**Infraestructura Computacional**

**Caso 3**

**Diana Camila Cepeda - 201613662**

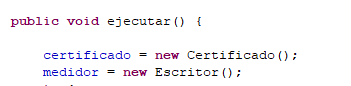
Juan Pablo Carreño – 201613487

1. Cambios en código para las medidas:



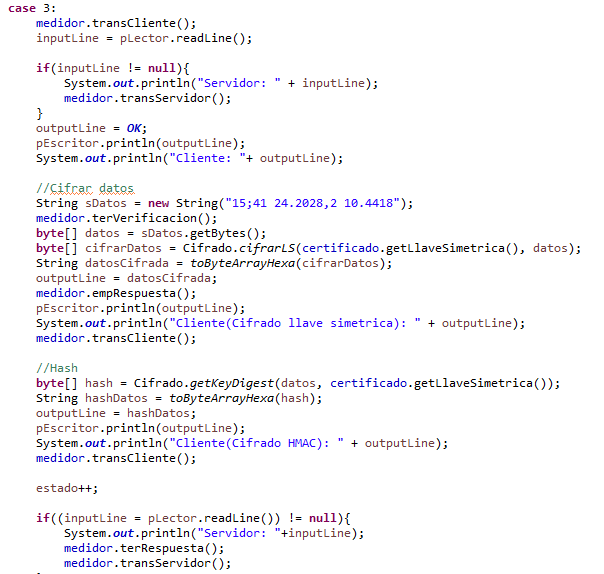
Se creo la clase *Escritor* que es el encargado de medir los tiempos y consecuentemente escribirlos en un archivo .txt para cada transacción realizada. Aquí es donde se encuentran las medidas para que se pueda realizar la toma de datos al correr la aplicación, se guardan los tiempos de verificación y respuesta, la cantidad te conexiones entre servidor y cliente que se pierden y también las que deberían llegar, también se tiene un atributo que indica si la transacción falla o tiene éxito. Es necesario notar que también se tiene en cuenta el atributo para guardar el valor del porcentaje de uso de la CPU. Se inicializan los atributos con valores de inicio para cada transacción, es decir, el tiempo inicial de verificación y respuesta es cero y no hay fallos de alguna transacción. En los métodos se encuentran las asignaciones de valores en nanosegundos para cada medida de tiempo y estas tienen su respectivo método de comienzo y final.

A continuación, mostraremos pantallazos del cliente en donde se llama a la medición.









Cambios necesarios en el cliente con seguridad para poder realizar las medidas de cada valor pedido de manera correcta:

Se crea el atributo **medidor** de tipo *Escritor* para realizar el llamado a cada método que fue creado en esta clase para que se realice el cálculo necesario. Los métodos *transCliente()* y *transServidor()* hacen referencia a la conexión ya sea desde el cliente hasta el servidor o desde el servidor hasta el cliente, estos métodos cuentan la cantidad de conexiones totales desde cada parte para cada parte del protocolo, esto se realizó con el fin de verificar que cada parte del protocolo se cumple para ambas partes. El método *getSystemCpuLoad()* es el encargado de medir el porcentaje de uso de la CPU, es por esto que se encuentra después de que el cliente haya comenzado la conexión con el servidor. Para el momento en que la ejecución llega a case 2 que es cuando el cliente comienza con la parte de descifrar lo que el servidor envía, es en donde se hace el llamado a *empVerificacion()* porque es en esta parte en la que se comienza a contar el tiempo que el cliente tarda en descifrar, cifrar y lo que el servidor tarda en descifrar, cuando llega el mensaje de confirmación por parte del servidor, se termina el tiempo de verificación, es decir, se realiza el llamado a *terVerificacion()*. Por último, se mide el tiempo de respuesta con el llamado a *empRespuesta()* justo antes de que el cliente envíe la consulta cifrada y este tiempo se detiene con el llamado a *terRespuesta()* después de recibir el mensaje del servidor, ya sea de que la transacción finalizó con éxito o si existieron errores. Después de terminar el protocolo, se realizan las verificaciones de fallos y se registra así sea correcto o fallido.

