing time, roughly by half, because two cores are working in parallel.

3. 4 threads: Further dividing the file into four parts allows four threads to process the data concurrently. The time reduction continues, but not as dramatically as from 1 to 2 threads.
4. 6 and 8 threads: With more threads, the file is divided into even smaller parts. However, the reduction in time becomes less noticeable because of the overhead involved in managing multiple threads and potential limits on available CPU cores.

Output Analysis

1. With 1 Thread: With a single thread, the entire file is processed sequentially. There is no parallelism, so the total time taken is the sum of all the processing steps (reading, tokenizing, counting). This typically takes longer compared to using multiple threads because there is no concurrent execution.
2. With 2 Threads: When using 2 threads, the file is divided into two chunks, and each chunk is processed by one thread. This allows the two threads to work concurrently, effectively halving the processing time compared to a single thread. The time taken with 2 threads is approximately half of that with 1 thread, demonstrating significant performance improvement due to parallel processing.
3. With 4 Threads: Using 4 threads divides the file into four chunks, with each chunk processed by a separate thread. This further reduces the time due to increased parallelism. However, the total time is not exactly half of that with 2 threads. This discrepancy arises due to the core limitations of the virtual machine. The system may have fewer physical cores than threads, leading to context switching and overhead, which impacts the total time reduction.
4. With 6 Threads: Dividing the file into six parts processed by six threads shows continued performance improvement. The processing time decreases because more chunks are processed simultaneously. However, the benefit may not be linear due to the increased overhead of managing more threads and the core limitations.
5. With 8 Threads: With 8 threads, the file is divided into eight chunks. While this allows for more parallel processing, the total time might not decrease

proportionally compared to 6 threads. This is because the virtual machine (or physical CPU) may have a limited number of cores, leading to context switching and increased overhead. Therefore, the performance gain may plateau or even slightly degrade.

Summary of Findings

- Parallel and Concurrent Execution: Using multiple threads allows the program to perform tasks in parallel, significantly reducing the total processing time. This is evident as the time decreases with an increasing number of threads.
- Time with 2 Threads: The time taken with 2 threads is approximately half of that with 1 thread. This is because the tasks are divided into two chunks, and the two threads can process these chunks in parallel, utilizing the CPU cores more effectively.
- Core Limitation Impact: The reduction in time is not strictly linear with the number of threads due to the limited number of CPU cores available. As seen with 4 and 8 threads, the time is not exactly half of that with 2 threads. The virtual machine's core limitations lead to context switching and overhead, offsetting some of the performance gains.
- Optimal Thread Count: There is an optimal number of threads where the performance gain is maximized. Beyond this point, adding more threads may not result in significant time reduction and could even increase the processing time due to overhead.
- Output Consistency: The word frequencies remain consistent across different runs, indicating that the program correctly tallies word counts regardless of the number of threads. This consistency also suggests that the critical sections are properly protected, preventing race conditions and ensuring accurate results.

Additional Observations

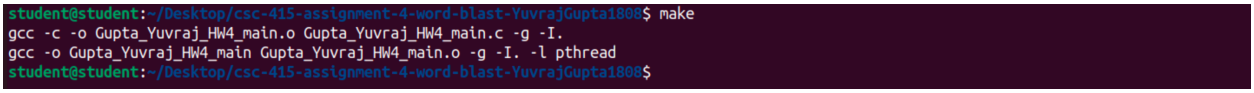
- Word Count Variability: Any minor differences in word counts across runs could be attributed to how the threads process their chunks and handle

word boundaries. However, the consistent top 10 words indicate that the program robustly handles chunk boundaries and correctly processes words.

- Impact of Thread Overhead: As the number of threads increases, the overhead associated with thread management (e.g., context switching, synchronization) becomes more pronounced. This can lead to diminishing returns in performance improvement, as seen with 4, 6, and 8 threads.
- Virtual Machine Constraints: The virtual machine's core limitations play a significant role in the observed performance. If the virtual machine has fewer physical cores than the number of threads, the expected speedup will not be fully realized due to the increased overhead of context switching and CPU scheduling.

In conclusion, the program effectively demonstrates the benefits of multithreading in processing large text files. By dividing the task into smaller chunks and processing them concurrently, the program leverages the parallelism offered by modern multi-core CPUs, leading to substantial reductions in processing time. However, the benefits are subject to the limitations of the system's core count and the overhead of thread management. The results show that while multithreading can significantly reduce processing time, the optimal number of threads depends on the underlying hardware and the nature of the task.

