OPERATING SYSTEMS- II

PROGRAMMING ASSIGNMENT-3: DYNAMIC MATRIX SQUARING

PRATHAMESH MALVE
CO22BTECH11010

GOAL: This assignment aims to perform parallel matrix multiplication through a Dynamic mechanism in C++.

Low level design of code:

In this assignment, we are implementing dynamic matrix squaring using 4 different methods, (a) TAS, (b) CAS, (c) Bounded CAS, and (d) atomic increment
I am submitting seperate source files for each method, but the low level design of each file is more or less same with some changes.

1. **Main Function:**
    - Handles file I/O for reading the input matrix, matrix size (N), number of threads (K), and row increment.
    - Allocates memory for input matrix (`matrixA`) and output matrix (`matrixC`).
    - Creates `K` threads, each assigned a unique ID.
    - Waits for all threads to finish.
    - Writes the result matrix (`matrixC`) to an output file along with the execution time.
2. **Matrix Multiplication Function (`matmul`):**

- For part (A), we use TAS lock (`tas_lock`) to synchronize access to the shared counter (`counter`)
- For part (B), we use a CAS lock (`cas_lock`) to synchronize access to the shared counter (`counter`).
- For part(c) , we use a Peterson Lock variant for synchronization.
- For part (D), we use an atomic operation (`compare_exchange_weak`) to increment the shared counter (`counter`) atomically.
- Each thread processes a portion of rows in the input matrix.
- Updates the shared counter with a row increment (`rowInc`) to determine the rows to process.
- Performs matrix multiplication for the assigned rows and updates the output matrix (`matrixC`).

Thread Synchronization:

- **Test-and-Set (TAS) Lock:**
  - `tas_lock` is used to control access to the shared counter (`counter`).
  - Threads use `__sync_lock_test_and_set` to acquire the lock and `__sync_lock_release` to release it.
  - Ensures that only one thread at a time increments the counter.
- **Compare-and-Swap (CAS) Lock:**
  - A single CAS lock (`cas_lock`) is used to ensure exclusive access to the shared counter (`counter`).

    - Threads use `__sync_val_compare_and_swap` to atomically acquire the lock (if unlocked).
    - Spins until the lock is acquired, then releases the lock using `__sync_lock_release`.
- **Bounded CAS:**
    - `cas_lock` is a shared variable indicating the lock status (0 unlocked, 1 locked).
    - `waiting` is an array indicating whether each thread is waiting for the lock.
    - Peterson's Algorithm variation ensures mutual exclusion.
- **Atomic Increment:**
    - Use `compare_exchange_weak` to atomically increment the counter, ensuring thread safety.
    - An atomic integer to ensure atomic increment operations for the shared counter.

Memory Management:

- **Dynamic Memory Allocation:**
    - Allocates memory for input matrix (`matrixA`) and output matrix (`matrixC`) using `new`.
    - Frees allocated memory after matrix multiplication is complete.

File I/O:

- **Input:**

    - Reads input matrix values, matrix size (N), number of threads (K), and row increment from an input file (`matrix.txt`).
- **Output:**
    - Writes the result matrix (`matrixC`) and execution time to an output file (`out1.txt`).

Timing:

- **Clock Measurement:**
    - Uses the `clock()` function to measure the execution time.
    - Calculates and writes the total execution time to the output file.