Федеральное агентство по образованию
Государственное образовательное учреждение
высшего профессионального образования
Петрозаводский государственный университет
Математический факультет

Кафедра информатики
и математического обеспечения

Проект программной системы.

"NGFILTER TOOLS"

Описание архитектуры модулей и структуры используемых файлов.

Петрозаводск — 2006

Оглавление

Глава 1

Составные части программной системы.

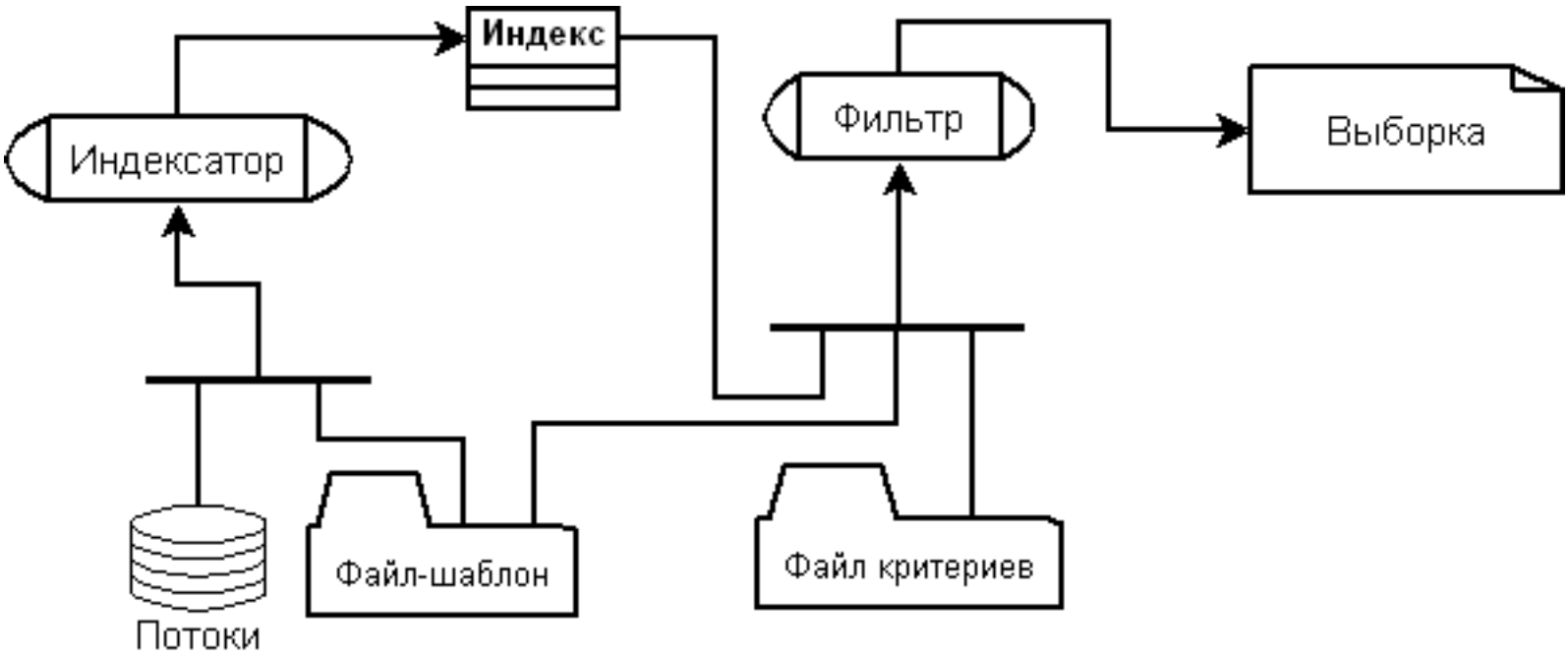

Рис. 1.1: Взаимодействие частей системы.

1.1 Файлы, используемые при работе.

1) Шаблон. Предполагается, что данные о потоках, собранные разными коллекторами, представ-Лены в разных форматах, но содержат сходные данные. (Предопределённые поля NetFlow) Для унификации работы с индексным файлом необходимо представлять данные в некотором внутреннем формате,

не зависящем от представления данных в хранилище. Внутренний формат описывается сущностью "шаблон". Шаблон выполняет три функции:

- установление символьных меток (имён) и длин полей внутреннего представления записи.
- указание типа данных, определяющего преобразования, которые необходимо произвести над данными при их чтении из файла условий и при выводе.
- определение дополнительных индексных баз данных для поиска по конкретным полям.

2) Файл условий.
Определяет условия фильтрации. Поля должны соответствовать полям заданным в шаблоне.

3) Индексные файлы.
Индексный файл является базой данных и созидается с помощью библиотеки BerkeleyDB. Существуют два типа баз данных:

- Первичные - в качестве ключа используется вся запись, представленная во внутреннем формате - как конкатенация всех полей. Первичная база одна. Она содержит все данные о потоках, которые выбраны из хранилища.
- Вторичные - в качестве ключа используется одно поле записи, представленной во внутреннем формате. Вторичные базы неразрывно связаны с первичной. Построение вторичных индексных файлов может дать выигрыш в скорости фильтрации. Для быстрого поиска нужных записей для каждого поля, которое указано в файле, задающем условия поиска, желательно создание вторичных баз данных.

4) Файлы, хранящие записи NetFlow.
Хранилище потоков - набор записей, описывающих потоки и собранных определённым коллектором.

1.2 Описание программ, входящих в состав пакета.

В составе программной системы - две утилиты с консольным интерфейсом:

1) ngindex
Строит индексный файл с использованием хранилища данных и шаблона.

Задача индексатора - с-конвертировать записи из хранилища во внутренний формат и добавить их в индексный файл. Одна из функций индексатора - ориентируясь по символьным меткам полей внутреннего представления записи заполнить поля этой записи данными из хранилища.

Основные функции:

- Доступ к файлам, содержащим записи в формате NetFlow v5 и созданным с помощью утилит из набора flow-tools.
  a) Функция непосредственного извлечения данных из хранилища жёстко привязана к его формату, то есть структуре представления данных в файле.

  b) Функций непосредственного извлечения данных может быть много - для доступа к хранилищам разного формата (должна быть предусмотрена возможность расширения для поддержки других форматов хранения)

  c) Используемая функция непосредственного извлечения данных определяется пользователем указанием опции командной строки.
- Доступ к базе данных в формате BerkeleyDB.
- Чтение шаблона.

2) ngfilter
Осуществляет выборку записей из индексного файла в соответствии с критериями, заданными в файле условий для полей, которые описаны в шаблоне.
Соответствие между полями записи во внутреннем представлении и условиями фильтрации по этим полям также устанавливается с помощью символьных меток.

Основные функции:

- Доступа к базе данных в формате BerkeleyDB.
- Чтения шаблона.
- Чтения условий фильтрации.
- Выборки записей из базы данных в формате BerkeleyDB.
- Вывода записей.

Глава 2

Описание форматов файлов.

2.1 Шаблон.

Файл шаблона потоков является текстовым файлом, задаваемым с помощью соответствующей опции в командной строке индексатора и фильтра и обязательным к указанию при их вызове.
Файл шаблона содержит описание формата внутреннего представления данных о потоках, используемого при работе фильтра и индексатора.

В каждой строке содержится описание полей в следующем формате:

[имя_поля] [тип] [длина_типа] [признак индексируемости]

В качестве разделителей используются символ табуляции и/или пробелы.
Имя поля - символьное представление имени поля, соответствующее спецификации NetFlow (например: IP_DEST).

Тип - тип поля:

- int - для целого числа в десятичном представлении.
- ipv4 - IP-адрес в формате IP v4.
- time - штамп времени.
- prot - тип протокола четвёртого уровня (TCP,UDP...).

Признак индексируемости - слово index в случае если по этому полю необходимо строить вторичную базу данных, иначе пусто.
Также файл может содержать комментарии. Комментарием считается строка начинающаяся с символа '#'.

2.1.1 Пример файлов шаблонов.

```
Пример для всех полей:
      IP_DEST          ipv4      4       index
      IP_SOUR          ipv4      4       index
      NEXT_HOP         ipv4      4
      INPUT            int       2
      OUTPUT           int       2
      D_PACKETS        int       4
      D_OCTETS         int       4
      START_TIME       time      4
      LAST_TIME        time      4
      PORT_DEST        int       4
      PORT_SOUR        int       4
      TCP_FLAGS        int       1
      PROT             prot      1
      TOS              int       1
      AS_SOUR          int       2
      AS_DEST          int       2
      MASK_SOUR        int       1
      MASK_DEST        int       1

Пример для части полей:
      IP_DEST          ipv4      4       index
      IP_SOUR          ipv4      4       index
      PORT_DEST        int       4
      PORT_SOUR        int       4
      START_TIME       time      4
```

2.2 Файл условий фильтрации.

По умолчанию файл ngfilter.conf в каталоге с исполняемым файлом ngfilter.
Файл с условиями поиска является текстовым файлом.
Он содержит названия полей, тип поиска, точные значения для поиска или интервал значений.
Формат файла следующий:

[Название поля] [Тип поиска] [Нижняя граница/Точное значение] [Верхняя граница]

Каждое условие описывается с начала новой строки. В качестве разделителей

внутри строки, описывающей условие, используются табуляция и/или пробелы.

