Санкт-Петербургский государственный университет

Шеверев Сергей Вячеславович

Выпускная квалификационная работа

Разработка библиотеки ввода-вывода с прозрачным сжатием данных и произвольным доступом для языка СИ

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование информационных систем»*

Основная образовательная программа *СВ.5006.2018 «Математическое обеспечение и администрирование информационных систем»*

Профиль *Системное программирование*

Научный руководитель:
Старший преподаватель кафедры информатики, к.ф.-м.н. Н. Ю. Ловягин

Рецензент:
Ассистент кафедры информационно-аналитических систем, Г. А. Чернышев

Санкт-Петербург
2022

Saint Petersburg State University

Sergei Sheverev

Bachelor's Thesis

Development of an I/O library with transparent data compression and random access for the C language

Education level: bachelor

Speciality *02.03.03 «Software and Administration of Information Systems»*

Programme *CB.5006.2018 «Software and Administration of Information Systems»*

Profile: *System Programming*

Scientific supervisor:
Senior Lecturer, C.Sc. N. U. Lovyagin

Reviewer:
Assistant Lecturer, G. A. Chernyshev

Saint Petersburg
2022

Оглавление

1. Введение

Программы, обрабатывающие большие массивы данных, т.е. объемы данных, превышающие размеры оперативной памяти, активно используют дисковую подсистему для хранения информации о промежуточных вычислениях и результатах своей работы. Примерами задач, требующих хранения таких объемов данных, могут служить задачи численного моделирования статических и динамических систем, задачи биоинформатики, системы хранения и анализа протоколов работы и другие. Во многих случаях требуется произвольный доступ к отдельным участкам данных на чтение и изменение.

Применение сжатия способно уменьшить требуемые объемы дискового пространства для хранения данных (например, в операционных системах семейства GNU/Linux широко применяется модуль ядра zram для сжатия данных процессов в памяти). Однако, прямое применение библиотек сжатия и архиваторов не всегда удобно, так как использование подобных инструментов требует освоения пользователем их интерфейсов и учета особенностей работы каждой из программ.

Более удобное решение представит собой библиотеку, обеспечивающую сохранение данных в сжатом виде, но имеющую при этом привычный для используемого языка программирования (аналогичный стандартной библиотеке языка) интерфейс файлового ввода-вывода.

Данная работа посвящена реализации библиотеки файлового ввода-вывода с произвольным доступом к данным, обеспечивающей прозрачные для пользователя (программиста) сжатие и распаковку сохраняемых в файл данных, для языка программирования СИ с интерфейсом, аналогичным интерфейсу программирования приложений библиотеки `stdio`. Библиотека должна представлять собой обертку над реализациями алгоритмов сжатия данных, предоставляемых библиотеками, выбираемыми программистом.

В ходе работы реализован функционал, обеспечивающий хранение данных в виде независимо сжимаемых блоков, произвольный доступ к данным на чтение и изменение. Также была проведена оптимизация

операций чтения и записи с помощью реализации механизма кеширования блоков данных. Пользователю предоставлена возможность конфигурирования работы библиотеки. Были проведены тесты производительности ввода-вывода и сделан вывод о применимости полученного решения на практике.

2. Обзор существующих решений

На данный момент существует несколько продуктов, предоставляющих сжатие и распаковку данных без потерь.

1. Файловые системы с прозрачным сжатием:

   - Fuse-zip [6] — FUSE-файловая система, работающая в пользовательском пространстве. Позволяет примонтировать zip-архив и работать с ним как со стандартной директорией. Позволяет производить чтение и запись файлов, сохраняя изменения при размонтировании.
   - ZIPFS [2] — библиотека, позволяющая работать с zip-архивами как с виртуальной файловой системой, обеспечивающая последовательный доступ на чтение. Загружает содержимое файла в память для ускорения работы.
   - PyFilesystem [11] — модуль языка Python, предоставляющий пользователю абстракцию над файловой системой. Позволяет работать с zip-архивами: манипулировать файлами внутри контейнера, удалять, создавать новые или перезаписывать имеющиеся файлы.
   - Btrfs [10] — файловая система с открытым исходным кодом, обеспечивающая ленивое прозрачное сжатие данных.
   - Ntfs [12] — проприетарная файловая система с прозрачным сжатием данных, разработанная компанией Microsoft.
   - ZFS [13] — проприетарная файловая система с прозрачным сжатием данных, разрабатываемая компанией Oracle.

   Использование файловых систем с прозрачным сжатием данных требует доступа к операционной системе. Также на пользователя возлагаются дополнительные задачи администрирования: необходимо совершить ряд действий перед началом работы со сжатым файлом, например, предварительно примонтировать его и размонтировать по окончанию использования.

2. Утилиты сжатия данных:

   - bgzip [7] (BGZF)
   - bzip2 [16]
   - 7-zip [14]
   - XZ Utils [15]
   - lzop [9]

   Применение утилит сжатия данных требует от программиста их вызова в коде программы, что приводит к проблемам с производительностью и безопасностью. Более того, архиваторы не предоставляют возможности доступа к данным без предварительной распаковки файла.

3. Библиотеки сжатия данных:

   - Miniz [5] — высокопроизводительная библиотека для потокового сжатия данных без потерь, распространяемая в виде одного файла с исходным кодом (и заголовочного файла). Высокая производительность обеспечивается за счет применения менее эффективного сжатия данных. Пользователю доступны функции для чтения и записи файлов в контейнеры формата ZIP.
   - LZ4 [3] — библиотека блочного сжатия с динамическим регулированием качества сжатия данных.
     Библиотечный код предоставляет возможность произвольного доступа к блокам сжатых данных на чтение.
   - C-Blosc [17] — библиотека блочного сжатия без потерь.
     За счет того, что файл представлен в виде блоков сжатых данных и указателей на эти блоки, у пользователя имеется возможность манипулировать данными произвольных блоков.

- zlib [8] — библиотека сжатия общего назначения, предоставляющая удобный и привычный пользователю(программисту) интерфейс файлового ввода-вывода с произвольным доступом к несжатым данным. Однако имеется ряд существенных недостатков таких как:
  - функция `gzseek` для смены позиции курсора в файле в режиме открытия файла на чтение эмулируется, что делает ее работу чрезвычайно медленной;
  - `gzseek` в режиме открытия файла на запись поддерживает смещение курсора только вперед;
  - становится невозможным применение функции `gzrewind` для "перемотки" файла, если он открыт для записи;
  - отсутствует возможность адресации от конца файла (`SEEK_END` не поддерживается).

Таблица 1: Сравнительная таблица библиотек сжатия данных.

