Trabajo de los threads

Ciencias de la Computación
Computación paralela y Distribuida
Nombre: Ruth Milagros Mamani Apucusi

Link del github:https://github.com/Milagros12345/computaci-n-paralela-y-distribuida/tree/main/lista_enlazada

1. **Multiplicación Matrix-Vector**

Thread	Components of y
0	y[0], y[1]
1	y[2], y[3]
2	y[4], y[5]

Se va a paralelizar dividiendo los trabajos entre los threads, entonces a cada thread le puede tocar un número fijo de filas.

2. **Las tres formas de sincronización**
   COMPARACIÓN DE TIEMPOS

Table 4.3 Linked List Times: 1000 Initial Keys, 100,000 ops, 99.9% Member, 0.05% Insert, 0.05% Delete

Implementation	Number of Threads			
	1	2	4	8
Read-Write Locks	0.213	0.123	0.098	0.115
One Mutex for Entire List	0.211	0.450	0.385	0.457
One Mutex per Node	1.680	5.700	3.450	2.700

Implementación	Linked List Times: 1000 Initial Keys, 100,000 ops, 99.9% Member, 0.05% Insert, 0.05% Delete **Número de threads**			
	1	2	4	8
Read-Write lock second: millisecond:	0 158	0 94	0 64	0 55

One Mutex for Entire list. second: millisecond:	0 159	0 268	0 335	0 375
One Mutex per Node second: millisecond:	0 918	0 876	0 551	0 601

Table 4.4 Linked List Times: 1000 Initial Keys, 100,000 ops, 80% `Member`, 10% `Insert`, 10% `Delete`

	Number of Threads			
Implementation	**1**	**2**	**4**	**8**
Read-Write Locks	2.48	4.97	4.69	4.71
One Mutex for Entire List	2.50	5.13	5.04	5.11
One Mutex per Node	12.00	29.60	17.00	12.00

Implementación	Linked List Times: 1000 Initial Keys, 100,000 ops, 80% Member, 10% Insert, 10% Delete **Número de threads**			
	1	2	4	8
Read-Write lock second: millisecond:	1 1607	1 1430	1 1553	1 1772
One Mutex for Entire list. second: millisecond:	1 1614	1 1856	1 1932	2 2222
One Mutex per Node second: millisecond:	5 5661	4 4464	2 2648	1 1912

3. **Thread-Safety**

   En memorias compartidas ocurre este problema, un bloque de código es Thread-Safety sí puede ser ejecutado simultáneamente por múltiples threads sin causar problemas. En el libro nos muestra un ejemplo de tokenización, en la que se utilizan varios threads y cada línea del texto es asignada a un thread en

forma de round-robin. Esto puede ser serializado con la semáfora. Otra forma de tokenizar es utilizando **strtok** que almacena en el caché la línea de entrada y la caché es compartida, **strtok** declara una variable de almacenamiento estático , el cuál hace que persista para la siguiente llamada, pero como tenemos una caché compartida, en la que puede haber sobrescritos, ejm una tokenización con un solo thread, en este caso funciona muy bién, pero si le añadimos más un thread hay posibles sobrescritos. Entonces el **strtok** no funciona con múltiples threads. Sin embargo hay una función .h de C que si funciona, esta es **strtok_r** una versión reentrante de strtok. **strtok_r** tiene un tercer argumento que es saveptr que es utilizado para mantener un registro de dónde se encuentra la función de la cadena entrante.