Данное решение предназначено для поблочного многопоточного сжатия/распаковки файлов с использованием GZip.

Если какие-то моменты в коде вам не понятны, вы можете прочитать следующую информацию, которой руководствовался я при написании данного решения.

Основу составляет класс BlockArchiver, который является базовым для Compressor и Decompressor.

BlockArchiver подразумевает следующие этапы работы: считывание блоков из входного файла, обработка считанных блоков и запись обработанных блоков в выходной файл. Все эти операции разделены по потокам: 1 поток на считывание, количество обрабатывающих потоков = количеству ядер процессора и 1 поток на запись. За управлением потоками и количеством памяти следит Dispatcher.

Контроль за памятью осуществляется следующим образом: при считывании мы проверяем количество памяти процесса с установленным лимитом, если оно больше, то останавливаем поток чтения и запускаем сборку мусора. В это время обрабатывающие потоки и записывающий продолжают свою работу и, когда считанные блоки закончатся, то происходит активация считывающего потока и сборка мусора. Также записывающий поток может приостанавливаться, когда очередной блок еще не был обработан. Каждый из обрабатывающих потоков дает знать записывающему, что отдал ему новый блок. Лимит памяти выбирается по следующей логике: для 32-битного процесса ограничение по памяти составляет около 2 Гб, но может срабатывать и на 1.5 Гб; Поэтому задаем лимит в 1.4 Гб с небольшим запасом; Также вычисляем количество доступной сейчас памяти и берем половину, чтобы оставить на другую работу; Если процесс 32-битный, то выбираем меньшее из них значение, а если 64-битный, то берем половину доступной памяти.

Теперь детально об алгоритмах архивации. Они задаются в конкретных классах: Compressor и Decompressor.

Изначально нужно упомянуть структуру сжатого блока GZip:

```
Offset   Length   Contents
  0      2 bytes  magic header  0x1f, 0x8b (\037 \213)
  2      1 byte   compression method
                     0: store (copied)
                     1: compress
                     2: pack
                     3: lzh
                     4..7: reserved
                     8: deflate
  3      1 byte   flags
                     bit 0 set: file probably ascii text
                     bit 1 set: continuation of multi-part gzip file, part number present
                     bit 2 set: extra field present
                     bit 3 set: original file name present
                     bit 4 set: file comment present
                     bit 5 set: file is encrypted, encryption header present
                     bit 6,7:   reserved
  4      4 bytes  file modification time in Unix format
  8      1 byte   extra flags (depend on compression method)
  9      1 byte   OS type
[
         2 bytes  optional part number (second part=1)
]?
[
         2 bytes  optional extra field length (e)
        (e)bytes  optional extra field
]?
[
           bytes  optional original file name, zero terminated
]?
[
           bytes  optional file comment, zero terminated
]?
[
        12 bytes  optional encryption header
]?
           bytes  compressed data
         4 bytes  crc32
         4 bytes  uncompressed input size modulo 2^32
```

Для сжатия алгоритм таков: считывания блоками входной файл, нумеруем их и отдаем дальше. Каждый блок сжимается стандартным алгоритмом, а потом мы записываем с 4 по 8 байты число, которое значит длину блока до сжатия, чтобы можно было легко распаковать блок. Почему именно туда? Там хранится время изменения файла в Unix формате, которое заполняется нулями, поэтому мы можем этим воспользоваться. Затем такой блок поступает на запись. Перед записью блоков происходит запись числа типа long для запоминания длины файла до сжатия. Это нужно для отображения прогресса выполнения (Пришлось пожертвовать 8 байт ради этой цели). И так пока все блоки не запишутся.

Для распаковки алгоритм такой: изначально считываем число = длине файла до сжатия, опять же для прогресса. А потом последовательно все блоки. Сначала первые 8 байт, с 4 по 8 из которых означают длину блока, вот ее то потом и дочитываем. Считываем блок, нумеруем и отдаем на обработку. Потом распаковка, сколько считывать из stream’а распаковки? А длина блока до сжатия хранится в последних 4 байтах блока. Разжимаем блок и отдаем на запись. Тут уже просто последовательная запись разжатых блоков.

Вот и вся информация, которой руководствовался я, чтобы реализовать данное решение.