Библиотека	Блочный доступ	Произвольный доступ
miniz	нет	нет
LZ4	чтение	нет
C-Blosc	чтение/запись	нет
zlib	нет	с ограничениями

Вышеперечисленные библиотеки сжатия данных не обладают требуемым функционалом, обеспечивающим произвольный доступ к данным на чтение и запись.

Представленные решения не предоставляют удобного интерфейса программирования приложений для полноценного произвольного доступа к данным в сжатом файле.

3. Постановка задачи

Целью работы является реализация библиотеки файлового ввода-вывода с произвольным доступом, обеспечивающую прозрачные для пользователя (программиста) сжатие и распаковку сохраняемых в файл данных, для языка программирования СИ с интерфейсом, аналогичным интерфейсу программирования приложений библиотеки `stdio`. В рамках данной работы поставлены следующие задачи:

1. Реализовать привычный пользователю интерфейс программирования приложений (API), обеспечивающий произвольный доступ к данным, предоставляющий аналоги функций стандартной библиотеки, перечисленные в Таблице 2.

Функция `stdio`	Аналог
`fopen`	`cfopen`
`fclose`	`cfclose`
`fseek`	`cfseek`
`ftell`	`cftell`
`rewind`	`crewind`
`fread`	`cfread`
`fwrite`	`cfwrite`
`fsync`	`cfsync`

Таблица 2: Реализуемые аналоги функций `stdio`.

2. Реализовать механизм кэширования блоков данных для чтения и записи.

3. Предоставить пользователю (программисту) возможность конфигурирования работы библиотеки, а именно:

   - размер блока несжатых данных;
   - стратегию обработки свободных блоков;
   - стратегию записи новых блоков;

- порог уплотнения файла в процентах;
- компрессор/декомпрессор данных.

4. Провести тестирование производительности файлового ввода-вывода и сделать вывод о применимости реализованной библиотеки.

Реализуемая библиотека является оберткой над алгоритмом сжатия данных, предоставленным библиотекой сжатия данных, используемой программистом (Рис. 1).

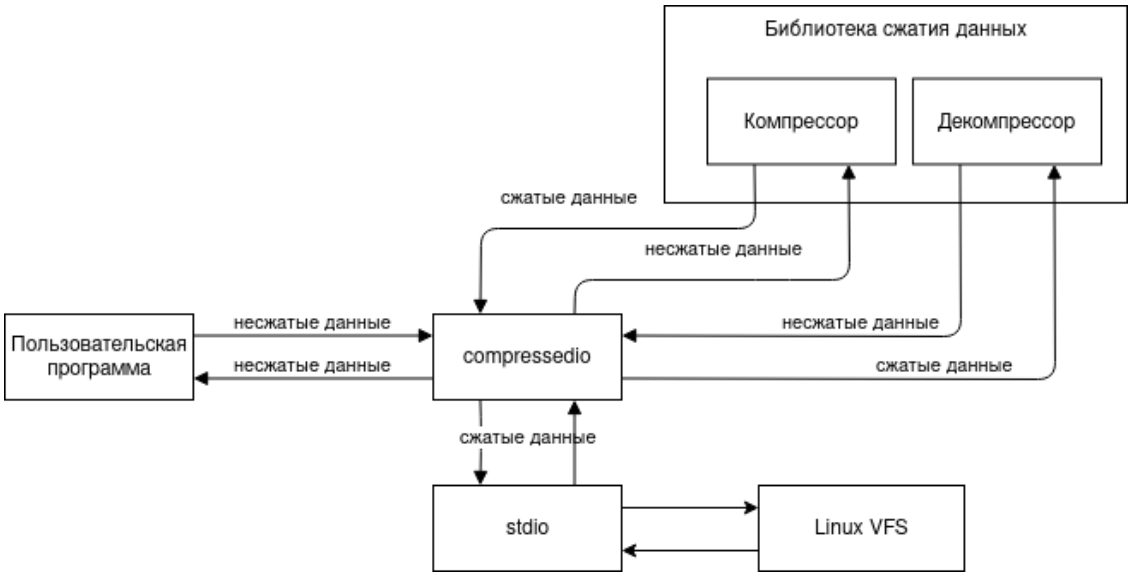

Рис. 1: Схема работы реализуемой библиотеки

4. Описание решения

Пользовательские данные разбиваются на отдельные блоки фиксированного размера, который может быть сконфигурирован. Блоки данных сжимаются и записываются в файл независимо друг от друга. Под сжатым файлом будем понимать файл, который хранится непосредственно на диске, а под несжатым файлом — абстракцию над сжатым файлом, с которой взаимодействует программист, использующий API реализуемой библиотеки. Далее под блоком сжатых данных будет пониматься блок, хранящийся в сжатом файле, а под блоком несжатых данных — распакованные данные сжатого блока и данные, поступившие от программы на запись (еще не сжатые). Сжатие данных прозрачно для пользователя: он взаимодействует с файлом как с несжатым. Далее пользователем будем называть программиста, который использует API библиотеки.

Каждый блок сжатых данных сопровождается служебной информацией, которая также записывается в файл непосредственно перед сжатыми данными этого блока. Служебная информация используется при последующей работе с соответствующим блоком сжатых данных.

При запросе информации, хранящейся в блоке, его сжатые данные считываются из файла в буфер библиотеки в оперативной памяти, распаковываются, и нужный фрагмент несжатых данных копируется в предоставленный пользователем буфер.

По мере выполнения программистом операций записи (чтение из сжатого файла не изменяет его содержимое) сжатый файл может фрагментироваться, так как возникнет необходимость сжать измененные несжатые данные блока повторно и перезаписать его вместе со служебной информацией. Кроме того, степень сжатия может ухудшиться и несмотря на сохранение размера несжатого блока размер нового сжатого блока может превысить размер старого. Свободным блоком назовем фрагмент сжатого файла, в который был записан блок сжатых данных до его перезаписи.

При записи данных в блок пользовательский буфер копируется в

буфер несжатых данных блока (по необходимому смещению), затем измененные несжатые данные блока сжимаются в буфер библиотеки в оперативной памяти, определяется подходящее место в сжатом файле (в конце или на место одного из свободных блоков в соответствии с конфигурацией) и блок сжатых данных записывается вместе с его служебной информацией. В том случае, когда блок сжатых данных был записан на место одного из свободных блоков, размер соответствующего свободного блока уменьшается (вплоть до нуля).

Поскольку фрагментация сжатого файла уменьшает эффективность использования дискового пространства, предусмотрена возможность уплотнения файла при превышении задаваемого пользователем порога переиспользования места в сжатом файле.