Screen shot of compilation:



Screen shot(s) of the execution of the program:

Test 1: Base Case

```
student@student:~/Desktop/csc-415-assignment-4-word-blast-YuvrajGupta1808$ make run
./Gupta_Yuvraj_HW4_main WarAndPeace.txt 2
Word Frequency Count on WarAndPeace.txt with 2 threads

Printing top 10 words 6 characters or more.
Number 1 is Pierre with a count of 1963
Number 2 is Prince with a count of 1928
Number 3 is Natásha with a count of 1213
Number 4 is Andrew with a count of 1143
Number 5 is himself with a count of 1020
Number 6 is Princess with a count of 916
Number 7 is French with a count of 881
Number 8 is before with a count of 833
Number 9 is Rostóv with a count of 776
Number 10 is thought with a count of 767
Total Time was 0.761311520 seconds
student@student:~/Desktop/csc-415-assignment-4-word-blast-YuvrajGupta1808$ make run RUNOPTIONS="WarAndPeace.txt 1"
```

Test 2: When RUNPOPTIONS=“WarAndPeace.txt 1”

```
student@student:~/Desktop/csc-415-assignment-4-word-blast-YuvrajGupta1808$ make run RUNOPTIONS="WarAndPeace.txt 1"
./Gupta_Yuvraj_HW4_main WarAndPeace.txt 1
Word Frequency Count on WarAndPeace.txt with 1 threads

Printing top 10 words 6 characters or more.
Number 1 is Pierre with a count of 1963
Number 2 is Prince with a count of 1928
Number 3 is Natásha with a count of 1213
Number 4 is Andrew with a count of 1143
Number 5 is himself with a count of 1020
Number 6 is Princess with a count of 916
Number 7 is French with a count of 881
Number 8 is before with a count of 833
Number 9 is Rostóv with a count of 776
Number 10 is thought with a count of 767
Total Time was 1.365701232 seconds
student@student:~/Desktop/csc-415-assignment-4-word-blast-YuvrajGupta1808$
```

Test 3: When RUNPOPTIONS=“WarAndPeace.txt 2”

```
student@student:~/Desktop/csc-415-assignment-4-word-blast-YuvrajGupta1808$ make run RUNOPTIONS="WarAndPeace.txt 2"
./Gupta_Yuvraj_HW4_main WarAndPeace.txt 2
Word Frequency Count on WarAndPeace.txt with 2 threads

Printing top 10 words 6 characters or more.
Number 1 is Pierre with a count of 1963
Number 2 is Prince with a count of 1928
Number 3 is Natásha with a count of 1213
Number 4 is Andrew with a count of 1143
Number 5 is himself with a count of 1020
Number 6 is Princess with a count of 916
Number 7 is French with a count of 881
Number 8 is before with a count of 833
Number 9 is Rostóv with a count of 776
Number 10 is thought with a count of 767
Total Time was 0.790858196 seconds
student@student:~/Desktop/csc-415-assignment-4-word-blast-YuvrajGupta1808$
```

Test 4: When RUNPOPTIONS=“WarAndPeace.txt 4”

```
student@student:~/Desktop/csc-415-assignment-4-word-blast-YuvrajGupta1808$ make run RUNOPTIONS="WarAndPeace.txt 4"
./Gupta_Yuvraj_HW4_main WarAndPeace.txt 4
Word Frequency Count on WarAndPeace.txt with 4 threads

Printing top 10 words 6 characters or more.
Number 1 is Pierre with a count of 1963
Number 2 is Prince with a count of 1928
Number 3 is Natásha with a count of 1212
Number 4 is Andrew with a count of 1143
Number 5 is himself with a count of 1020
Number 6 is Princess with a count of 916
Number 7 is French with a count of 881
Number 8 is before with a count of 833
Number 9 is Rostóv with a count of 776
Number 10 is thought with a count of 767
Total Time was 0.779393859 seconds
student@student:~/Desktop/csc-415-assignment-4-word-blast-YuvrajGupta1808$
```

Test 5: When RUNPOPTIONS=“WarAndPeace.txt 6”

```
student@student:~/Desktop/csc-415-assignment-4-word-blast-YuvrajGupta1808$ make run RUNOPTIONS="WarAndPeace.txt 6"
./Gupta_Yuvraj_HW4_main WarAndPeace.txt 6
Word Frequency Count on WarAndPeace.txt with 6 threads

Printing top 10 words 6 characters or more.
Number 1 is Pierre with a count of 1963
Number 2 is Prince with a count of 1928
Number 3 is Natásha with a count of 1213
Number 4 is Andrew with a count of 1143
Number 5 is himself with a count of 1020
Number 6 is Princess with a count of 916
Number 7 is French with a count of 881
Number 8 is before with a count of 833
Number 9 is Rostóv with a count of 776
Number 10 is thought with a count of 767
Total Time was 0.763135112 seconds
student@student:~/Desktop/csc-415-assignment-4-word-blast-YuvrajGupta1808$
```

Test 6: When RUNPOPTIONS=“WarAndPeace.txt 8”

```
student@student:~/Desktop/csc-415-assignment-4-word-blast-YuvrajGupta1808$ make run RUNOPTIONS="WarAndPeace.txt 8"
./Gupta_Yuvraj_HW4_main WarAndPeace.txt 8
Word Frequency Count on WarAndPeace.txt with 8 threads

Printing top 10 words 6 characters or more.
Number 1 is Pierre with a count of 1963
Number 2 is Prince with a count of 1928
Number 3 is Natásha with a count of 1213
Number 4 is Andrew with a count of 1143
Number 5 is himself with a count of 1020
Number 6 is Princess with a count of 916
Number 7 is French with a count of 881
Number 8 is Before with a count of 833
Number 9 is Rostóv with a count of 776
Number 10 is thought with a count of 767
Total Time was 0.801061073 seconds
student@student:~/Desktop/csc-415-assignment-4-word-blast-YuvrajGupta1808$
```