UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA
Facultad de Ingeniería

Ingeniería en Ciencias de la Computación

COMPUTO PARALELO Y DISTRIBUÍDO
1.9 Actividad 2: Manejo de Threads

Trabajo de: ADRIAN A. GONZÁLEZ DOMÍNGUEZ *[359834]*
Asesora: JOSE SAUL DE LIRA MIRAMONTES

25 de agosto de 2024

Java

En **Java**, el manejo de threads se puede realizar utilizando la clase `Thread` o implementando la interfaz `Runnable`.

```
import java.util.Random;

class FillMatrixTask implements Runnable {
    private int[][] matrix;
    private int row;

    public FillMatrixTask(int[][] matrix, int row) {
        this.matrix = matrix;
        this.row = row;
    }

    @Override
    public void run() {
        Random random = new Random();
        for (int i = 0; i < matrix[row].length; i++) {
            matrix[row][i] = random.nextInt(100); // Genera un número
aleatorio entre 0 y 99
        }
    }
}

public class Main {
    public static void main(String[] args) throws InterruptedException {
        int rows = 5;
        int cols = 5;
        int[][] matrix = new int[rows][cols];
        Thread[] threads = new Thread[rows];

        // Crear y lanzar los threads
        for (int i = 0; i < rows; i++) {
            threads[i] = new Thread(new FillMatrixTask(matrix, i));
            threads[i].start();
        }

        // Esperar a que todos los threads terminen
        for (int i = 0; i < rows; i++) {
            threads[i].join();
        }
```

```
        // Imprimir la matriz
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Explicación:

- `FillMatrixTask`: Implementa `Runnable` y llena una fila de la matriz con valores aleatorios.
- `Thread`: Se crea un thread para cada fila de la matriz.
- `join()`: Se espera a que cada thread termine antes de imprimir la matriz.

C/C++

En **C++**, el manejo de threads se realiza a través de la librería estándar `<thread>`.

```
#include <iostream>
#include <thread>
#include <vector>
#include <random>

void fillRowWithRandom(int* row, int cols) {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dist(0, 99);

    for (int i = 0; i < cols; i++) {
        row[i] = dist(gen); // Genera un número aleatorio entre 0 y 99
    }
}

int main() {
    const int rows = 5;
    const int cols = 5;
    int matrix[rows][cols];
    std::vector<std::thread> threads;
```

```
    // Crear y lanzar los threads
    for (int i = 0; i < rows; i++) {
        threads.emplace_back(fillRowWithRandom, matrix[i], cols);
    }

    // Esperar a que todos los threads terminen
    for (auto& th : threads) {
        th.join();
    }

    // Imprimir la matriz
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            std::cout << matrix[i][j] << " ";
        }
        std::cout << std::endl;
    }

    return 0;
}
```

Explicación:

- `fillRowWithRandom`: Función que llena una fila de la matriz con valores aleatorios.
- `std::thread`: Se crea un thread para cada fila de la matriz.
- `join()`: Se espera a que cada thread termine antes de continuar.