Помимо служебной информации блоков в сжатом файле дополнительно хранятся заголовок файла с общей информацией о сжатом файле и две таблицы позиций блоков в сжатом файле: таблица позиций блоков сжатых данных и таблица позиций свободных блоков. Обе таблицы представляют собой последовательно записанные в сжатый файл позиции первых байт служебной информации блоков. Расположение таблиц позиций блоков в сжатом файле не фиксировано — для их поиска используется заголовок, в который записано текущее расположение таблиц.

4.1. Механизм выбора блока

Блокам сжатых данных соответствуют непересекающиеся фрагменты несжатого файла. Во время исполнения блоки сжатых данных представлены в виде упорядоченного по левым границам фрагментов несжатого файла массива в памяти. Индекс соответствующего блока сжатых данных вычисляется по формуле:

$$b_{ind} = f_{pos}\, div\, b_{size}$$

где

- b_{ind} — индекс искомого блока;

- f_{pos} — позиция курсора внутри несжатого файла;
- b_{size} — размер блоков несжатых данных.

4.2. Загрузка блока в память

Загрузка блока данных в память производится в три этапа. На первом этапе из таблицы позиций блоков сжатых данных загружается позиция первого байта служебной информации о блоке. Далее на втором этапе выполняется чтение служебной информации из сжатого файла, инициализация памяти экземпляра структуры блока и запись этой структуры в массив блоков по соответствующему индексу. Третий этап загрузки блока разделен на три подэтапа.

1. Инициализация памяти буфера несжатых данных блока.
2. Чтение сжатых данных блока в буфер библиотеки в оперативной памяти.
3. Сжатые данные из буфера библиотеки распаковываются в буфер несжатых данных блока.

Стоит отметить, что загрузка блока выполняется при необходимости выполнения операций чтения или записи с этим блоком, а буфер сжатых данных библиотеки инициализируется при первой операции чтения или записи. Подобный способ управления загрузкой блоков сжатых данных значительно сокращает время открытия файла, так как загружается служебная информация только тех блоков сжатых данных, которые действительно необходимы для работы пользовательской программы.

4.3. Перезапись/запись блока

Как было упомянуто ранее, изменение несжатых данных блока влечет за собой необходимость его перезаписи в сжатый файл и появление свободных блоков. Для обеспечения безопасности изменения данных в

сжатом файле блок сжатых данных не перезаписываются по его текущей позиции даже в случае, когда размер сжатых данных не изменился или даже уменьшился, поскольку при возникновении исключительных ситуаций сжатые данные могут быть повреждены, что сделает невозможным их последующее использование (невозможность распаковки поврежденных сжатых данных).

Данные блока (нового или измененного) сжимаются и записываются вместе со служебной информацией по позиции в сжатом файле, которая вычисляется в зависимости от конфигурации стратегии записи блоков, выбранной пользователем. Доступны следующие стратегии записи блоков сжатых данных:

- запись в конец сжатого файла, игнорируя свободные блоки;
- запись на место любого подходящего по размеру свободного блока;
- запись на место свободного блока с наименьшим подходящим размером.

Служебная информация о позиции перезаписанного блока сжатых данных в таблице позиций обновляется. Однако в случае ошибки перезаписи блока сжатых данных обновление служебной информации не происходит. Фрагмент сжатого файла, в который ранее был записан блок сжатых данных объявляется свободным и добавляется в таблицу свободных блоков. Свободный блок в зависимости от конфигурации может быть перезаписан нулевыми байтами для оптимизации хранения сжатого файла файловой системой (если поддерживаются разреженные файлы).

4.4. Запись таблиц блоков

В случае записи в файл нового блока размер таблицы позиций блоков сжатых данных увеличивается, что приводит к необходимости перезаписи этой таблицы по новой позиции в сжатом файле. В случае невозможности перезаписать таблицу позиций блоков сжатых данных

происходит возвращение к использованию уже имеющейся. Стоит отметить, что несмотря на неудачу перезаписи таблицы информация о новых блоках сжатых данных остается доступной до закрытия файла.

Таблица позиций сжатых блоков хранится по принципу массивов с избыточной вместимостью и перезаписывается только тогда, когда в ней не хватает места для новых блоков. Таблица позиций свободных блоков перезаписывается только при закрытии файла.

Ошибка перезаписи таблицы позиций свободных блоков не является критической, поскольку потеря информации о новых свободных блоках не приводит к ошибкам учета переиспользования места, так как в заголовке сжатого файла дополнительно хранится число всех свободных байтов. Потеря информации о новых блоках сжатых данных при ошибках перезаписи таблицы позиций сжатых блоков к моменту закрытия файла приведет к потере данных только новых блоков.

4.5. Уплотнение сжатого файла

При превышении установленного пользователем лимита переиспользования места в сжатом файле (суммы размера всех свободных блоков) происходит уплотнение сжатого файла при его закрытии. Под уплотнением понимается перезапись сжатого файла таким образом, что блоки сжатых данных записываются непосредственно друг за другом, не оставляя свободных блоков. Уплотнение сжатого файла реализовано следующим образом.

1. В директории сжатого файла создается пустой файл.
2. В созданный пустой файл записывается заголовок сжатого файла, в котором размер переиспользованного места полагается пустым.
3. Записывается пустая таблица свободных блоков (таблица размера 0) и создается копия таблицы блоков сжатых данных в памяти.
4. Последовательно записываются блоки сжатых данных и для каждого блока обновляется его позиция в копии таблицы блоков сжатых данных в памяти.

5. Из памяти записывается таблица блоков сжатых данных в новый файл.

6. Прежний сжатый файл удаляется, созданный ранее файл переименовывается.

Необходимо отметить, что повторное сжатие неизмененных данных не производится — сжатые данные копируются из сжатого файла в новый файл.

4.6. Механизм кэширования блоков

Для оптимизации операций ввода-вывода был реализован кэш блоков несжатых данных. Кэш блоков данных представлен в виде LRU-кэша с двумя очередями (нижней и верхней). Запрашиваемые в библиотечном коде блоки данных сначала попадают в нижнюю очередь. После этого происходит загрузка сжатых данных этого блока, эти данные распаковываются в память и хранятся в ней, пока данный блок не покинет кэш. При повторном запросе блоков, находящихся в нижней очереди кэша, они перемещаются в верхнюю очередь. Блоки, которые были вытеснены из верхней очереди кэша, снова попадают в нижнюю очередь и при вытеснении их из нее (нижней очереди) вовсе покидают кэш.