В качестве <Тип поиска> указывается:

- e (или equal)
  - по точному значению, тогда воспринимается только первое из указанных значений
- i (или interval)
  - по нижней и верхней границе значений

Также файл может содержать комментарии. Комментарием считается строка начинающаяся с символа '#'.

2.2.1 Пример содержания файла условий фильтрации.

```
# Some comments here
IP_DEST e 192.168.0.100
IP_SOUR i 10.10.0.0 10.10.255.255
D_PACKETS e 100
```

Глава 3

Описание интерфейсов утилит.

3.1 Интерфейс ngindex.

Индексатор представляет собой консольную программу, которая по заданному списку файлов с информацией о потоках строит базу данных формате BerkeleyDB. Передача параметров происходит через аргументы командной строки.
Должны поддерживаться как длинная, так и короткая форма записи опций. Программа должна поддерживать опцию '--version', которая приводит к печати номера версии программы на стандартный вывод и последующему успешному завершению, и опцию '--help', которая печатает информацию об использовании опций на стандартный вывод и так же успешно завершает выполнение программы. При использовании этих опций не должна выполняться ничего, кроме печати запрошенной информации. При запуске программы без опций должно выполняться тоже действие, что и при запуске программы с опцией '--help'.

Формат вызова программы:
ngindex [опции [параметры]] -t <template> -d <dirname> <filenames>
где filenames - имена файлов с информацией о потоках в формате Netflow.

Список опций программы:

-d, --db-dir {dirname}
Имя директории, в которой будут сохранены файлы базы данных в формате BerkeleyDB. Обязательная опция.
-h, --help
Вывод справки по опциям. Необязательная опция.
-t, --template {template}
Имя файла шаблона с описанием внутренней структуры записи о потоке. Обязательная опция.

-v, --verbose
Режим выдачи подробной информации о работе программы в стандартный поток вывода. Необязательная опция.

-V, --version
Вывод версии программы. Необязательная опция.

-w, --infunc {function name}
Указывается имя функции для разбора исходного файла. Необязательная опция. По умолчанию используется функция для разбора файлов в формате flow-tools, с данными Netflow версии 5.

В случае ошибки в опциях или возникновения ошибки в процессе работы на стандартный поток ошибок происходит вывод кода ошибки и поясняющего сообщения. В случае успешного завершения работы создается директория, содержащая файлы базы данных.

Формат вывода сообщений о версии и помощи по использованию:

1) $ ngindex --version
   ngindex (ngfilter tools) 0.1

2) $ ngindex -h

```
ngindex usage: ngindex [options [parameters]]
               -t <template> -d <dirname> <filenames>

-d, --db-dir  {dirname}       Output database directory
-h, --help                    Print this information
-t, --template {template}     Template file name
-v, --verbose                 Verbose mode
-V, --version                 Print program version
-w, --infunc {function name}  Parse function
```

3.2 Интерфейс ngfilter.

Программа фильтр представляет собой консольную утилиту, которая осуществляет отбор значений из базы данных BerkeleyDB по заданным критериям и выводит набор удовлетворяющих записей. Поддерживаются длинная и короткая формы записи опций. При запуске необходимо обязательно указать каталог, в котором заранее были построены индексные файлы с помощью утилиты ngindex, на основе которых программа ngfilter будет осуществлять поиск необходимых записей, а также указать файл-шаблон, на основе которого программа будет формировать записи.

Формат вызова программы: ngfilter [опции [параметры]] -d <dirname> -t <template>

Список опций программы:

-d, --db-dir {dirname}
Имя директории, из которой будут прочитаны файлы базы данных в формате BerkeleyDB. Обязательная опция.

-t, --template {template}
Имя файла шаблона с описанием внутренней структуры записи о потоке. Обязательная опция.

-V, --version
Вывод версии программы. Необязательная опция.

-h, --help Вывод справки по опциям. Необязательная опция.

-c, --criterion
Указываем имя входного файла с критериями для выборки. По умолчанию программа берёт файл ngfilter.conf из текущего каталога.

-v, --verbose
Режим выдачи подробной информации о работе программы на stdout. Необязательная опция.

-o, --output
Указать файл, в который будет записан результат выборки. По умолчанию результат выводится на stdout. Необязательная опция.

Формат вывода сообщений о версии и подсказки по использованию:

1) $ ngfilter --version
   ngfilter (ngfilter tools) 1.1

2) $ ngfilter -h

```
ngfilter usage: ngfilter [options [parameters]] -t <template> -d <dirname>

-d, --db-dir {dirname}     Database directory
-t, --template {template}  Template file name
-V, --version              Print program version
-h, --help                 Print this information
-c, --criterion {filename} Set manualy file name with search criterion.
                           Default file is ngfilter.conf
-v, --verbose              Show process information. Default it
                           is hidden.
-o, --output {filename}    Set manualy file name where program will
```

output results.

Глава 4

Высокоуровневое описание архитектур утилит.

4.1 Архитектура индексатора.

- Интерфейсная часть:

  -> Вход:
  параметры из командной строки
  <- Выход:
  результат работы

  Функция осуществляет разбор и анализ командной строки, переданной программе при запуске, и проводит проверку корректности параметров.
  В случае, если программа вызвана правильно и параметры верны, управление передается индексирующей части, которой сообщаются необходимые для её работы данные.
  Также к задачам интерфейсной части относится вывод справки и информационного сообщения о программе в случае некорректного запуска или соответствующего запроса.

- Функция чтения и анализа шаблона записи:

  -> Вход:
  имя файла-шаблона
  <- Выход:
  результат работы
  инициализированные структуры описывающие запись и её поля

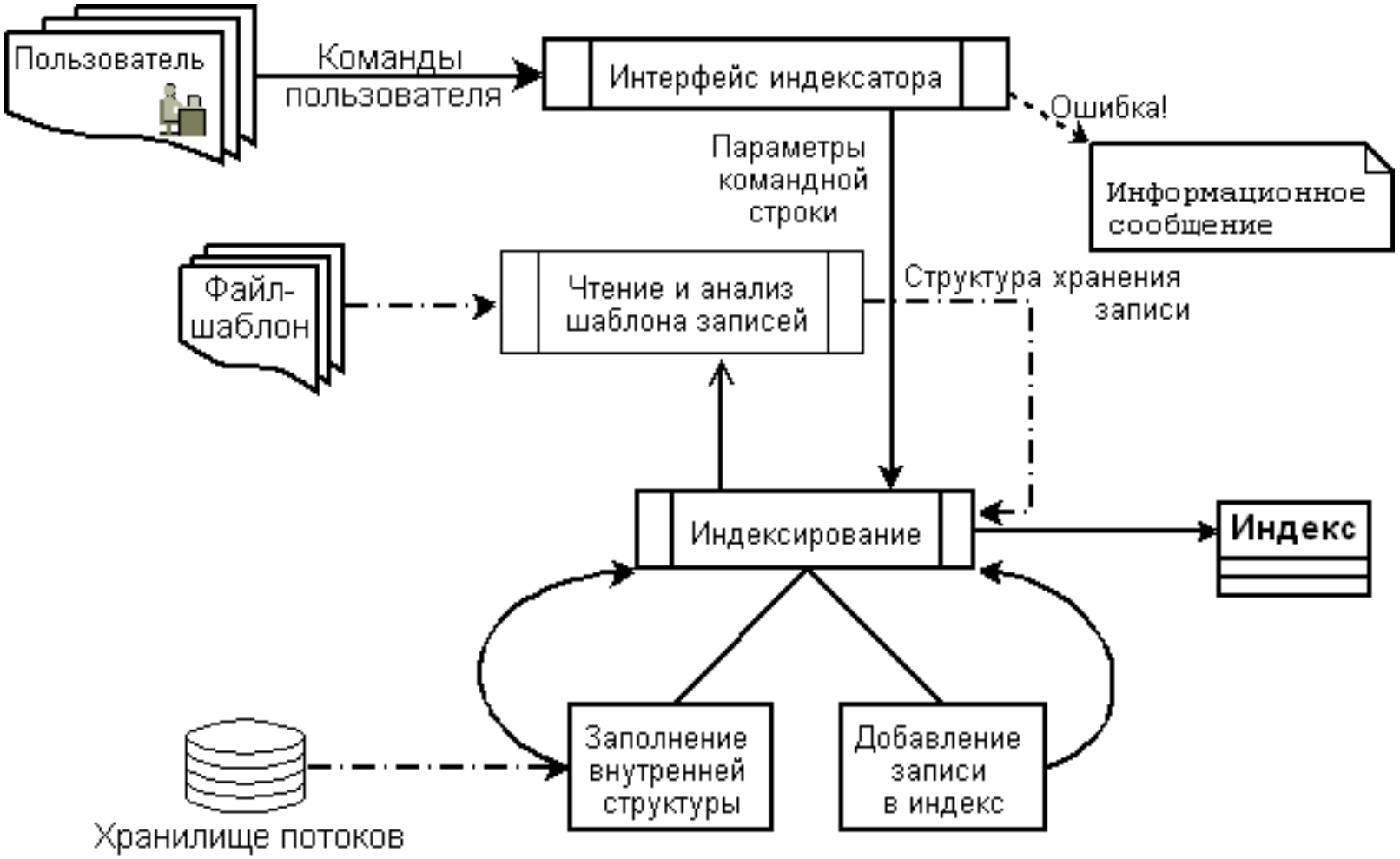

Рис. 4.1: Архитектура индексатора.

