Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему
УТИЛИТА ПОИСКОВ ОДИНАКОВЫХ ФАЙЛОВ
(аналог fdupes, но с фильтрацией по именам и типам)
БГУИР КР 1-40 02 01 306 ПЗ

Студент: Григорик И. А.

Руководитель: Глоба А. А.

Минск 2022

Оглавление

ВВЕДЕНИЕ

В текущее время у каждого пользователя компьютера, вне зависимости от его опыта работы с компьютером, рано или поздно появляются одинаковые файлы. По различным причинам пользователь может не замечать их, или просто игнорировать, т.к. искать вручную все дублирующиеся файлы не так уж и просто. Именно поэтому используются специальные утилиты для автоматического поиска одинаковых файлов.

К примеру, fdupes – утилита, написанная Андрианом Лопесом, может искать одинаковые файлы из любого каталога в системе. Для этого используется получение хеша файла из его дескриптора. Следует пояснить, что такое хеш файла и файловый дескриптор:

- *Хеш файла* – это уникальный идентификатор файла, который высчитывается системой посредством определённых преобразований хранящейся в файле информации.
- *Дескриптор файла* – это целое неотрицательное число, с помощью которого процесс может обращаться к потоку ввода-вывода. Дескриптор может быть связан с файлом, сокетом или каталогом.

С помощью данных средств можно сверять файлы по содержанию, что и является целью поиска одинаковых файлов.

Данный курсовой проект представляет собой такую утилиту командной строки, подобную уже известной утилите fdupes, которая будет искать файлы с одинаковым содержимым, основываясь на хеше файла. Так же данная утилита будет сравнивать расширения файлов и их имена, помимо содержимого. Так же, с помощью некоторых управляющих флагов, утилита сможет производить действия над данными файлами, помимо простого поиска.

В рамках данной курсовой работы необходимо ознакомится с библиотекой openSSL/md5.h, которая используется для получения хеша файлов, или же с командой MD5, которые используются для получения MD5 хеша, с помощью которого файлы и будут проверяться на идентичность. В процессе разработки следует углубить знания по языку C / C++, а также осуществить взаимодействие пользовательского приложения с системой. В конце следует протестировать приложение и провести эксперименты на нескольких устройствах.

OpenSSL – полноценная криптографическая библиотека, с открытым исходным кодом, которая используется во многих проектах для хеширования MD5, MD2, SHA. Библиотека написана Эриком Янгом и Тимом Хадсоном, и получила популярность благодаря расширениям SSL/TLS, которые используются в веб-протоколе HTTPS.

Для качественного выполнения курсового проекта следует рассмотреть аналоги данной утилиты. Ключевой аналог – так же консольная утилита *fdupes*. Данная утилита использует хеш файлов, для их сравнивания. Утилита написана полностью на языке программирования C, и находится под лицензией MIT. Она же является примером подражания моего курсового проекта.

Следующая утилита – *CloneSpy*. Это так же бесплатный инструмент для очистки дискового пространства от дублирующихся файлов. Так же может находить файлы нулевой длины, у которых нету содержимого. Данный аналог представляет собой приложение с GUI, что не является целью проекта. Так же приложение разрабатывалось под операционную систему (ОС) Windows, что так же не предпочтительно.

FSlint – так же утилита с графическим интерфейсом, но для ОС Linux. Данная утилита представляет собой поиск одинаковых файлов, но с расширенным функционалом. Так же тут присутствуют флаги поиска для одинаковых имён, архивов и пустых директорий.

Рассмотрев все аналоги можно сделать вывод, каковой должна быть программа для корректной конкурентоспособности. Данная утилита должна объединять в себя лучшие качества вышеперечисленных программ, коими были выбраны следующие:

– Обязательный поиск одинаковых файлов из конкретной директории.
– Быстрый поиск файлов, посредством сравнивания их хеша.
– Присутствие флагов управления для поиска одинаковых имён.

Данный функционал является минимальным требованием, которое в последующем будет дополнено.

1 ОБЗОР ЛИТЕРАТУРЫ

Следует начать с определения утилиты.

Утилита – это некая вспомогательная компьютерная программа в составе программного обеспечения, которая используется для выполнения типовых задач, связанной с работой оборудования или ОС. В данном случае утилита используется для облегчения пользования компьютером.

Данная утилита основывается на получении хеша файла из его дескриптора, определения которых давались выше. Процедура получения хеша называется *хешированием*.