Операции записи производятся с несжатыми данными закэшированного блока. Модифицированные данные блока будут записаны в файл при его вытеснении из кэша. Данные сжимаются непосредственно перед записью в файл. В том случае, когда запись происходит в новый блок, которого еще нет в файле, его данные также будут записаны в файл в сжатом виде. Данные блоков, вытесненных из кэша, освобождаются и будут вновь подгружены при необходимости. Схематическое устройство кэша блоков представлено на Рис. 2.

Отметим, что вариант кэша с одной очередью обладает следующим недостатком: новые единожды запрошенные блоки данных, попадая в кэш, могут привести к тому, что другие неоднократно запрошенные

блоки данных вытесняются (находясь в конце очереди) из кэша. Кэш с двумя очередями частично решает эту проблему [4].

При добавлении блока данных в кэш у него выставляется специальное поле-указатель `cache_node` на элемент очереди (верхней или нижней), соответствующий этому блоку. При вытеснении блока из кэша данное поле устанавливается в `NULL`-значение, что соответствует отсутствию блока в кэше. Таким образом, поиск блока данных в кэше сводится к проверке поля `cache_node` этого блока (Рис. 3).

Пользователю доступна функция синхронизации данных блоков в кэше со сжатым файлом.

```
int cfsync(cFILE* cfile)
```

Она в принудительном порядке записывает данные блоков в кэше. Данные блоков из кэша сохраняются в сжатый файл при его закрытии функцией `cfclose`. Следует отметить, что в случае возникновения исключительных ситуаций в пользовательском коде (сбоя в работе) кэш не будет сброшен в файл и несохраненные данные будут утеряны.

Размер кэша задается количеством кэшируемых блоков. Для этого необходимо в конфигурационной структуре `cCONFIG` выставить соответствующее поле `cache_size`. Использование кэша блоков можно отключить, выставив `cache_size` в `0`.

4.7. Настраиваемый компрессор данных

Программисту предлагается самостоятельно выбрать компрессор данных, который будет использоваться библиотекой для соответствующих операций сжатия и распаковки данных. Для этого необходимо заполнить структуру компрессора `cCOMPRESSOR`, предоставив следующие функции:

- `int compress(void* dst, size_t* dst_len, void* source, size_t source_len)`

  Обеспечивает сжатие буфера данных `source` в буфер `dst`. Буфер `dst` в качестве размера должен иметь значение, возвращаемое функцией `compressBound`.

- `int uncompress(void* dst, size_t dst_len, void* source, size_t source_len)`

  Обеспечивает распаковку буфера данных `source` в буфер `dst`. Буфер `dst` должен иметь достаточный размер, чтобы разместить распакованные данные;

- `size_t compressBound(size_t source_len)`

  Возвращает верхнюю границу размера буфера, в который будут помещаться данные при сжатии.

Функции `compress`/`uncompress`, определяемые пользователем, в случае успеха должны возвращать 0 и -1, если возникла ошибка.

Для установки компрессора для сжатого файла пользователь должен вызвать функцию с сигнатурой

`int set_compressor(cCOMPRESSOR* compressor, cFILE* cfile).`

Пример заполнения структуры компрессора с использованием функций общего назначения библиотеки `zlib`:

```
#include <zlib.h>
...
/* compress, uncompress и compressBound from zlib.h */
cCOMPRESSOR compressor;
compressor.compress = compress;
compressor.uncompress = uncompress;
compressor.compressBound = compressBound;
int res = set_compressor(&compressor, fin);
if (res == -1){
    puts(''Error'');
} else {
    puts(''Success'');
}
```

Соответствующие действия для установки пользовательского компрессора для предварительно открытого (с помощью функции `cfopen`) сжатого файла должны быть выполнены перед его использованием.

4.8. Стандартная конфигурация

Пользователю доступна возможность использования стандартной конфигурации. Для этого ему необходимо вместо адреса экземпляра

конфигурационной структуры `cCONFIG` передать `NULL`-значение, либо воспользоваться функцией `get_default_cconfig`, генерирующей стандартную конфигурацию.

В качестве стандартной конфигурации предлагается следующая:

- размер несжатого блока данных — 16384 байта (16 Кбайт);
- по умолчанию свободные блоки в сжатом файле заполняются нулевыми байтами;
- порог уплотнения равен нулю;
- новые блоки сжатых данных размещаются в конце файла;
- кэш отключен.

4.9. Описание API

Предоставляемый библиотекой интерфейс программирования приложений включает следующие функции:

- `cFILE* cfopen(const char* filename, const char* mode, cCONFIG* config)`

  Открывает сжатый файл. Аргумент `filename` — это строка в стиле языка Си, в которой указывается путь до требуемого сжатого файла, аргумент `mode` определяет режим открытия файла, аналогичный режимам `r`, `r+`, `w`, `w+`, `a` и `a+` функции `fopen` библиотеки `stdio`. Аргумент `config` — указатель на конфигурационную структуру, может быть `NULL`-значением в том случае, когда программист желает использовать сжатый файл со стандартной конфигурацией. При открытии сжатого файла экземпляр конфигурационной структуры копируется, либо создается новый с использованием вызова функции `get_default_cconfig`. В процессе открытия сжатого файла считывается его заголовок, выделяется память под экземпляр структуры `cFILE` и происходит ее инициализация. Если файла по пути `filename` не существует, то создается новый сжатый файл с необходимой структурой (только в режимах `w`,

w+, a и a+). В случае успешной инициализации структуры cFILE вызывающей стороне возвращается указатель на эту структуру, либо NULL-значение при ошибке.

- `int cfclose(cFILE* cfile)`

  Закрывает сжатый файл. Аргумент `cfile` — это указатель на экземпляр структуры `cFILE`, который ранее был получен средствами вызова функции `cfopen`. Закрытие сжатого файла сопровождается синхронизацией блоков несжатых данных в кэше, записью таблицы свободных блоков в файл и вызовом функции уплотнения файла, если превышен пользовательский порог переуплотнения. Далее происходит освобождение памяти экземпляра структуры `cFILE`. Вызывающей стороне возвращается 0 в случае успеха и -1 иначе.

- `long int cftell(cFILE* cfile)`

  Возвращает текущую позицию курсора в несжатом файле или -1 в случае ошибки. Аргумент `cfile` — это указатель на экземпляр структуры `cFILE`, который ранее был получен средствами вызова функции `cfopen`.

- `void crewind(cFILE* cfile)`

  Устанавливает текущую позицию курсора в несжатом файле на нулевой байт и инициирует запуск механизма выбора блока, которому соответствует позиция 0 в несжатом файле. Аргумент `cfile` — это указатель на экземпляр структуры `cFILE`, который ранее был получен средствами вызова функции `cfopen`. Функция `crewind` не имеет возвращаемого значения.

