FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi

Lab Duration: 3 hrs. CS417 Parallel Processing Lab Marks: 10

Lab No: 04 Instructor: Ms. Anmol Sanaullah Khan Dated: 04/10/2024

Before performing tasks, keep in mind the following rules:

1. **CHEATING IS NOT ALLOWED. Looking at someone’s else screen is also cheating.**
2. **Mobile phone and internet usage are not allowed.**
3. **If you have any queries related to the task, you can ask instructors only. Never talk to each other until you are allowed.**
4. **Do not answer any query until you are asked.**
5. **Perform all the tasks.**
6. **Avoiding any of the above rules will lead to marks deduction.**

**Task 1 - Shared Bank Account Simulation**

Simulate a shared bank account where multiple threads (representing customers) either deposit or withdraw money. Each thread will randomly choose to either deposit or withdraw an amount, and a mutex will ensure the operations happen safely without race conditions.

**Steps:**

1. Create a shared bank account balance initialized to 1000.
2. Spawn 4 threads, where each thread randomly chooses to deposit or withdraw amounts between 1 and 100 for a total of 5 transactions.
3. Use a mutex to ensure only one thread accesses the balance at a time.
4. Print the account balance after all transactions are complete.

**Expected Output:**

* Each thread prints whether it deposited or withdrew money and the updated account balance.

**Task 2 - Racing Game Simulation**

Simulate a racing game where multiple cars (threads) race through multiple checkpoints. Each car must wait at a barrier for all the other cars before moving to the next checkpoint.

Steps:

1. Create 4 threads representing cars in a race.
2. Each car must pass through 3 checkpoints. Use a barrier to ensure all cars reach a checkpoint before moving to the next one.
3. Print the progress of each car as it reaches the checkpoints.

Expected Output:

* Threads print messages indicating when they reach each checkpoint, and they proceed to the next checkpoint only after all cars have reached the current one.

**Task 3 – Matrix Multiplication**

Implement a multithreaded matrix multiplication program where different threads are responsible for calculating different parts of the result matrix. Use a mutex to control access to the shared matrix and a barrier to ensure that threads synchronize at specific points (e.g., at the end of row calculations).

Steps:

1. Generate two random matrices, A and B, of size 4x4.
2. Spawn four threads, each responsible for computing a specific row of the result matrix C.
3. Use a mutex to protect access to the shared result matrix C to avoid race conditions.
4. Implement a barrier to ensure that all threads have completed their row calculation before continuing.
5. Print the result matrix after all threads have finished their calculations.

Expected Output:

* Matrix A, matrix B, and the resulting matrix C printed after all calculations are done.

For example:

Matrix A:

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

Matrix B:

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

Result Matrix C:

90 100 110 120

202 228 254 280

314 356 398 440

426 484 542 600