Introducción

El presente trabajo tiene como objetivo principal aplicar los conceptos vistos en clase sobre Threads en un programa, específicamente en un compresor de archivos que funcione con varios hilos. El programa se aprovecha del conocimiento, de ante mano, de que los números a comprimir dentro de la secuencia fluctúan suavemente. Para realizar la compresión se usa la técnica de Frame Of Reference. Dicha técnica consiste en que dentro de cada bloque se buscara el mínimo, luego se restara su valor a todo el resto de números. Seguidamente se calculará la cantidad mínima de bits necesaria para representar al mayor numero luego de la sustracción y con esta cantidad se guardaran todos los números. Finalmente se escribirá en el archivo de salida la referencia (menor numero), la cantidad de bits por número y la secuencia de números comprimida.

Descripción textual:

Para la realización del trabajo se siguió la recomendación de los docentes de la cátedra de dividir el problema a resolver en tres partes: la compresión en si del archivo con un hilo, el uso de varios hilos leyendo el archivo y usar un hilo escritor que haya sacando los datos a escribir de varias colas en las que los hilos lectores pondrán dichos datos.

Primero se ataco la compresión del archivo con un hilo. Está parte del problema si bien no fue complicada en sí, fue demorada. Ésto se debe a que los lenguajes de programación están hechos para el fácil manejo de bytes y es lo que a los programadores nos resulta más natural. Con lo cual se necesito de considerable lectura de documentación de C++ para poder atacar correctamente el subproblema de turno.

Si bien C++ tiene una clase llamada bitset, la cual permite trabajar con bits, su uso resulto algo incómodo pues requiere saber en tiempo de compilación cuantos bits se van a alojar en ella. Si bien existe una clase de bitset dinámico que acaba con esta contrariedad, dicha clase no pertenece a ninguna librería estándar por lo cual se opto por no usarla.

Como se menciono, el manejo de bits fue lo más dificultoso de ésta parte. Para lograr resolverlo, se redujo el uso de bitsets lo más posible y se opto por guardar chars a los cuales se les irían colocando los valores de cada bit según lo necesario para la compresión.

Seguido a esto, se decidió por tener un hilo escritor y un hilo lector. El hilo escritor colocaría BitBlock’s (objeto producto de la compresión y que sabe escribirse) en una cola de donde el hilo lector los sacaría para escribirlos en el archivo de salida. La clase FileCompressor en éste punto era considerablemente grande y conocía a casi todas las clases realizadas de una u otra forma.

Éste subproblema se logró resolver de de forma rápida aun que más tarde se evidenció un problema derivado de él que llevo un tiempo considerable y se necesito de la ayuda de uno de los docentes para resolverlo.

