**Programación Concurrente**

**Trabajo práctico final**

**Año: 2021**

**Cuatrimestre: 1º cuatrimestre**

**Profesores:**

* **Daniel Ciolek**
* **Pablo Terlisky**

**Integrantes**

**Nombre y Apellido:**

**Mail:**

**Legajo Nº:**

Nahuel Gómez

Mail

Legajo

Matías N. Fuentes

matiasnfuentes@gmail.com

48.423

**Introducción**

Siguiendo las especificaciones técnicas y el modelo base dado, buscamos realizar una solución eficiente y funcional para poder cumplir con ellos.

Nos decidimos por una implementación en la que tomáramos como tarea cada uno de los cien hashes del archivo a decodificar, ya que nos parecía una unidad de trabajo que permitiría distribuir la carga del mismo entre los thread de forma bastante equitativa. Por otro lado, esto permite un flujo de trabajo continuo y minimiza la carga sobre el buffer.

Nuestro método main es el encargado de instanciar un buffer, para alocar las tareas, con un tamaño pasado por parámetro.

Implementamos el mismo de forma muy similar a lo que vimos en la teórica, utilizando un array como estructura de datos (en este caso de Runnable), y haciendo que la lectura y escritura sobre el mismo sean métodos synchronized para garantizar que cada thread podrá tomar una tarea de este en exclusión mutua.

Otra de las funciones de nuestro método main es instanciar un threadpool que tiene como parámetros el buffer, y la cantidad de threads con la cual se quiere trabajar.

Este threadpool será el encargado de instanciar los threads que necesitemos y también de ponerlos a correr. Además, cuenta con los métodos launch y stop:

* Launch: permite poner una tarea en el buffer para que sea ejecutada por los threads.
* Stop: lanza tantas tareas PoisonPill como threads instanciados haya. Cuando estas tareas son ejecutadas por los workers, se lanza una excepción y su ejecución se detiene.

Una vez que los thread están corriendo, es el momento de lanzar las tareas de decodificación. Para esto tenemos la clase “Reader” que se encarga de leer los hashes a decodificar de un archivo, y retornárnoslos uno a uno. Leído el hash, se crea y se escribe una “DecodeTask” en el buffer, que toma como parámetro el hash, el salt máximo (también pasado por parámetro al main) y el path al diccionario que queremos utilizar. Este proceso es repetido hasta agotar todos los hashes del archivo “passwd.txt”.

Terminada esta etapa se llama al método stop del threadpool, para que lance las PoisonPill que sean necesarias teniendo en cuenta que, una vez terminadas las tareas de decodificación, todos los threads deben tomar una de ellas y finalizar su ejecución.

Nuestra DecodeTask consiste en formar con cada uno de los salt y los posibles passwords dados un hash, y verificar si este coincide con el que buscamos. A penas se obtiene el hash buscado, se sale del ciclo (no se prueban más combinaciones) y se escribe el password asociado a ese hash, en el archivo cracked.txt.

Para hacer la escritura de este, utilizamos la clase archivo de salida. La misma tiene el método sychronized write, que permite hacer la escritura de las contraseñas decodificadas en exclusión mutua, garantizando de esta forma que no vamos a perder ninguna de ellos cuando los threads llamen a la escritura.

**Evaluación**

[Especificaciones técnicas del equipo de las pruebas]

[Resultados en forma de tabla y gráficos]

**Análisis**

[Interpretación de los gráficos y tablas.]

[Ver cuando funciona mejor]