Осуществляет открытие и чтение шаблона записи, проверка его корректности.
На основании информации из шаблона строятся внутренние структуры данных, предназначенные для хранения характеристик записи и её полей.

- Функция индексирования:

  -> Вход:
  имена файлов для индексации
  инициализированные структуры, описывающие запись и её поля
  путь к каталогу, в который будет сохранен индексный файл
  идентификатор функции, которую следует использовать для доступа к базе потоков
  <- Выход:
  результат работы

индексный файл

При индексации в цикле последовательно выполняется чтение некоторого блока записей и сохранение их в индексе. Из основной программы выполняется запуск этих функций и передача заполненных структур между ними. Данные индексируются средствами библиотеки BerkeleyDB.

- Функция открытия индексных файлов:

  -> Вход:
  путь к каталогу, в который будет сохранен индексный файл
  идентификатор способа доступа к индексным файлам
  инициализированные структуры описывающие поля записи
  <- Выход:
  результат работы
  созданные индексные файлы
  инициализированные структуры, описывающие индексные файлы

  Функция открывает индексные файлы с заданными атрибутами доступа.

- Функция закрытия индексных файлов:

  -> Вход:
  структуры BerkeleyDB, описывающие индексные файлы
  <- Выход:
  результат работы

  Функция закрывает индексные файлы.

- Функции заполнения внутренней структуры из файла, содержащего записи о потоках:

  -> Вход:
  идентификатор текущего файла с потоками
  массив структур хранения записи
  <- Выход:
  результат работы (количество прочитанных записей или индикатор ошибки)
  заполненный массив структур

  Данная функция привязана к формату хранения данных о потоках в базе. Она последовательно извлекает из текущей позиции в файле данные о записях и заполняет ими массив внутренних структур. Для со-

ответствия функциональному требованию F7, возможно использование исходного кода пакета flow-tools.

– Функция добавления записи в индекс:

-> Вход:
структуры BerkeleyDB, описывающие индексные файлы
массив заполненных структур
количество структур для записи
<- Выход:
результат работы (количество добавленных записей или индикатор ошибки)

Заполненные структуры добавляются к существующему индексу.

4.2 Архитектура фильтра.

- Интерфейсная часть:
-> Вход:
параметры из командной строки
<- Выход:
результат работы

Функция осуществляющая разбор и анализ параметров командной строки, переданных программе. В этой функции проводится проверка корректности параметров и вызова программы. В случае, если программа вызвана правильно и параметры верны, управление передается фильтру с необходимыми входными данными. Также к задачам интерфейсной части относится вывод справки и сообщения о ошибке в случае некорректного запуска или запроса.

- Функция чтения и анализа шаблона записи:

-> Вход:
имя файла-шаблона
<- Выход:
результат работы
инициализированные структуры описывающие запись и её поля

Осуществляет открытие и чтение шаблона записи, проверка его корректности.
На основании информации из шаблона строятся внутренние структуры данных, предназначенные для хранения характеристик записи и её полей.

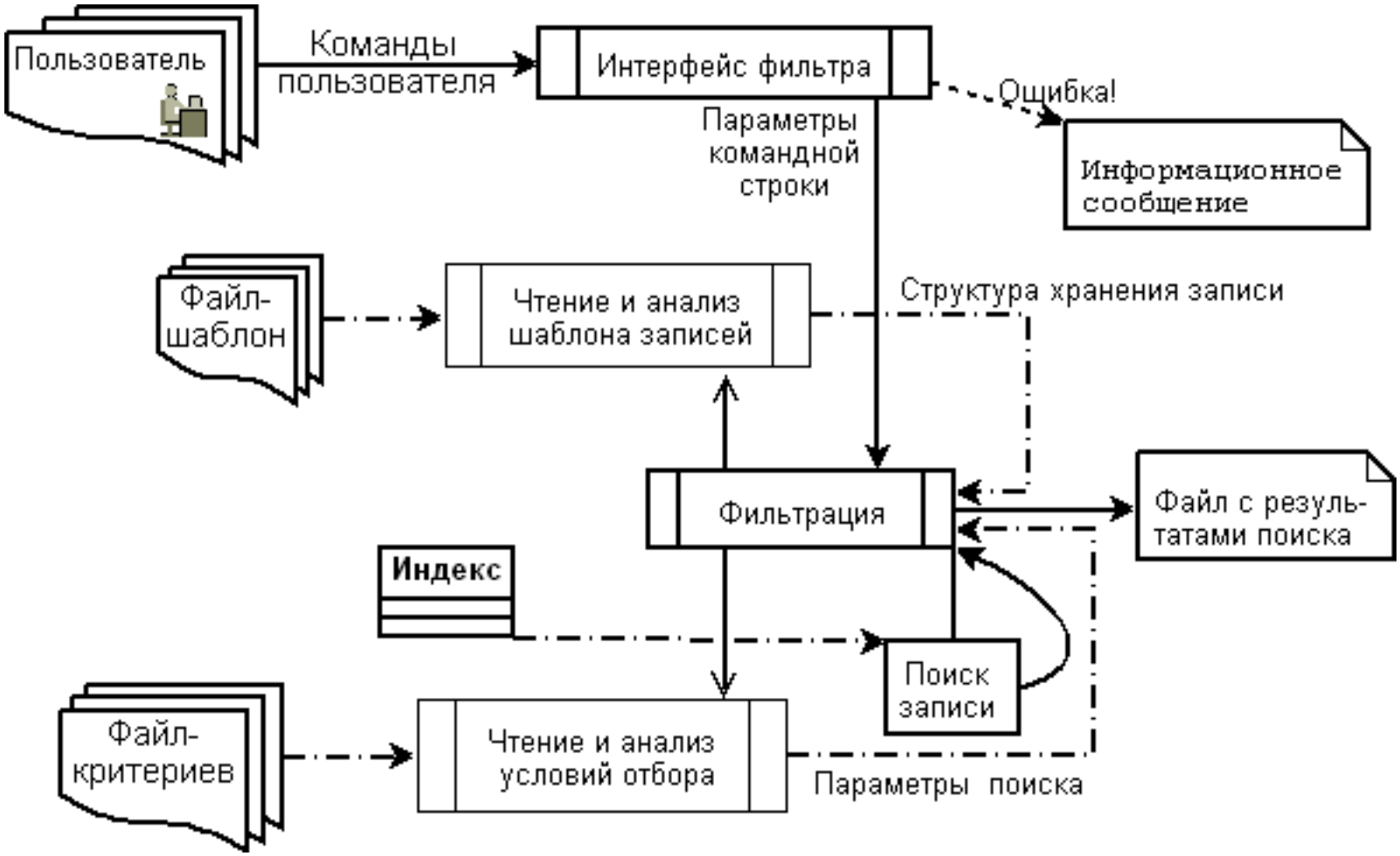

Рис. 4.2: Архитектура фильтра.

- Функция чтения и анализа условий отбора:
  -> Вход:
  имя файла-критериев
  <- Выход:
  результат работы
  заполненная структура критериев для сравнения записей

  Осуществляет открытие и чтение файла-критериев. Проверка его корректности. На основании полученной информации получается структура для сравнения записей.

- Фильтр:
  -> Вход:
  имя директории, содержащей индексные файлы
  заполненная структура критериев для сравнения записей

инициализированные структуры описывающие запись и её поля
<- Выход:
результат работы
выборка записей

При фильтрации происходит разбор индекса по условиям, заданным в файле, и вывод найденных данных, удовлетворяющих критериям. Условие может точно идентифицировать запись, не по всем полям точными значениями и по подмножеству полей. В этой функции реализуется поиск записей в базе потоков по индексному файлу. Поиск производится согласно условиям, заданным в файле критериев.

- Функция открытия индексных файлов:
  -> Вход:
  путь к каталогу, в который будет сохранен индексный файл
  идентификатор способа доступа к индексным файлам
  инициализированные структуры описывающие поля записи
  <- Выход:
  результат работы
  созданные индексные файлы
  инициализированные структуры, описывающие индексные файлы

  Функция открывает индексные файлы с заданными атрибутами доступа.
- Функция закрытия индексных файлов:
  -> Вход:
  структуры BerkeleyDB, описывающие индексные файлы
  <- Выход:
  результат работы

  Функция закрывает индексные файлы.
- Функция чтения блока записей, удовлетворяющих условиям:
  -> Вход:
  массив структур представления записи
  структуры BerkeleyDB, описывающие индексные файлы
  <- Выход:
  результат работы (количество прочитанных записей)
  заполненный массив структур представления записи

Функция осуществляет чтение блока данных из индексного файла и заполняет ими массив записей.

– Функция вывода записей, удовлетворяющих условиям:
-> Вход:
заполненный массив структур представления записи
количество записей в массиве
<- Выход:
результат работы

Функция производит вывод в файл всех найденных данных по мере их обнаружения согласно заданным критериям.

Глава 5

Описание переменных.

Переменные, используемые при работе программы.