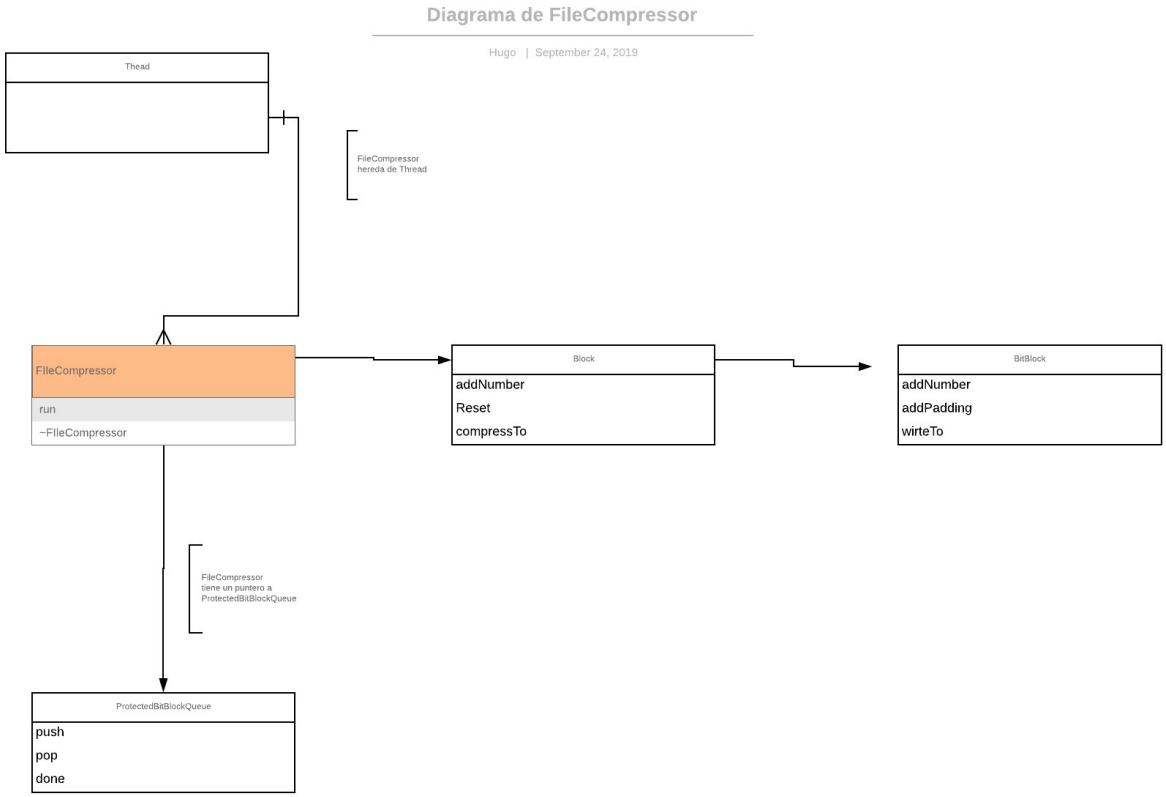
Para la realización de la ProtectedBitBlockQueue se leyó el ejemplo sobre conditional variables que aparece en cppreference (https://es.cppreference.com/w/cpp/thread/condition_variable) y se lo aplico a una cola protegida de enteros. Luego de varias correcciones se la modificó para que pueda almacenar BitBlocks. Se necesitó también definir el constructor por movimiento y el operador ‘=’ por movimiento.

El mencionado problema fue solucionado de forma aparentemente satisfactoria (aun que más tarde en el siguiente problema se evidenciaría que ésto no fue así) y en relativamente poco tiempo. Fue una gran idea el primero realizar una cola protegida que almacene enteros y probarla pues es más fácil encontrar errores. La clase FileCompressor redujo su complejidad notablemente, resultando en una clase mucho más sencilla y fácil de leer.

Sin embargo, no todos los errores fueron encontrados en la etapa previa. Lo cual traería muchas complicaciones la momento de agregar varios hilos. El problema se debía a que si el Writer intentaba sacar algo de ProtectedBitBlockQueue y ésta no tenía más elementos, resultaba en un Segmentation Fault. La causa era que nuca se revisaba al momento de extraer elementos dentro de la cola si ésta estaba vacía o no.

Dicho problema fue el principal motivo de que la parte final del trabajo se demorara considerablemente más de lo que debía pues tomo cuatro días y la ayuda de un docente el encontrar el problema. Dejando ésto atrás, se tuvo un buen análisis de las secciones críticas evitando así race conditions y aún que se tuvieron algunos deadlocks durante el desarrollo fueron relativamente obvios y no tan complejos de reparar.

Esquema de Diseño



En el diagrama mostrado se muestra de manera algo resumida como FileCompressor delega las responsabilidades para comprimir un bloque. Se pide disculpas por el diagrama fuera del formato UML, la herramienta utilizada no fue la mejor.

Conclusiones:

Se ejecutó el programa en sus tres versiones con el comando time con lo cual se concluye que, el realizar programas que trabajen con múltiples hilos puede traer un muy amplio beneficio en cuanto a rendimiento pero aumenta también la complejidad de la realización de un programa debido a que los errores que se pueden tener por la mala utilización de threads son bastante difíciles de encontrar. Con lo cual se debe tener en cuanta tanto los beneficios de performance como los costos de producción al momento de realizar un programa multi-hilos pues de no necesitar de una reducción en tiempo de ejecución podría resultar contraproducente.