Хеширование может проводиться по разным алгоритмам. Основной смысл хеширования заключается в безопасности и надёжности, возможности сжимать любые куски информации в короткий стандарт сообщений, являющихся уникальными для каждой доли информации. Даже если один байт информации будет изменён – хеш так же поменяется.

Таким образом не надо хранить всё содержимое файла или каждый раз его открывать, чтобы сверить его информацию с другим файлом. Достаточно будет просто сгенерировать его хеш и запомнить его.

Недостатком хеширования является неизбежность коллизии.

Коллизия – это равенство значений хеш-функций на двух различных кусках информации. В данном случае это означает, что если функция сгенерирует одинаковый хеш для двух разных файлов, то в системе они будут считаться за одинаковые. Для решения вопроса коллизий создаются современные хеш-функции, в которых шанс появления коллизий стремится к минимуму.

Так же стоит пояснить, что длина строки зависит от конкретной хеш-функции, но одна функция не может сгенерировать две строки разной длины.

На данный момент популярны следующие хеш-функции:

- *SHA256* – одна из наиболее устойчивых к коллизиям функция. Недостаток: по сравнению с другими, имеет довольно большое время выполнения и большая длина хеш-слова (256 байт).
- *RIPEMD160* – так же устойчивая к коллизиям функция, которая, к тому же, имеет длину хеш-слова почти в два раза меньше (160 байт), чем SHA256. Время выполнения примерно такое же, как у SHA256.
- *MD5* – самая быстрая криптографическая хеш-функция из широко используемых, к тому же имеет наименьший размер хеша (128 байт). Недостаток: небезопасна. Легко подвергается коллизиям, поэтому не стала использоваться в проектах, по типу криптовалютных кошельков.

Так же хеширование используется для сокрытия данных, так как хеширование одностороннее (т.е. нельзя преобразовать хеш в первоначальные данные). Получить первоначальные данные можно только сгенерировав такую же строку, или создать коллизию, которая приведёт конвертации другой строки

В данном случае сокрытие не требуется, и могут допускаться коллизии, поэтому может использоваться функция хеширования MD5.

Теперь же о файловых дескрипторах.

Файловый дескриптор в данном случае используется для отображения файла на память. Данный метод является эффективным, ибо помогает разгрузить систему, и вообще не использовать физическую память, чем помогает снизить нагрузку на диск для нескольких программ, обращающимся к одному и тому же файлу.

Так же следует заметить, что ОС Linux придерживается правила «всё есть файл», поэтому тип файла тут – понятие, которое отличается и часто путается с расширением файла в ОС Windows. В Linux существуют всего три типа файлов: обыкновенные, специальные и директории. Следовательно, можно сделать вывод, что условие реализации фильтрации поиска по типам файла будет реализовываться относительно их расширения, что является подтипом обыкновенных файлов.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Сначала следует поставить конкретные функциональные требования разрабатываемой программе, разбить утилиту на модули и функциональные блоки. Данный подход в большинстве упростит понимание проектирования, сможет помочь устранить проблемы в архитектуре и обеспечить гибкость каждого из модулей. После того, как аналоги рассмотрены и проведен краткий экскурс в основные понятия, можно поставить конкретные цели разрабатываемому программному обеспечению.

2.1 Постановка задачи

Минимальное требование – просто поиск одинаковых файлов с определённой директории, с из выводом на экран. Так как в условии курсового проекта сказано, что должна быть фильтрация по типам, то это так же будет являться обязательным условием выполнения. В качестве поиска будет использоваться алгоритм хеширования MD5 из-за своей скорости и маленькой длине хешируемого слова, что является условием быстрого выполнения утилиты. Так же для оптимизации будут использоваться отображения в память, для упрощённого, для системы, получения хеша файла. Данный подход поможет не затрачивать лишнее дисковое пространство и в несколько раз ускорить работу утилиты. Так как присутствует условие фильтрации по типам, то утилита должна подстраиваться под передаваемые ей пользовательские флаги, следовательно, должен быть установлен обработчик входных флаговых значений.

