September 29, 2019

Introducción

El diseño del trabajo consiste básicamente en:

1. Hilos compresores
2. Hilo escritor
3. Cola protegida

Los hilos compresores se encargan de leer cada uno sus respectivos bloques del archivo de entrada, comprimirlos y agregarlos a una de las colas, que luego son recorridas por el hilo escritor mediante un *round robin*, que permite retirar cada bloque en el orden correcto. Finalmente, cada bloque comprimido es escrito en el archivo de salida.

Organización de los objetos

El compresor es modelado por una clase **Compressor**, que contiene un vector de colas protegidas (**ProtecedQueue**), un vector de hilos compresores (**CompressorThread**), un hilo escritor (**WriterThread**) además de dos atributos para el manejo de archivos (tanto de entrada como de salida), que son los objetos **Reader** y **Writer**.

Los objetos Reader y Writer leen y escriben por defecto por entrada y salida estándar respectivamente, con la opcion de asignarles un archivo especifico para la lectura y/o escritura.

Cada CompressorThread tiene una referencia a su respectiva cola y al objeto Reader (de esta manera todos los hilos acceden al mismo archivo), además de tener un atributo llamado **BlockBuffer**, que le permite agregar los números leídos de sus respectivos bloques y generar un bloque comprimido (**Block**).

Respecto al WriterThread, este objeto tiene acceso a todo el vector de colas, ademas de tener una referencia al Writer.

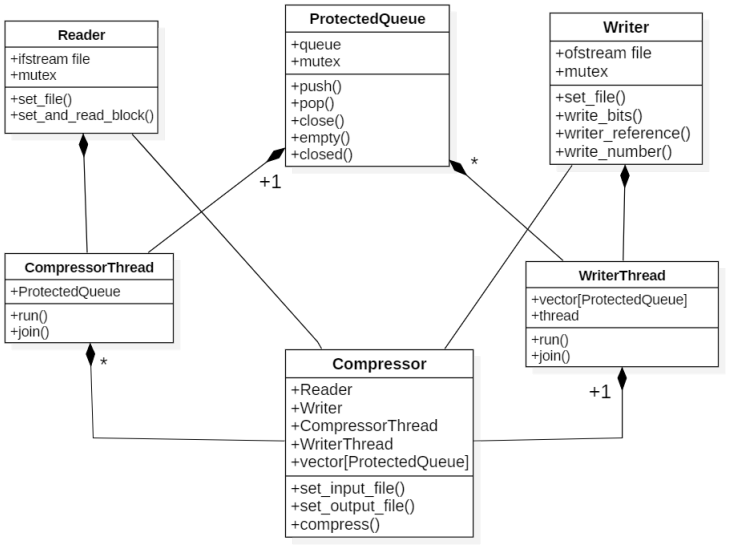

Figure 1: Composición del compresor

Lectura

Al iniciar la compresión, cada CompressorThread se crea pasándole como información cual bloque debe leer y un offset de avance (que equivale a la cantidad de hilos usados para la compresión). Cada bloque accede al archivo por medio del Reader (cuando es posible, ya que esta protegido por mutex), lee el bloque e correspondiente y actualiza la posición del siguiente bloque a leer. La implementación de la lectura esta pensada de manera que cada hilo bloquee el menor tiempo posible al archivo. Esto se logra haciendo que el procesamiento del bloque se haga luego de liberarse el mutex del archivo, dándole lugar a los otros hilos para acceder.

