FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi

Lab Duration: 3 hrs.	**CS417 Parallel Processing Lab (CS)**	**Marks: 20**
Lab No: 02	**Instructor: Mr. Usman Haider**	**Dated: 20/09/2023**

Before performing tasks, keep in mind the following rules:

1. **CHEATING IS NOT ALLOWED. Looking at someone else's 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.**

Note: For all questions, do all necessary exception handling. While making comparisons, make sure to use the array/matrix of large size. Moreover, generate a report showing the time comparison of all tasks. Don't calculate the time for Task 3.

Task 1: Write a multithreaded C program using CLI arguments to search for a given element in a large array. Compare the time taken by the multithreaded code and sequential code.
CLI arguments:

- array_size: The size of the array.
- Key: The element to be searched for.
- num_threads: The number of threads to be used for parallel searching.

Measure the time the multithreaded and sequential codes take to search for the element in the array. Print the time taken by both codes (this should be done in a single program).

Task 2: Write a multithreaded C program to find a matrix's maximum and minimum elements using CLI arguments. The CLI arguments should be:

- matrix_size: The size of the matrix.
- num_threads: The number of threads to be used for parallel searching.
- max_min: A choice of max or min to indicate whether to find the maximum or minimum element in the matrix.

Measure the time taken by the multithreaded code and sequential code.

Task 3: Using CLI arguments, write a multithreaded C program that simulates a car production factory. The program should have multiple part suppliers and multiple consumers car assemblers. Producers supply parts, and consumers assemble cars using those parts. Use a shared buffer to hold parts and cars, protected using mutexes. The program should use mutexes to protect access to the shared buffer to prevent race conditions. The program should ideally:

1. Create a shared buffer (an array) to hold parts supplied by producers and cars assembled by consumers.
2. Create a specified number of producer threads responsible for supplying parts (e.g., tires, engines).
3. Create a specified number of consumer threads responsible for assembling cars using the supplied parts.
4. Producers should add parts to the shared buffer and print messages indicating the supplied part.
5. Consumers should assemble cars using the available parts from the buffer and print messages indicating the car's assembly.
6. Ensure that producers do not add to the buffer if it's full, and consumers do not remove it if it's empty.

Sample Output:

```
Producer 1 supplied part: 567
Producer 2 supplied part: 321
Producer 3 supplied part: 234
Consumer 1 assembled a car with parts: 567, 321, 234
Producer 1 supplied part: 876
Producer 2 supplied part: 432
Consumer 2 assembled a car with parts: 876, 432, 567
Producer 1 supplied part: 789
Producer 3 supplied part: 654
Consumer 1 assembled a car with parts: 789, 654, 234
Producer 2 supplied part: 987
Consumer 2 assembled a car with parts: 987, 876, 432
```