1. **Identificación de la plataforma:**

* **Arquitectura de:** 64 bits
* **Número de núcleos (cores): 2 nucleos**
* **Velocidad del procesador: 2,20 GHz**
* **Tamaño de la memoria RAM: 6G**
* **Espacio de memoria asignado a la JVM: 2G**

1. **Comportamiento de la aplicación con diferentes estructuras de administración de la concurrencia.**

Mediciones: tiempo de verificación, tiempo de consulta, número de transacciones perdidas y porcentaje de uso de la CPU.

* 1 thread y 400 transacciones iniciadas con retardos de 20 ms, 1 thread y 200 transacciones iniciadas con retardos de 40 ms y 1 thread y 80 transacciones iniciadas con retardos de 100 ms.



* 2 threads y 400 transacciones iniciadas con retardos de 20 ms, 2 threads y 200 transacciones iniciadas con retardos de 40 ms y 2 threads y 80 transacciones iniciadas con retardos de 100 ms.



* 8 threads y 400 transacciones iniciadas con retardos de 20 ms, 8 threads y 200 transacciones iniciadas con retardos de 40 ms y 8 threads y 80 transacciones iniciadas con retardos de 100 ms.



1. **Graficas Con Seguridad:**
2. **#Threads vs Tiempo de Verificación:**

**Pool 1:**

**Pool 2:**

**Pool 8**

1. **#Threads Vs Transacciones perdidas**

**Pool 1:**

**Pool 2:**

**Pool 3:**

**c) #Threads Vs Rendimiento de la CPU**

**1. 1 Thread**

1. **2 Threads**
2. **8 Threads**
3. **Tiempo de verificación vs CPU(2 Threads)**
4. **Tiempo de verificación Vs CPU(8 Threads)**

**Conclusiones:**

**-** Entre más grande sea el número de threads, ayuda a que, independientemente del incremento que se aplique sobre la cantidad de peticiones, se use en promedio el mismo porcentaje de cpu. Por lo tanto, debería implementarse solamente cuando hay una gran cantidad de transacciones por responder.

- Se facilito el poder dar respuesta a todas las transacciones con la facilidad de tener pocas maquinas conectadas a la misma red donde estaban alojados el servidor y el cliente.

- En promedio, al variar el número de threads el tiempo de respuesta es muy cercano entre las iteraciones realizadas, está entre 22 y 26 segundos aproximadamente.

- Ya que ninguna transacción perdida, se puede demostrar que el servidor estuvo disponible y funcionando en todo momento cuando se realizaron las pruebas de carga, de lo contrario no habrían sido atendidas algunas o ninguna de estas.

1. **Sin Seguridad:**

**¿Cuál es el resultado esperado sobre el comportamiento de una aplicación que implemente funciones de seguridad vs. una aplicación que no implementa funciones de seguridad?**

Se esperaría que los datos en comparación con una aplicación con seguridad sean menores dado que se esta quitando la verificación del certificado y la encriptación de los datos, los cuales hacen que el tiempo y uso del procesador aumente, y tambien que las peticiones del usuario suban en tiempo de desempeño y disponibilidad por parte del servidor.

1. **#Threads Vs TiempoVerificacion**

**1 Thread:**

**2 Threads:**

**8 Threads:**

1. **#Threads Vs Perdidas**
2. **#Threads Vs CPU**

**1 Thread:**

**2 Threads:**

**8 Threads:**

1. **Tiempo de verificación vs CPU(2 Threads)**
2. **Tiempo de verificación vs CPU(8 Threads)**

**Conclusiones:**

Estas gráficas confirman los resultados esperados, debido que a pesar de que en algunas métricas como el porcentaje de CPU utilizado la diferencia no sea tan grande, en las pruebas implementadas con seguridad se emplea un porcentaje de CPU mayor, y también un tiempo de espera y verificación más largo que en la implementación sin seguridad.