MultiThreading and Synchronization

AVIK PRAMANICK (23CS60R78)

Assignment Date: 16.10.2023

Introduction

The primary objective of this assignment is to gain a comprehensive understanding of multithreading and synchronization. The focus is on leveraging the capabilities of the pthread library, addressing the complexities introduced by race conditions, and familiarizing oneself with synchronization primitives.

Objective

The core aim of this assignment is to determine the approximate shortest path within a social graph. This task becomes intricate due to the dynamic nature of the graph, which undergoes continuous changes.

Problem Statement

The assignment revolves around a social graph. This graph is inherently dynamic, with edges being added and removed frequently. The format of the graph is based on an adjacency list representation. For further reference, the graph can be accessed [here](https://snap.stanford.edu/data/loc-brightkite$_edges.txt.gz$).

Implementation Details

Graph Data Structure

The graph is represented using an adjacency list. This choice is motivated by the efficient representation of sparse graphs. Each node maintains a list of its neighbors, implemented using linked lists.

Graph Update Threads

A probabilistic approach is employed for updating the graph. Based on a predetermined probability, edges are either added or removed. This randomness ensures a dynamic nature of the graph, simulating real-world scenarios.

Path Finder Threads

The task of these threads is to find paths between designated landmarks within the graph. Dijkstra's algorithm, known for its efficiency in finding the shortest path, is the algorithm of choice for this purpose.

Path Stitcher Threads

The primary role of these threads is to combine paths to deduce approximate paths between pairs of nodes. This stitching process is crucial for determining paths in a dynamically changing graph.

Logging

Two log files, namely *update.log* and *path_found.log*, are maintained. The former captures the updates made to the graph, while the latter logs the paths that have been identified.

Report Questions

- **Choice of Data Structure:** The adjacency list was chosen for representing the graph due to its space efficiency for sparse graphs. Linked lists are used to store neighbors, providing O(1) insertion times.
- **Additional Data Structures:** No additional data structures were employed for this assignment.
- **Synchronization:** Mutex locks and semaphores are utilized to ensure synchronization, preventing race conditions and ensuring thread safety.
- **Errors and Debugging:** During the implementation, several race conditions were encountered. These were resolved using synchronization primitives, ensuring the integrity of the graph and the accuracy of the paths found.

Conclusion

This assignment provided invaluable insights into the intricacies of multithreading and synchronization. The challenges faced during implementation underscored the importance of thread safety and the complexities introduced by race conditions. Furthermore, it highlighted the significance of multithreading and synchronization in real-world applications, where data is dynamic, and efficiency is paramount.