Новокрещенов Константин

09.08.2015

Тестовое задание на вакансию C# разработчика

Описание работы программы **GZipTest.exe**

В своей работе программа для выполнения преобразования входного файла, включающую в себя как сжатие исходного файла в архив, так и распаковку исходного архива в прежний файл, использует следующую базовую идею, а именно: входной файл разбивается на «куски» (**FileChunk**) определенного размера, далее над каждым из полученных «кусков» осуществляется требуемое преобразование (сжатие / распаковка), после чего преобразованные «куски» сшиваются для получения выходного файла. Процедура разделения входного файла и процедура сшивки выходного файла различаются в зависимости от режима работы программы.

При работе программы в режиме Compress исходный файл делится на куски равного размера (кроме последнего куска, размер которого будет меньше в случае, если размер исходного файла не кратен размеру «куска»). По умолчанию, исходный файл разделяется на куски по 15 МБ. Для осуществления разделения файла используется класс **FixedSplitInfoExtractor**, результатом работы которого является объект типа **FileSplitInfo**, который хранит массив элементов типа **FileChunkInfo**. Класс **FileChunkInfo** предназначен для описания местоположения «куска» внутри файла: поля **Id**, **Length**, **Position** определяют порядковый номер куска файла, его длину и начальную позицию соответственно.

Сжатие каждого из полученных кусков исходного файла представляет собой «конвейер», состоящий из трех главных этапов: чтение куска в память, преобразование (сжатие) куска с помощью класса **GZipStream**, запись полученного сжатого (архивного) куска на диск в выходной архив. Выполнение данных операций реализовано в виде задач **ReadFileChunkTask** (приоритет 0), **ConvertFileChunkTask** (приоритет 1), **FileAssembleTask** (приоритет 2). Данные задачи выполняются многопоточно с помощью реализованного пула потоков **PrioryTaskPool**. Каждая из задача имеет приоритет (указан в скобках). Приоритет отражает «важность» скорейшего выполнения задачи с точки зрения освобождения памяти, которая в данном случае является критическим ресурсом (особенно при обработке файлов с размером, превышающим размер доступной оперативной памяти). В первую очередь выполняются задачи с наивысшим приоритетом и далее по порядку. После того, как все задачи выполнены и все преобразованные (сжатые) куски собраны и записаны в архив, в конец полученного архива добавляется метаинформация (**ArchiveHeader**), в которой указывается количество сжатых кусков в архиве и длина каждого из них. Сборка архива и запись метаинформации реализована к классе **ArchiveFileAssembler**.

Работа программы в режиме **Decompress** во многом аналогична работе в режиме **Compress**, описанном выше. Основные различия заключаются в процедуре разделения архива и сшивке выходного файла. Для разделения архива на куски (а именно получения размера каждого из них и начальной позиции) используется метаинформация (**ArchiveHeader**), записанная при сборке архива. Логика получения «кусков» архива в соответствии с данной метаинформацией реализована в классе **ArchiveSplitInfoExtractor**.

Далее каждый из «кусков» проходит через «конвейер» задач, в результате распаковывается (с помощью **GZipStream**) в исходный (прежний) кусок (для определения размера выходного куска используются последние 4 байта сжатого GZipStream-ом куска, в которые GZipStream записывает длину исходного куска, если она меньше 4 ГБ). Далее все распакованные куски собираются в выходной файл. В данном случае в конец полученного файла метаинформация уже не записывается. Сборка выходного файла реализована в классе **RegularFileAssembler**.

Как было упомянуто выше, выполнение задач по преобразованию кусков файла осуществляется с помощью пула потоков, количество параллельных потоков исполнения в котором по умолчанию равно количеству логических процессоров в системе.

Также реализована возможность корректной остановки программы по **Ctrl+C**.