Из вышеперечисленных методов можно составить определённые модули программы, которые будут обеспечивать полную функциональность выполнения. Так же следует выделить, что в данном случае расширение файла, и его имя является одним и тем же, следовательно, для этого не имеет смысла делать двух разных флагов, и будет организован только один флаг: –n (name). Так же к флагам добавляется флаг статистики (–s), и флаг удаления файлов (–d). Первый будет использоваться для показа текущей статистики сбора файлов, второй для полного удаления дубликатов файлов после их поиска. Дополнительный флаг – флаг сбора всех файлов –a (all), используется в различных утилитах, для поиска системных файлов, или скрытых файлов, начинающихся с точки. Последний флаг – флаг примера (–t), который будет отображать корректное поведение программы. С него и стоит начать описание модулей программы.

2.2 Разбиение программы на модули

Для корректного решения излагаемой проблемы всю программу следует разбить на модули, которые впоследствии будут реализованы в функциях или блоках кода. Модулем будет являться полноценная функция или подфункция, а блоком – блок кода, который может содержаться в модуле.

2.2.1 Модуль тестирования

Данный модуль будет полностью симулировать применение программы. Он будет представлять собой открытие какого-то каталога, создания там нескольких файлов с одинаковым содержимым, выводом этого содержимого на экран, задержкой для проверки содержимого и удалением или поиском одинаковых файлов. Модуль необходим для отображения корректного поведения программы, и будет использовать максимальное количество модулей, описанных ниже.

2.2.2 Модуль хранения данных о файлах

Данный модуль будет представлять собой некий массив самостоятельно написанной структуры, которая будет использоваться для хранения всех метаданных о каждом проверяемом файле. В данном случае метаданными будут служить хеш файла и его имя, для уникальных файлов, и имя файла и путь до него, для дублирующихся файлов. Данный модуль будет представлять из себя два вектора данных структур.

2.2.3 Модуль сбора информации о файлах

Модуль блок будет представлять собой рекурсивную функцию, используемую для прохождения по всем файлам с задающей директории. Модуль необходим для обновления информации модуля хранения данных о файлах. В нём же можно выделить ещё несколько блоков и модулей, которые будут описаны в заголовках **2.2.5**, **2.2.6** и **2.2.7.**

2.2.4 Блок чтения информации о файле

Блок будет реализовывать собой отображение данных файла в виртуальное адресное пространство, после чего будут задействован блок получения хеша файла и модуль преобразования хеша в шестнадцатеричную систему счисления (СС).

2.2.5 Блок хеширования данных файла

Данный блок кода просто будет представлять собой получения хеша определённого файла основываясь не его отображении в виртуальном адресном пространстве. После выполнения этого блока необходимо освободить адресное пространство.

2.2.6 Модуль преобразования хеша

Данный модуль преобразовывает полученный хеш, который может состоять из нечитаемых символов или же из символов, неудобных для работы, в

строку, определённой длины, состоящую из шестнадцатеричных чисел. Преобразование необходимо для корректной работы, ибо данные в неудобных форматах для чтения могут неправильно сравниваться.

2.2.7 Модуль обработки флагов.

Модуль представляет собой преобразование входных данных в определённые флаги, которые в дальнейшем будут использоваться в блоке проверки данных и выводе.

2.2.8 Блок проверки о хранении файла.

Блок является проверкой данных конкретного файла со всеми файлами, которые хранятся в модуле хранения данных. Модуль может изменяться, в связи с изменением некоторых флагов. В случае выполнения определённых условий, данные файла будут заноситься в вектор дублирующихся файлов.

2.2.9 Модуль удаления файлов

Модуль является простым считыванием вектора дублирующихся файлов и их последующего удаления. Так же возможно не полное, а выборочное удаление файлов.

Все вышеперечисленные модули позволяют обеспечить полноту выполняемых действий, соответственно необходимы для выполнения курсовой работы.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Данная глава будет представлять собой ключевой раздел, дающий понимание работы моей утилиты, её структуру, с точки зрения описания отдельных функций и модулей, обработки данных, с приведением листинга и структурной диаграммой, вынесенной в приложение «А».

3.1 Описание структур утилиты

Ключевыми структурами утилиты являются file_data, file_to_delete и flags.

Первые две структуры представляют собой шаблоны для хранения имени файла и его хеша или пути до него, в зависимости от структуры. Они разделены, и по логике программы, объект, находящийся в массиве с типом данных одной структуры не может находится в массиве, с типом данных другой. Данные структуры являются ключевыми в поиске и удалении файлов, тем не менее их можно рассматривать и по-отдельности.