- `int cfseek(cFILE* cfile, long int offset, int origin)`

  Перемещает курсор в несжатом файле и инициирует запуск механизма выбора блока, которому соответствует новое положение курсора несжатом файле. Аргумент `cfile` — указатель на экземпляр структуры `cFILE`, полученный ранее средствами вызова функции `cfopen`. Аргументы `offset` и `origin` используются

для вычисления новой позиции курсора. Доступно позиционирование курсора относительно начала несжатого файла, текущей позиции курсора и конца несжатого файла. Соответствующие позиции определяются метками `CSEEK_SET`, `CSEEK_CUR`, `CSEEK_END`. Возвращает 0 в случае успеха и -1 иначе.

- `size_t cfread(void* ptr, size_t size, size_t nmemb, cFILE* cfile)`

  Производит чтение из несжатого файла. Аргументы функции:

  - `ptr` — указатель на пользовательский буфер, в который будут копироваться несжатые данные;
  - `size` — размер в байтах элемента, считываемого из несжатого файла;
  - `nmemb` — количество элементов размера `size`, которые необходимо считать из несжатого файла;
  - `cfile` — указатель на экземпляр структуры `cFILE`, полученный средствами вызова функции `cfopen`, из которого будет происходить чтение несжатых данных.

  Функция `cfread` загружает блок несжатых данных, соответствующий текущей позиции курсора в несжатом файле согласно описанной ранее процедуре загрузки блока. Однако, если используется кэш блоков, то функция `cfread` сначала попытается получить несжатые данные блока из кэша. В пользовательский буфер `ptr` копируется нужный фрагмент несжатых данных блока, смещается курсор в несжатом файле вперед. При необходимости происходит загрузка следующих блоков, пока не будет считано требуемое количество данных, либо не будет достигнут конец несжатого файла. Вызывающей стороне возвращается количество успешно прочитанных элементов размера `size`, которое не превосходит `nmemb`.

- `size_t cfwrite(void* ptr, size_t size, size_t nmemb, cFILE* cfile)`

  Производит запись в несжатый файл. Аргументы функции:

- `ptr` — указатель на пользовательский буфер, из которого будут копироваться несжатые данные;
- `size` — размер в байтах элемента, записываемого в несжатый файл;
- `nmemb` — количество элементов размера `size`, которые необходимо записать в несжатый файл;
- `cfile` — указатель на экземпляр структуры `cFILE`, полученный средствами вызова функции `cfopen`, из которого будет происходить чтение несжатых данных.

Функция `cfwrite` загружает блок несжатых данных, соответствующий текущей позиции курсора в несжатом файле согласно описанной ранее процедуре загрузки блока. В случае нахождения блока данных в кэше, данные блока будут получены из него. Из пользовательского буфера `ptr` копируется нужный фрагмент данных в несжатые данные блока, смещается курсор в несжатом файле вперед. При необходимости происходит загрузка следующих блоков, пока не будет записано требуемое количество данных, либо не будет достигнут конец несжатого файла. В случае, когда достигается конец несжатого файла и записаны не все элементы пользовательского буфера, происходит создание новых блоков сжатых данных. Измененные или новые блоки записываются в файл, либо их запись откладывается при использовании кэша и происходит в момент вытеснения этих блоков из кэша, либо при вызове функции `cfsync` или `fclose`. Вызывающей стороне возвращается количество успешно записанных элементов размера `size`, которое не превосходит `nmemb`.

- `int cfsync(cFILE* cfile)`

  Синхронизирует блоки несжатых данных в кэше со сжатым файлом. Новые блоки в кэше записываются в файл. Возвращает 0 в случае успеха и -1 иначе.

- `set_compressor(cCOMPRESSOR* compressor, cFILE* cfile)`

Устанавливает пользовательский компрессор для сжатого файла. Возвращает 0 в случае успеха и -1 иначе.

- `cCONFIG* get_default_cconfig()`

  Возвращает экземпляр конфигурационной структуры `cCONFIG` со значениями по умолчанию.

4.10. Архитектура библиотеки

Архитектура библиотеки представлена в виде схемы на Рис. 4. Сжатый файл во время исполнения представлен экземпляром структуры `cFILE`, полями которого являются указатели на экземпляры остальных сервисных структур:

- `cBTABLE` — таблица блоков сжатых данных;
- `cFBTALBE` — таблица свободных блоков;
- `cCONFIG` — конфигурационная структура;
- `cCOMPRESSOR` — структура компрессора;
- `cCACHE` — кэш блоков;
- `cHEADER` — структура заголовка сжатого файла.

4.11. Формат сжатого файла

В сжатом файле поддерживается определенный формат (Таблица 3). В начало файла записан заголовок файла, в котором хранится следующая служебная информация: размер блока несжатых данных, позиция первого байта таблицы блоков сжатых данных, позиция первого байта таблицы свободных блоков, смещение до конца сжатого файла, размер в байтах сжатых данных, хранимых в файле, размер в байтах перерасходуемого места в файле. Размер заголовка равен 48 байтам.

Таблицы позиций блоков сжатых данных и свободных блоков имеют одинаковую структуру в файле и записаны следующим образом: по

соответствующей позиции, указанной в заголовке сжатого файла, записаны размеры таблицы в блоках (реальный и виртуальный). Под реальным размером понимается фактически выделенное количество блоков, а под виртуальным — количество используемых блоков. Виртуальный размер не превосходит реальный. Далее последовательно записаны позиции первых байтов служебной информации каждого блока. Размер таблицы длины n в байтах вычисляется по формуле:

$$T_{bytesize} = 8 \cdot (n + 2)$$

Блоки состоят из двух частей: метаданных и данных (сжатых или объявленных свободными). Метаданные блока сжатых данных содержат следующую информацию о блоке: размер несжатых данных блока в байтах, размер сжатых данных блока в байтах, левая граница фрагмента несжатого файла, которому соответствует данный блок, а метаданные свободного блока — только размер в байтах фрагмента сжатого файла, обслуживаемый свободным этим блоком. Размеры блоков в байтах:

- $B_{bytesize} = 32 + B_{csize}$ для блока сжатых данных с размером сжатых данных B_{csize} байт;
- $FB_{bytesize} = 8 + FB_{size}$ для свободного блока с размером FB_{size} байт.

4.12. Тесты производительности

В качестве конфигурации библиотеки использовалась стандартная с размером блока в 16 Килобайт. В качестве библиотеки сжатия использовалась библиотека для сжатия общего назначения `zlib` с уровнем сжатия 6. Замеры времени производились командой `time`, использовалась сумма значений `user` и `sys`. Каждый тест проводился 5 раз, а значения усреднялись. Также была вычислена несмещенная оценка

дисперсии по формуле

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(v_i - v_{avg})^2}$$

где

- v_i — скорость работы ввода-вывода i-го запуска теста;
- v_{avg} — усредненная скорость работы ввода-вывода всех запусков теста;
- N — количество запусков теста.