5.1 Структура, описывающая представление записи.

```
struct record_description
  {
    int attr_count; /* Количество полей в записи. */
    int lenght;        /* Длина записи в байтах. */
  } g_rec_description;
```

Доступ к полям структуры осуществляется посредством специализированных функций.

```
int get_rec_attr_count(void);
  /* Возвращает число полей записи. */
int get_rec_length(void);
  /* Возвращает длину записи в байтах. */

int set_rec_description(int attr_count, int lenght);
  /*
   * Устанавливает соответствующие характеристики записи.
   * Используется только при чтении шаблона.
   */
```

5.2 Внутреннее представление записи.

Записи передаются и обрабатываются блоками фиксированной длины. Каждая запись в блоке - массив unsigned char. Каждое поле записи представляет собой по-

следовательность байт. Т.е. определённые подмножества байт интерпретируются как поля записи.

```
unsigned char ** block_of_records;
  /* блок записей для их передачи между функциями */

#DEFINE BLOCK_REC_COUNT <number>
  /* максимальное число число записей в блоке */
```

5.3 Описание полей записи.

Массив структур, описывающих каждое поле записи.

```
struct attribute_description
  {
    char *symbolic_name;
      /* символическое название поля */

    int offset;
      /* смещение поля внутри записи */

    short int type;
      /*
       *   константа, определяющая формат чтения данных из
       *   файла условий фильтрации и их вывод.
       */

    int lenght;
      /* длина поля в байтах */

    char secondary_db_build_flag;
      /*
       * Флаг, указывающий на необходимость создания
       * вторичного индекса по данному атрибуту.
       */
  } g_attr_description[ ];
```

Массив объявлен глобально. Доступ к структурам осуществляется с помощью функций - по имени поля или по его порядковому номеру.

```
int get_attr_num_by_name(const char * symbolic_name); - Возвращает номер структуры в
struct attr_description * get_attr_by_num(const int attr_num); - Возвращает указател
```

Добавление структуры в массив осуществляется с помощью следующей функции.

```
int add_rec_to_array(struct attr_description * next_attr_description)
```

Массив структур, описывающих переменные для доступа ко вторичным базам данных.

```
struct attribute_db
  {
    DB * secondary_db_handle;
      /*
       * указатель на вторичную базу данных для данного
       * атрибута, если её нет = NULL
       */

    DBC * secondary_db_cursor;
      /* курсор для обхода вторичного индекса */
  } attr_db[ ];
```

5.4 Общая структура описания базы данных для обеих утилит.

```
struct db_description
  {
    DB * primary_handle;
      /* указатель на первичную базу данных */

    DBC * primary_cursor;
      /* курсор для обхода первичного индекса */

    DBC * joined_cursor;
      /* объединённый курсор для обхода нескольких индексов */

    DBC ** cursors_joining_array;
      /* Массив курсоров для их объединения. */
  };
```

5.5 Переменные, определяющие условия фильтрации.

```
struct search_param
  {
    char *search_flags;
      /* Массив флагов, показывающий по каким полям заданы условия. */

    unsigned char * search_low_border;
      /* Нижняя граница для полей при поиске. */

    unsigned char * search_high_border;
      /* Верхняя граница для полей при поиске. */
  } search_parameters;
```

В файле с условиями поиска могут быть указаны условия не для всех полей, поэтому необходимо ввести массив, содержащий индикаторы поиска по каждому полю. Если поиск должен производится по точному заданному значению поля, то соответствующие этому полю части строк search_low_border и search_high_border будут равны. Оба массива имеют структуру, полностью аналогичную записи о потоке.

5.6 Флаг подробного вывода действий.

```
int g_verbose;
```

Определяется глобально. Значение устанавливается при анализе командной строки.

Глава 6

Описание архитектуры функций индексатора.

Функции, осуществляющие чтение записей из хранилища и их добавление в индексные файлы возвращают число обработанных записей или отрицательное значение в случае ошибки.

Все остальные функции, равно как и сама программа при своём завершении возвращают число, заданное следующими определёнными в заголовочном файле stdlib.h константами:

EXIT_SUCCESS при успешном завершении.

EXIT_FAILURE при ошибке в работе.

6.1 Интерфейсная часть.

Роль интерфейсной части выполняет функция, осуществляющая разбор и анализ командной строки переданной программе при запуске. Данная функция располагается в основной подпрограмме (main{}) в файле ngindex.c.

Необходимые для работы заголовочные файлы:

```
#include <stdlib.h>
#include <unistd.h>
#include <getopt.h>
#include <stdio.h>
```

6.1.1 Описание функции.

Получаемые параметры:
количество аргументов, содержащееся в командной строке, переданной программе

при вызове. (int argc)
массив, содержащий указанные аргументы. (char **argv)

Прототип функции:

```
int main (int argc,    /* количество аргументов командной строки */
          char **argv) /* массив аргументов командной строки */
```

Описание используемых функций и переменных, которые описаны в библиотеках операционной системы unix:

Используется следующая функция разбора командной строки, определённая в заголовочном файле getopt.h, также необходимо подключение unistd.h

```
int getopt_long (int argc,
                 /* количество аргументов командной строки */

                 char * const argv[],
                 /* массив аргументов командной строки */

                 const char *optstring,
                 /* строка, определяющая короткие опции */

                 const struct option *longopts,
                 /* строка, определяющая длинные опции */

                 int *longindex);
                 /* номер распознанной длинной опции */
```

Детальное описание всех переменных и функции содержится на странице справочного руководства в unix системах. (man 3 getopt, man 3 getopt_long).

6.1.2 Схема работы функции:

- Задаются параметры функции getopt_long ():
  optstring
  longopts

- В цикле производится выбор аргументов с их параметрами из командной строки.

- С помощью оператора switch происходит установка переменных, необходимых для функции индексации или определяются иные действия программы.

a) Если параметров нет - действия программы аналогичны действиям при задании опции --help
b) Если задана опция --help - производится вывод подсказки и независимо от наличия других опций программа завершает работу.
Возвращаемое значение EXIT_SUCCESS
c) Если задана опция --version - производится вывод информации о версии программы, независимо от наличия других опций.
Возвращаемое значение EXIT_SUCCESS
d) Если обнаружена опция, которая не определена, или отсутствуют необходимые параметры опции, программа завершается с выводом поясняющего сообщения в стандартный поток ошибок.
Возвращаемое значение EXIT_FAILURE
e) После первого встреченного параметра, не являющегося опцией разбор прекращается. Оставшаяся после разбора часть командной строки будет передана функции индексирования, в предположении, что в нем содержатся имена файлов, которые следует обработать.

- Вызов функции чтения шаблона.
- Проверка значения, возвращаемого функцией чтения шаблона.

  a) При ошибочном значении программа завершает работу с выводом поясняющего сообщения в стандартный поток ошибок.
  Возвращаемое значение EXIT_FAILURE

- Вызов функции, индексирующей данные.
- Проверка значения, возвращаемого функцией, индексирующей данные.

  a) При ошибочном значении программа завершает работу с выводом поясняющего сообщения в стандартный поток ошибок.
  Возвращаемое значение EXIT_FAILURE

- Завершение работы программы.
  Возвращаемое значение EXIT_SUCCESS

6.2 Функция чтения файла-шаблона.

Описана отдельно как функция, используемая в обеих программах.

6.3 Функция индексирования.

Функция индексирования осуществляет передачу прочитанных из хранилища потоков записей функции, добавляющей их в индексный файл.
Название функции: build_index(), файл build_index.c

Необходимые для работы заголовочные файлы:

```
#include <stdlib.h>
#include <stdio.h>
```

6.3.1 Описание функции:

Получаемые параметры:
массив строк - имён файлов, содержащих записи о потоках. (char **flow_files)
путь к каталогу, в который будет сохранен индексный файл. (char * path_to_index_dir)
ссылка на функцию, которую следует использовать для доступа к файлу, содержащему записи о потоках.

```
(int (*raw_data_access_func)
     (unsigned char ** block_of_records,
        /* блок записей, в который будут считаны записи */

      int fd_raw_db_file,
        /* файловый дескриптор обрабатываемого файла */

      void * internal_data))
        /*
         * указатель, на данные, которые будут сохраняться
         * между вызовами функции доступа к хранилищу
         * данных
         * при первом обращении к функции должен быть установлен в NULL
         */
```

Прототип функции:

```
int build_index(char **flow_files,
                char * path_to_index_dir,
                int (*raw_data_access_func) (unsigned char **, int, void *));
```

6.3.2 Схема работы функции:

- Вызов функции создания баз данных.
- Создание массива структур хранения записей. (unsigned char ** block_of_records)
- Цикл по всем обрабатываемым файлам.
  а) Цикл пока не достигнут конец текущего файла.
  - Вызов функции, осуществляющей чтение блока данных из файла.
  - Проверка значения, возвращаемого функцией чтения данных.
  - При ошибочном значении программа завершает работу с выводом поясняющего сообщения в стандартный поток ошибок.
    Возвращаемое значение EXIT_FAILURE
  - Считанный блок данных добавляется в индексный файл с помощью соответствующей функции.
  - Проверка значения, возвращаемого функцией добавления данных в индекс.
  - При ошибочном значении программа завершает работу с выводом поясняющего сообщения в стандартный поток ошибок.
    Возвращаемое значение EXIT_FAILURE
- Вызов функции закрытия используемых баз данных.
- Возврат значения EXIT_SUCCESS

6.4 Функция открытия используемых баз данных.

Описана отдельно как функция, используемая в обеих программах.

6.5 Функция закрытия используемых баз данных.

