INFORME DE LABORATORIO - THREADS

I. Consideraciones Previas

1. Hardware

   Marca: HP Probook
   Procesador: Intel R ©CoreTM i5-4200M CPU
   Velocidad de Procesador: 2.5Ghz
   Numero de nucleos: 4
   Disco Duro:750 GB
   Arquitectura: 64 Bits

2. Sistema Operativo

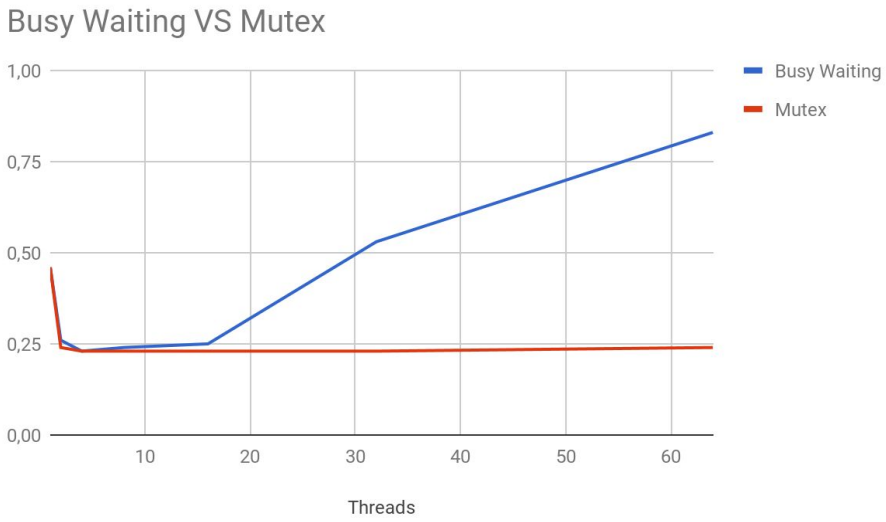
   Fedora 27 - Linux

II. BUSY WAITING VS MUTEX

Para comparar estas dos técnicas usaremos algoritmos para generar el número irracional pi, y el fin de utilizar estos dos abordajes es eliminar el problema de “raice condition” en el cual dos threads pueden acceder a una misma zona crítica la cual puede ser una variable alterandola sin que el otro thread sepa de esta operación.

- COMPARACIÓN DE TIEMPOS

Threads	Busy Waiting	Mutex
1	0,46	0,46
2	0,26	0,24
4	0,23	0,23
8	0,24	0,23
16	0,25	0,23
32	0,53	0,23
64	0,83	0,24

