CS3006 Parallel and Distributed Computing
Assignment 1-Pthreads

Instructions:

- For your assignment, make a code file and pdf file with Roll Number. For e.g. make ROLL-NUM_SECTION_A1.cpp (23i-0001_A_A1.c) and so on. Each file that you submit must contain your name, student-id, and assignment # on top of the file in comments.
- Combine all your work in one folder. The folder must contain only code and pdf file (no binaries, no exe files etc.).
- Rename the folder as ROLL-NUM_SECTION (e.g. 23i-0001_A) and compress the folder as a zip file. (e.g. 23i-0001_A.zip). Do not submit .rar file.
- All the submissions will be done on Google classroom within the deadline. Submissions other than Google classroom (e.g. email etc.) will not be accepted.
- The student is solely responsible to check the final zip files for issues like corrupt files, viruses in the file, mistakenly exe sent. If we cannot download the file from Google classroom due to any reason, it will lead to zero marks in the assignment.
- Displayed output should be well mannered and well presented. Use appropriate comments and indentation in your source code.
- **Be prepared for viva or anything else after the submission of assignment.**
- If there is a syntax error in code, zero marks will be awarded in that part of assignment.
- Understanding the assignment is also part of assignment.
- **Zero marks will be awarded to the students involved in plagiarism. (Copying from the internet is the easiest way to get caught).**
- **Late Submission policy will be applied as described in course outline.**

Tip: For timely completion of the assignment, start as early as possible.
Note: Follow the given instruction to the letter, failing to do so will result in a zero.

Statistical Summaries Using Pthreads

The goal of this assignment is to refresh your knowledge of Pthreads by implementing a multithreaded solution to analyze large files. You will use Pthreads to divide the workload across multiple threads and compute statistical summaries efficiently. This assignment will also introduce you to concepts like thread synchronization and performance evaluation using varying numbers of threads.

Tasks:

You are required to choose any two of the following problems and implement a multithreaded solution using Pthreads. Each task will require you to process a large input file and produce statistical summaries.

Task Options:

1. **Graph File Analysis (Adjacency List Format):**
   - Input: A large file containing a graph in adjacency list format, where each line represents a node followed by its neighbors, e.g., A: B, C, D.
   - Output:
     - Total number of nodes and edges in the graph.
     - Top 10 nodes with the highest number of neighbors (degree).
   - Instructions: Divide the file into chunks, assign each thread to process a chunk, and use thread synchronization to combine results.

   Dataset https://snap.stanford.edu/data/com-Orkut.html
https://github.com/liuxu77/LargeST/tree/main?tab=readme-ov-file
2. **Word Analysis in a Large Text File:**
   - Input: A large text file containing sentences or paragraphs.
   - Output:
     - Total word count.
     - Count of words that start with vowels (a, e, i, o, u).
     - Top 10 most frequently occurring words.
   - Instructions: Split the text into chunks, process each chunk with a thread, and use thread-safe data structures for word counts.

Dataset https://www.euromatrixplus.net/multi-un/
Use English Dataset
3. **Term Frequency Analysis in a Corpus:**
   - Input: A large text file with multiple paragraphs or articles.
   - Output:
     - Term frequency for each word in the file.
     - Total number of unique words.
   - Instructions: Use a hash table or dictionary to store term frequencies, with proper thread synchronization to avoid race conditions.

Dataset https://www.euromatrixplus.net/multi-un/
Use English Dataset

4. **Log File Analysis:**
   - Input: A large log file where each line contains a timestamp and a message (e.g., [2024-12-21 10:30:00] ERROR: Something went wrong).
   - Output:
     - Total number of log entries.
     - Number of log entries per log level (e.g., ERROR, WARN, INFO).
     - Number of errors for each typeper log level (E1-E64) check log template available given at dataset below.
     - Count of unique log messages.

- Instructions: Divide the log file by lines, assign chunks to threads, and aggregate results in shared structures.