Описана отдельно как функция, используемая в обеих программах.

6.6 Функция добавления записи в индексный файл.

```
 int add_records_to_db(struct db_description *db_descr, unsigned char **records_bloc
Функция для добавления массива записей в базу данных
Получаемые параметры:
  указатель на заполненную структуру, описывающую переменные, связанные с первичной
  указатель на заполненную структуру записи (формат, описан в файле
  шаблона).
  общее число добавляемых записей.
При успешном завершении функция возвращает количество добавленных
в базу данных записей.
При неудачном завершении функция возвращает число -1.

Схема работы функции:
int
add_records_to_db(struct db_description *db_descr, char **records_block, const int tot
{
        Функция добавляет записи только в первичную БД, BerkeleyDB обновляет
        вторичные базы сама, поэтому нам не надо добавлять запись во вторичные
        базы данных.

        Цикл по количеству записей в блоке
        {
                Инициализация ключа и данных (структуры типа DBT).

                Добавляем запись в БД методом put, если не удалось добавить -
                возвращаем -1.
        }

        Возвращаем количество успешно добавленных записей.
}
```

6.7 Функция чтения записей NetFlow v.5 из файла в формате FLOW-TOOLS.

Данная функция осуществляет чтение и интерпретацию обрабатываемого файла.
Название: netflow_v5_flow_tools_access()
Описана в файле: flow_tools_netflow_v5/access_main.c

Необходимые для работы заголовочные файлы:
необходимость подключения тех или иных системных заголовочных файлов будет выяснена в процессе разработки.

```
flow_tools_netflow_v5/structures.h
  содержит описания структур, перенесенных из исходного кода пакета flow-tools, а
  также структуры data_file_description (см. ниже)
flow_tools_netflow_v5/defines.h
  содержит определения (DEFINE), перенесенных из исходного кода пакета flow-tools
```

6.7.1 Описание функции:

Получаемые параметры:
Блок записей, в который будут считаны записи - unsigned char ** block_of_records
Файловый дескриптор обрабатываемого файла - int fd_raw_db_file
Указатель, на данные, которые будут сохраняться между вызовами функции доступа к хранилищу данных - void * internal_data. Эти данные передаются в следующей структуре:

```
struct data_file_description
  {
    struct ftio * in_file_descr;
      /*
       * Структура, описывающая файл с данными.
       * Перенесена из исходного кода
       * пакета flow-tools.
       */
    int * offset_of_attr_in_ft_struct;
      /*
       * Массив, в котором каждый элемент содержит
       * смещение поля в статической структуре, описывающей
       * запись о потоке в формате NetFlow v5. Данная
       * структура в точности скопирована из исходного кода
```

```
            * пакета flow-tools.
            * Соответствие поля во внутреннем представлении и
            * данного смещения устанавливается по порядковому номеру.
            */
        }
```

Прототип функции:

```
int netflow_v5_flow_tools_access (unsigend char ** block_of_records,
                                  int fd_raw_db_file,
                                  void * internal_data))
```

6.7.2 Схема работы функции:

- Если в качестве параметра передан новый файл.

  a) Вызов функции формирования структуры, описывающей файл с данными
  Проверка возвращённого значения. В случае ошибки - возвращение значения 0.

  b) Вызов функции, устанавливающей соответствие полей внутреннего представления данных и полей статической структуры, описывающей внешние данные.
  Проверка возвращённого значения. В случае ошибки - возвращение значения -1.

- Цикл по числу записей в блоке с проверкой достижения конца файла на каждой итерации.

  a) Вызов функции чтения одной записи из файла.
  Проверка возвращённого значения. В случае ошибки - возвращение значения -1.

  b) Копирование полей записи из структуры, принятой во flow-tools для её хранения, во внутреннее представление записи и сохранение её в блоке.
  Проверка возвращённого значения. В случае ошибки - возвращение значения -1.

- Возврат счётчика цикла чтения записей - числа прочитанных записей.

6.8 Функция формирования структуры, описывающей файл с данными.

Данная функция формирует структуру, описывающую обрабатываемый файл.
Название: ftio_init()
Описана в файле: flow_tools_netflow_v5/ftio_init.c

Функция перенесена из пакета flow-tools. Удалена поддержка различного порядка байт, записей в формате отличном от NetFlow v5 и возможность создания структуры, предназначенной для записи. (В данном контексте необходимо только чтение.)

Необходимые для работы заголовочные файлы:

```
необходимость подключения тех или иных системных заголовочных файлов будет
  выяснена в процессе разработки.
flow_tools_netflow_v5/structures.h
  содержит описания структур, перенесенных из исходного кода пакета flow-tools, а
  также структуры data\_file\_description (см. ниже)
flow_tools_netflow_v5/defines.h
  содержит определения (DEFINE), перенесенных из исходного кода пакета flow-tools
```

6.8.1 Описание функции:

Получаемые параметры: Указатель на структуру, описывающую файла в формате flow-tools - struct * ftiheader
Файловый дескриптор обрабатываемого файла - int fd_flow_file

Прототип функции:

```
int ftio_init ( struct ftio *ftio, int fd_flow_file)
```

6.8.2 Схема работы функции:

- Получение информации о файле с помощью системного вызова fstat и копирование её в структуру, описывающую обрабатываемый файл.

  Проверка возвращённого значения. В случае ошибки - возвращение значения - завершение работы exit(EXIT_FAILURE).

- Вызов mmap().

Проверка возвращённого значения. В случае ошибки - возвращение значения - завершение работы exit(EXIT_FAILURE).

- Вызов функции чтения заголовка.

  Проверка значения, возвращаемого функцией чтения заголовка. Если формат не распознан или заголовок повреждён. Возвращаемое значение - EXIT_FAILURE.

- Инициализация zlib.

  Проверка значения, возвращаемого функцией. В случае ошибки - завершение работы exit(EXIT_FAILURE).

- Возвращает значение EXIT_SUCCESS.

6.9 Функция чтения заголовка файла в формате flow-tools.

Данная функция осуществляет чтение и интерпретацию обрабатываемого файла. Название: read_header() Описана в файле: flow_tools_netflow_v5/read_header.c

Функция перенесена из пакета flow-tools. Удалена поддержка различного порядка байт, записей в формате отличном от NetFlow v5.

Необходимые для работы заголовочные файлы:
необходимость подключения тех или иных системных заголовочных файлов будет выяснена в процессе разработки.
flow_tools_netflow_v5/structures.h содержит описания структур, перенесённых из исходного кода пакета flow-tools, а также структуры data_file_description (см. ниже)
flow_tools_netflow_v5/defines.h содержит определения (DEFINE), перенесённых из исходного кода пакета flow-tools

6.9.1 Описание функции:

Получаемые параметры: Указатель на структуру, описывающую заголовок файла в формате flow-tools. struct * ftiheader Файловый дескриптор обрабатываемого файла. int fd_flow_file

Прототип функции:

```
int read_header ( int fd_flow_file,
                  struct ftiheader *internal_header )
```

6.9.2 Схема работы функции:

- Попытка чтения части заголовка, одинаковой для обоих версий файлов (stream version 1,3).

- Проверка значения, возвращаемого функцией чтения заголовка.

  Если формат не распознан - завершение работы. Возвращаемое значение - EXIT_FAILURE.

- Если версия файла - 1.

  Чтение заголовка версии 1 и заполняющей поля внутренней структуры.

  Если возникли ошибки при чтении файла или заголовок повреждён. Возвращаемое значение - EXIT_FAILURE.

- Если версия файла - 3.

  Чтение заголовка версии 3 и заполняющей поля внутренней структуры.

  Если возникли ошибки при чтении файла или заголовок повреждён. Возвращаемое значение - EXIT_FAILURE.

- Возврат значения EXIT_SUCCESS.

6.10 Функция установления соответствия полей.

Функция, устанавливающая соответствие полей внутреннего представления данных и полей статической структуры, описывающей запись в формате flow-tools. Название: set_correspond() Описана в файле: flow_tools_netflow_v5/set_correspond.c

Необходимые для работы заголовочные файлы: необходимость подключения тех или иных системных заголовочных файлов будет выяснена в процессе разработки. flow_tools_netflow_v5/structures.h содержит описания структур, перенесённых из исходного кода пакета flow-tools, а также структуры data_file_description (см. ниже) flow_tools_netflow_v5/defines.h содержит определения (DEFINE), перенесённые из исходного кода пакета flow-tools

6.10.1 Описание функции:

Получаемые параметры: Массив, в котором каждый элемент содержит смещение поля в статической структуре, описывающей запись о потоке в формате NetFlow v5. Данная структура в точности скопирована из исходного кода пакета flow-tools. Соответствие поля во внутреннем представлении и данного смещения устанавливается по порядковому номеру.
int * offset_of_attr_in_ft_struct;

Прототип функции:

```
int set_correspond(int * offset_of_attr_in_ft_struct)
```

6.10.2 Схема работы функции:

- Выделение памяти под массив.
- Цикл по числу полей в записи

  Определение соответствующего смещения в структуре из flow-tools для очередного поля в массиве структур, описывающих поля записи, по символьной метке.

- Возврат значения EXIT_SUCCESS.

6.11 Функция чтения записи из файла.

Название: ftio_read() Описана в файле: flow_tools_netflow_v5/ftio_read.c

Функция перенесена из пакета flow-tools. Удалена поддержка различного порядка байт, записей в формате отличном от NetFlow v5.