Для тестирования производительности ввода-вывода использовались данные трех видов:

- текстовые данные;
- случайные числа типа `double` в интервале $[0, 1)$;
- случайные числа типа `uint64_t`.

Спецификации устройства, на котором проводились тесты:

- процессор — Ryzen 5 3600 с заблокированным множителем 36 (частота всех ядер составляла во время тестов 3600 MHz);
- оперативная память — два модуля (для обеспечения работы контроллера памяти процессора в двухканальном режиме) одноранговой памяти DDR4 Ballistix Sport LT с частотой модулей 2400 MHz с первичными таймингами 16-16-16-39;
- системная плата — Asus Tuf Gaming B450 Pro Gaming с частотой системной шины 99.8 MHz;
- накопитель — твердотельный диск Samsung 970 Evo Plus объемом 250 гигабайт (nvme 1.3).

Тестирование проводилось в виртуальной машине QEMU [1] с использованием модуля KVM [19], твердотельный накопитель был проброшен в виртаульную машину средстави OVMF [18].

Для тестирования производительности последовательных чтения и записи производились замеры времени сжатия и распаковки всего файла. Для сравнения результатов тестирования библиотечного кода также проводились соответствующие замеры времени работы утилиты `zip` с уровнем сжатия 6. Выбор именно этой утилиты обоснован тем, что она использует `zlib` в качестве библиотеки сжатия данных.

Для тестирования производительности произвольного чтения и записи проводились замеры времени чтения и записи фрагментов случайного размера, не превосходящих размера 160 Кбайт (10 блоков несжатых данных) по случайному смещению в несжатом файле.

Результаты тестирования представлены в Таблице — 4.

Скорость работы последовательного ввода-вывода библиотеки не уступает скорости работы архиватора `zip`. Однако сжатие данных является менее эффективным в среднем на 1.37%.

Произвольный доступ к несжатым данным снижает эффективность ввода-вывода. Произвольное чтение несжатых данных работает на 17.88% медленнее, а произвольная запись — на 10.08%. Легко заметить, что производительность утилиты `zip` и предоставленной библиотеки зависит от типа, используемых данных.

Скорость работы ввода-вывода реализованной библиотеки с вышеописанной конфигурацией сопоставима со скоростью работы USB-накопителей (десятки мегабайт в секунду) и скоростью работы HDD-дисков (100-200 мегабайт в секунду), что делает ее применимой на практике.

5. Заключение

В ходе проведения данной работы была реализована библиотека файлового ввода-вывода, обеспечивающая прозрачное для пользователя (программиста) сжатие и распаковку сохраняемых в файл данных, для языка программирования Си.

Были решены следующие задачи:

1. Реализован интерфейс программирования приложений (API), обеспечивающий произвольный доступ к данным, в который вошли следующие аналоги функций стандартной библиотеки:
   - `cFILE* cfopen(const char*, const char*, cCONFIG*);`
   - `int cfclose(cFILE*);`
   - `int cfseek(cFILE*, long int, int);`
   - `long int cftell(cFILE*);`
   - `void crewind(cFILE*);`
   - `size_t cfread(void*, size_t, size_t, cFILE*);`
   - `size_t cfwrite(void*, size_t, size_t, cFILE*);`
   - `int cfsync(cFILE*).`
2. Реализован механизм кэширования блоков.
3. Пользователю предоставлена возможность конфигурирования работы библиотеки.
4. Выполнено тестирование производительности файлового ввода-вывода и сделан вывод о пригодности реализованной библиотеки ввода-вывода для использования в задачах, требующих произвольного доступа к данным на чтение и изменение.

Исходный код библиотеки можно найти на github-репозитории: https://github.com/rousewayse/compressedio

Список литературы

[1] Bellard Fabrice. QEMU. — 2022. — Access mode: https://wiki.qemu.org/ (online; accessed: May 22, 2022).

[2] Cappelletti Luca. ZIPFS. — 2022. — Access mode: https://www.androwish.org/home/wiki?name=ZIP+virtual+file+system (online; accessed: May 22, 2022).

[3] Collet Yann. LZ4. — 2022. — Access mode: https://lz4.github.io/lz4/ (online; accessed: May 22, 2022).

[4] Dennis Shasha Theodore Johnson. A Low Overhead High Performance Buffer Management Replacement Algorithm. — P. 440. — Access mode: http://www.vldb.org/conf/1994/P439.PDF (online; accessed: May 22, 2022).

[5] Geldreich Rich. Miniz. — 2022. — Access mode: https://github.com/richgel999/miniz (online; accessed: May 22, 2022).

[6] Gonzalez Ryan. Fuse-zip. — 2022. — Access mode: https://github.com/refi64/fuse-zip (online; accessed: May 22, 2022).

[7] Handsaker Bob. bgzip. — 2022. — Access mode: http://www.htslib.org/doc/bgzip.html (online; accessed: May 22, 2022).

[8] Jean-loup Gailly Mark Adler. ZLIB. A Massively Spiffy Yet Delicately Unobtrusive Compression Library. — 2022. — Access mode: https://zlib.net/ (online; accessed: May 22, 2022).

[9] Markus Franz Xaver Johannes Oberhumer. Lzop. — 2017. — Access mode: https://www.lzop.org/ (online; accessed: May 22, 2022).

[10] Mason Chris. Btrfs. — 2022. — Access mode: https://btrfs.wiki.kernel.org/index.php/Main_Page (online; accessed: May 22, 2022).

[11] McGugan Will. PyFilesystem. — 2022. — Access mode: https://www.pyfilesystem.org/ (online; accessed: May 22, 2022).

[12] Microsoft. NT File System. — 2022. — Access mode: https://www.microsoft.com/ (online; accessed: May 22, 2022).

[13] Oracle. ZFS. — 2022. — Access mode: https://www.oracle.com/solaris/ (online; accessed: May 22, 2022).

[14] Pavlov Igor. 7-Zip. — 2022. — Access mode: https://www.7-zip.org/ (online; accessed: May 22, 2022).

[15] Project The Tukaani. XZ Utils. — 2022. — Access mode: https://tukaani.org/xz/ (online; accessed: May 22, 2022).

[16] Seward Julian. bzip2 is a freely available, patent free, high-quality data compressor. — 2019. — Access mode: https://www.sourceware.org/bzip2/ (online; accessed: May 22, 2022).