Log Dataset 1GB or higher https://github.com/logpai/loghub?tab=readme-ov-file

5. **Matrix File Analysis:**
   - Input: A large file representing a matrix in row-major format, where each line contains space-separated integers representing a row of the matrix.
   - Output:
     - Total sum of all elements in the matrix.
     - Maximum and minimum elements in the matrix.
     - Row-wise and column-wise sums.
   - Instructions: Divide the rows of the matrix among threads and synchronize results.

https://github.com/liuxu77/LargeST/tree/main?tab=readme-ov-file

Implementation Details:

1. **Multithreading with Pthreads:**
   - Use pthread_create to spawn threads.
   - Assign each thread a chunk of the file to process. In case of multiple files, you may read files using more than 1 thread. Reading 1 file with multiple threads may be counter productive.
   - Use pthread_join to wait for all threads to complete.
   - Primary task of parallel to perform computation based on the data read from the files.
   - Use pthread_join to wait for all threads to complete.

1. Use pthread_setaffinity_np (or a similar API) to bind threads to specific CPU cores.
   2. Implement two versions of your program:
      - **Without Affinity:** Allow the operating system to decide thread-to-core mapping.
      - **With Affinity:** Explicitly bind threads to specific cores using a round-robin or other mapping strategy

2. **Synchronization:**
   - Use mutexes (pthread_mutex) to manage access to shared variables.
   - Use condition variables if required to synchronize thread operations.
3. **Performance Measurement:**
   - Run your program with different numbers of threads (e.g., 1, 2, 4, 8).
   - Measure and report the execution time for each configuration.
   - Provide a brief analysis of the speedup achieved by increasing the number of threads.
4. **Input File Handling:**
   - The input file size must be > 1 GB.
   - Ensure your program can handle files larger than system memory by processing them in chunks.

Thread Affinity and Performance Testing:

- **Objective:** Explore the effect of thread affinity on performance when processing large datasets.

Instructions:

1. Use pthread_setaffinity_np (or a similar API) to bind threads to specific CPU cores.
2. Implement two versions of your program:
   - **Without Affinity:** Allow the operating system to decide thread-to-core mapping.
   - **With Affinity:** Explicitly bind threads to specific cores using a round-robin or other mapping strategy.
3. Measure the execution time and CPU utilization for both cases with varying numbers of threads (e.g., 1, 2, 4, 8).
4. Analyze the impact of thread affinity on performance and scalability.

Deliverables for This Task:

- A table or graph showing execution time for both cases with varying thread counts.

- A brief analysis of how thread affinity influences performance in terms of cache locality, core utilization, and context switching overhead.
- Include the implementation of thread affinity logic in your program.

Deliverables:

1. **Code:** Submit a well-commented C program implementing your chosen task.
   - A table or graph showing execution time for both cases with varying thread counts.
   - A brief analysis of how thread affinity influences performance in terms of cache locality, core utilization, and context switching overhead.
   - Include the implementation of thread affinity logic in your program.
2. **Report:**
   - Brief explanation of your solution.
   - Execution time and speedup analysis for different thread configurations.
   - Hotspot analysis for both serial and parallel version using appropriate tool (a few sample tools have been suggested in the links. Select a tool based on your hardware)
   - Challenges faced and how you addressed them.
   - Demo of the working of your code with voice over not more than 30 sec uploaded on YouTube and link added to the report. Also add discussion of the code in the same video not. More than 60 sec, discussing the parallelization strategy used for the 2 selected problems. Camera and screensharing for code must be on while recording.

Evaluation Criteria:

1. **Correctness:** Does your program produce accurate results for the chosen task?
2. **Efficiency:** How well does your program utilize multiple threads?
3. **Thread Safety:** Have you correctly used synchronization primitives like mutexes?
4. **Performance Analysis:** Have you analyzed the impact of thread count on performance? Have you performed Hotspot analysis using appropriate tool?
5. **Code Quality:** Is your code clean, modular, and well-documented?