Необходимые для работы заголовочные файлы: необходимость подключения тех или иных системных заголовочных файлов будет выяснена в процессе разработки.
flow_tools_netflow_v5/structures.h
содержит описания структур, перенесённых из исходного кода пакета flow-tools, а также структуры data_file_description (см. ниже)
flow_tools_netflow_v5/defines.h
содержит определения (DEFINE), перенесённых из исходного кода пакета flow-tools

6.11.1 Описание функции:

Получаемые параметры: Указатель на структуру, описывающую файл в формате flow-tools - struct ftio *ftio
Возвращаемое значение - указатель на прочитанную структуру.

Прототип функции:

```
void * ftio_read ( struct ftio *ftio )
```

6.11.2 Схема работы функции:

- Если достигнут конец файла, возврат NULL.

- Чтение данных.

- Возврат указателя на начало очередной записи в блоке прочитанных данных.

Глава 7

Описание архитектуры функций фильтра.

Функции, осуществляющие чтение записей из индексных файлов и вывод записей возвращают число обработанных записей или отрицательное значение в случае ошибки.
Все остальные функции, равно как и сама программа при своем завершении возвращают число, заданное следующими определёнными в заголовочном файле stdlib.h константами:
EXIT_SUCCESS при успешном завершении.
EXIT_FAILURE при ошибке в работе.

7.1 Интерфейсная часть.

Роль интерфейсной части выполняет функция, осуществляющая разбор и анализ командной строки переданной программе при запуске. Данная функция располагается в основной подпрограмме (main) в файле ngfilter.c.

Необходимые для работы заголовочные файлы:

```
#include <stdlib.h>
#include <unistd.h>
#include <getopt.h>
```

7.1.1 Описание функции:

Получаемые параметры:
количество аргументов, содержащееся в командной строке, переданной программе

при вызове. (int argc)
массив, содержащий указанные аргументы. (char **argv)

Прототип функции:

```
int main (int argc,     /* количество аргументов командной строки */
          char **argv)  /* массив аргументов командной строки */
```

Описание используемых функций и переменных, которые описаны в библиотеках операционной системы unix:

Используется следующая функция разбора командной строки, определённая в заголовочном файле getopt.h, также необходимо подключение unistd.h

```
int getopt_long (int argc,
                 /* количество аргументов командной строки */

                 char * const argv[],
                 /* массив аргументов командной строки */

                 const char *optstring,
                 /* строка, определяющая короткие опции */

                 const struct option *longopts,
                 /* строка, определяющая длинные опции */

                 int *longindex);
                 /* номер распознанной длинной опции */
```

Детальное описание всех переменных и функции содержится на странице справочного руководства в unix системах. (man 3 getopt, man 3 getopt_long)

7.1.2 Схема работы функции:

- Задаются параметры функции getopt_long ():
  optstring
  longopts

- В цикле производится выбор аргументов с их параметрами из командной строки.

- С помощью оператора switch происходит установка переменных, необходимых для функции индексации или определяются иные действия программы.

a) Если параметров нет - действия программы аналогичны действиям при задании опции --help
b) Если задана опция --help - производится вывод подсказки и независимо от наличия других опций программа завершает свою работу.
Возвращаемое значение EXIT_SUCCESS
c) Если задана опция --version - производится вывод информации о версии программы, независимо от наличия других опций.
Возвращаемое значение EXIT_SUCCESS
d) Если обнаружена опция, которая не определена, или отсутствуют необходимые параметры опции, программа завершается с выводом поясняющего сообщения в стандартный поток ошибок.
Возвращаемое значение EXIT_FAILURE

- Вызов функции чтения шаблона.
- Проверка значения, возвращаемого функцией чтения шаблона.

  a) При ошибочном значении программа завершает работу с выводом поясняющего сообщения в стандартный поток ошибок.
  Возвращаемое значение EXIT_FAILURE
- Вызов функции чтения файла условий.
- Проверка значения, возвращаемого функцией чтения файла условий.

  a) При ошибочном значении программа завершает работу с выводом поясняющего сообщения в стандартный поток ошибок.
  Возвращаемое значение EXIT_FAILURE
- Открытие файла, в который будет осуществляться вывод.
- Вызов функции, фильтрующей данные.
- Проверка значения, возвращаемого функцией, фильтрующей данные.

  a) При ошибочном значении программа завершает работу с выводом поясняющего сообщения в стандартный поток ошибок.
  Возвращаемое значение EXIT_FAILURE
- Завершение работы программы.
  Возвращаемое значение EXIT_SUCCESS

7.2 Функция чтения файла-шаблона.

Описана отдельно как функция, используемая в обеих программах.

7.3 Функция чтения файла условий фильтрации.

Название функции: process_conditions(), файл process_conditions.c

Необходимые для работы заголовочные файлы:

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
```

7.3.1 Описание функции:

Получаемые параметры:
строка, содержащая путь к файлу, описывающему условия фильтрации.
char *path_to_conditions

Прототип функции:

```
int process_conditions ( const char *path_to_conditions)
```

Для открытия файла используется функция:

```
FILE *fopen(const char *path,   /* путь к файлу */
            const char *mode);  /* режим, в котором следует открыть файл */
```

Возвращаемое значение - указатель на файловый поток.

Для чтения строки из файла используется функция:

```
char *fgets(char *string,  /* переменная, в которую будет прочитана строки */
            int size,      /*
                            * максимально возможное число символов, которое
                            * можно считать в строку
                            */
            FILE *stream); /* файловый поток, из которого осуществляется чтение */
```

Для разбора строки файла на лексемы используется следующая функция:

```
char *strtok(char *string,        /* анализируемая строка */
             const char *delim);  /* разделитель лексем */
```

В качестве разделителя лексем используются: табуляция, пробел, конец строки '\t\n '.
Возвращаемое значение - очередная лексема.

7.3.2 Схема работы функции:

- Производится попытка открытия файла, содержащего условия.

  a) В случае ошибки - завершение работы функции.
  Возвращаемое значение: EXIT_FAILURE

- Инициализация переменной conditions_count - счётчика заданных условий и переменной conditions_token - разделителя лексем в строке условия (табуляция, конец строки и пробел).

- В цикле производится построчное чтение файла условий. При чтении одной строки файла условий находится соответствующее поле записи и в переменные, описывающие границы(условия) фильтрации, с необходимыми смещениями записываются заданные значения.

  a) Строка, содержащая признак комментария - первый символ "#", пропускается.

  b) Каждая прочитанная строка разбивается на лексемы. Обработка лексемы зависит от места её появления.

  - Считывается название поля, проверяется на корректность. Если поля с таким названием нет, то завершение работы с возвращаемым значением EXIT_FAILURE. По названию поля определяется номер структуры, описывающей его с помощью функции get_attr_num_by_name (char *symbolic_name).
  - Считывается тип поиска, проверяется на корректность значения (может принимать только значения e, equal, i, interval. Если не задано корректного значения, то работа функции завершается с возвращаемым значением EXIT_FAILURE.
  - Если условие задается на точное совпадение, считывается одно значение, на диапазон - два.

– От строки берется часть, которая не содержит символа '.' (с помощью strtok). Затем делается попытка аналогичным образом отделить ещё одну часть этой же строки. Если вторая строка пуста, то указанное значение не имеет символов '.', то есть указано обычное число. Иначе, указанное значение является IP адресом. В таком случае отделяются оставшиеся 2 его части и числа, разделённые точками интерпретируются и записываются в массивы как 4 символа.

с) Если в строке обнаружены лишние лексемы - завершение работы.
Возвращаемое значение: EXIT_FAILURE

- Завершение работы функции.
  Возвращаемое значение: EXIT_SUCCESS

7.4 Функция фильтрации.

Функция фильтрации осуществляет передачу прочитанных из индексного файла записей, которые удовлетворяют условиям фильтрации, функции, выводящей эти записи в файл.
Название функции: filtrate(), файл filtrate.c

Необходимые для работы заголовочные файлы:

```
#include <stdlib.h>
#include <stdio.h>
```

7.4.1 Описание функции:

Получаемые параметры:
путь к каталогу, который содержит индексный файл. (char * path_to_index_dir)
файловый дескриптор, в который будет осуществлён вывод записей. (int fd_output)

Прототип функции:

```
int filtrate(char * path_to_index_dir,
             int fd_output);
```

7.4.2 Схема работы функции:

- Вызов функции открытия баз данных.

- Создание массива структур хранения записей. (unsigned char ** block_of_records)
- Цикл до тех пор пока не выведены все удовлетворяющие условиям записи.

  a) Вызов функции, осуществляющей поиск записей в индексе и копирование удовлетворяющих условиям записей в блок данных.

  b) Проверка значения, возвращаемого функцией поиска в индексном файле. При ошибочном значении программа завершает работу с выводом поясняющего сообщения в стандартный поток ошибок.
  Возвращаемое значение EXIT_FAILURE

  c) Заполненный блок записей передается функции вывода.

  d) Проверка значения, возвращаемого функцией вывода. При ошибочном значении программа завершает работу с выводом поясняющего сообщения в стандартный поток ошибок.
  Возвращаемое значение EXIT_FAILURE

- Вызов функции закрытия используемых баз данных.
- Возврат значения EXIT_SUCCESS

7.5 Функция вывода блока записей.

Тело данной функции расположено в отдельном файле, используемом при сборке фильтра.
Название функции: print_records().
Файл: print_records.c

Необходимые для работы заголовочные файлы:

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
```

7.5.1 Описание функции:

Получаемые параметры:
идентификатор файла, в который будет осуществлён вывод.