[17] Team The Blosc Development. Blosc. — 2022. — Access mode: https://github.com/Blosc/c-blosc (online; accessed: May 22, 2022).

[18] TianoCore. OVMF. — 2022. — Access mode: https://github.com/tianocore/tianocore.github.io/wiki/OVMF (online; accessed: May 22, 2022).

[19] community The Linux Kernel. KVM. — 2022. — Access mode: https://www.linux-kvm.org/page/ (online; accessed: May 22, 2022).

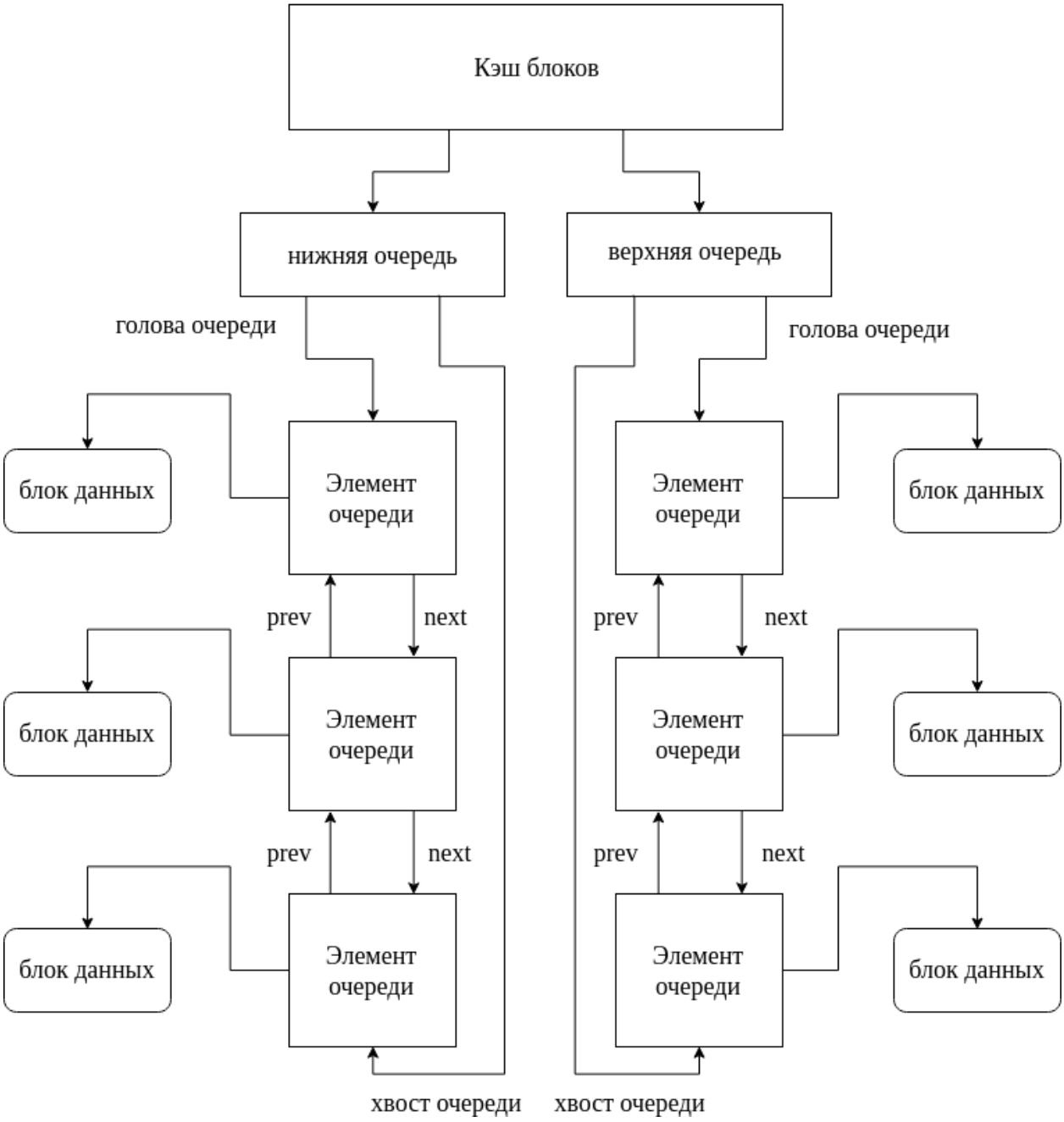

Рис. 2: Схема устройства кэша блоков

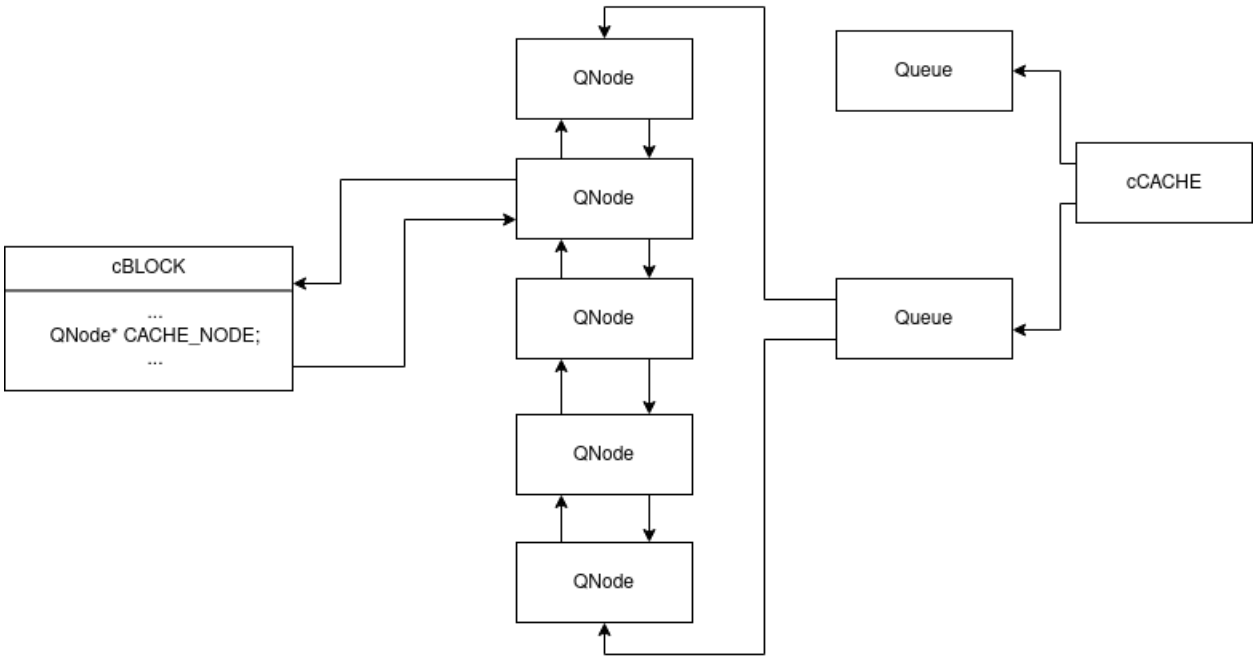

Рис. 3: Схема поиска блока в кэше

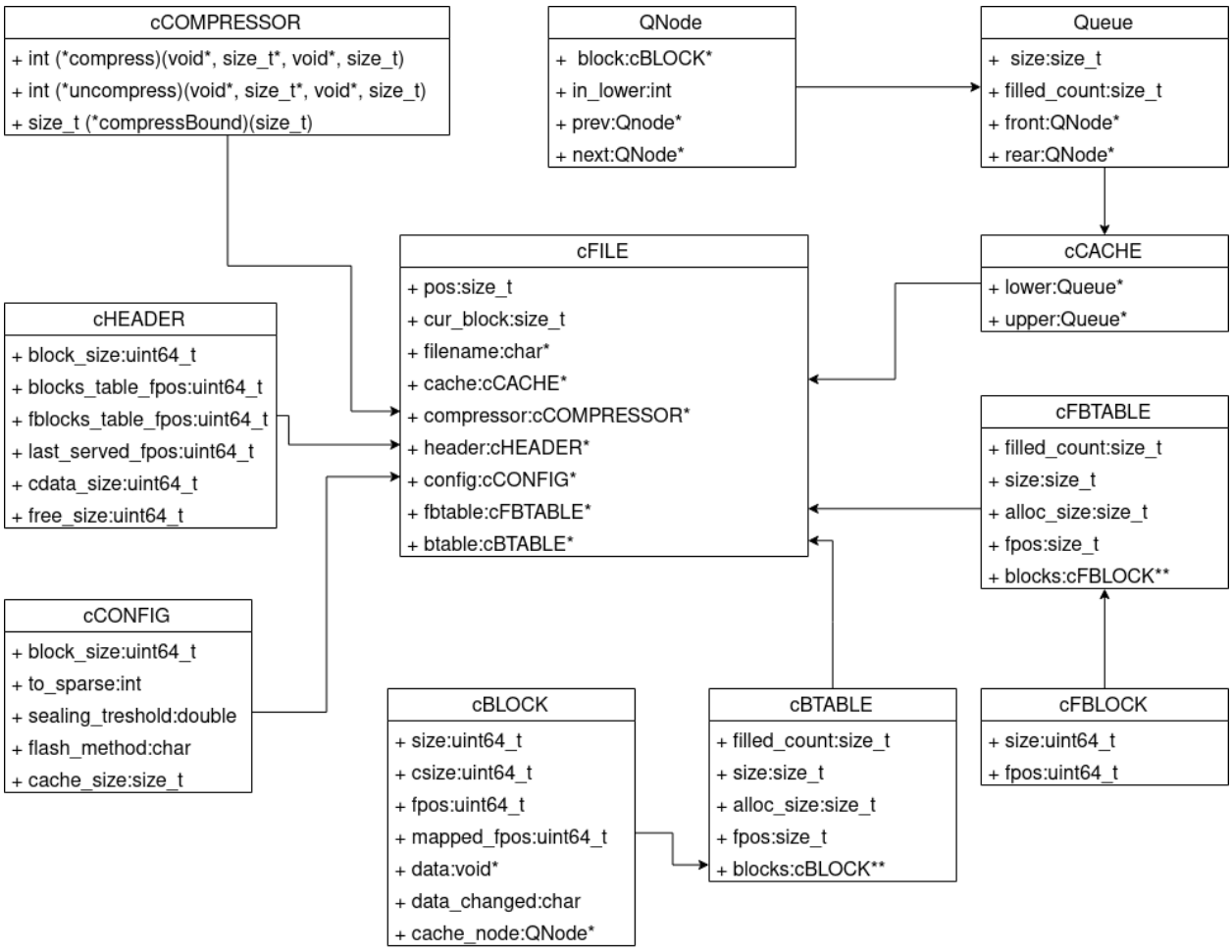

Рис. 4: Схема архитектуры библиотеки

	Поле	Длина, байт	Описание
Заголовок сжатого файла	`block_size`	8	Размер блока несжатых данных, который используется в сжатом файле
	`btable_fpos`	8	Позиция таблицы блоков сжатых данных
	`fbtable_fpos`	8	Позиция таблицы свободных блоков
	`last_served_fpos`	8	Смещение до конца сжатого файла
	`csize`	8	Размер в байтах сжатых данных в сжатом файле
	`free_size`	8	Размер в байтах свободных блоков в сжатом файле
Таблица блоков	`size`	8	Реальный размер таблицы в блоках
	`alloc_size`	8	Виртуальный размер таблицы в блоках