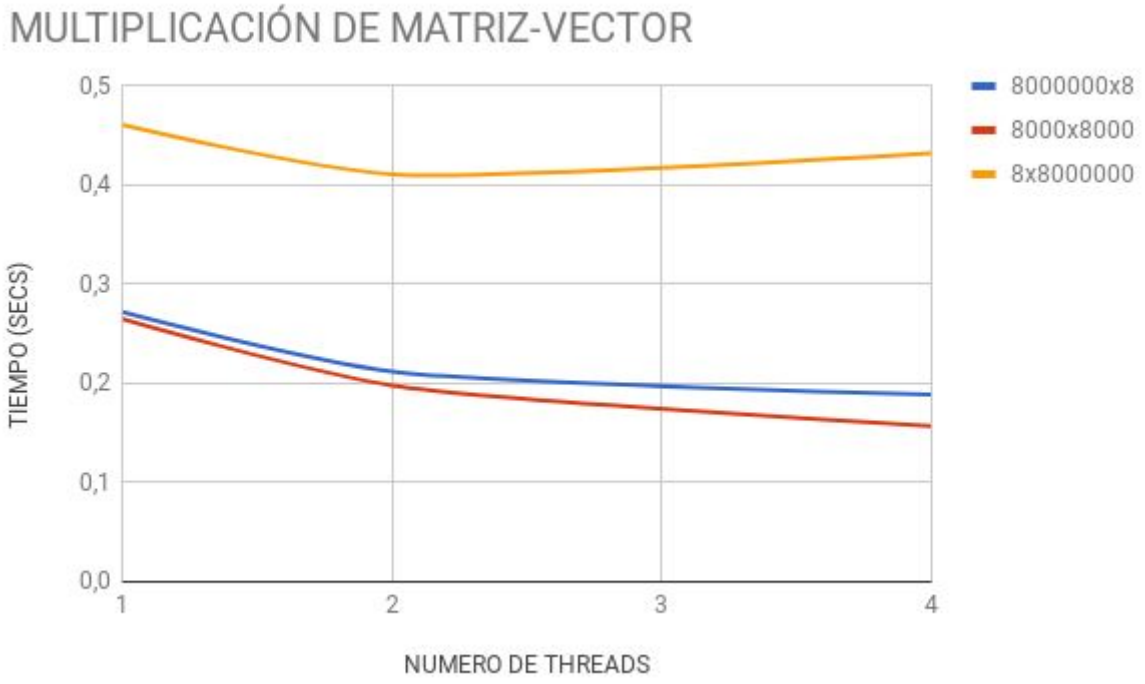


En la anterior figura se muestra un comparación de tiempos, en el cual es notable que la técnica usando mutex es mejor que usando la técnica de busy waiting, ya que este último reduce el acceso a la zona crítica a esperas secuenciales, por ejemplo el primer thread espera a que el segundo termine, el segundo espera a que el tercero termine y así sucesivamente, por lo que el resultado es el esperado y la línea de tiempo de la técnica busy waiting seguirá aumentando linealmente con respecto al número de threads usados.

III. MULTIPLICACIÓN MATRIZ-VECTOR

La siguiente prueba que se muestra son multiplicaciones de matrices de diferentes tamaños, la peculiaridad es que todas las matrices tienen el mismo número de elementos (64 millones de elementos).

	Dimensiones de Matrices		
Threads	8000000x8	8000x8000	8x8000000
1	0,272	0,265	0,461
2	0,212	0,198	0,411
4	0,189	0,157	0,432



Y como se puede notar en el gráfico aunque todas tengan el mismo número de elementos, el tiempo usado en realizar la multiplicación es distinta; los mejores tiempos se obtienen con la matriz de 8000x8000 y esto se puede explicar por el principio de localidad espacial en el uso de la memoria caché que se vió en un reporte de laboratorio anterior.

IV. LISTA ENLAZADA CON PTHREADS

Ahora veremos cómo se pueden realizar operaciones paralelos dentro de una estructura de datos como lo es una lista enlazada simple.

El principal problema es el mismo que cuando generamos el número pi, es decir el problema es el de “raice condition” ya que si un thread T1 quisiera modificar un nodo N1 y otro thread T2 quisiera acceder y realizar alguna operación en el mismo nodo N, y esto representa un problema.

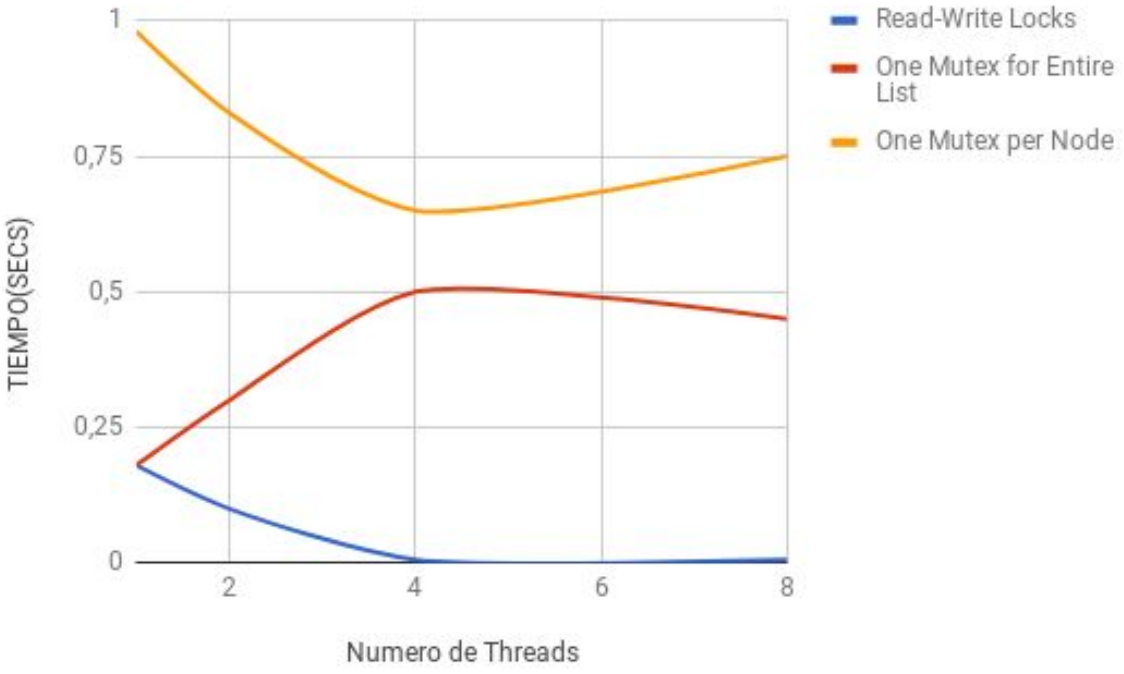
Para solucionar este problema, se presenta 3 formas:

1. Un mutex por lista
2. Un mutex por nodo
3. Read-Write Lock

Se realizaron dos pruebas:

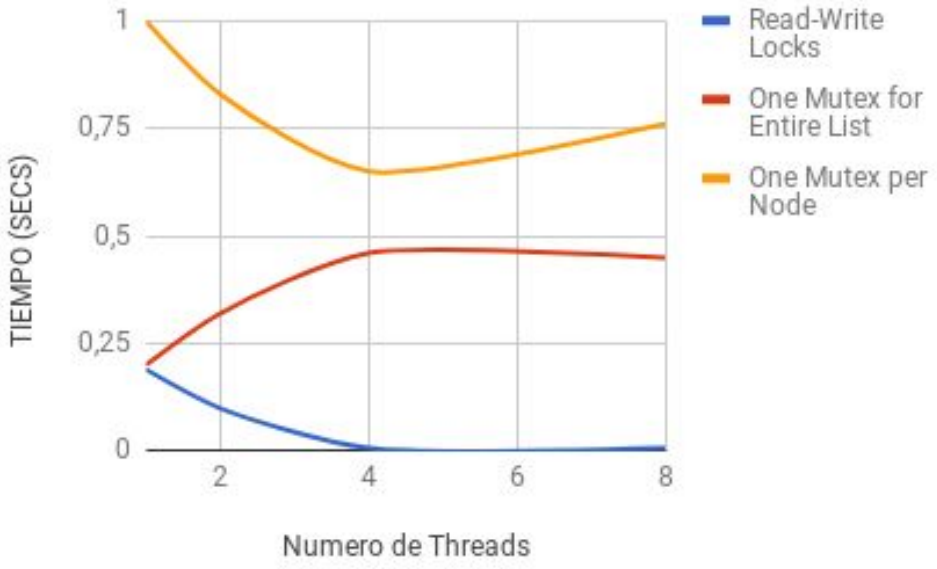
A. Prueba 1
   1. Número de elementos: 1000
   2. Operaciones: 100,000
   3. Busquedas: 80%
   4. Inserciones: 10%
   5. Borrados: 10%

	Número de Threads			
Numero de Threads	1	2	4	8
Read-Write Locks	0,18	0,1	0,006	0,006
One Mutex for Entire List	0,18	0,3	0,5	0,45
One Mutex per Node	0,98	0,83	0,65	0,75



B. Prueba 2
    1. Número de elementos: 1000
    2. Operaciones: 100,000
    3. Búsquedas: 99.99%
    4. Inserciones: 0.05%
    5. Borrados: 0.05%

	Número de Threads			
Implementacion	1	2	4	8
Read-Write Locks	0,19	0,1	0,007	0,007
One Mutex for Entire List	0,2	0,32	0,46	0,45
One Mutex per Node	1	0,83	0,65	0,76


1
0,75
0,5
0,25
0
TIEMPO (SECS)
2
4
6
8
Numero de Threads
Read-Write Locks
One Mutex for Entire List
One Mutex per Node