указатель на структуру, содержащую запись.

Прототип функции:

```
int print_records (const int *fd_output,
                   /*
                    * файловый дескритор, в который будут
                    * выведены записи
                    */

                  unsigned char **block_of_records,
                    /* блок записей для вывода */

                  const int rec_count)
                    /* количество записей для вывода */
```

7.5.2 Схема работы функции:

- Вывод поясняющего заголовка.
- Цикл по всем переданным записям.
  а) Цикл по всем полям записи.
  - Установление границ поля в записи.
  - Применение к числу необходимых преобразований.
  - Вывод поля. В случае возникновения ошибки на каком-либо этапе - завершение работы.
    Возвращаемое значение EXIT_FAILURE
- Завершение работы функции.
  Возвращаемое значение EXIT_SUCCESS

7.6 Функция открытия используемых баз данных.

Описана отдельно как функция, используемая в обеих программах.

7.7 Функция закрытия используемых баз данных.

Описана отдельно как функция, используемая в обеих программах.

7.8 Функция, осуществляющая поиск записей в индексном файле.

Данная функция осуществляет поиск записей, соответствующих условиям фильтрации, в базе данных, созданной с помощью библиотеки BerkeleyDB.
Название: search_records()
Описана в файле: search_records.c

Необходимые для работы заголовочные файлы:

```
#include <stdlib.h>
#include <string.h>

#include <db.h>
```

7.8.1 Описание функции:

Получаемые параметры:
Блок записей, в который будут скопированы записи – unsigned char ** block_of_records
Структура, описывающая условия поиска – struct search_param *search_parameters
Структура, содержащая переменные, необходимые для работы с первичной базой данных – struct db_description *db_vars
Массив структур, содержащих переменные, необходимые для работы со вторичными базами данных – struct attribute_db **db_attrib

Прототип функции:

```
int
search_records (  /* block of records to fill */
                  unsigned char **records_block,

                  /* search borders and flags */
                  struct search_param *search_parameters,

                  /* primary database related variables */
                  struct db_description *db_vars,

                  /* secondary database related variables */
                  struct attribute_db **db_attrib )
```

7.8.2 Схема работы функции:

1) Если функция вызвана в первый раз в текущем процессе - инициализация необходимых переменных.

   - Если есть условия для полей, по которым построены вторичные базы данных, инициализация курсоров для работы с ними, а затем объединение этих курсоров.

   - В противном случае установка курсора для поиска по первичной база данных.

2) Если инициализация не удалась - возврат значения -1.

3) Цикл по числу записей в блоке.

   - Извлечение очередной записи с использованием курсора певичной или объединенных курсоров вторичных баз данных.
   - Проверка того, что запись удовлетворяет условиям поиска.
   - Если запись подходит, она добавляется в блок. Переход на очередную итерацию.
   - Если запись не подходит - поиск прекращается. Возвращается число записей добавленных в блок.
   - При возникновении ошибок - прекращение работы.

Глава 8

Архитектуры функций, общих для обеих утилит.

8.1 Архитектура функции чтения и анализа шаблона записи.

Данная функция расположена в отдельном файле, используемом при сборке как фильтра, так и индексатора.
Название функции: process_template(); файл: process_template.c

8.1.1 Описание функции:

Получаемые параметры:
строка, содержащая путь к файлу, описывающему шаблон записи.

```
char *path_to_template;
```

указатель на структуру

Функция при завершении своей работы возвращает вызывающей подпрограмме число, определяемое следующими определёнными в заголовочном файле stdlib.h константами:
EXIT_SUCCESS при успешном завершении.
EXIT_FAILURE при ошибке в работе.

Прототип функции:

```
int process\_template (const char *path\_to\_template)
```

Для открытия файла используется функция:

```
FILE *fopen(const char *path,   /* путь к файлу */
            const char *mode);  /* режим, в котором следует открыть файл */
```

Возвращаемое значение - указатель на файловый поток.

Для чтения строки из файла используется функция:

```
char *fgets(char *string,
              /* переменная, в которую будет осуществлено чтение строки */
            int size,
              /* максимальное число символов, которое можно считать в строку */
            FILE *stream);
              /* файловый поток, из которого осуществляется чтение */
```

Для разбора строки файла на лексемы используется следующая функция:

```
    char *strtok(char *string,        /* анализируемая строка */
                 const char *delim);  /* разделитель лексем */
```

Возвращаемое значение - очередная лексема.

Детальное описание всех переменных и функции содержится на странице справочного руководства в unix системах. (man 3 strtok, man 3 fopen, man 3 fgets)

Необходимые для работы заголовочные файлы:

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
```

8.1.2 Схема работы функции:

- Производится попытка открытия файла, содержащего шаблон. В случае ошибки - завершение работы функции, возвращаемое значение: EXIT_FAILURE
- Установка переменной token_separator, которая определяет разделитель поля в одной записи.
- Выделяется память для структуры attribute_description, описывающей формат представления поля. Если невозможно выделить память для этой структуры, то программа завершает свою работы с возвращаемым значением EXIT_FAILURE.

- В цикле производится построчное чтение файла шаблона.

  a) В цикле производится разбор строки на лексемы.
  Каждая прочитанная строка разбивается на лексемы с использованием разделителя. Строка, содержащая признак комментария, пропускается.

  – В ходе чтения файла поля структуры attribute_description заполняются соответствующими значениями. Если для поля указано, что по нему нужно построить индекс, то соответствующий элемент структуры secondary_db_build_flag принимает значение true, если не нужно строить индекс - значение false.

  b) Заполненная структура, описывающая поле, добавляется в конец массива специальной функцией add_rec_to_array. В случае ошибки - завершение работы.
  Возвращаемое значение: EXIT_FAILURE

- На основании счётчиков числа полей и числа байт в записи заполняются поля структуры record_description с помощью соответствующей функции set_rec_description.

- Завершение работы функции.
  Возвращаемое значение: EXIT_SUCCESS

8.2 Архитектура функции добавления структуры, описывающей поле, к массиву.

Данная функция расположена в файле с функцией чтения и анализа шаблона и используется при сборке как фильтра, так и при сборке индексатора.
Название функции: add_rec_to_array(); файл: attr_description.c

8.2.1 Описание функции:

Получаемые параметры:
Указатель на структуру, описывающую одно поле записи. struct_attr_description *next_attr_description

Функция при завершении своей работы возвращает вызывающей подпрограмме число, определяемое следующими определёнными в заголовочном файле stdlib.h константами:
EXIT_SUCCESS при успешном завершении.

EXIT_FAILURE при ошибке в работе.

Прототип функции:

```
int add_rec_to_array (struct_attr_description *next_attr_description)
```

Для переопределения размера выделяемой памяти для массива структур используется функция:

```
void *realloc (void *ptr,      /* Указатель на область переопределяемой памяти */
               size_t size);   /* Новый размер выделенной памяти в байтах */
```

Возвращаемое значение - указатель на выделенную область памяти.

Детальное описание функции содержится на странице справочного руководства в unix системах. (man 3 realloc)

Необходимые для работы заголовочные файлы:

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
```

8.2.2 Схема работы функции:

- Получение длины записи
- Переопределение области памяти для массива структур с помощью вызова realloc, если невозможно выделить память, то завершение работы программы с возвращаемым значением EXIT_FAILURE
- Заполнение полей последнего элемента массива структур значениями из полученной в качестве параметра структуры
- Завершение работы функции.
  Возвращаемое значение: EXIT_SUCCESS

8.3 Архитектура функции, осуществляющей поиск номера атрибута в массиве по названию атрибута.

Название функции: get_attr_by_name();
Файл: attr_description.c

8.3.1 Описание функции:

Получаемые параметры:
Указатель на строку, представляющей собой название атрибута. char *symbolic_name

Результат: Функция при завершении своей работы, в случае, если атрибут был найден в массиве, возвращает вызывающей подпрограмме номер атрибута в массиве - число, большее 0. Если атрибут не был найден, то функция возвращает значение -1. (Данное значение выбрано потому, что стандартное значение EXIT_FAILURE незафиксировано на стадии проектирования и, следовательно, может принимать в том числе значения , большие или равные 0, что в данном случае будет означать номер элемента в массиве).