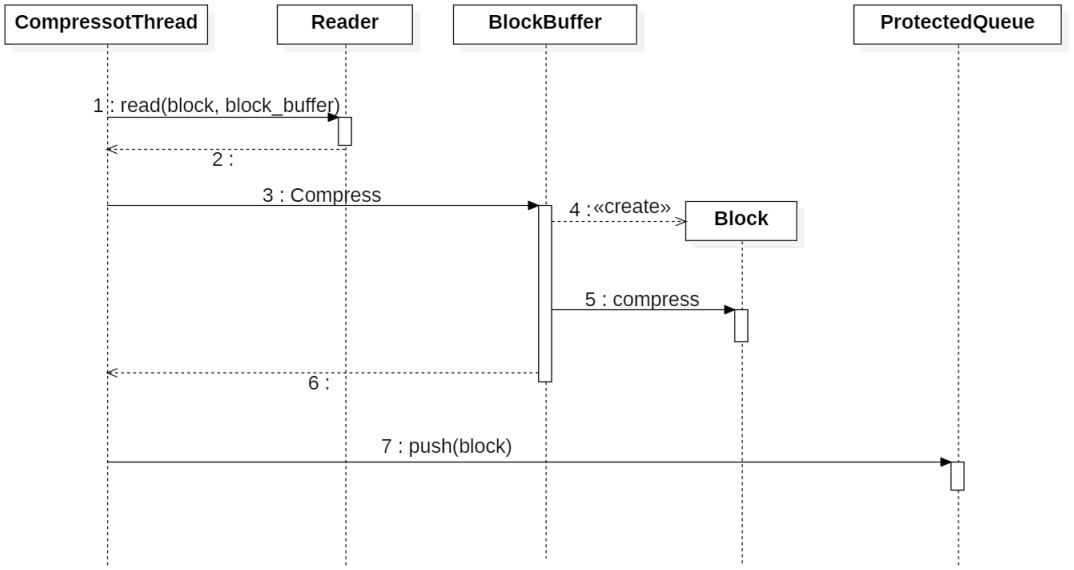

Figure 2: Proceso de lectura de un bloque

Procesamiento del bloque

Cada numero leído del bloque se agrega al objeto de la clase BlockBuffer. Una vez completo el buffer o llegado al fin de archivo (si el archivo no es multiplo de la cantidad de bloques), se llama al metodo del buffer *create_compressed_block*, que crea un bloque comprimido y lo aloca en el heap, devolviendo un puntero al objeto.

La compresión del bloque es hecha por la clase Block, que se apoya en un atributo de la clase **Bitset**, que permite el manejo de números al nivel de bits, utilizando un vector de *booleans*.

Cola Bloqueante

La cola bloqueante posee 4 estados posibles. Estos se modelan mediante dos variables booleanas Estas variables son "cerrada" (**bool closed**) y "vacía" (**bool empty**). Si la cola no esta cerrada se desencola un elemento si posible, o se espera a que se encole uno para luego desencolarlo. Si la cola esta cerrada, se desencola hasta que no quede mas elementos o se devuelve un *nullptr* si se encuentra vacía.

Escritura

La escritura es realizada por el WriterThread. Este realiza un *round robin* por todas las colas hasta que se encuentren todas cerradas y vacías. A medida que se lee un bloque, este se delega al Writer para que lo imprima en el archivo de salida en la posición correcta.

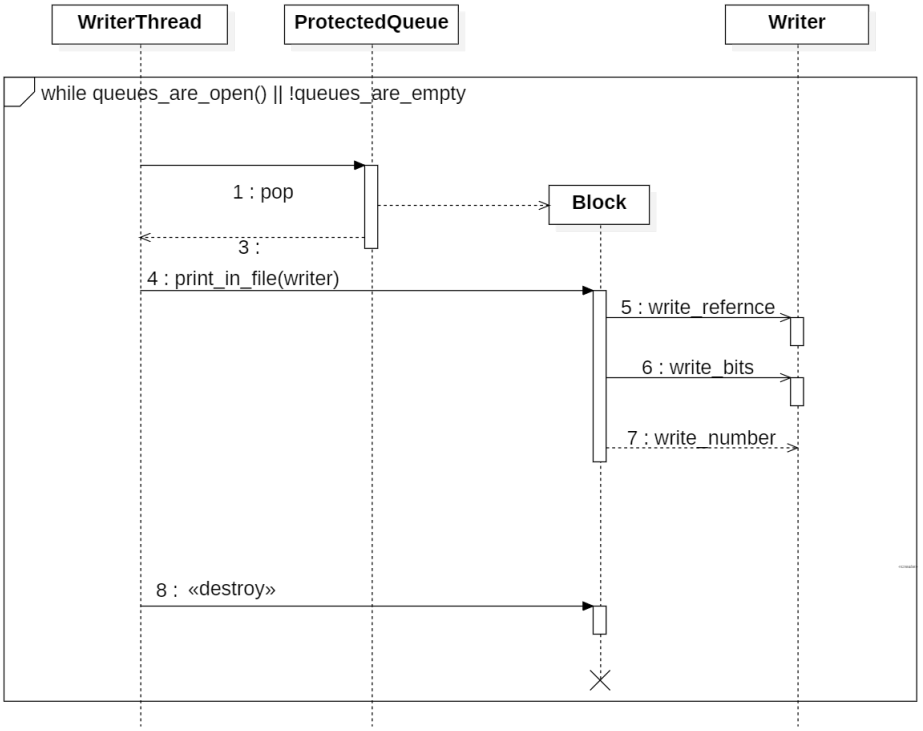

Figure 3: Escritura del bloque