3.1.1 file_data

Структура представляет собой всего два поля – сгруппированные данные о файле, из которых можно определить его уникальность. Поля:

– std::string file_name – строка, которая хранит имя файла с его относительным путём.
– std::string file_hash – строка, хранящая хеш файла в шестнадцатеричном коде, с помощью которой и определяется уникальность файла.

С помощью всего лишь двух полей можно реализовать систему поиска одинаковых файлов. Но если бы дублирующиеся файлы никуда не записывались, то смысл этой структуры и утилиты в целом теряется, так что для полного функционала необходима следующая структура:

3.1.2 file_to_delete

Данная структура является минимальным требованием, для полного удаления файла из системы. Она так же содержит всего два поля, которые позволяют удалить файл. Поля:

– std::string file_name – необязательное поле, используемое для понятного представления. Хранит имя файла, которое в последующем используется для показа удаляемого файла.
– std::string file_path – строка, хранящая относительный путь до файла, с помощью которой и происходит удаление файла в модуле удаления. Обязательна для данной структуры.

Кроме этого в данном проекте представлена система управления программой посредством флагов, поэтому необходимость структуры, содержащей флаги в виде булевых переменных крайне важна.

3.1.3 flags

Структура flags будет реализовывать собой простой набор булевых переменных, которые будут выставляться, в зависимости от ввода пользователя. Флаги же могут выставляться как отдельно, так и совместимо. Это будет описано в полях:

- bool stats – вывод статистики по собираемым файлам как в момент сборки и поиска, так и в момент конечного вывода.
- bool name_flag – флаг фильтрации по именам. Позволяет управлять сборкой информации о файлах.
- bool delete_flag – флаг для удаления повторяющихся файлов. Активирует модуль удаления, который без выставления данного флага будет являться недействительным.
- bool all_files – флаг, позволяющий собирать информацию в скрытых файлах, так же известных как dot-файлы в Linux.
- bool test_flag – данный флаг позволяет полностью симулировать работу программы. Указываемый путь после этого файла не будет использоваться и не обязателен.

Стоит отметить, что все вышеперечисленные флаги могут совмещаться, но отдельный флаг test_flag будет игнорировать все поставленные флаги, т.к. после его установки пользователь ожидает увидеть полную работу программы со всеми доступными флагами по очереди. Чтобы сделать это возможным программа последовательно выставляет все флаги (по одному), и запускает модули поиска дублирующихся файлов, который в себе вызывает остальные блоки и подмодули. Данные модули будут описываться в следующем разделе.

3.2 Описание модулей и блоков программы

Все вышеописанные структуры являются полностью бесполезными, без средств работы с ними, так называемыми «модулями», или функциями программы, которые содержат блоки кода. Рассматривать их следует от самого простого к самому сложному:

3.2.1 get_dir

Данный модуль один из самых простых, и позволяет преобразовать строку, введённую пользователем одной из аргументов командной строки, в правильный формат. К примеру, если введена директория с относительным путём /home/, то она преобразуется в строку /home. Этот модуль необходим для корректного открытия директории и вывода пути до файла в будущем.

3.2.2 parse_flags

Модуль представляет собой функцию, возвращающую структуру типа file. Получает аргументы командной строки и их количество. Создаётся новый

объект структуры file_data, заполняется на основании содержания аргументов командной строки и возвращается.

3.2.3 md5_to_stirng

Функция представляет собой конвертацию не всегда читаемых символов в строку определённой длины (длины хеш-слова md5). Строка конвертируется посимвольно путём преобразования каждого символа в шестнадцатеричную СС. В конце конвертации эта строка возвращается.

3.2.4 get_size_by_fd

Функция получает размер файла, на основе его дескриптора, создавая переменную типа struct stat и вызывая функции fstat, находящиеся в библиотеке sys/types.h, sys/stats.h, unistd.h. Возвращает значение поля st_size структуры stat, в удачном случае, и завершает программу с кодом –1 в неудачном.