Прототип функции:

```
int get_attr_by_name (char *symbolic_name)
```

Необходимые для работы заголовочные файлы:

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
```

8.3.2 Схема работы функции:

- Инициализация номера проверяемого атрибута i
- В цикле, пока структура из массива attribute_description не будет указывать на NULL Если имя атрибута из i-той структуры совпадает с заданным symbolic_name, то функция завершает свою работы и возвращает значение i
- Функция завершает свою работу и возвращает значение -1 (это означает, что проверены все элементы массива и среди них не найдено атрибута с названием symbolic_name.

8.4 Архитектура функции, возвращающей описание атрибута с определённым номером.

Название функции: get_attr_by_num();
Файл: attr_description.c

8.4.1 Описание функции:

Получаемые параметры:
Порядковый номер структуры, описывающей атрибут, в массиве * g_attr_description;
int attr_num;

Результат: Указатель на структуру, описывающую атрибут с указанным номером (struct attr_description *). В случае ошибки возвращается NULL.

Прототип функции:

```
struct attr_description * get_attr_by_num(int attr_num);
```

8.4.2 Схема работы функции:

- Если полученный номер не положителен или превышает число атрибутов записи - завершение работы.
  Возвращаемое значение NULL.
- Возвращение указателя с номером attr_num в массиве g_attr_description[].

8.5 Архитектура функции, возвращающей число атрибутов в записи.

Название функции: get_rec_attr_count();
Файл: rec_description.c

8.5.1 Описание функции:

Получаемые параметры:
Нет.

Результат:
Число атрибутов в записи, представленной во внутреннем формате.

Прототип функции:

```
int get_rec_attr_count (void)
```

8.5.2 Схема работы функции:

- Из глобально объявленной структуры

```
struct record_description  g_rec_description
```

возвращается значение поля attr_count. Упомянутая структура должна быть инициализирована функцией set_rec_description().

8.6 Архитектура функции, возвращающей длину записи.

Название функции: get_rec_length();
Файл: rec_description.c

8.6.1 Описание функции:

Получаемые параметры:
Нет.

Результат:
Длина (в байтах) записи о потоке во внутреннем представлении.
00 Прототип функции:

```
int get_rec_length (void)
```

8.6.2 Схема работы функции:

- Из глобально объявленной структуры

```
struct record_description  g_rec_description
```

возвращается значение поля lenght. Упомянутая структура должна быть инициализирована функцией set_rec_description().

8.7 Архитектура функции, устанавливающей поля глобальной структуры, описывающей запись.

Название функции: set_rec_description();
Файл: rec_description.c

8.7.1 Описание функции:

Получаемые параметры:
число атрибутов в записи int attr_count
длина записи в байтах int lenght
Результат:
EXIT_SUCCESS при успешном завершении.
EXIT_FAILURE при ошибке в работе.

Прототип функции:

```
int set_rec_description(const int attr_count, const int lenght);
```

8.7.2 Схема работы функции:

- Если хотя бы одно из полученных значений не положительно - завершение работы.
  Возвращаемое значение EXIT_FAILURE.

- Присвоение значений полям глобально объявленной структуры

  ```
  struct record_description  g_rec_description
  ```

  Имена полей структуры и получаемых параметров совпадают.
  Возвращаемое значение EXIT_SUCCESS.

8.8 Функции-обёртки для системных функций free() и calloc()

Фунции-обертки используются для обработки ошибок, взникающих в результате использования системных функций, которые могут вызывать нестандартные ситуации.
Внутри "оберток"производится вызов соответствующей системной функций, проверка ее работы, возвращаемого значения и выполнение сответствующих действий.
критичных систесных функций

8.8.1 Функция получения памяти

Данная функция выделяет область памяти и очищает её для некоторого количества элементов. Название: _calloc

Описана в файле: wrappers.c
Необходимые для работы заголовочные файлы:

```
#include <stdlib.h>
```

Описание функции: Получаемые параметры:
Количество элементов - size_t nitems
Размер каждого элемента - size_t size
Прототип функции:

```
void *_calloc(size_t nitems, size_t size)
```

Схема работы функции:

- Если в качестве параметра передано отрицательное количество элементов.

  а) Выход, возвращаемое значение - EXIT_FAILURE

- Вызов функции calloc (nitems, size), проверка возвращаемого значения.

  а) Если функция возвращает NULL: выход, возвращаемое значение - EXIT_FAILURE

- выход, возвращаемое значение - указатель на начало выделенной области памяти.

8.8.2 Функция освобождения памяти

Данная функция освобождает область памяти, выделенную при помощи функции _calloc. Название: _free
Описана в файле: wrappers.c
Необходимые для работы заголовочные файлы:

```
#include <stdlib.h>
```

Описание функции: Получаемые параметры:
Указатель на начало освобождаемой области памяти - void *ptr
Прототип функции:

```
void _free(void *ptr)
```

Схема работы функции:

- Если в качестве параметра передан указатель не указывающий на NULL.

  a) Вызов функции free(ptr)

  b) Указатель ptr = NULL

8.9 Функции для работы с BerkeleyDB

```
Для хранения информации используется библиотека BerkeleyDB.
Для хранения файлов баз данных создается директория, имя
которой выбирается пользователем. В этой директории сохраняется
копия шаблона описания полей с именем 'template' и файлы баз данных.
Основная база данных носит имя 'primary.db', вторичные базы данных
носят имена 'имя_индексируемого_поля.db' (пример: ip_dest.db)

Флаги открытия баз данных (функция DB open) для индексатора: DB_CREATE | DB_EXCL.
Флаги открытия баз данных (функция DB open) для фильтра:     DB_RDONLY.

Флаги метода доступа к базе данных: DB_BTREE.

Данные функции расположены в отдельном файле, используемом
  при сборке индексатора и фильтра.
    Название функций:
      open_databases()
      universal_key_extractor()
      close_databases()

Данная функция расположены в отдельном файле, используемом
  при сборке фильтра.
      search_records()

Данная функция расположены в отдельном файле, используемом
  при сборке индексатора.
      add_records_to_db

      Функции open_databases(), close_databases(), universal_key_extractor() являются
      общими для индексатора и фильтра.
      Функция add_records_to_db() используется в индексаторе.
      Функция search_records() используется в фильтре.
```

8.9.1 Функция открытия баз данных

```
int open_databases(const char *db_dir_path_path, const char db_open_mode,
         struct attribute_db *attrib_db,
         struct db_description * primary_db_descr);
  Открыть первичную и вторичные базы данных.

  Получаемые параметры:
    Имя директории в которую будут помещены файлы баз данных BerkeleyDB.
    Режим открытия БД.
    Структуры содержащие дескрипторы БД.

  Возможные режимы открытия БД:
  1. 'w' - режим открытия БД для записи, этот режим используется индексатором
  (флаги открытия DB_CREATE и DB_EXCL).
  2. 'r' - режим открытия БД для чтения, этот режим используется фильтром
  (флаг открытия DB_RDONLY).

  Функция при завершении своей работы возвращает, вызывающей
  подпрограмме число, определяемое следующими определёнными в
  заголовочном файле stdlib.h константами:
      EXIT_SUCCESS при успешном завершении.
      EXIT_FAILURE при ошибке в работе.

Схема работы функции:

int
open_databases(char *db_dir_path, char db_open_mode,
        struct attribute_db *attrib_db,
        struct db_description * primary_db_descr)
{
        ВАЖНО: Вторичные базы данных должны открываться только после УСПЕШНОГО
        открытия первичной БД.

        Установить флаги открытия БД, в соответствии с переданным режимом.

        Инициализировать дескриптор первичной БД, методом db_create.

        Открыть первичную базу данных, методом open (функция BerkeleyDB).

        Если не удалось, то возвращаем EXIT_FAILURE.
```

```
        Цикл по всем полям записи.
        {
                Если надо строить вторичную БД
                {
                        Инициализировать дескриптор вторичной БД.

                        Открываем вторичную БД, если не удалось,
                        то возвращаем EXIT_FAILURE.

                        В app_private(поле структуры DB) вторичной БД  заносим
                        указатель на соответствующую этому полю(индексируемому
                        в этой базе) структуру attribute_description.

                        Связываем БД с соответствующим полем первичной БД
                        методом associate, если не удалось,
                        то возвращаем EXIT_FAILURE.
                }

        }

        Возвращаем EXIT_SUCCESS.
}
```

8.9.2 Функция закрытия базы данных

```
int close_databases(struct attribute_db *attrib_db,
        struct db_description * primary_db_descr);
  Функция для закрытия баз данных.
  Получаемые параметры: структуры содержащие дескрипторы БД.

  Функция при завершении своей работы возвращает, вызывающей
  подпрограмме число, определяемое следующими определёнными
  в заголовочном файле stdlib.h константами:
      EXIT_SUCCESS при успешном завершении.
      EXIT_FAILURE при ошибке в работе.

Схема работы функции:
int
close_databases(void)
```

```
{
        ВАЖНО: Все вторичные базы данных должны закрываться до закрытия
        первичной базы данных.
        Цикл по всем полям записи.
        {
                Если надо строить вторичную БД
                {
                        Закрываем дескриптор вторичной БД.
                }
        }

        Закрываем дескриптор первичной БД.

        Возвращаем EXIT_SUCCESS.
}
```

8.9.3 Функция Universal key extractor

Функция служит для получения значения конкретного поля структуры в зависимости от вторичной базы данных.
Необходима для связывания основной и вторичных баз данных.
Для нормальной работы BerkeleyDB возвращаемое значение функции должно равняться 0.

```
int universal_key_extractor(DB *secondary_db, const DBT *primary_key,
                            const DBT *primary_data, DBT *secondary_key)
{
        Получение ссылки на структуру attribute_description (описание атрибута)
        из поля app_private вторичной базы данных secondary_db.

        Получение длины поля(length) и смещения(offset) из структуры
        attribute_description для нужного поля.

        Обнуление структуры secondary_key.

        Копируем length байт начиная со смещения offset из primary_key->data в
        secondary_key->data.

        Устанавливаем значение secondary_key->size равным length + 1 (терминальный 0
тоже является частью строки).

        Возвращаем 0.
}
```