	`fposes`	`size` $\cdot$ 8	Последовательно записанные позиции (размера 8 байт) блоков
Блок сжатых данных	`size`	8	Размер в байтах несжатых данных блока
	`csize`	8	Размер в байтах сжатых данных блока
	`mapped_fpos`	8	Левая граница фрагмента несжатого файла, соответствующая несжатым данным блока
	`cdata`	`csize`	Байты несжатых данных блока
Свободный блок	`size`	8	Размер в байтах фрагмента сжатого файла, обслуживаемый свободным блоком
	`fdata`	`size`	Байты объявленного свободным блоком фрагмента сжатого файла

Таблица 3: Формат сжатого файла.

Таблица 4: Результаты тестирования последовательных чтения и записи

Решение	Данные	Чтение, Мб/сек	Запись, Мб/сек	Сжатие %
`zip`	Текст	154.72 ± 0.23	25.78 ± 0.02	32.25%
	`double`	132.74 ± 0.38	28.25 ± 0.02	6.6%
	`uint64_t`	155.02 ± 0.36	8.65 ± 0.01	38.7%
`compressedio`	Текст	221.092 ± 0.20	45.99 ± 0.06	31.04%
	`double`	193.61 ± 0.33	44.18 ± 0.04	6.2%
	`uint64_t`	240.39 ± 0.57	16.50 ± 0.004	36.2%
`compressedio` (произвольный доступ)	Текст	180.23 ± 0.26	41.65 ± 0.03	21.41%
	`double`	151.08 ± 0.20	39.39 ± 0.03	2.5%
	`uint64_t`	208.97 ± 0.93	15.55 ± 0.006	33.48%