Programación Concurrente

Trabajo Práctico

Se requiere implementar en Java un programa concurrente que a partir de un archivo `passwd.txt` con una lista de nombres de usuario y hashes SHA-256 de sus passwords, genere un nuevo archivo `cracked.txt` donde los passwords se encuentran en texto plano. Para descifrar los passwords, se cuenta con un archivo `diccionario.txt`[1], que contiene una lista de passwords comunes. Sin embargo, los passwords en `passwd.txt` tienen un *salt*, es decir, contienen una cadena de texto adicional a la palabra para dificultar su crackeo. Por simplicidad, supondremos que esta cadena siempre es un número entero (sin ceros adelante) seguida de un caracter `#` (numeral). El formato de cada línea del archivo `passwd.txt` podría ser:

```
user;54efce51afdc3dc8d4f662daf314fa5511624749817c3deda35a795a67b90bec
```

donde la cadena después del punto y coma es el hash del password `27#prueba`.

Dado que la función de hash SHA-256 no es reversible, se debe probar, para cada palabra del diccionario, computar el hash con cada *salt* posible hasta encontrar la que coincida con el que se encuentra en `passwd.txt`. Se desea que esta tarea se encuentre repartida entre múltiples threads, cuya cantidad debe ser configurable por medio de la línea de comando. También debe ser posible configurar el número máximo a probar como *salt*.

Para realizar un hash de una palabra, se puede utilizar la clase `MessageDigest` de `java.security`, que proporciona un método `digest()` para realizar el hash. Esta función opera utilizando como entrada y salida un **array de bytes**, mientras que las palabras del diccionario son **String**, y los hashes en `passwd.txt` están en **formato hexa**, por lo que será necesario hacer las conversiones pertinentes para verificar si se ha dado con el password (Se puede usar el método `getBytes()` de la clase `String` para obtener el arreglo de bytes a partir del input).

No es necesario que `cracked.txt` posea los passwords crackeados en el mismo orden en el que figuran en `passwd.txt`. El programa puede opcionalmente también imprimir por consola cada vez que logra encontrar un password (pero no debería imprimir información de debug por default).

Junto con este enunciado se incluye el archivo `diccionario.txt` a utilizar y el archivo `passwd.txt`. Todos los passwords del `passwd.txt` entregado son son palabras que se encuentran en `diccionario.txt`, y el *salt* utilizado es un número menor a 100 en todos los casos.

Elementos a entregar

Programa Java

Se deberán entregar los fuentes de la resuolución del TP en un archivo `.zip`. El programa debe respetar la siguiente estructura:

1. Una clase `Main` con el punto de entrada del programa. El programa debe terminar una vez que haya obtenido todos los passwords (ie. no debe quedarse intentando

[1] Este archivo es uno de uso real para ataques de diccionario, utilizado por el programa *Cain & Abel*

hashes de passwords que ya solucionó). Al terminar el programa debe informar el tiempo transcurrido desde el inicio de la ejecución.

2. Una clase `Buffer` (implementada como un monitor utilizando métodos *synchronized*) que actúa como una cola FIFO concurrente de capacidad acotada (configurable). Es decir, bloquea a un lector intentando sacar un elemento cuando está vacía y bloquea a un productor intentando agregar un elemento cuando está llena. La capacidad del Buffer también debe ser parametrizable en el constructor del Buffer.
3. Una clase `DictAttackWorker` que extiende de `Thread` trabaja en el descifrado de los passwords. Los `DictAttackWorker` deben consumir "unidades de trabajo" de una instancia de `Buffer` compartida con `Main`. Se puede elegir diseñar qué carga de trabajo poner en cada unidad, una posibilidad es que en cada una haya un hash a descifrar. La clase `Main` tiene la responsabilidad de instanciar el `Buffer` y agregar las unidades de trabajo de a una.
4. Una clase `ThreadPool`, que se encarga de instanciar e iniciar la cantidad de `DictAttackWorkers` pedida por un usuario. La clase `Main` debe instanciar los `DictAttackWorkers` a través del `ThreadPool` pasando como argumento el `Buffer` que le permitirá enviar las "unidades de trabajo" a cada trabajador.
5. Una clase `ArchivoSalida`, que por medio de métodos *synchronized* coordine la escritura de `cracked.txt`. El manejo de este archivo se puede hacer por medio de la clase `BufferedWriter`.
6. Las clases adicionales que crea necesarias, por ejemplo para el manejo de los archivos `passwd.txt` y `dictionary.txt` (Que se pueden leer por medio de la clase `Scanner`).

Al iniciar el programa se debe delegar la iniciación de los threads necesarios en la clase `ThreadPool` y luego introducir en el `Buffer` las "unidades de trabajo" de a una. Cada `DictAttackWorker` en funcionamiento debe tomar unidades de trabajo del `Buffer` de a una y buscar la combinación de salt y password que genera el hash dado. Cabe destacar que es inadmisible utilizar una cantidad de threads menor a la solicitada por el usuario (aunque a la hora de hacer pruebas se recomienda no usar más de 16 threads).

Informe

Además del código, se requiere un informe corto en formato `pdf`, respetando el modelo presentado a continuación, que incluya los tiempos de ejecución del programa para encontrar los passwords del `passwd.txt` provisto por la cátedra y con los siguientes parámetros:

- **Threads:** 1, 2, 4, 6, 8, 10, 12 y 16
- **Tamaño de Buffer:** 2, 4, 8 y 20

El informe, que es **condición necesaria para la aprobación del TP**, debe respetar el siguiente formato:

1. **Autoría:** Nombres, e-mail y número de legajo de quienes hicieron el trabajo (máximo 2 personas salvo excepciones acordadas con los docentes).
2. **Introducción:** Sección explicando el dominio del TP, el diseño del código y cualquier consideración adicional que considere relevante para la correcta interpretación de los resultados.

3. **Evaluación:** Sección explicando el proceso de evaluación, incluyendo los detalles del equipo donde se ejecutan las pruebas (marca de micro, cantidad de memoria disponible y versión del sistema operativo). En esta sección deben incluirse dos tablas (una por cada tamaño de buffer) reportando los tiempos en los que se logró encontrar la totalidad de los passwords adecuando para cada cantidad de threads configurada, y el gráfico de la curva mostrando la progresión (con todos los tamaños de buffer en superposición).
4. **Análisis:** Sección en la que debe hacerse un análisis de los datos obtenidos en la evaluación, en particular destacando la cantidad de threads con la que se obtuvo el tiempo de ejecución mínimo.

Forma de Entrega

Se deben enviar el archivo `.zip` con el código del programa y el `.pdf` con el informe por email a las casillas de correo electrónico de *ambos* profesores con el subject: `Entrega TP PCONC 1S2021 - (apellido1) - (apellido2)` (donde `apellido1` y `apellido2` son los apellidos de las personas que constituyen el grupo). El mensaje debe ir *con copia* a la otra persona que integre el grupo, de forma tal que se pueda hacer la devolución respondiendo ese correo.

Es posible trabajar en un repositorio *git* compartido por las personas que compongan el grupo. En este caso, el repositorio debe ser **privado**. Al entregar, se deberá agregar a los profesores al repositorio. De todos modos se solicita que envíen el mail con las condiciones especificadas, aunque en vez de tener los archivos adjuntos incluirán el link al respositorio.

Fecha de Entrega

El TP debe entregarse antes del **Sábado 3 de Julio** a las **23:59hs**. En caso de ser necesario, se cuenta con una fecha de reentrega el **Martes 6 de Julio**.