3.2.5 collect_files

Этот модуль является одним из ключевых. Представлен в виде функции, которая может вызываться рекурсивно, и получает на вход ссылки на массивы структур file_data, file_to_delete и flags, которые используются для сбора информации и её фильтрации. В целом структура модуля разбивается на вышеописанные (3.2.(3-4)), и внутренние условия. Изначально модуль создаёт некоторые переменную dir – указатель на директорию, в которой работает программу. Далее создаётся поток каталога, в котором работает программа, и пока он не будет полностью считан – не произойдёт выход из функции. Этот цикл и называется модулем сбора информации, внутри которого и выполняются все блоки программы (2.2.4 – 2.2.8). Данные блоки будут описываться ниже:

3.2.6 Блок чтения информации о файле

Если файл существует и выполняются все условия (файл не «..»/ «.» и не директория), то программа будет читать файловый дескриптор, и получать посредством вызова get_size_by_fd длину файла и в случае, если она меньше 1 Гб отображает файл в память и активирует следующий блок:

3.2.7 Блок получения хеша файла

Блок представляет собой вызов функции MD5 из библиотеки openssl/md5.h, которая заполняет строку result – строку для записи хеша, на основе отображения файла в памяти и его размера. После успешного хеширования удаляет отображение файла из виртуальных адресов и конвертирует хеш в строку, с помощью модуля md5_to_string. После этого выводится статистика, если подобный флаг был выставлен и обработан в parse_flags, и функция приступает к главному: блоку проверки о хранении файла.

3.2.8 Блок проверки о хранении файла

Данный блок является циклом, проходящим по массиву unique_files со структурой file_data, что позволяет сравнить конкретный файл со всеми хранящимися в массиве. Блок сверяет хеш текущего файла и хранимого. В случае эквивалентности, информация о файле заносится в массив files_to_delete. Если же все файлы проверены, то информация заносится в массив unique_files. Если же в массиве ничего не хранится – то файл записывается, и программа переходит к другому файлу.

ЛИТЕРАТУРА

1. Лафоре, Р. Объектно-ориентированное программирование в C++ / Р. Лафоре. – СПб. : Питер, 2004.
2. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Джекобсон; пер. с англ. – СПб. : ДМК, 2004.
3. Лав Р. Системное программирование на Linux/ 2-е издание 2014.
4. Керниган Б. Язык программирования С/ 4-е издание М.:Питер, 2004. – 923 с.
5. Рочкинд М. Программирование для UNIX, 2-е изд. СПб, БХВ-Петербург, 2005.
6. Калле Р. Грокаем технологию биткоин / Р. Калле – СПб. : Питер, 2020.

Контрльный лист (то что помещается 40 строк на лист)
111111111111111111111111111111111111111111111111111111111111111111
112222222222222222222222222222222222222222222222222222222222333333
333333333333333333333333333333333333333333333333333333333333344444
444444444444444444444444444444444444444444444444444444444444555555
555555555555555555555555555555555555555555555555555555555555556666
666666666666666666666666666666666666666666666666666666666666666777
777777777777777777777777777777777777777777777777777777777777777777
777888888888888888888888888888888888888888888888888888888888889999
999999999999999999999999999999999999999999999999999999999999999900
000000000000000000000000000000000000000000000000000000000000000001
111111111111111111111111111111111111111111111111111111111111111222
222222222222222222222222222222222222222222222222222222222222222222
333333333333333333333333333333333333333333333333333333333333333444
444444444444444444444444444444444444444444444444444444444444444445
555555555555555555555555555555555555555555555555555555555555555666
666666666666666666666666666666666666666666666666666666666666666667
777777777777777777777777777777777777777777777777777777777777777778
888888888888888888888888888888888888888888888888888888888888888888
999999999999999999999999999999999999999999999999999999999999999999
000000000000000000000000000000000000000000000000000000000000000000
111111111111111111111111111111111111111111111111111111111111111122
222222222222222222222222222222222222222222222222222222222222222223
333333333333333333333333333333333333333333333333333333333333333333
344444444444444444444444444444444444444444444444444444444444444444
455555555555555555555555555555555555555555555555555555555555555556
666666666666666666666666666666666666666666666666666666666666666667
777777777777777777777777777777777777777777777777777777777777777888
888888888888888888888888888888888888888888888888888888888888888889
999999999999999999999999999999999999999999999999999999999999999999
000000000000000000000000000000000000000000000000000000000000111111
111111111111111111111111111111111111111111111111111111111111111222
222222222222222222222222222222222222222222222222222222222222222233
333333333333333333333333333333333333333333333333333333333333333333
444444444444444444444444444444444444444444444444444444444444444444
455555555555555555555555555555555555555555555555555555555555555566
666666666666666666666666666666666666666666666666666666666666777777
777777777777777777777777777777777777777777777777777777777777777788
888888888888888888888888888888888888888888888888888888888888888889

999999999999999999999999999999999999999999999999999999999999999999999000
0000000000000000000000000000000000000000000000000000000000000000000000